

Mihaiela-Ana Lupea

Andreea-Diana Mihiș

A Computational Approach to Classical Logics and Circuits

Editura RISOPRINT
Cluj-Napoca • 2015

© 2015 RISOPRINT

Toate drepturile rezervate autorilor & Editurii Risoprint

*Editura RISOPRINT este recunoscută de C.N.C.S.
(Consiliul Național al Cercetării științifice).
www.risoprint.ro www.cnccs-uefiscd.ro*



Opiniile exprimate în această carte aparțin autorului și nu reprezintă punctul de vedere al Editurii Risoprint. Autorul își asumă întreaga responsabilitate pentru forma și conținutul cărții și se obligă să respecte toate legile privind drepturile de autor.

Toate drepturile rezervate. Tipărit în România. Nicio parte din această lucrare nu poate fi reproducă sub nicio formă, prin niciun mijloc mecanic sau electronic, sau stocată într-o bază de date fără acordul prealabil, în scris, al autorului.

All rights reserved. Printed in Romania. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the author.

**Descrierea CIP a Bibliotecii Naționale a României
LUPEA, MIHAIELA-ANA**

A computational approach to classical logics and circuits / Mihaela-Ana Lupea, Andreea-Diana Mihiș. - Cluj-Napoca : Risoprint, 2015

Bibliogr.

Index

ISBN 978-973-53-1636-5

I. Mihiș, Andreea-Diana

004.312(075.8)

004.312:62.001.63(075.8)

Director editură: GHEORGHE POP
Consilier editorial: VOICHIȚA-MARIA CLINCI

*Referenți științifici:
Prof. univ. dr. DOINA TĂTAR
Prof. univ. dr. GABRIELA CZIBULA*

Tiparul executat la:
S.C. ROPRINT®S.R.L.

400 188 Cluj-Napoca • Str. Cernavodă nr. 5-9
Tel./Fax: **0264-590651** • roprint@roprint.ro

PREFACE

The purpose of this book is to present the logical foundations of computer science: classical logics and logic circuits.

Fundamental concepts and results of classical logics are introduced in a formal style and in an explicitly computational way. Applications to automated theorem proving in propositional logic and first-order logic are presented. The studied proof methods are: the resolution method, the semantic tableaux method and the sequent/anti-sequent calculi. The formalization of mathematical reasoning and human reasoning using propositional logic and first-order logic is also an objective of this paper.

The design of logic circuits, based on Boolean functions is an important part of this book.

The paper combines the theoretical presentation of classical logics and logic circuits with numerous examples explained and a rich base of proposed exercises.

Chapter 1 is dedicated to *propositional logic*. The semantic issues discussed are: truth tables, validity, consistency, inconsistency, logical equivalence, logical consequence, normal forms. From a syntactic perspective propositional logic is introduced as an axiomatic (deductive) system, with the purpose of reasoning modeling.

First-order predicate logic is the topic of **chapter 2** of the paper. A Hilbert axiomatic system is used to present predicate calculus in a syntactic approach. The semantics of predicate logic is introduced in order to provide a meaning in terms of the modeled universe for each formula from the language. Normal forms, substitutions and unifiers used in predicative resolution are discussed. The method based on Herbrand's theorem is applied with the aim of reducing the problem of inconsistency of a set of predicate formulas to the problem of inconsistency in propositional logic.

Chapter 3 treats the *semantic tableaux method*, a refutation proof method. The formulas are decomposed in order to determine their models. The classical approach through graphic representation using a binary tree is suggestive, but hard to implement. The new approach, using the TP function allows an easy and efficient implementation of this proof method. An automated theorem prover based on this method, including the corresponding algorithms and implementation details are presented.

Chapter 4 presents two complementary axiomatic systems: *the sequent* and *anti-sequent calculi*. As syntactic and direct proof methods, they are used to check

the validity/derivability and non-validity/non-derivability in propositional logic and first-order logic.

The topic of **chapter 5** is *resolution*, a syntactic and refutation proof method, very efficient and easily to implement. Resolution is introduced as an axiomatic (formal) system and as a procedure. In order to increase the efficiency of the resolution process, the strategies (deletion, set-of-support, unit preference, level saturation, linear) and the refinements (lock, semantic, linear-ordered) of resolution are studied.

In **chapter 6** examples of reasoning modeling in mathematics and daily life, using propositional logic and predicate logic, are presented. An approach of the program verification task using resolution in first-order logic is also described.

Chapter 7 introduces the basic concepts of *Boolean algebras* and *Boolean functions* and presents three *simplification methods* of Boolean functions. Veitch-Karnaugh diagrams graphical method, Quine's analytical method and Moisil's algebraic method are described and applied to numerous examples.

Chapter 8 deals with *logic circuits*. Defined on Boolean algebras, Boolean functions are very useful in the design of logic circuits. Some of the combinational circuits used in the hardware of computers (comparator, adder, subtractor, encoder, decoder) are presented.

By its content, this book is usefull to all those interested in classical logics and logic circuits, fundamental areas of computer science. Professionals in computer science are offered a theoretical basis in the applicative direction of building automated proof systems used in mathematics, software engineering, intelligent agents, robotics, natural language, artificial vision.

We wish to acknowledge our deep appreciation to Prof. Dr. Doina Tătar for many valuable scientific discussions and guidance during all the years of study and research in the field of classical logics. Special thanks for all her constructive comments made during the preparation of this paper.

Cluj-Napoca
2015

Mihaiela-Ana Lupea
Andreea-Diana Mihiş

A Computational Approach to Classical Logics and Circuits

CONTENTS

Introduction	7
1. Propositional logic	9
1.1. Syntax	9
1.2. Semantics of propositional logic.....	9
1.3. Logical equivalences.....	13
1.4. Normal forms in propositional logic	18
1.5. Formal (axiomatic) system of propositional logic	22
1.6. The theorem of deduction and its reverse	26
1.7. Properties of propositional logic	29
1.8. Decision problems and proof methods.....	30
1.9. Exercises	31
2. First-order logic	35
2.1. The axiomatic (formal) system of first-order logic	35
2.2. Transformation of natural language sentences into predicate formulas...	41
2.3. The semantics of first-order (predicate) logic	43
2.4. Logical equivalences in predicate logic	46
2.5. Normal forms in first-order logic.....	49
2.6. Substitutions and unification.....	52
2.7. Herbrand – based procedure.....	56
2.8. Exercises	61
3. Semantic tableaux method	67
3.1. A Classical Approach to the Semantic Tableaux Method.....	67
3.2. A New Approach to the Semantic Tableaux Method	81
3.3. An Automated Theorem Prover for Propositional Logic, based on the Semantic Tableaux Method.....	86
3.4. Considerations for implementation	89
3.5. Exercises	93
4. Sequent and Anti-Sequent calculi	97
4.1. The sequent calculus	97
4.2. The anti-sequent calculus.....	106
4.3. Exercises	115
5. Resolution proof method.....	118
5.1. Resolution method for propositional logic	118
5.2. Strategies for propositional resolution	122
5.3. Lock resolution	126
5.4. Linear resolution	131

Contents

5.5	Resolution in first-order (predicate) logic	136
5.6	Semantic resolution	143
5.7	Exercises	151
6.	Reasoning modeling and program verification.....	155
6.1.	Common-sense reasoning modeling using propositional logic.....	155
6.2.	Reasoning modeling using predicate logic	162
6.3.	Program verification using predicate logic	178
7.	Boolean algebras and Boolean functions	184
7.1.	Boolean algebras	184
7.2.	Boolean functions	186
7.3.	Canonical forms of Boolean functions.....	188
7.4.	Simplification of Boolean functions	193
7.5.	Veitch-Karnaugh diagrams method	196
7.6.	Quine-Mc'Clusky's method.....	208
7.7.	Moisil's simplification method	215
7.8.	Application.....	217
7.9.	DCF versus CCF in simplification.....	218
7.10.	Exercises	221
8.	Logic circuits.....	224
8.1.	Basic concepts.....	224
8.2.	Examples of useful combinational circuits	233
8.3.	Exercises	244
	Index	248
	Bibliography	252

A Computational Approach to Classical Logics and Circuits

INTRODUCTION

The science that studies the principles of reasoning and valid inference is called *logic*. Philosophical logic and mathematical logic are commonly associated with *deductive reasoning*, which determines whether the truth of a conclusion can be determined for an inference rule, based solely on the truth of the set of premises.

In antiquity, earlier philosophers introduced *propositional logic*, but only in the 3rd century B.C. was it developed into a formal logic by the Stoics. In 322 B.C., Aristotle proposed the *syllogism law* that became the dominant model of correct argumentation in philosophy for more than two thousand years. This law was used later as an inference rule in symbolic logic.

The philosopher Gottfried Leibniz (1646 -1716) is considered to be the founder of *symbolic logic*. His research aimed at finding a general decision procedure, but his important results were unknown to the larger logical community.

Completely independent of Leibniz, two well-known mathematicians and logicians reached in their works the same results as Leibniz:

- George Boole: *Mathematical Analysis of Logic, giving an algebraic approach to Aristotelian logic* (1847) and *The Laws of Thought* (1854). He applied methods from the field of symbolic algebra to logic.
- Augustus De Morgan: *Formal Logic* (1847). His major contribution was to the development of the theory of relations and the rise of modern symbolic, or mathematical, logic.

In 1879 Gottlob Frege proposed a formal system called *predicate* or *first-order logic* (FOL), introducing quantified statements and proposing the notion of a *proof* in terms that are still accepted today.

Propositional logic and predicate logic are called *classical logics* and they model a valid type of deductive reasoning. The decision problems in these logics are: to check the validity of a statement and to check if a *conclusion* is derivable (inferable) from a set of statements (*axioms* and *hypotheses*). The inferential process is a monotonic one, meaning that once a conclusion is deduced (derived) from a set of hypotheses (premises), new premises will not invalidate it.

Once the formal framework (syntax and semantics) of logic was developed, the efforts of researchers focused on finding efficient *proof methods* that would solve the decision problems and implement them. The proof methods are classified into *direct methods* and *refutation methods*. The direct methods build the proof of the conclusion directly from the axioms and the hypotheses using the inference rules. The refutation methods are based on the ‘reductio ad absurdum’ principle and the idea is to show that the negation of the conclusion together with the hypotheses and the axioms lead to a contradiction.

Introduction

The invention of *truth-table*, an important tool in propositional logic, is of controversial attribution because many mathematicians-logicians (Frege, Russell, Philo, Boole, Peirce, Schröder, Łukasiewicz, Whitehead, Venn, Lewis) had ideas related to truth-tables. However, the well-known tabular structure is credited to either Ludwig Wittgenstein (1922) or Emil Post (1921).

In 1936 G.K.Gentzen introduced *natural deduction*, a direct and syntactic method, based on an axiomatic system that formalizes the deductive (inferential) process. As an improvement of this method, two complementary axiomatic systems *sequent* and *anti-sequent calculi* were developed and employed to prove the validity/derivability and non-validity/non-derivability respectively in classical logics.

In 1965 J.A.Robinson proposed a refutation and syntactic proof method, simple and easy to implement, called *resolution*. Later, this method was refined (semantic resolution, linear resolution, lock resolution) in order to increase the efficiency.

The *semantic tableaux method* introduced by R. Smullyan in 1968 is based on semantic considerations. The proof by contradiction of the conclusion consists of decomposing the conjunction of the hypotheses and the negation of the conclusion, searching for models and showing that they do not exist.

Automated Theorem Proving (ATP) deals with the development of computer programs which show that some statement (the *conjecture*) is a *logical consequence* of a set of statements (the *axioms* and the *hypotheses*). These automated systems were used in a lot of domains such as mathematics (*EQP*, *Otter*, *Geometry Expert*), software generation (*KIDS*, *AMPHION*), software verification (*KIV*, *PVS*), hardware verification (*ACL2*, *HOL*, *ANALYTICA*). As potential fields we enumerate biology, social science, medicine, commerce.

Also developed were dedicated (educational) automated theorem provers:

- based on semantic tableaux method: 3TAP, pTAP, leanTAP, Cassandra;
- based on resolution method: OTTER, PCPROVE, AMPHION, Jape;
- based on semantic trees + Herbrand theorem: HERBY;
- based on model elimination calculus: SETHEO;

In 1938 Claude Shannon proved that a two-valued binary Boolean algebra can describe the operations of two-valued electrical switching circuits. Propositional logic is used to minimize the number of gates in a circuit, and to show the equivalence of *combinational circuits*.

The Romanian mathematician Grigore Moisil invented the three-stable circuits and had important contributions in the fields of algebraic logic and differential equations. Moisil used propositional logic to minimize Boolean functions.

In modern times *Boolean algebras* and *Boolean functions* are indispensable in the design of computer chips and digital circuits.

A computational approach, providing methods and algorithms to solve different tasks in classical logics and circuits is the specific purpose of this book.

1. PROPOSITIONAL LOGIC

In this chapter the propositional logic is presented, first from a semantic perspective and then as an axiomatic system. This formalism is useful in reasoning modeling and the design of logic circuits. We used as references the following papers [1, 2, 9, 16, 19, 23, 37, 38, 40, 42, 43, 47, 48, 59, 60, 61, 62, 63, 64].

1.1. Syntax

The *syntax* introduces the entities used to define well-formed propositional formulas.

- $\Sigma_P = \text{Var_propos} \cup \{F, T\} \cup \text{Connectives} \cup \{(,)\}$ is the *vocabulary*;
- $\text{Var_propos} = \{p, q, r, \dots\}$ is a finite *set of propositional variables*;
- $\text{Connectives} = \{\neg(\text{negation}), \wedge(\text{conjunction}), \vee(\text{disjunction}), \rightarrow(\text{implication}), \leftrightarrow(\text{equivalence})\}$:
The negation is a unary connective and all the others are binary connectives.
The decreasing order of precedence of the connectives is as follows:
 $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.
- F_P is the *set of well-formed formulas* built using the propositional variables, the connectives and the parentheses (to avoid ambiguity).
example: $(p \rightarrow \neg q) \wedge (r \vee q \leftrightarrow p) \wedge s$ is a propositional formula.

1.2. Semantics of propositional logic

Logical propositions are models of propositional assertions from natural language, which can be “true” or “false”.

The aim of the semantics is to give a meaning (to assign a truth value) to the propositional formulas. The *semantic domain* is the set of *truth values*: { F (false), T (true) }, which satisfy the relations: $\neg F = T$, $\neg T = F$.

New connectives \uparrow (“nand”), \downarrow (“nor”), \oplus (“xor”) are introduced using the following definitions:

$$p \uparrow q := \neg(p \wedge q),$$

$$p \downarrow q := \neg(p \vee q),$$

$$p \oplus q := \neg(p \leftrightarrow q)$$

These new connectives are used in the design of logic circuits.

Propositional Logic

The semantics of the connectives are provided by the following **truth tables**:

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$p \uparrow q$	$p \downarrow q$	$p \oplus q$
T	T	F	T	T	T	T	F	F	F
T	F	F	F	T	F	F	T	F	T
F	T	T	F	T	T	F	T	F	T
F	F	T	F	F	T	T	T	T	F

Remarks:

- A **conjunction** is *true* exactly when both its operands are *true*. As a generalization, the conjunction $p_1 \wedge p_2 \wedge \dots \wedge p_n$ is *true* exactly when all its n operands are *true*.
- A **disjunction** (“*inclusive or*”) is *false* only when both its operands are *false*. As a generalization, the conjunction $p_1 \vee p_2 \vee \dots \vee p_n$ is *false* only when all its n operands are *false*.
- The implication $p \rightarrow q$ is *false* only when the hypothesis p is *true* and the conclusion q is *false* (*true* cannot imply *false*).
- The equivalence $p \leftrightarrow q$ is *true* only when p and q have the same truth value.
- The connective \oplus (“*exclusive or*”) is the negation of equivalence and it is *true* only when one operand is *true* and the other one is *false*.

Definition 1.1.

An **interpretation** of a formula $U(p_1, p_2, \dots, p_n) \in F_P$ is a function $i: \{p_1, p_2, \dots, p_n\} \rightarrow \{F, T\}$ which can be extended to $i: F_P \rightarrow \{F, T\}$ using the following relations:

$$i(\neg p) = \neg i(p)$$

$$i(p \wedge q) = i(p) \wedge i(q)$$

$$i(p \vee q) = i(p) \vee i(q)$$

$$i(p \rightarrow q) = i(p) \rightarrow i(q)$$

$$i(p \leftrightarrow q) = i(p) \leftrightarrow i(q)$$

Interpretations assign truth values to propositional variables and using the semantics of the connectives evaluate formulas assigning truth values to them. The semantics is compositional, meaning that the truth value of a formula is obtained from the truth values of its subformulas.

Definition 1.2. (semantic concepts)

Let $U(p_1, p_2, \dots, p_n)$ be a propositional formula.

1. An interpretation i which evaluates the formula U as *true*, $i: \{p_1, \dots, p_n\} \rightarrow \{T, F\}$ such that $i(U) = T$, is called a **model** of U .

A Computational Approach to Classical Logics and Circuits

2. An interpretation i which evaluates the formula U as *false*, $i : \{p_1, \dots, p_n\} \rightarrow \{T, F\}$ such that $i(U) = F$, is called an *anti-model* of U .
3. A formula U is called *consistent (satisfiable)* if it has at least one model: $\exists i : \{p_1, \dots, p_n\} \rightarrow \{T, F\}$ such that $i(U) = T$.
4. The formula U is called *valid (tautology)* and we use the notation: $\models U$, if U is evaluated as true in all its interpretations: $\forall i : \{p_1, \dots, p_n\} \rightarrow \{T, F\}, i(U) = T$.
All 2^n interpretations of U are *models* of U .
5. The formula U is called *inconsistent (unsatisfiable)* if U does not have any model, thus U is evaluated as *false* in all its interpretations:
 $\forall i : \{p_1, \dots, p_n\} \rightarrow \{T, F\}, i(U) = F$. All 2^n interpretations of U are *anti-models* of U .
6. The formula U is called *contingent* if U is consistent, but it is not valid. A contingent formula has at least one model and at least one anti-model.

Remarks:

- In order to evaluate a propositional formula $U(p_1, p_2, \dots, p_n)$ in all its 2^n interpretations, a truth table is built. The table has 2^n lines (rows), corresponding to all interpretations and the first n columns are filled with all the possible assignments of *true* and *false* values to the n propositional variables. The column of U is obtained using the semantics and the precedence of the connectives.
- If the truth table (column) of U contains only “*T*”, then U is a *tautology*.
- If the truth table (column) of U contains only “*F*”, then U is an *inconsistent formula*.
- For a propositional formula, its *models* are the interpretations (the table’s rows) which evaluate the formula as *true* and its *anti-models* are the interpretations (the table’s rows) which evaluate the formula as *false*.

The notion of *logical consequence* captures the essence of logical thinking and it is a generalization of the *tautology* notion, as we shall see in the following definition.

Definition 1.3.

The formula $V \in F_P$ is a *logical consequence* of the formula $U \in F_P$, notation: $U \models V$, if all the models of U are also models of V :
 $\forall i : F_P \rightarrow \{T, F\}$ such that $i(U) = T$, we have that $i(V) = T$.

Definition 1.4.

The formulas $U \in F_P$ and $V \in F_P$ are *logically equivalent*, notation: $U \equiv V$, if U and V have identical truth tables: $\forall i : F_P \rightarrow \{T, F\}$ we have that $i(U) = i(V)$.

Propositional Logic

Note that “ \models ” and “ \equiv ” are meta-symbols used to express logical relations between propositional formulas.

Example 1.1.

Build the truth tables of the following formulas:

$$U(p,q,r) = (\neg p \vee q) \wedge (r \vee p)$$

$$V(p,q,r) = (\neg p \wedge r) \vee (q \wedge r) \vee (q \wedge p)$$

$$W(p,q,r) = (p \uparrow (\neg p \wedge q)) \vee r$$

$$Z(p,q,r) = p \wedge ((\neg q \vee r) \downarrow q)$$

In order to evaluate U , the formula is decomposed in two subformulas: $\neg p \vee q$ and $r \vee p$, which are evaluated and then we apply the conjunction between their corresponding truth tables (columns). In the same manner the columns of V, W, Z are built.

	p	q	r	$\neg p \vee q$	$r \vee p$	$U(p,q,r)$	$V(p,q,r)$	$W(p,q,r)$	$Z(p,q,r)$
i_1	T	T	T	T	T	T	T	T	F
i_2	T	T	F	T	T	T	T	T	F
i_3	T	F	T	F	T	F	F	T	F
i_4	T	F	F	F	T	F	F	T	F
i_5	F	T	T	T	T	T	T	T	F
i_6	F	T	F	T	F	F	F	T	F
i_7	F	F	T	T	T	T	T	T	F
i_8	F	F	F	T	F	F	F	T	F

- U, V, W and Z have $2^3 = 8$ interpretations: i_1, i_2, \dots, i_8 .
- The formula $U(p,q,r)$ is contingent, having four models: i_1, i_2, i_5, i_7 and four anti-models: i_3, i_4, i_6, i_8 .
- $i_1 : \{p, q, r\} \rightarrow \{T, F\}$, $i_1(p) = T$, $i_1(q) = T$, $i_1(r) = T$ and $i_1(U) = T$
- $i_6 : \{p, q, r\} \rightarrow \{T, F\}$, $i_6(p) = F$, $i_6(q) = T$, $i_6(r) = F$ and $i_6(U) = F$
- The formula $W(p,q,r)$ is a tautology, all its eight interpretations are models.
- The formula $Z(p,q,r)$ is inconsistent being evaluated as *false* in all its eight interpretations, which are the anti-models of Z .
- $U \equiv V$, because U and V have identical truth tables (columns).
- $U \models \neg p \vee q$, because all the models (i_1, i_2, i_5, i_7) of U are also models of the formula: $\neg p \vee q$.

A Computational Approach to Classical Logics and Circuits

1.3. Logical equivalences

The following logical equivalences express properties of connectives, relations between them and they are used to transform syntactically a propositional formula, preserving its meaning (semantics).

- **Simplification laws:**

$\neg\neg U \equiv U$	and	$U \rightarrow U \equiv T$
$U \wedge \neg U \equiv F$	and	$U \vee \neg U \equiv T$
$T \wedge U \equiv U$	and	$F \vee U \equiv U$
$U \rightarrow T \equiv T$	and	$U \rightarrow F \equiv \neg U$
$T \rightarrow U \equiv U$	and	$F \rightarrow U \equiv T$
$U \leftrightarrow T \equiv U$	and	$U \leftrightarrow F \equiv \neg U$
$U \oplus T \equiv \neg U$	and	$U \oplus F \equiv U$
$U \leftrightarrow U \equiv T$	and	$U \oplus U \equiv F$

- **Idempotency laws:**

$$U \wedge U \equiv U \quad \text{and} \quad U \vee U \equiv U$$

- **Commutative laws:**

$$\begin{aligned} U \wedge V &\equiv V \wedge U & \text{and} & \quad U \vee V \equiv V \vee U \\ U \uparrow V &\equiv V \uparrow U & \text{and} & \quad U \downarrow V \equiv V \downarrow U \\ U \leftrightarrow V &\equiv V \leftrightarrow U & \text{and} & \quad U \oplus V \equiv V \oplus U \end{aligned}$$

- **Absorption laws:**

$$U \wedge (U \vee V) \equiv U \quad \text{and} \quad U \vee (U \wedge V) \equiv U$$

- **Associative laws:**

$$(U \wedge V) \wedge Z \equiv U \wedge (V \wedge Z) \quad \text{and} \quad (U \vee V) \vee Z \equiv U \vee (V \vee Z)$$

- **Distributive laws:**

$$U \wedge (V \vee Z) \equiv (U \wedge V) \vee (U \wedge Z) \quad \text{and} \quad U \vee (V \wedge Z) \equiv (U \vee V) \wedge (U \vee Z)$$

- **De Morgan's laws:**

$$\neg(U \wedge V) \equiv \neg U \vee \neg V \quad \text{and} \quad \neg(U \vee V) \equiv \neg U \wedge \neg V$$

- **Relations between connectives:**

$U \rightarrow V \equiv \neg U \vee V$	and	$U \rightarrow V \equiv \neg(U \wedge \neg V)$
$U \rightarrow V \equiv U \leftrightarrow (U \wedge V)$	and	$U \rightarrow V \equiv V \leftrightarrow (U \vee V)$
$U \leftrightarrow V \equiv (U \rightarrow V) \wedge (V \rightarrow U)$	and	$U \oplus V \equiv \neg(U \rightarrow V) \vee \neg(V \rightarrow U)$
$U \leftrightarrow V \equiv (U \vee V) \rightarrow (U \wedge V)$	and	$U \wedge V \equiv \neg(\neg U \vee \neg V)$
$U \vee V \equiv \neg(\neg U \wedge \neg V)$	and	$U \wedge V \equiv \neg(U \rightarrow \neg V)$
$U \vee V \equiv \neg U \rightarrow V$	and	$\neg U \equiv U \downarrow U$
$\neg U \equiv U \uparrow U$	and	$U \wedge V \equiv (U \downarrow U) \downarrow (V \downarrow V)$
$U \vee V \equiv (U \uparrow U) \uparrow (V \uparrow V)$	and	$U \wedge V \equiv (U \uparrow V) \uparrow (U \uparrow V)$
$U \vee V \equiv (U \downarrow V) \downarrow (U \downarrow V)$	and	

Propositional Logic

The duality principle: For every logical equivalence $U \equiv V$ containing only the connectives $\neg, \wedge, \vee, \uparrow, \downarrow, \leftrightarrow, \oplus$, there is another logical equivalence $U' \equiv V'$, where U', V' are formulas obtained from U, V by interchanging the truth values (T, F) and the connectives from the following pairs: $(\wedge, \vee), (\uparrow, \downarrow), (\leftrightarrow, \oplus)$.

Note that some of the above laws are pairs of *dual logical equivalences*.

Dual connectives: $(\wedge, \vee), (\uparrow, \downarrow), (\leftrightarrow, \oplus)$.

Dual truth values: (T, F) .

Definition 1.5.

A set of connectives is *functionally complete* if there is no truth table that cannot be expressed as a formula involving only these connectives. All the other connectives can be expressed using the connectives from such a set.

The following sets of propositional connectives are functionally complete.

- | | | | |
|-------------------------|--------------------------------|------------------------------|---------------------------|
| 1. $\{\neg, \wedge\}$; | 2. $\{\neg, \vee\}$; | 3. $\{\neg, \rightarrow\}$; | 4. $\{\oplus, \wedge\}$; |
| 5. $\{\oplus, \vee\}$; | 6. $\{\oplus, \rightarrow\}$; | 7. $\{\uparrow\}$; | 8. $\{\downarrow\}$; |

The following definitions show that from the semantic point of view, a set of formulas is equivalent to the conjunction of its elements.

Definition 1.6.

Let U_1, U_2, \dots, U_n, V be propositional formulas.

1. The set $\{U_1, U_2, \dots, U_n\}$ is called *consistent* if the formula $U_1 \wedge U_2 \wedge \dots \wedge U_n$ is consistent:
$$\exists i : F_P \rightarrow \{T, F\} \text{ such that } i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = T.$$
2. The set $\{U_1, U_2, \dots, U_n\}$ is called *inconsistent* if the formula $U_1 \wedge U_2 \wedge \dots \wedge U_n$ is inconsistent:
$$\forall i : F_P \rightarrow \{T, F\}, i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = F.$$
3. The formula V , called *conclusion*, is a *logical consequence* of the set $\{U_1, U_2, \dots, U_n\}$ of formulas called *premises (hypotheses, facts)*, notation: $U_1, U_2, \dots, U_n \models V$, if $\forall i : F_P \rightarrow \{T, F\}$ such that $i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = T$, we have $i(V) = T$.

The models of the set of premises are also models of the conclusion.

Theorem 1.1.

Let $U_1, U_2, \dots, U_n, U, V$ be propositional formulas.

1. $\models U$ if and only if $\neg U$ is inconsistent.
A formula is a tautology if and only if its negation is inconsistent.
2. $U \models V$ if and only if $\models U \rightarrow V$ if and only if the set $\{U, \neg V\}$ is inconsistent.

A Computational Approach to Classical Logics and Circuits

3. $U \equiv V$ if and only if $\models U \leftrightarrow V$.

U and V are logically equivalent if and only if the formula

$U \leftrightarrow V$ is a tautology.

4. $U_1, U_2, \dots, U_n \models V$ if and only if

$\models U_1 \wedge U_2 \wedge \dots \wedge U_n \rightarrow V$ if and only if

the set $\{U_1, U_2, \dots, U_n, \neg V\}$ is inconsistent.

The proofs of the previous assertions are simple and we shall prove only some of them.

Proofs:

2. If $U \models V$ then $\models U \rightarrow V$.

If $U \models V$, using the definition of logical consequence relation (Definition 1.3), then: $\forall i : F_P \rightarrow \{T, F\}$ such that $i(U) = T$ we have that $i(V) = T$.

We have to evaluate the formula $U \rightarrow V$ in all the possible interpretations.

There are two cases: a) and b).

a) $i(U) = T$ and using the hypothesis $i(V) = T$,

we have that $i(U \rightarrow V) = i(U) \rightarrow i(V) = T \rightarrow T = T$

b) $i(U) = F$ thus $i(U \rightarrow V) = i(U) \rightarrow i(V) = F \rightarrow i(V) = T$

From a) and b) we have that $\forall i : F_P \rightarrow \{T, F\}, i(U \rightarrow V) = T$, therefore $\models U \rightarrow V$.

3. If $U \equiv V$ then $\models U \leftrightarrow V$.

If $U \equiv V$, using the definition of logical equivalence relation (Definition 1.4):

$\forall i : F_P \rightarrow \{T, F\}, i(U) = i(V)$, then

$\forall i : F_P \rightarrow \{T, F\}, i(U \leftrightarrow V) = T$ and according to the definition of a tautology we have that $\models U \leftrightarrow V$

4. If the set $\{U_1, U_2, \dots, U_n, \neg V\}$ is inconsistent then $U_1, U_2, \dots, U_n \models V$ holds.

If the set $\{U_1, U_2, \dots, U_n, \neg V\}$ is inconsistent, using the definition of inconsistency of a set of formulas:

$\forall i : F_P \rightarrow \{T, F\}, i(U_1 \wedge U_2 \wedge \dots \wedge U_n \wedge \neg V) = F$, then

$\forall i : F_P \rightarrow \{T, F\}$ there are two possible cases: a) and b).

a) $\exists k, 1 \leq k \leq n$, such that $i(U_k) = F$ then

$i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = F$, thus i is an anti-model of the set of premises of the logical consequence: $U_1, U_2, \dots, U_n \models V$ and it is an irrelevant case.

b) $\forall k, 1 \leq k \leq n, i(U_k) = T$ and $i(\neg V) = F$ then

$i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = T$ and $i(V) = T$ then

Propositional Logic

any model i of the set of premises $\{U_1, U_2, \dots, U_n\}$ is also a model of the conclusion V , so according to the definition of logical consequence relation we have that: $U_1, U_2, \dots, U_n \models V$ holds.

Remark:

The assertions 1, 2 and 4 from Theorem 1.1 are used in the refutation proof methods, such as *resolution* and *semantic tableaux method* and they model the proof by contradiction ("reductio ad absurdum").

Theorem 1.2.

Let $S = \{U_1, U_2, \dots, U_n\}$ be a set of propositional formulas.

1. If S is a consistent set, then $\forall j, 1 \leq j \leq n, S - \{U_j\}$ is a consistent set.
2. If S is a consistent set and V is a valid formula, then $S \cup \{V\}$ is a consistent set.
3. If S is an inconsistent set and U_j is valid, where $1 \leq j \leq n$, then the set $S - \{U_j\}$ is inconsistent.
4. If S is an inconsistent set, then $\forall V \in F_P$, the set $S \cup \{V\}$ is inconsistent.

Proofs: The proofs of the above assertions are simple, the Definition 1.6 is used.

1. If S is a consistent set then

$\exists i : F_P \rightarrow \{T, F\}$ such that $i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = T$ then

$\exists i : F_P \rightarrow \{T, F\}$ such that $\forall k, 1 \leq k \leq n, i(U_k) = T$ then

$\exists i : F_P \rightarrow \{T, F\}$ such that $\forall j, 1 \leq j \leq n,$

$$i(U_1 \wedge U_2 \wedge \dots \wedge U_{j-1} \wedge U_{j+1} \wedge \dots \wedge U_n) = T,$$

therefore $\forall j, 1 \leq j \leq n, S - \{U_j\}$ is a consistent set.

4. If S is an inconsistent set then

$\forall i : F_P \rightarrow \{T, F\}, i(U_1 \wedge U_2 \wedge \dots \wedge U_n) = F$ then

$\forall i : F_P \rightarrow \{T, F\}$ and $\forall V \in F_P, i(U_1 \wedge U_2 \wedge \dots \wedge U_n \wedge V) = F$,

therefore the set $S \cup \{V\}$ is inconsistent.

Theorem 1.3.

Let R, S be sets of propositional formulas and U, V, Z , be propositional formulas.

The logical consequence relation has the following properties:

1. **monotonicity:**

if $R \models U$ and $R \subseteq S$ then $S \models U$;

2. **cut:**

if $S \models V_j, \forall j \in \{1, \dots, n\}, n \in \mathbb{N}$ and $S \cup \{V_1, V_2, \dots, V_n\} \models U$ then $S \models U$;

A Computational Approach to Classical Logics and Circuits

3. **transitivity:**
if $S \models U$ and $\{U\} \models V$ then $S \models V$;
4. **conjunction in conclusions (right "and"):**
if $S \models U$ and $S \models V$ then $S \models U \wedge V$;
5. **disjunction in premises (left "or"):**
if $S \cup \{U\} \models Z$ and $S \cup \{V\} \models Z$ then $S \cup \{U \vee V\} \models Z$;

Proofs: These properties can be easily proved using the definition of logical consequence relation.

1. **monotonicity:**

If $R \models U$, then $\forall i : F_P \rightarrow \{T, F\}$ such that $i(R) = T$, we have that $i(U) = T$.

If $R \subseteq S$ then $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$ then $i(R) = T$.

From these two relationships, we have that $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$ then $i(R) = T$, so $i(U) = T$ and thus $S \models U$.

2. **cut:**

If $S \models V_j$, $\forall j \in \{1, \dots, n\}, n \in \mathbb{N}$, then $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, we have that $i(V_j) = T$.

So, $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, $\forall j \in \{1, \dots, n\}, n \in \mathbb{N}$, we have that $i(V_j) = T$.

In other words, $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, $i(S \cup \{V_1, V_2, \dots, V_n\}) = T$.

But $S \cup \{V_1, V_2, \dots, V_n\} \models U$, which means that $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S \cup \{V_1, V_2, \dots, V_n\}) = T$, we have that $i(U) = T$.

From the last two statements we obtain that $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, $\forall j \in \{1, \dots, n\}, n \in \mathbb{N}$, we have: $i(S \cup \{V_1, V_2, \dots, V_n\}) = T$ and that $i(U) = T$.

So, $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, we have that $i(U) = T$, which is $S \models U$.

3. **transitivity:**

If $S \models U$, then $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, we have that $i(U) = T$.

If $\{U\} \models V$, then $\forall i : F_P \rightarrow \{T, F\}$ such that $i(U) = T$, we have that $i(V) = T$.

So, $\forall i : F_P \rightarrow \{T, F\}$ such that $i(S) = T$, we have that $i(U) = T$ and also $i(V) = T$, which is $S \models V$.

Propositional Logic

1.4. Normal forms in propositional logic

Some of the proof methods need as input data formulas having a certain character of “normal” or “canonical” form.

Definition 1.7.

1. A **literal** is a propositional variable or its negation. ($p, \neg q, r$).
2. A **clause** is a disjunction of a finite number of literals. (examples: $p, \neg p \vee q, r \vee q \vee s$).
3. A **cube** is a conjunction of a finite number of literals. (examples: $q, p \wedge \neg q, r \wedge s \wedge p$).
4. The **empty clause**, denoted by \square , is the clause without any literal and it is the only inconsistent clause.
5. A formula is in **disjunctive normal form (DNF)**, if it is written as a disjunction of cubes: $\vee_{i=1}^P (\wedge_{j=1}^q l_{ij})$, where l_{ij} are literals.
6. A formula is in **conjunctive normal form (CNF)**, if it is written as a conjunction of clauses: $\wedge_{i=1}^n (\vee_{j=1}^m l_{ij})$, where l_{ij} are literals.

Example 1.2.

p - DNF with one unit cube;

$p \vee \neg q \vee r$ - DNF with three unit cubes;

$p \wedge q$ - DNF with one cube;

$p \vee (q \wedge r) \vee (\neg p \wedge \neg r \wedge s)$ - DNF with three cubes.

Example 1.3.

p - CNF with one unit clause;

$p \vee \neg q \vee r$ - CNF with one clause;

$p \wedge q$ - CNF with two unit clauses;

$\neg p \wedge (q \vee \neg r) \wedge (p \vee r \vee \neg s)$ - CNF with three clauses.

Property:

Let $\{l_1, l_2, \dots, l_n\}$ be a set of literals. The following assertions are equivalent:

1. The clause $\vee_{j=1}^n l_j$ is a tautology.
2. The cube $\wedge_{j=1}^n l_j$ is inconsistent.
3. The set $\{l_1, l_2, \dots, l_n\}$ of literals contains at least one pair of opposite literals:
 $\exists j, k \in \{1, \dots, n\}$ such that $l_j = \neg l_k$.

A Computational Approach to Classical Logics and Circuits

Example 1.4.

1. The clause $U = p \vee q \vee r \vee \neg p$ is a tautology ($U \equiv T$) since $p, \neg p$ are opposite literals.
2. The cube $V = p \wedge q \wedge r \wedge \neg p$ is an inconsistent formula ($U \equiv F$), because it is a conjunction with $p, \neg p$ as opposite literals.

Theorem 1.4.

Every propositional formula admits an equivalent conjunctive normal form (CNF) and an equivalent disjunctive normal form (DNF).

The normalization algorithm consists of transformations which preserve the logical equivalence and are applied to the initial formula in order to obtain the corresponding normal form.

Step 1:

- The formulas of " $U \rightarrow V$ " type are replaced by the logical equivalent form $\neg U \vee V$.
- The formulas of " $U \leftrightarrow V$ " type are replaced by the logical equivalent form $(\neg U \vee V) \wedge (\neg V \vee U)$.

Step 2:

- De Morgan's laws are applied: the negations are pushed in until they apply only to propositional variables.
- Multiple negations are eliminated by the reduction rule: $\neg\neg U \equiv U$.

Step 3:

- The distributive laws are applied.

Theorem 1.5.

1. A formula in CNF is a tautology if and only if all its clauses are tautologies.
2. A formula in DNF is inconsistent if and only if all its cubes are inconsistent.

Remarks:

- The first part of the above theorem provides a direct method to prove that a formula is a tautology.
- The DNF of a propositional formula provides all the models of that formula, finding all the interpretations which evaluate, one by one, the cubes as true.
- The CNF of a propositional formula provides all the anti-models of that formula, finding all the interpretations which evaluate, one by one, the clauses as false.

Dual concepts: clause-cube, DNF-CNF.

Propositional Logic

Example 1.5.

Write the equivalent CNF of the second axiom of propositional logic:

$$A_2 = ((U \rightarrow (V \rightarrow Z)) \rightarrow ((U \rightarrow V) \rightarrow (U \rightarrow Z))) .$$

The normalization algorithm is applied as follows:

$$A_2 = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ ((U \rightarrow (V \rightarrow Z)) \rightarrow ((U \rightarrow V) \rightarrow (U \rightarrow Z))) \end{matrix} \equiv$$

(replace \rightarrow from the inside formulas, denoted by 2,4 and 6)
 $\equiv ((U \rightarrow (\neg V \vee Z)) \rightarrow ((\neg U \vee V) \rightarrow (\neg U \vee Z))) \equiv$

(replace the main connective \rightarrow , denoted by 3)
 $\equiv \neg(U \rightarrow (\neg V \vee Z)) \vee ((\neg U \vee V) \rightarrow (\neg U \vee Z)) \equiv$

(replace both \rightarrow connectives, denoted by 1 and 5)
 $\equiv \neg(\neg U \vee \neg V \vee Z) \vee (\neg(\neg U \vee V) \vee \neg U \vee Z) \equiv$

(apply De Morgan's laws)
 $\equiv (U \wedge V \wedge \neg Z) \vee (U \wedge \neg V) \vee \neg U \vee Z - \text{DNF with 4 cubes}$

(apply the distributive laws)
 $\equiv (U \vee \underline{U} \vee \underline{\neg U} \vee Z) \wedge (\underline{U} \vee \neg V \vee \underline{\neg U} \vee Z) \wedge (V \vee \underline{U} \vee \underline{\neg U} \vee Z) \wedge$
 $\wedge (\underline{V} \vee \underline{\neg V} \vee \neg U \vee Z) \wedge (\underline{\neg Z} \vee U \vee \neg U \vee Z) \wedge (\underline{\neg Z} \vee \neg V \vee \neg U \vee Z) \equiv T$

We have obtained a CNF with 6 clauses which are tautologies. Each clause contains at least one pair of opposite formulas (underlined).

Thus, according to the previous theorem, the formula A_2 is a tautology.

Example 1.6.

Write the equivalent DNF of the formula: $X = \neg A_1 = \neg(U \rightarrow (V \rightarrow U))$

$$X = \neg(U \rightarrow (V \rightarrow U)) \equiv \neg(\neg U \vee (\neg V \vee U)) \equiv U \wedge \neg(\neg V \vee U) \equiv \underline{U} \wedge V \wedge \underline{\neg U}$$

We have obtained a CNF (with three unit clauses) or a DNF (with one cube). DNF contains an inconsistent cube, having a pair of opposite formulas, thus X is an inconsistent formula and A_1 (the first axiom of the propositional logic) is a tautology.

Example 1.7.

Using the appropriate normal form write all the models of the propositional formula: $U = (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q$.

We apply the normalization algorithm:

$$\begin{aligned} U &= (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q \equiv \neg(p \wedge q \rightarrow r) \vee (p \rightarrow r) \wedge q \equiv \\ &\equiv \neg(\neg(p \wedge q) \vee r) \vee (\neg p \vee r) \wedge q \equiv (p \wedge q \wedge \neg r) \vee (\neg p \vee r) \wedge q \equiv \\ &\equiv (p \wedge q \wedge \neg r) \vee (\neg p \wedge q) \vee (r \wedge q) - \text{DNF with 3 cubes} \end{aligned}$$

The models of U are the interpretations which evaluate one by one the cubes of DNF as true. In a cube we assign the truth value T to all its literals.

A Computational Approach to Classical Logics and Circuits

Cube: $p \wedge q \wedge \neg r$ provides one model assigning to all the literals $\{p, q, \neg r\}$ of the cube the truth value T .

$$i_1 : \{p, q, r\} \rightarrow \{T, F\}, i_1(p) = T, i_1(q) = T, i_1(r) = F \text{ and } i_1(p \wedge q \wedge \neg r) = T$$

Cube: $\neg p \wedge q$.

The propositional variable r is missing in the cube's expression, so the truth value of the cube does not depend on the truth value assigned to r : we need to assign to p the value F , to q the value T , but to r we can assign either T or F such that the cube is evaluated as true.

Therefore this cube provides the following models:

$$i_2 : \{p, q, r\} \rightarrow \{T, F\}, i_2(p) = F, i_2(q) = T, i_2(r) = T \text{ and } i_2(\neg p \wedge q) = T$$

$$i_3 : \{p, q, r\} \rightarrow \{T, F\}, i_3(p) = F, i_3(q) = T, i_3(r) = F \text{ and } i_3(\neg p \wedge q) = T$$

Cube: $r \wedge q$ provides the models:

$$i_4 : \{p, q, r\} \rightarrow \{T, F\}, i_4(p) = T, i_4(q) = T, i_4(r) = T \text{ and } i_4(r \wedge q) = T$$

$$i_5 : \{p, q, r\} \rightarrow \{T, F\}, i_5(p) = F, i_5(q) = T, i_5(r) = T \text{ and } i_5(r \wedge q) = T$$

Note that $i_2 = i_5$, thus U has four distinct models i_1, i_2, i_3, i_4 :

$$i_1(U) = i_2(U) = i_3(U) = i_4(U) = T$$

All the other four interpretations, from the total of eight possible interpretations, evaluate the formula U as false, they are anti-models of U .

Example 1.8.

Using the conjunctive normal form of $U = p \vee q \rightarrow p \wedge q$ find all its anti-models.

The normalization algorithm is applied in order to obtain $\text{CNF}(U)$.

$$\begin{aligned} U &= p \vee q \rightarrow p \wedge q \equiv \neg(p \vee q) \vee (p \wedge q) \equiv (\neg p \wedge \neg q) \vee (p \wedge q) \text{ apply distributivity} \\ &\equiv (\neg p \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p) \wedge (\neg q \vee q) \equiv \\ &\equiv T \wedge (\neg p \vee q) \wedge (\neg q \vee p) \wedge T \equiv (\neg p \vee q) \wedge (\neg q \vee p) - \text{CNF with 2 clauses} \end{aligned}$$

The anti-models of U are the interpretations which evaluate one by one the clauses of CNF as false. In a clause we assign the truth value F to all its literals.

Clause: $\neg p \vee q$ provides the anti-model:

$$i_1 : \{p, q\} \rightarrow \{T, F\}, i_1(p) = T, i_1(q) = F, \text{ thus } i_1(\neg p \vee q) = F \text{ and } i_1(U) = F$$

Clause: $\neg q \vee p$ provides the anti-model:

$$i_2 : \{p, q\} \rightarrow \{T, F\}, i_2(p) = F, i_2(q) = T \text{ thus } i_2(\neg q \vee p) = F \text{ and } i_2(U) = F$$

U has two anti-models: i_1 and i_2 , which evaluate the formula as false. The total number of its interpretations is $2^2 = 4$, therefore U has two models which evaluate the formula as true, and they correspond to the other two possible assignments of T and F to the variables p and q .

Propositional Logic

1.5. Formal (axiomatic) system of propositional logic

In this section we present the formalization of propositional logic proposed in paper [62]. From a syntactic perspective propositional logic is defined using the axiomatic system: $P = (\Sigma_P, F_P, A_P, R_P)$ where:

1. $\Sigma_P = \text{Var_propos} \cup \{T, F\} \cup \text{Connectives} \cup \{(,)\}$ is the **vocabulary**
 - $\text{Var_propos} = \{p_1, p_2, \dots\}$ is the **set of propositional variables**
 - $\text{Connectives} = \{\neg(\text{negation}), \wedge(\text{conjunction}), \vee(\text{disjunction}), \rightarrow(\text{implication}), \leftrightarrow(\text{equivalence})\}$ - the set of **logical connectives**
2. F_P is the **set of well-formed formulas**, which is the smallest set of formulas satisfying the rules:
 - **base:** $p_i \in F_P, i = 1, 2, \dots$;
 - **induction:**
 - if $U, V \in F_P$ then:
 - $\neg U \in F_P, U \wedge V \in F_P, U \vee V \in F_P, U \rightarrow V \in F_P, U \leftrightarrow V \in F_P$
 - **closure:** all formulas from F_P are obtained by applying the above rules in a finite number of steps.
3. $A_P = \{A_1, A_2, A_3\}$ is the set of axioms of propositional logic

$A_1 : U \rightarrow (V \rightarrow U)$
 $A_2 : (U \rightarrow (V \rightarrow Z)) \rightarrow ((U \rightarrow V) \rightarrow (U \rightarrow Z))$
 $A_3 : (U \rightarrow V) \rightarrow (\neg V \rightarrow \neg U) - \text{modus tollens}$

These axioms are in fact axiomatic schemes, where U, V, Z are arbitrary formulas, generating an infinite number of axioms.
4. $R_P = \{\text{mp}\}$ is the set of **inference (deduction) rules** containing **modus ponens rule**. Notation: $U, U \rightarrow V \vdash_{\text{mp}} V$, with the meaning: “from the formulas U and $U \rightarrow V$ we deduce (infer) V ”.

Remark:

- All the axioms obtained using the axiomatic schemes A_1, A_2, A_3 are tautologies (valid formulas).

Definition 1.8.

Let U_1, U_2, \dots, U_n be formulas, called **hypotheses** and V be a formula called **conclusion**. V is deducible (derivable, inferable) from U_1, U_2, \dots, U_n and we denote this by $U_1, U_2, \dots, U_n \vdash V$, if there is a sequence (f_1, f_2, \dots, f_m) of formulas such that $f_m = V$ and $\forall i \in \{1, \dots, m\}$ we have a) or b) or c).

A Computational Approach to Classical Logics and Circuits

- a) $f_i \in A_P$ (axiom);
- b) $f_i \in \{U_1, U_2, \dots, U_n\}$ (hypothesis formula);
- c) $f_{i_1}, f_{i_2} \vdash_{mp} f_i, \quad i_1 < i \text{ and } i_2 < i$ (formula f_i is inferred using *modus ponens* from two existing formulas)

The sequence (f_1, f_2, \dots, f_m) is called the *deduction of V from the hypotheses U_1, U_2, \dots, U_n* .

Definition 1.9.

A formula $U \in F_P$, such that $\emptyset \vdash U$ (or $\vdash U$) is called *theorem*.

Remark: The theorems are the formulas deducible (inferable) only from the axioms and using *modus ponens* as inference rule.

This axiomatic system (P) belongs to the group of Hilbert axiomatic systems [24], all of them having only *modus ponens* as inference rule. The formal system (P) presented above has as a basic binary connective *implication* (\rightarrow). Other Hilbert systems change the axioms or use also *disjunction* (\vee) as a basic connective. All these axiomatic systems are equivalent: the axioms of such a system can be proved as theorems in all the other axiomatic systems.

Example 1.9.

Prove that: $U, U \rightarrow V, V \rightarrow Z \vdash Z$ using the definition of deduction. $U, V, Z \in F_P$

We build the sequence $(f_1, f_2, f_3, f_4, f_5)$ of formulas as follows:

$$\begin{aligned} f_1 &: U \\ f_2 &: U \rightarrow V \\ &\quad f_1, f_2 \vdash_{mp} V \\ f_3 &: V \\ f_4 &: V \rightarrow Z \\ &\quad f_3, f_4 \vdash_{mp} Z \\ f_5 &: Z \end{aligned}$$

Note that f_1, f_2, f_4 are initial formulas (hypotheses), and f_3, f_5 are obtained using *modus ponens* rule. According to Definition 1.8 the sequence: $(f_1, f_2, f_3, f_4, f_5)$ is the deduction of Z from the hypotheses $U, U \rightarrow V, V \rightarrow Z$.

Example 1.10.

Prove that: $\neg p \vee q, p \vee r, \neg q \vdash r$ using the definition of deduction.

In order to apply *modus ponens* the ' \vee ' connective must be written in an equivalent form using ' \rightarrow '.

Propositional Logic

The sequence $(f_1, f_2, f_3, f_4, f_5, f_6, f_7)$ of formulas is built as follows:

$$f_1 : \neg p \vee q \equiv p \rightarrow q \text{ (hypothesis)}$$

$$f_2 : p \vee r \equiv \neg p \rightarrow r \text{ (hypothesis)}$$

$$f_3 : \neg q \text{ (hypothesis)}$$

$$f_4 : (p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p) - \text{axiom } A_3 \text{ (*modus tollens*)}$$

$$f_1, f_4 \vdash_{mp} \neg q \rightarrow \neg p$$

$$f_5 : \neg q \rightarrow \neg p$$

$$f_3, f_5 \vdash_{mp} \neg p$$

$$f_6 : \neg p$$

$$f_2, f_6 \vdash_{mp} r$$

$$f_7 : r \text{ (conclusion)}$$

Using only the hypotheses we cannot prove the conclusion, so we need to use axioms. In the deduction process modelled by the sequence $(f_1, f_2, f_3, f_4, f_5, f_6, f_7)$, the formulas f_1, f_2, f_3 are hypotheses, f_4 is an instantiation of axiom A_3 and f_5, f_6, f_7 are derived using *modus ponens*. According to Definition 1.8, we have proved that the deduction holds.

Example 1.11.

Prove that $\vdash U \wedge V \rightarrow V$ using the definition of deduction.

We transform syntactically the formula $U \wedge V \rightarrow V$ such that the only connectives are \neg, \rightarrow .

The conjunction is replaced using the following logical equivalence:

$$U \wedge V \equiv \neg(\neg U \vee \neg V) \equiv \neg(U \rightarrow \neg V)$$

$$U \wedge V \rightarrow V \equiv \neg(U \rightarrow \neg V) \rightarrow V$$

We use the axioms A_1 and A_3 as follows:

$$f_1 : \neg V \rightarrow (U \rightarrow \neg V) - \text{an instance of axiom } A_1$$

$$f_2 : (\neg V \rightarrow (U \rightarrow \neg V)) \rightarrow (\neg(U \rightarrow \neg V) \rightarrow V) - \text{axiom } A_3 \text{ (*modus tollens*)}$$

$$f_1, f_2 \vdash_{mp} \neg(U \rightarrow \neg V) \rightarrow V \equiv \neg(\neg U \vee \neg V) \rightarrow V \equiv U \wedge V \rightarrow V$$

The formula $U \wedge V \rightarrow V$ was deduced only from the axioms and *modus ponens* inference rule, therefore it is a theorem.

Example 1.12.

Prove that $\vdash U \rightarrow U$, building the deduction of $U \rightarrow U$ from the axioms only.

The sequence of formulas $(f_1, f_2, f_3, f_4, f_5)$ is built.

$$f_1 : U \rightarrow ((U \rightarrow U) \rightarrow U) \text{ obtained from } A_1 : U \rightarrow (V \rightarrow U)$$

replacing V by $U \rightarrow U$;

A Computational Approach to Classical Logics and Circuits

$f_2 : (U \rightarrow ((U \rightarrow U) \rightarrow U)) \rightarrow ((U \rightarrow (U \rightarrow U)) \rightarrow (U \rightarrow U))$ obtained
 from $A_2 : (U \rightarrow (V \rightarrow Z)) \rightarrow ((U \rightarrow V) \rightarrow (U \rightarrow Z))$
 using the replacements: V by $U \rightarrow U$ and Z by U ;

$$f_1, f_2 \vdash_{mp} (U \rightarrow (U \rightarrow U)) \rightarrow (U \rightarrow U)$$

$$f_3 : (U \rightarrow (U \rightarrow U)) \rightarrow (U \rightarrow U)$$

$f_4 : U \rightarrow (U \rightarrow U)$ obtained from $A_1 : U \rightarrow (V \rightarrow U)$ replacing V by U
 $f_3, f_4 \vdash_{mp} U \rightarrow U$

$$f_5 : U \rightarrow U$$

Therefore the sequence $(f_1, f_2, f_3, f_4, f_5)$ is the deduction of $U \rightarrow U$ from the axioms, so $U \rightarrow U$ is a theorem.

Example 1.13.

Prove that $\neg U \vdash U \rightarrow V$ using the definition of deduction.

In the inferential process the sequence of formulas $(f_1, f_2, f_3, f_4, f_5)$ is built as follows:

$$f_1 : \neg U - \text{hypothesis formula}$$

$f_2 : \neg U \rightarrow (\neg V \rightarrow \neg U)$ obtained from $A_1 : U \rightarrow (V \rightarrow U)$
 replacing U by $\neg U$ and V by $\neg V$;

$$f_1, f_2 \vdash_{mp} \neg V \rightarrow \neg U$$

$$f_3 : \neg V \rightarrow \neg U$$

$f_4 : (\neg V \rightarrow \neg U) \rightarrow (U \rightarrow V)$ obtained from $A_3 : (U \rightarrow V) \rightarrow (\neg V \rightarrow \neg U)$
 replacing U by $\neg V$ and V by $\neg U$;

$$f_3, f_4 \vdash_{mp} U \rightarrow V$$

$$f_5 : U \rightarrow V$$

The sequence $(f_1, f_2, f_3, f_4, f_5)$ is the deduction of $U \rightarrow V$ from $\neg U$ and the axioms.

In the inferential (deductive) process, beside *modus ponens*, other inference rules are used. These rules can be also expressed as theorems.

<i>addition</i>	$U \vdash U \vee V$ $\vdash U \rightarrow U \vee V$	inference rule theorem
<i>simplification</i>	$U \wedge V \vdash U$ $\vdash U \wedge V \rightarrow U$	inference rule theorem
	$U \wedge V \vdash V$ $\vdash U \wedge V \rightarrow V$	inference rule theorem

Propositional Logic

<i>conjunction</i>	$U, V \vdash U \wedge V$	inference rule
<i>modus ponens</i>	$U, U \rightarrow V \vdash V$ $\vdash U \wedge (U \rightarrow V) \rightarrow V$	inference rule theorem
<i>modus tollens</i>	$\neg V, U \rightarrow V \vdash \neg U$ $\vdash \neg V \wedge (U \rightarrow V) \rightarrow \neg U$	inference rule theorem
	$U \rightarrow V \vdash \neg V \rightarrow \neg U$ $\vdash (U \rightarrow V) \rightarrow (\neg V \rightarrow \neg U)$	inference rule theorem
<i>syllogism</i>	$U \rightarrow V, V \rightarrow Z \vdash U \rightarrow Z$ $\vdash (U \rightarrow V) \wedge (V \rightarrow Z) \rightarrow (U \rightarrow Z)$	inference rule theorem
	$U, U \rightarrow V, V \rightarrow Z \vdash Z$ $\vdash U \wedge (U \rightarrow V) \wedge (V \rightarrow Z) \rightarrow Z$	inference rule theorem
<i>resolution</i>	$U \vee V, \neg U \vee Z \vdash V \vee Z$ $\vdash (U \vee V) \wedge (\neg U \vee Z) \rightarrow (V \vee Z)$	inference rule theorem

1.6. The theorem of deduction and its reverse

Some theorems, even if they are syntactically simple, are very difficult to be proved using only the axioms and *modus ponens*, because we need to guess the starting-point axiomatic schemes and the corresponding replacements of the arbitrary formulas U, V, Z .

The combination of the following two theoretic results is used to help us prove at the syntactic level theorems having a special syntactic form (the connectives are mainly ' \rightarrow ').

Theorem 1.6. Theorem of deduction:

If $U_1, \dots, U_{n-1}, U_n \vdash V$, then $U_1, \dots, U_{n-1} \vdash U_n \rightarrow V$.

Theorem 1.7. Reverse of the theorem of deduction:

If $U_1, \dots, U_{n-1} \vdash U_n \rightarrow V$ then $U_1, \dots, U_{n-1}, U_n \vdash V$.

Applying n times the theorem of deduction and its reverse we obtain:

$U_1, \dots, U_{n-1}, U_n \vdash V$ if and only if

$U_1, \dots, U_{n-1} \vdash U_n \rightarrow V$ if and only if

$U_1, \dots, U_{n-2} \vdash U_{n-1} \rightarrow (U_n \rightarrow V)$ if and only if

$U_1 \vdash U_2 \rightarrow (\dots \rightarrow (U_{n-1} \rightarrow (U_n \rightarrow V)) \dots)$ if and only if

$\vdash U_1 \rightarrow (U_2 \rightarrow (\dots \rightarrow (U_n \rightarrow V) \dots))$

A Computational Approach to Classical Logics and Circuits

Note: The application of the theorem of deduction and its reverse means that a premise (hypothesis) of a deduction can be moved from the left-hand side of ' \vdash ' to its right-hand side as a premise of a main ' \rightarrow ' and vice-versa.

Consequences of the theorem of deduction:

1. $\vdash U \rightarrow ((U \rightarrow V) \rightarrow V)$;
2. $\vdash (U \rightarrow V) \rightarrow ((V \rightarrow Z) \rightarrow (U \rightarrow Z))$ the *syllogism* law;
3. $\vdash (U \rightarrow (V \rightarrow Z)) \rightarrow (V \rightarrow (U \rightarrow Z))$ the *permutation of the premises* law;
4. $\vdash (U \rightarrow (V \rightarrow Z)) \rightarrow (U \wedge V \rightarrow Z)$ the *reunion of the premises* law;
5. $\vdash (U \wedge V \rightarrow Z) \rightarrow (U \rightarrow (V \rightarrow Z))$ the *separation of the premises* law.

A syntactic and direct proof method based on these theorems consists of three steps:

Step 1: The initial theorem (formula) to be proved is written in an equivalent form as a deduction, applying the reverse of the theorem of deduction several times.

Step 2: The deduction obtained in Step 1 is proved using the axiomatic system.

Step 3: Starting with the deduction proved in Step 2 we apply the theorem of deduction several times. All the premises moved in Step 1 from right to left are now moved from left to right in the reverse order to obtain the initial formula.

Remarks:

- Step 1 may be skipped if we can guess the starting-point deduction in order to prove the initial formula.
- In Step 3, if we change the order in which the premises of the deduction are moved from left to right, new theorems can be proved.

Example 1.14.

The consequences 1 and 2 are proved in the following.

1. We begin with the *modus ponens* rule: $U, U \rightarrow V \vdash_{mp} V$

Applying the theorem of deduction we obtain: $U \vdash (U \rightarrow V) \rightarrow V$ and we continue with another application of the same theorem: $\vdash U \rightarrow ((U \rightarrow V) \rightarrow V)$.

2. We begin with the deduction: $U, U \rightarrow V, V \rightarrow Z \vdash Z$ proved in Example 1.9.

Application of the theorem of deduction (U is moved from left to right):

$$U \rightarrow V, V \rightarrow Z \vdash U \rightarrow Z$$

Application of the theorem of deduction ($V \rightarrow Z$ is moved from left to right):

$$U \rightarrow V \vdash (V \rightarrow Z) \rightarrow (U \rightarrow Z)$$

Application of the theorem of deduction ($U \rightarrow V$ is moved from left to right):

$$\vdash (U \rightarrow V) \rightarrow ((V \rightarrow Z) \rightarrow (U \rightarrow Z))$$

Propositional Logic

Example 1.15.

Using the theorem of deduction and its reverse prove:

$$\vdash (p \rightarrow r) \rightarrow ((p \wedge r \rightarrow q) \rightarrow (p \rightarrow q))$$

Step 1:

The reverse of the theorem of deduction is applied to obtain the starting-point deduction:

if $\vdash (p \rightarrow r) \rightarrow ((p \wedge r \rightarrow q) \rightarrow (p \rightarrow q))$ then

$$p \rightarrow r \vdash (p \wedge r \rightarrow q) \rightarrow (p \rightarrow q) \text{ then}$$

$$p \rightarrow r, p \wedge r \rightarrow q \vdash p \rightarrow q \text{ then } p \rightarrow r, p \wedge r \rightarrow q, p \vdash q$$

Step 2:

We prove the deduction obtained in Step 1.

Using Definition 1.8 we build the sequence of formulas: $(f_1, f_2, f_3, f_4, f_5, f_6)$:

$$f_1 : p \text{ — premise (hypothesis)}$$

$$f_2 : p \rightarrow r \text{ — premise}$$

$$f_1, f_2 \vdash_{mp} r$$

$$f_3 : r$$

$$f_4 : f_1 \wedge f_3 = p \wedge r \text{ (conjunction of the conclusions)}$$

$$f_5 : p \wedge r \rightarrow q \text{ — premise}$$

$$f_4, f_5 \vdash_{mp} q$$

$$f_6 : q$$

The sequence $(f_1, f_2, f_3, f_4, f_5, f_6)$ is the deduction of q from the premises: $p \rightarrow r, p \wedge r \rightarrow q, p$.

Step 3:

To the deduction $p \rightarrow r, p \wedge r \rightarrow q, p \vdash q$ we apply three times the theorem of deduction.

There are $3! = 6$ such possibilities (to move the premises to the right-hand side of the meta-symbol \vdash) and we prove 6 theorems: $T_1, T_2, T_3, T_4, T_5, T_6$

1. *the premises are moved to the right-hand side of ' \vdash ' in the following order:*

$$p, p \wedge r \rightarrow q, p \rightarrow r .$$

if $p \rightarrow r, p \wedge r \rightarrow q, p \vdash q$ then

$$p \rightarrow r, p \wedge r \rightarrow q \vdash p \rightarrow q \text{ then}$$

$$p \rightarrow r \vdash (p \wedge r \rightarrow q) \rightarrow (p \rightarrow q) \text{ then}$$

$$\vdash T_1 = (p \rightarrow r) \rightarrow ((p \wedge r \rightarrow q) \rightarrow (p \rightarrow q)) \text{ — the theorem to be proved.}$$

A Computational Approach to Classical Logics and Circuits

2. the premises are moved to the right-hand side of ' \vdash ' in the following order:

$p, p \rightarrow r, p \wedge r \rightarrow q.$

if $p \rightarrow r, p \wedge r \rightarrow q, p \vdash q$ then

$p \rightarrow r, p \wedge r \rightarrow q \vdash p \rightarrow q$ then

$p \wedge r \rightarrow q \vdash (p \rightarrow r) \rightarrow (p \rightarrow q)$ then

$$\vdash T_2 = (p \wedge r \rightarrow q) \rightarrow ((p \rightarrow r) \rightarrow (p \rightarrow q))$$

The following theorems can be also proved:

$$\vdash T_3 = (p \wedge r \rightarrow q) \rightarrow (p \rightarrow ((p \rightarrow r) \rightarrow q)))$$

$$\vdash T_4 = p \rightarrow ((p \wedge r \rightarrow q) \rightarrow ((p \rightarrow r) \rightarrow q))$$

$$\vdash T_5 = (p \rightarrow r) \rightarrow (p \rightarrow ((p \wedge r \rightarrow q) \rightarrow q))$$

$$\vdash T_6 = p \rightarrow ((p \rightarrow r) \rightarrow ((p \wedge r \rightarrow q) \rightarrow q))$$

1.7. Properties of propositional logic

The properties of propositional logic are: **compactness, soundness, completeness, coherence, non-contradiction and decidability**.

Theorem 1.8. (compactness 1)

An infinite set of propositional formulas has a model if and only if each of its subsets has a finite model.

Theorem 1.9.

Let $S = \{U_1, U_2, \dots, U_m, \dots\}$ be an infinite set of propositional formulas.

1. S is inconsistent if and only if $\exists k \in N^*$, such that $\{U_1, U_2, \dots, U_k\}$ is inconsistent.

2. S is consistent if and only if

$\{U_1\}$ is consistent and

$\{U_1, U_2\}$ is consistent and

...

$\{U_1, U_2, \dots, U_m\}$ is consistent and

...

Remarks:

- An infinite set of propositional formulas is inconsistent if and only if it has an inconsistent finite subset, therefore *inconsistency can be proved in a finite number of steps*.
- An infinite set of propositional formulas is consistent if and only if all its subsets (an infinite number) are consistent, therefore *consistency cannot be proved in a finite number of steps*.

Propositional Logic

Theorem 1.10. (compactness 2)

A propositional formula V is a logical consequence of an infinite set of propositional formulas S (notation: $S \models V$) if and only if there is a finite subset $\{U_1, U_2, \dots, U_n\} \subset S$ such that $U_1, U_2, \dots, U_n \models V$.

Theorem 1.11. Soundness theorem

(syntactic validity implies semantic validity):

If $\vdash U$ then $\models U$ (a theorem is a tautology).

Theorem 1.12. Completeness theorem

(semantic validity implies syntactic validity):

If $\models U$ then $\vdash U$ (a tautology is a theorem).

Theorem 1.13. Soundness and completeness for propositional logic:

$\vdash U$ if and only if $\models U$ (a formula is a theorem if and only if it is a tautology).

Consequences of the last theorem are the following properties:

1. **Propositional logic is non-contradictory:** we cannot have simultaneously: $\vdash U$ and $\vdash \neg U$.
2. **Propositional logic is coherent:** not every propositional formula is a theorem.
3. **Propositional logic is decidable:** always we can decide if a propositional formula is a theorem or not. The truth table method is a decision method.

1.8. Decision problems and proof methods

Decision problems in propositional logic:

1. Is a propositional formula a tautology/theorem?

$$\models^? V \quad \text{or} \quad \vdash^? V$$

2. Is a propositional formula a logical/syntactic consequence of a set of hypotheses?

$$U_1, \dots, U_n \models^? V \quad \text{or} \quad U_1, \dots, U_n \vdash^? V$$

In order to solve these two decision problems, theorem proving methods are applied. These methods will be studied in the next chapters.

In the following we provide w the classifications of the studied proof methods from two perspectives:

First classification: **semantic versus syntactic methods**

1. **semantic proof methods:**

- the truth table method;
- the semantic tableaux method;
- CNF- conjunctive normal form.

A Computational Approach to Classical Logics and Circuits

2. *syntactic proof methods:*

- the definition of deduction;
- the theorem of deduction and its reverse;
- the resolution method;
- the sequent calculus method.

Second classification: **direct versus refutation methods**

1. *direct methods:* they use directly the formula to be proved:

- the truth table method;
- the CNF- conjunctive normal form;
- the definition of deduction;
- the theorem of deduction and its reverse;
- the sequent calculus method.

2. *refutation methods:* they model the “reductio ad absurdum” (proof by contradiction) using the negation of the formula to be proved:

- the semantic tableaux method;
- the resolution method.

1.9. Exercises

Exercise 1.1.

Check the following properties for \downarrow ('nor'), \uparrow ('nand') and \oplus ('xor') connectives using the truth table method.

1. associativity of ' \uparrow ' connective:

$$p \uparrow (q \uparrow r) \equiv (p \uparrow q) \uparrow r ;$$

2. associativity of ' \downarrow ' connective:

$$p \downarrow (q \downarrow r) \equiv (p \downarrow q) \downarrow r ;$$

3. associativity of ' \oplus ' connective:

$$p \oplus (q \oplus r) \equiv (p \oplus q) \oplus r ;$$

4. distribution of ' \uparrow ' connective over ' \downarrow ' connective:

$$p \uparrow (q \downarrow r) \equiv (p \uparrow q) \downarrow (p \uparrow r) ;$$

5. distribution of ' \downarrow ' connective over ' \uparrow ' connective:

$$p \downarrow (q \uparrow r) \equiv (p \downarrow q) \uparrow (p \downarrow r) ;$$

6. De Morgan's laws for ' \downarrow ' and ' \uparrow :

$$\neg(p \downarrow q) \equiv \neg p \uparrow \neg q \quad \text{and} \quad \neg(p \uparrow q) \equiv \neg p \downarrow \neg q ;$$

$$7. \quad p \uparrow (q \vee r) \equiv (p \uparrow q) \wedge (p \uparrow r) \quad \text{and} \quad p \downarrow (q \wedge r) \equiv (p \downarrow q) \vee (p \downarrow r) .$$

$$8. \quad p \downarrow (q \uparrow p) \equiv F \quad \text{and} \quad p \uparrow (q \downarrow p) \equiv T ;$$

Propositional Logic

Exercise 1.2.

Using the truth table method decide what kind of formula (consistent, inconsistent, tautology, contingent) is $U_j, j \in \{1, 2, \dots, 8\}$. Write all the models and anti-models of $U_j, j \in \{1, 2, \dots, 8\}$.

1. $U_1 = q \wedge \neg p \wedge r \rightarrow \neg p \vee \neg(q \wedge r);$
2. $U_2 = \neg p \vee \neg(q \wedge r) \rightarrow q \wedge \neg p;$
3. $U_3 = \neg p \wedge (\neg q \vee r) \rightarrow q \vee \neg p \vee r;$
4. $U_4 = \neg(\neg p \vee q) \vee r \rightarrow \neg p \vee (\neg q \vee r);$
5. $U_5 = \neg p \vee (\neg q \vee \neg r) \rightarrow q \wedge \neg p;$
6. $U_6 = \neg p \vee (\neg q \wedge \neg r) \rightarrow q \wedge \neg p \wedge r;$
7. $U_7 = p \rightarrow (q \wedge r) \vee q \wedge \neg p;$
8. $U_8 = (p \vee q) \wedge \neg r \rightarrow p \wedge q \wedge r.$

Exercise 1.3.

Using the truth table method, check if the following logical consequences hold:

1. $p \rightarrow q \models (p \rightarrow r) \rightarrow (p \rightarrow q \wedge r);$
2. $p \rightarrow q \models (q \rightarrow r) \rightarrow (p \rightarrow r);$
3. $p \rightarrow (q \rightarrow r) \models (p \rightarrow q) \rightarrow (p \rightarrow r);$
4. $p \rightarrow r \models (q \rightarrow r) \rightarrow ((p \vee q) \rightarrow r);$
5. $p \rightarrow q \models (\neg p \rightarrow q) \rightarrow q;$
6. $p \rightarrow q \models (q \rightarrow r) \rightarrow (p \rightarrow q \wedge r);$
7. $p \rightarrow q \models (q \rightarrow r) \rightarrow (p \rightarrow q \vee r);$
8. $r \rightarrow (q \rightarrow p) \models (r \rightarrow q) \rightarrow (r \rightarrow p).$

Exercise 1.4.

Prove that the following formulas are tautologies using the truth table method.

1. the left-distribution of ' \rightarrow ' over ' \wedge ': $(p \rightarrow (q \wedge r)) \rightarrow ((p \rightarrow q) \wedge (p \rightarrow r));$
2. the permutation of the premises law: $(p \rightarrow (q \rightarrow r)) \rightarrow (q \rightarrow (p \rightarrow r));$
3. the reunion of the premises law: $(p \rightarrow (q \rightarrow r)) \rightarrow (p \wedge q \rightarrow r);$
4. the separation of the premises law: $(p \wedge q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r));$
5. the 'cut' law: $(p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r);$
6. the left-distribution of ' \vee ' over ' \rightarrow ': $p \vee (q \rightarrow r) \rightarrow ((p \vee q) \rightarrow (p \vee r));$
7. the syllogism law: $(p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r);$
8. the left-distribution of ' \rightarrow ' over ' \vee ': $(p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r)).$

Exercise 1.5.

Transform the formulas $U_j, j \in \{1, 2, \dots, 8\}$ into their equivalent conjunctive and disjunctive normal forms. Using one of these forms prove that $U_j, j \in \{1, 2, \dots, 8\}$ are valid formulas in propositional logic.

1. $U_1 = (p \rightarrow (q \leftrightarrow r)) \rightarrow ((p \rightarrow q) \leftrightarrow (p \rightarrow r));$
2. $U_2 = (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r);$
3. $U_3 = (p \wedge q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r));$

A Computational Approach to Classical Logics and Circuits

4. $U_4 = (p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r));$
5. $U_5 = (p \vee (q \leftrightarrow r)) \rightarrow ((p \vee q) \leftrightarrow (p \vee r));$
6. $U_6 = (p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r));$
7. $U_7 = (p \rightarrow (q \wedge r)) \rightarrow ((p \rightarrow q) \wedge (p \rightarrow r));$
8. $U_8 = p \vee (q \rightarrow r) \rightarrow ((p \vee q) \rightarrow (p \vee r)).$

Exercise 1.6.

Using the appropriate normal form write all the models of the following formulas:

1. $U_1 = (p \vee q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q;$
2. $U_2 = \neg(\neg p \vee q) \vee r \rightarrow \neg p \wedge \neg(q \wedge r);$
3. $U_3 = (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q;$
4. $U_4 = (p \vee q) \wedge \neg r \rightarrow p \wedge q \wedge r;$
5. $U_5 = p \vee \neg(q \wedge \neg r) \rightarrow p \wedge q \wedge \neg r;$
6. $U_6 = (p \vee q \rightarrow r) \rightarrow (q \rightarrow r) \wedge p;$
7. $U_7 = (q \vee r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q;$
8. $U_8 = (q \wedge r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q.$

Exercise 1.7.

Using the appropriate normal form, prove that the following formulas are inconsistent:

1. $U_1 = (p \rightarrow (q \rightarrow r)) \wedge \neg((p \rightarrow q) \rightarrow (p \rightarrow r));$
2. $U_2 = (\neg p \vee q) \wedge \neg(\neg q \rightarrow \neg p);$
3. $U_3 = (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \wedge (p \wedge \neg r);$
4. $U_4 = (p \rightarrow (q \vee r)) \wedge (\neg(p \rightarrow q) \wedge \neg(p \rightarrow r));$
5. $U_5 = p \wedge (q \rightarrow r) \wedge ((p \wedge q) \wedge \neg(p \wedge r));$
6. $U_6 = (p \rightarrow (q \rightarrow r)) \wedge (p \wedge q \wedge \neg r);$
7. $U_7 = (p \rightarrow (q \rightarrow r)) \wedge \neg(q \rightarrow (p \rightarrow r));$
8. $U_8 = (p \wedge q \rightarrow r) \wedge \neg(p \rightarrow (q \rightarrow r)).$

Exercise 1.8.

Write all the anti-models of the following formulas using CNF.

1. $U_1 = (q \wedge r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q;$
2. $U_2 = (q \vee r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q;$
3. $U_3 = (p \vee q \rightarrow r) \rightarrow (q \rightarrow r) \wedge p;$
4. $U_4 = p \vee \neg(q \wedge \neg r) \rightarrow p \wedge q \wedge \neg r;$
5. $U_5 = p \vee \neg(q \wedge \neg r) \rightarrow p \wedge q \wedge \neg r;$

Propositional Logic

6. $U_6 = (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q;$
7. $U_7 = \neg(\neg p \vee q) \vee r \rightarrow \neg p \wedge \neg(q \wedge r);$
8. $U_8 = (p \vee q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q.$

Exercise 1.9.

Using the definition of deduction, prove the following deductions:

- | | |
|---|---|
| 1. $p \rightarrow q, r \rightarrow t, p \vee r, \neg q \vdash t;$ | 2. $p \rightarrow r, p \vee r \rightarrow q, r \vdash q;$ |
| 3. $q \rightarrow p, t \rightarrow r, q \vee t, \neg p \vdash r;$ | 4. $p \vee (q \rightarrow r), p \vee q, \neg p \vdash r;$ |
| 5. $\neg p \vee \neg q \vee r, q, p \vdash r;$ | 6. $p \rightarrow \neg q \vee r, p \wedge q, p \vdash r;$ |
| 7. $r \vee (q \rightarrow p), r \vee q, \neg r \vdash p;$ | 8. $p \rightarrow q, q \rightarrow r, r \rightarrow t, p \vdash t.$ |

Exercise 1.10.

Prove the following theorems using the theorem of deduction and its reverse.

1. $\vdash p \vee (q \rightarrow r) \rightarrow ((p \vee q) \rightarrow (p \vee r));$
2. $\vdash (p \rightarrow (\neg r \rightarrow q)) \rightarrow (r \vee \neg p \vee q);$
3. $\vdash (p \rightarrow (q \rightarrow r)) \rightarrow (p \wedge q \rightarrow r);$
4. $\vdash (p \wedge q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r));$
5. $\vdash (p \rightarrow (q \rightarrow r)) \rightarrow (q \rightarrow (p \rightarrow r));$
6. $\vdash (p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r));$
7. $\vdash (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r);$
8. $\vdash (p \rightarrow q) \rightarrow ((p \rightarrow r) \rightarrow (p \rightarrow q \wedge r)).$

Exercise 1.11.

Using the theorem of deduction and its reverse prove that:

1. $\vdash (p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r));$
2. $\vdash (p \rightarrow q) \rightarrow ((\neg r \vee p) \rightarrow (r \rightarrow q));$
3. $\vdash p \vee (q \rightarrow r) \rightarrow ((p \vee q) \rightarrow (p \vee r));$
4. $\vdash (p \rightarrow r) \rightarrow ((q \rightarrow r) \rightarrow (p \vee q \rightarrow r));$
5. $\vdash (p \rightarrow q) \rightarrow ((r \rightarrow t) \rightarrow (p \wedge r \rightarrow q \wedge t));$
6. $\vdash (p \rightarrow r) \rightarrow ((p \wedge r \rightarrow q) \rightarrow (p \rightarrow q));$
7. $\vdash (\neg q \vee p) \rightarrow ((s \rightarrow q) \rightarrow (s \rightarrow p));$
8. $\vdash (p \rightarrow (q \rightarrow r)) \rightarrow (p \rightarrow (\neg r \rightarrow \neg q)).$

2. FIRST-ORDER LOGIC

First-order (predicate) logic was introduced by Gottlob Frege in 1879. It is an extension of propositional logic and allows reasoning about the objects of some non-empty universe and the relations among these objects. As references for the theoretical concepts and results presented in this chapter, the following papers were used [1, 2, 12, 16, 19, 20, 21, 23, 26, 37, 43, 44, 47, 54, 58, 60, 61, 62, 63].

2.1. The axiomatic (formal) system of first-order logic

In predicate logic new syntactic categories: quantifiers (existential and universal), constants, variables, function symbols and predicate symbols, are introduced.

We present an axiomatic (deductive) system for predicate logic as was proposed in paper [62]: $\text{Pr} = (\Sigma_{\text{Pr}}, F_{\text{Pr}}, A_{\text{Pr}}, R_{\text{Pr}})$, where:

- $\Sigma_{\text{Pr}} = \text{Var} \cup \text{Const} \cup (\bigcup_{j=1}^n F_j) \cup (\bigcup_{j=1}^m P_j) \cup \text{Connectives} \cup \text{Quantifiers}$ is the **vocabulary**
 - **Var** is the set of variable symbols $\{x, y, z, \dots\}$; the variables take different values in a specific domain D and they are generic terms, type definitions: *book*, *child*, *event*.
 - **Const** is the set of constants $\{a, b, c, \dots\}$; the constants take fixed values in a domain D and they usually specify objects names, persons names: *Paul*, *book_3*.
 - $F_i = \{f \mid f : D^i \rightarrow D\}$ is the set of function symbols of arity “ i ”
 - $P_i = \{p \mid p : D^i \rightarrow \{T, F\}\}$ is the set of predicate symbols of arity “ i ” and usually represent connection rules among variables and constants.
 - **Connectives** = $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$;
 - **Quantifiers** = $\{\forall (\text{universal quantifier}), \exists (\text{existential quantifier})\}$
- To define the well-formed predicate formulas we need to introduce first the concepts: term, atom, literal.
- **TERMS** is the set of terms defined as follows:
 - $\text{Var} \subset \text{TERMS}; \text{Const} \subset \text{TERMS};$
 - if $f \in F_k$ and $t_1, t_2, \dots, t_k \in \text{TERMS}$ then $f(t_1, t_2, \dots, t_k) \in \text{TERMS}$
 - examples: $x, a, f(x), g(x, a), g(f(x), y)$

First-order Logic

- **ATOMS** is the set of atomic formulas (atoms):
 - $T, F \in ATOMS$
 - if $P \in P_k$ and $t_1, \dots, t_k \in TERMS$ then $P(t_1, \dots, t_k) \in ATOMS$
 - examples: $T, F, P(x, y, a), Q(f(x), a), R(a, g(f(x), y))$
where: $x, y \in Var, a \in Const, f \in F_1, g \in F_2, P, R \in P_3, Q \in P_2.$
- **literal** - an atom or its negation.
 - examples: $P(f(x), a, y, g(x, b)), \neg Q(x, a, f(x)),$
where: $x, y \in Var, a, b \in Const, f \in F_1, g \in F_2, P \in P_4, Q \in P_3.$
- F_{Pr} is the set of well-formed predicate (first-order) formulas:
 - if $U, V \in F_{Pr}$ then
 $\neg U \in F_{Pr}, U \wedge V \in F_{Pr}, U \vee V \in F_{Pr}, U \rightarrow V \in F_{Pr}, U \leftrightarrow V \in F_{Pr}$
 - if $U \in F_{Pr}$, $x \in Var$, and x is not within the scope of a quantifier then
 $(\forall x)U(x) \in F_{Pr}$ and $(\exists x)U(x) \in F_{Pr}.$
- $A_{Pr} = \{A_1, A_2, A_3, A_4, A_5\}$ is the set of axioms:
 - $A_1 : U \rightarrow (V \rightarrow U)$
 - $A_2 : ((U \rightarrow (V \rightarrow Z)) \rightarrow ((U \rightarrow V) \rightarrow (U \rightarrow Z)))$
 - $A_3 : (U \rightarrow V) \rightarrow (\neg V \rightarrow \neg U)$ (*modus tollens*)
 - $A_4 : (\forall x)U(x) \rightarrow U(t)$, where t is a term, (*universal instantiation*)
 - $A_5 : (U \rightarrow V(y)) \rightarrow (U \rightarrow (\forall x)V(x))$, where x is not a free variable in U
or V , y is free in V and y does not appear in U .
- $R_{Pr} = \{mp, univ_gen\}$ is the set of inference rules:
 - *modus ponens* rule: $U, U \rightarrow V \vdash_{mp} V$
 - *universal generalization* rule:
 $U(x) \vdash_{univ_gen} (\forall x)U(x)$, where x is a free variable in U .

Remarks:

1. $A_P \subset A_{Pr}$, $F_P \subset F_{Pr}$, $R_P \subset R_{Pr}$ and thus $Theorems_P \subset Theorems_{Pr}$. The theorems in propositional logic are also theorems in predicate logic.
2. In axiom A_4 (*universal instantiation*), $(\forall x)U(x)$ says that $U(x)$ holds true for all the objects in the domain (universe), so U holds for any specific object in the domain. The term t used for instantiation can be a variable or a constant of the domain.
3. The *universal generalization* rule says that if U holds for any arbitrary element x of the universe, then we can conclude that $(\forall x)U(x)$.

A Computational Approach to Classical Logics and Circuits

Definition 2.1.

1. In a predicate formula the variables which are within the scope of a quantifier are called *bound variables*, all the others are called *free variables*.
2. A first-order formula is called a *closed formula* if all its variables are bound.
3. If a predicate formula contains at least one free variable, the *formula is open*.

Example 2.1.

- The predicate formula $(\forall x)(\exists z)(P(x, z, a) \vee (\exists y)Q(x, f(y)))$ is closed (all variables are bound), where: $x, y, z \in Var$, $a \in Const$, $f \in F_1$, $P \in P_3$, $Q \in P_2$.
- The predicate formula $(\forall x)P(x, y) \wedge Q(z, a)$ is open, $a \in Const$, $x, y, z \in Var$, $P, Q \in P_2$, the variables y and z are free and x is a bound variable (universal quantified - within the scope of \forall).

Definition 2.2. [62]

Let U_1, U_2, \dots, U_n, V be first-order formulas, U_1, U_2, \dots, U_n are the hypotheses. V is deducible (inferable, derivable) from U_1, U_2, \dots, U_n , notation: $U_1, U_2, \dots, U_n \vdash V$, if there is a sequence of formulas (f_1, f_2, \dots, f_m) such that $f_m = V$ and $\forall i \in \{1, \dots, m\}$ we have a) or b) or c) or d).

1. $f_i \in A_P$ (axiom of predicate logic);
2. $f_i \in \{U_1, U_2, \dots, U_n\}$ (hypothesis formula);
3. $f_{i_1}, f_{i_2} \vdash_{mp} f_i$, $i_1 < i$ and $i_2 < i$ (formula f_i is inferred, using *modus ponens* rule, from two formulas that are already in the sequence);
4. $f_j \vdash_{univ_gen} f_i$, $j < i$ (formula f_i is obtained using the *universal generalization* rule from a formula that exists already in the sequence).

The sequence (f_1, f_2, \dots, f_m) is called the *deduction* of V from U_1, U_2, \dots, U_n .

Definition 2.3.

A formula $U \in F_{Pr}$, such that $\emptyset \vdash U$ (notation: $\vdash U$) is called a *theorem*.

Remark:

- The theorems are the formulas deducible from the axioms, using modus ponens and the universal generalization rule.

In the inferential (deduction) process the inference rules specific to propositional logic (*modus ponens*, *addition*, *simplification*, *modus tollens*, *syllogism*) and the rules specific to predicate logic (*universal instantiation*, *universal generalization*, *existential instantiation*, *existential generalization*) are used.

First-order Logic

Inference rules for quantifiers	
<i>universal instantiation</i>	$(\forall x)U(x) \vdash_{univ_inst} U(t)$, t is a term (variable, constant of the domain)
<i>universal generalization</i>	$U(x) \vdash_{univ_gen} (\forall x)U(x)$, x is a free variable in U
<i>existential instantiation</i>	$(\exists x)U(x) \vdash_{exist_inst} U(c)$, c is a constant of the domain
<i>existential generalization</i>	$U(t) \vdash_{exist_gen} (\exists x)U(x)$, t is a variable or a constant of the domain, x must not appear free in U

Remarks:

1. The *existential instantiation rule* says that if U holds for some element of the universe, then we can give that element a name such as c . When selecting symbols, one must select them one at a time and must not use a symbol that has already been selected within the same reasoning/proof.
2. The *existential generalization rule* says that if there is some element t in the universe and t has the property U , then there exists something in the universe that has the property U .

Example 2.2.

Using the definition of deduction, prove that the formula $(\forall x)(P(x) \wedge Q(x)) \rightarrow (\forall x)P(x)$ is a theorem.

We build the sequence $(f_1, f_2, f_3, f_4, f_5)$ of formulas as follows:

$$f_1 : (\forall x)(P(x) \wedge Q(x)) \rightarrow P(y) \wedge Q(y) \quad - A_4 \text{ axiom where } y \text{ is the term}$$

used for the *instantiation of the universal quantified variable x*

$$f_2 : P(y) \wedge Q(y) \rightarrow P(y) \quad - \text{theorem (from propositional logic)}$$

$$f_1, f_2 \vdash_{syllogism_rule} f_3 = (\forall x)(P(x) \wedge Q(x)) \rightarrow P(y)$$

$$f_4 : ((\forall x)(P(x) \wedge Q(x)) \rightarrow P(y)) \rightarrow ((\forall x)(P(x) \wedge Q(x)) \rightarrow (\forall x)P(x))$$

- A_5 axiom

$$f_3, f_4 \vdash_{mp} f_5 = (\forall x)(P(x) \wedge Q(x)) \rightarrow (\forall x)P(x) \quad \text{modus ponens is applied}$$

$(f_1, f_2, f_3, f_4, f_5)$ is the deduction (proof) of the theorem $(\forall x)(P(x) \wedge Q(x)) \rightarrow (\forall x)P(x)$.

Example 2.3.

Using the definition of deduction, prove that:

$$(\forall y)(\forall z)(P(y) \vee Q(z)) \vdash_{-} (\forall x)(P(x) \vee Q(x))$$

The sequence $(f_1, f_2, f_3, f_4, f_5, f_6)$ of predicate formulas is obtained:

$$f_1 : (\forall y)(\forall z)(P(y) \vee Q(z)) \quad - \text{hypothesis}$$

A Computational Approach to Classical Logics and Circuits

$f_2 : (\forall y)(\forall z)(P(y) \vee Q(z)) \rightarrow (\forall z)(P(x) \vee Q(z))$ - A_4 axiom where x is the term used for the instantiation of the universal quantified variable y

$f_1, f_2 \vdash_{mp} f_3 = (\forall z)(P(x) \vee Q(z))$ - modus ponens is applied

$f_4 : (\forall z)(P(x) \vee Q(z)) \rightarrow P(x) \vee Q(x)$ - A_4 axiom where x is the term used for the instantiation of the universal quantified variable z

$f_3, f_4 \vdash_{mp} f_5 = P(x) \vee Q(x)$ - modus ponens is applied

$f_5 \vdash_{univ_gen} f_6 = (\forall x)(P(x) \vee Q(x))$ - universal generalization is applied

$(f_1, f_2, f_3, f_4, f_5, f_6)$ is the deduction (the proof) of $(\forall x)(P(x) \vee Q(x))$ from the hypothesis $(\forall y)(\forall z)(P(y) \vee Q(z))$.

Example 2.4.

Prove that the formula $(\forall x)(P(x) \vee Q(x))$ is derivable from the formula $(\forall y)(\forall z)(P(y) \vee Q(z))$, using the definition of deduction.

We have to prove that: $(\forall y)(\forall z)(P(y) \vee Q(z)) \vdash (\forall x)(P(x) \vee Q(x))$

The sequence $(f_1, f_2, f_3, f_4, f_5, f_6)$ of predicate formulas is build as follows:

$f_1 : (\forall y)(\forall z)(P(y) \vee Q(z))$ – hypothesis

$f_2 : (\forall y)(\forall z)(P(y) \vee Q(z)) \rightarrow (\forall z)(P(x) \vee Q(z))$ - A_4 axiom where x is the term used for the instantiation of the universal quantified variable y

$f_1, f_2 \vdash_{mp} f_3 = (\forall z)(P(x) \vee Q(z))$ - modus ponens is applied

$f_4 : (\forall z)(P(x) \vee Q(z)) \rightarrow P(x) \vee Q(x)$ - A_4 axiom where x is the term used for the instantiation of the universal quantified variable z

$f_3, f_4 \vdash_{mp} f_5 = P(x) \vee Q(x)$ modus ponens is applied

$f_5 \vdash_{univ_gen} f_6 = (\forall x)(P(x) \vee Q(x))$ universal generalization is applied

$(f_1, f_2, f_3, f_4, f_5, f_6)$ is the deduction (the proof) of $(\forall x)(P(x) \vee Q(x))$ from $(\forall y)(\forall z)(P(y) \vee Q(z))$.

Theorem 2.1. Theorem of deduction [20]

Let X be a set of predicate formulas and U, V predicate formulas.

If $X \cup \{U\} \vdash V$ then $X \vdash U \rightarrow V$.

Theorem 2.2. Refutation theorem [20]

Let X be a set of predicate formulas and U a predicate formula.

If $X \cup \{\neg U\}$ is inconsistent then $X \vdash U$.

The last theorem is used in proof methods such as: *resolution* and *semantic tableaux method*, called **refutation proof methods** and they model ‘reductio ad absurdum’ (proof by contradiction).

First-order Logic

Example 2.5.

Using the theorem of deduction, prove that the formula $(\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\forall x)B(x))$ is a theorem.

Step1:

We apply the reverse of the theorem of deduction:

If $\vdash (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\forall x)B(x))$ then

$(\forall x)(A(x) \rightarrow B(x)) \vdash ((\forall x)A(x) \rightarrow (\forall x)B(x))$ then

$(\forall x)(A(x) \rightarrow B(x)), (\forall x)A(x) \vdash (\forall x)B(x)$

Step2:

We prove that $(\forall x)(A(x) \rightarrow B(x)), (\forall x)A(x) \vdash (\forall x)B(x)$ building the sequence (f_1, f_2, \dots, f_8) of predicate formulas:

$f_1 : (\forall x)(A(x) \rightarrow B(x))$ - hypothesis

$f_2 : (\forall x)(A(x) \rightarrow B(x)) \rightarrow (A(y) \rightarrow B(y))$ - axiom A_4 , x is instantiated with the term y

$f_1, f_2 \vdash_{mp} f_3 = A(y) \rightarrow B(y)$

$f_4 : (\forall x)A(x)$ - hypothesis

$f_5 : (\forall x)A(x) \rightarrow A(y)$ - axiom A_4 , x is instantiated with the term y

$f_4, f_5 \vdash_{mp} f_6 = A(y)$

$f_3, f_6 \vdash_{mp} f_7 = B(y)$

$f_7 \vdash_{univ_gen} f_8 = (\forall x)B(x)$

(f_1, f_2, \dots, f_8) is the deduction of $(\forall x)B(x)$ from the hypothesis $(\forall x)(A(x) \rightarrow B(x))$ and $(\forall x)A(x)$.

Step3:

Using the deduction proved in Step2 and applying twice the theorem of deduction we obtain:

If $(\forall x)(A(x) \rightarrow B(x)), (\forall x)A(x) \vdash (\forall x)B(x)$ then

$(\forall x)(A(x) \rightarrow B(x)) \vdash ((\forall x)A(x) \rightarrow (\forall x)B(x))$ then

$\vdash (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\forall x)B(x))$

Thus we have proved that the initial formula is a theorem.

A Computational Approach to Classical Logics and Circuits

2.2. Transformation of natural language sentences into predicate formulas

In the following examples, sentences from natural language are translated into first-order language. We explain how the quantifiers, the variables, the constants, the function symbols and the predicate symbols are used in predicate formulas in order to introduce objects and to express properties and relations among objects.

1. All Computer Science (CS) students are smart.

$$(\forall x)(CS_student(x) \rightarrow smart(x))$$

- *CS_student* and *smart* are unary predicate symbols, expressing properties of the person x .

2. There is someone who studies at Babes-Bolyai University (*BBU*) and is smart.

$$(\exists x)(student_BBU(x) \wedge smart(x))$$

- *student_BBU* and *smart* are unary predicate symbols, expressing properties of the person x .

3. Every child loves anyone who gives the child any present.

$$(\forall x)(\forall y)(\forall z)(child(x) \wedge present(y) \wedge gives(z, x, y) \rightarrow loves(x, z))$$

- *child*, *present* are unary predicates symbols;
- *loves* is a binary predicate symbol (*loves*(x, y) is true if x loves y);
- *gives* is a ternary predicate symbol (*gives*(z, x, y) is true if z gives to x the *present* y).

4. John's relatives, except Mary, live in Cluj, some of them like opera music, but all of them like to dance.

$$(\forall x)(relative(John, x) \rightarrow likes_to_dance(x) \wedge (\neg equal(x, Mary)$$

$$\rightarrow lives(x, Cluj))) \wedge (\exists y)(relative(John, y) \wedge likes_opera(y))$$

- *John, Mary, Cluj* are constants;
- *likes_to_dance, likes_opera* are unary predicate symbols;
- *relative, lives, equal* are binary predicate symbols;
- *relative* is a symmetric binary relation, thus in a reasoning process involving this relation we have to add the formula:
 $(\forall x)(\forall y)(relative(x, y) \rightarrow relative(y, x));$
- the predicate *equal* is defined by the following axioms:
 $(\forall x)equal(x, x)$ – reflexivity;
 $(\forall x)(\forall y)(equal(x, y) \rightarrow equal(y, x))$ – symmetry;
 $(\forall x)(\forall y)(\forall z)(equal(x, y) \wedge equal(y, z) \rightarrow equal(x, z))$ – transitivity.

First-order Logic

5. If x and y are nonnegative integers and x is greater than y , then x^2 is greater than y^2 .

$$(\forall x)(\forall y)(\text{nonneg}(x) \wedge \text{nonneg}(y) \wedge \text{greater}(x, y) \rightarrow \text{greater}(\text{square}(x), \text{square}(y)))$$

- function symbol: $\text{square} \in F_1$, $\text{square}(x) = x^2$;
- predicate symbols: $\text{nonneg} \in P_1$, $\text{nonneg}(x): "x > 0"$ and $\text{greater} \in P_2$, $\text{greater}(x, y): "x > y"$.

6. In a plane if a line x is parallel to a constant line d then all the lines perpendicular to x are also perpendicular to d .

$$(\forall x)(\text{parallel}(x, d) \rightarrow (\forall y)(\text{perpendicular}(y, x) \rightarrow \text{perpendicular}(y, d)))$$

- perpendicular and parallel are binary predicate symbols corresponding to the geometric relations. In a reasoning process involving these relations we have to add the formulas expressing the properties of reflexivity (for parallel), symmetry (for parallel and perpendicular), transitivity (for parallel).

7. The axioms which define the natural numbers:

- a₁. Every natural number has a unique immediate successor.

existence: $(\forall x)(\exists y)\text{equal}(y, \text{successor}(x))$

uniqueness:

$$(\forall x)(\forall y)(\forall z)(\text{equal}(y, \text{successor}(x)) \wedge \text{equal}(z, \text{successor}(x)) \rightarrow \text{equal}(y, z))$$

- a₂. The number 0 is not the immediate successor of a natural number.

$\neg(\exists x)\text{equal}(0, \text{successor}(x))$, 0 is a constant.

- a₃. Every natural number, except 0, has a unique immediate successor.

existence: $(\forall x)(\exists y)(\neg \text{equal}(0, x) \wedge \text{equal}(y, \text{successor}(x)))$

uniqueness:

$$(\forall x)(\forall y)(\forall z)(\text{equal}(y, \text{successor}(x)) \wedge \text{equal}(z, \text{successor}(x)) \rightarrow \text{equal}(y, z))$$

- unary functions: successor , predecessor ;

- binary predicate: equal is reflexive, symmetric and transitive.

The following formulas express the equality of the successors and the predecessors of two equal numbers:

$$(\forall x)(\forall y)(\text{equal}(x, y) \rightarrow \text{equal}(\text{successor}(x), \text{successor}(y)))$$

$$(\forall x)(\forall y)(\text{equal}(x, y) \rightarrow \text{equal}(\text{predecessor}(x), \text{predecessor}(y)))$$

A Computational Approach to Classical Logics and Circuits

2.3. The semantics of first-order (predicate) logic

The semantics of predicate logic realize the connection between the constant symbols, the function symbols, the predicate symbols and the real constants, functions, predicates from the modeled universe. Also, a meaning for each formula from the language, in terms of the modeled universe, is provided.

Definition 2.4.

An **interpretation** for a language L of predicate logic is a pair $I = \langle D, m \rangle$, where:

1. D is a nonempty set called the domain of interpretation.
2. m is a function that assigns:
 - a fixed value $m(c) \in D$ to the constant c .
 - a function $m(f): D^n \rightarrow D$ to each n -ary function symbol f ;
 - a predicate $m(P): D^n \rightarrow \{T, F\}$ to each n -ary predicate symbol P .

Notations: $I = \langle D, m \rangle$ be an interpretation.

- $|I| = D$ is the domain of I , Var is the set of variables.
- $I|X|$ is $m(X)$ where X is a predicate symbol or a function symbol.
- $As(I)$ is the set of assignment functions for variables over the domain of interpretation I .
An assignment function $fa \in As(I)$ is defined as follows: $fa : Var \rightarrow |I|$.
- $[fa]_x = \{fa' | fa' \in As(I) \text{ and } fa'(y) = fa(y), \text{ for every } y \neq x\}$.

Definition 2.5.

Let U and V be first-order formulas, A a formula free of the variable x , I an interpretation and $fa \in As(I)$ an assignment function. The evaluation function v_{fa}^I is defined inductively as follows:

1. $v_{fa}^I(x) = fa(x), x \in Var$;
2. $v_{fa}^I(c) = I|c|, c \in Const$;
3. $v_{fa}^I(f(t_1, t_2, \dots, t_n)) = I|g|(v_{fa}^I(t_1), v_{fa}^I(t_2), \dots, v_{fa}^I(t_n)), g \in F_n, n > 0$;
4. $v_{fa}^I(P(t_1, t_2, \dots, t_n)) = I|P|(v_{fa}^I(t_1), v_{fa}^I(t_2), \dots, v_{fa}^I(t_n)), P \in P_n, n > 0$;
5. $v_{fa}^I(\neg U) = \neg v_{fa}^I(U)$;
6. $v_{fa}^I(U \wedge V) = v_{fa}^I(U) \wedge v_{fa}^I(V)$;
7. $v_{fa}^I(U \vee V) = v_{fa}^I(U) \vee v_{fa}^I(V)$;
8. $v_{fa}^I(U \rightarrow V) = v_{fa}^I(U) \rightarrow v_{fa}^I(V)$;
9. $v_{fa}^I((\exists x)A(x)) = T$ if and only if $v_{fa'}^I(A(x)) = T$ for a function $fa' \in [fa]_x$
10. $v_{fa}^I((\forall x)A(x)) = T$ if and only if $v_{fa'}^I(A(x)) = T$ for any function $fa' \in [fa]_x$

First-order Logic

Definition 2.6.

1. A first-order formula U is *satisfiable (consistent)* if there exists an interpretation I and an assignment function $fa \in As(I)$ such that $v_{fa}^I(U) = T$. Otherwise, the formula is called *unsatisfiable (inconsistent)*.
2. A first-order formula U is *true under the interpretation I* if for any assignment function $fa \in As(I)$, $v_{fa}^I(U) = T$, notation: $\models_I U$, and I is called *model* of U .
3. A first-order formula U is *false under the interpretation I* if for any assignment function $fa \in As(I)$, $v_{fa}^I(U) = F$, and I is called *anti-model* of U .
4. A first-order formula U is *valid (tautology)* if U is true under all the possible interpretations, notation: $\models U$.
5. The first-order formulas U and V are *logically equivalent* if $v_{fa}^I(U) = v_{fa}^I(V)$ for any interpretation I and any assignment function fa , notation: $U \equiv V$.
6. A set of first-order formulas S *logically implies* the first-order formula V if all the models of the set S (the models of the conjunction of all formulas from S) are also models of the formula V . We say that V is a *logical consequence* of the set S , notation: $S \models V$.
7. A *set* of first-order formulas is *consistent* if the conjunction of all its formulas has at least one model. A *set* of formulas is *inconsistent* if the conjunction of all its formulas does not have a model.

Remarks:

1. The evaluation of a closed formula U depends only on the interpretation in which we want to evaluate it, notation: $v^I(U)$.
2. Any first-order (predicate) formula has an infinite number of interpretations.

Example 2.6.

Build a model and an anti-model for the closed predicate formula:

$$U = (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall x)P(x) \vee (\forall x)Q(x)$$

- Let us consider the interpretation $I_1 = \langle D_1, m \rangle$, where:

$D_1 = \mathbb{N}$ (the set of natural numbers)

$m(P) : \mathbb{N} \rightarrow \{T, F\}, m(P)(x) = "x:2"$

$m(Q) : \mathbb{N} \rightarrow \{T, F\}, m(Q)(x) = "x:3"$.

$$\begin{aligned}v^{I_1}(U) &= v^{I_1}((\forall x)(P(x) \vee Q(x))) \rightarrow v^{I_1}((\forall x)P(x) \vee (\forall x)Q(x)) \\&= v^{I_1}((\forall x)(P(x) \vee Q(x))) \rightarrow v^{I_1}((\forall x)P(x)) \vee v^{I_1}((\forall x)Q(x)) \\&= (\forall x)_{x \in \mathbb{N}}(x:2 \vee x:3) \rightarrow (\forall x)_{x \in \mathbb{N}}(x:2) \vee (\forall x)_{x \in \mathbb{N}}(x:3) \\&= F \rightarrow F \vee F = F \rightarrow F = T.\end{aligned}$$

A Computational Approach to Classical Logics and Circuits

$v^{I_1}(U) = T$, U is evaluated as true under the interpretation I_1 which is a model of U .

- Let us consider the interpretation $I_2 = \langle D_2, m \rangle$, where:

$D_2 = \{4, 9\}$ – the domain of interpretation;

$m(P) : \{4, 9\} \rightarrow \{T, F\}, m(P)(x) = "x:2"$

$m(Q) : \{4, 9\} \rightarrow \{T, F\}, m(Q)(x) = "x:3".$

To evaluate the formula U under the interpretation I_2 , with the finite domain $D_2 = \{4, 9\}$, the universally quantified subformulas are replaced by the conjunction of their instances for $x = 4$ and $x = 9$.

$$\begin{aligned} v^{I_2}(U) &= v^{I_2}((\forall x)(P(x) \vee Q(x))) \rightarrow v^{I_2}((\forall x)P(x) \vee (\forall x)Q(x)) \\ &= v^{I_2}((\forall x)(P(x) \vee Q(x))) \rightarrow v^{I_2}((\forall x)P(x)) \vee v^{I_2}((\forall x)Q(x)) \\ &= (4:2 \vee 4:3) \wedge (9:2 \vee 9:3) \rightarrow (4:2 \wedge 9:2) \vee (4:3 \wedge 9:3) \\ &= (T \vee F) \wedge (F \vee T) \rightarrow (T \wedge F) \vee (F \wedge T) = T \wedge T \rightarrow F \vee F. \\ &= T \rightarrow F = F \end{aligned}$$

I_2 evaluates the formula U as false, I_2 is an anti-model of U and thus U is not a valid formula, it is a contingent one (I_1 is a model of U).

Example 2.7.

Evaluate the open formula $U(z)$ under the interpretations I_1 and I_2

$$U(z) = (\exists x)(\exists y)P(f(x, y), z)$$

- $I_1 = \langle D_1, m_1 \rangle$, $D = \mathbb{Z}$ (the set of integer numbers),

$$m_1(f)(x, y) = (x + y)^2 \text{ and } m_1(P)(x, y) : "x = y".$$

Because $U(z)$ is an open formula, its evaluation depends on the assignment of values (integers) to the free variable z , $fa \in As(I_1)$, where:

$$v_{fa}^{I_1}(U(z)) = (\exists x)_{x \in \mathbb{Z}} (\exists y)_{y \in \mathbb{Z}} "(x + y)^2 = fa(z)" = \begin{cases} T, & \text{if } fa(z) \text{ is a square} \\ F, & \text{otherwise} \end{cases}$$

$U(z)$ is a consistent formula, but it is not a true formula under the interpretation I_1 , therefore I_1 is not a model of the formula.

- $I_2 = \langle D_2, m_2 \rangle$, $D = \mathbb{Z}$ (the set of integer numbers),

$$m_2(f)(a, b) = a + b \text{ and } m_2(P)(a, b) : "a = b".$$

$$v_{fa}^{I_2}(U(z)) = (\exists x)_{x \in \mathbb{Z}} (\exists y)_{y \in \mathbb{Z}} "x + y = fa(z)" = T, \forall fa(z) \in \mathbb{Z}.$$

The formula $U(z)$ is true under the interpretation I_2 : every integer number $fa(z)$ is the sum of two integer numbers x and y .

$\models_{I_2} U(z)$, therefore I_2 is a model of $U(z)$.

First-order Logic

2.4. Logical equivalences in predicate logic

- **Expansion laws**

$(\forall x)A(x) \equiv (\forall x)A(x) \wedge A(t)$ – the universal quantifier is an infinitary conjunction,
 $(\exists x)A(x) \equiv (\exists x)A(x) \vee A(t)$ – the existential quantifier is an infinitary disjunction,
 where t is a term which does not contain x .

- **DeMorgan's infinitary laws**

$$\neg(\exists x)A(x) \equiv (\forall x)\neg A(x) \quad \neg(\forall x)A(x) \equiv (\exists x)\neg A(x)$$

- **Quantifiers interchanging laws**

$$(\exists x)(\exists y)A(x, y) \equiv (\exists y)(\exists x)A(x, y) \quad (\forall x)(\forall y)A(x, y) \equiv (\forall y)(\forall x)A(x, y)$$

Remark: $(\exists x)(\forall y)B(x, y) \neq (\forall y)(\exists x)B(x, y)$

Quantifiers of the same type commute, but quantifiers of different type do not commute.

- **The extraction of quantifiers in front of the formula laws**

$$A \vee (\exists x)B(x) \equiv (\exists x)(A \vee B(x)) \quad A \vee (\forall x)B(x) \equiv (\forall x)(A \vee B(x))$$

$$A \wedge (\exists x)B(x) \equiv (\exists x)(A \wedge B(x)) \quad A \wedge (\forall x)B(x) \equiv (\forall x)(A \wedge B(x))$$

where A does not contain x as a free variable.

$$(\exists x)A(x) \vee B \equiv (\exists x)(A(x) \vee B) \quad (\forall x)A(x) \vee B \equiv (\forall x)(A(x) \vee B)$$

$$(\exists x)A(x) \wedge B \equiv (\exists x)(A(x) \wedge B) \quad (\forall x)A(x) \wedge B \equiv (\forall x)(A(x) \wedge B)$$

where B does not contain x as a free variable.

- **Distributive laws**

$$(\exists x)(A(x) \vee B(x)) \equiv (\exists x)A(x) \vee (\exists x)B(x) \text{ distribution of "}\exists\text{" over "}\vee\text{"}$$

$$(\forall x)(A(x) \wedge B(x)) \equiv (\forall x)A(x) \wedge (\forall x)B(x) \text{ distribution of "}\forall\text{" over "}\wedge\text{"}$$

The above laws are pairs of dual equivalences. " \forall " and " \exists " are dual quantifiers.

Remark: The distribution of

" \exists " over " \wedge ", " \forall " over " \vee ", " \exists " over " \rightarrow ", " \forall " over " \rightarrow " do not provide valid distributive laws. These combinations of quantifiers and connectives express only semi-distributive laws as follows:

Semi-distributivity of " \exists " over " \wedge ":

$$\models (\exists x)(A(x) \wedge B(x)) \rightarrow (\exists x)A(x) \wedge (\exists x)B(x) \text{ and}$$

$$\not\models (\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\exists x)(A(x) \wedge B(x))$$

Semi-distributivity of " \forall " over " \vee ":

$$\models (\forall x)A(x) \vee (\forall x)B(x) \rightarrow (\forall x)(A(x) \vee B(x)) \text{ and}$$

$$\not\models (\forall x)(A(x) \vee B(x)) \rightarrow (\forall x)A(x) \vee (\forall x)B(x)$$

A Computational Approach to Classical Logics and Circuits

Semi-distributivity of " \exists " over " \rightarrow ":

$$\models ((\exists x)A(x) \rightarrow (\exists x)B(x)) \rightarrow (\exists x)(A(x) \rightarrow B(x)) \text{ and}$$
$$\not\models (\exists x)(A(x) \rightarrow B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\exists x)B(x))$$

Semi-distributivity of " \forall " over " \rightarrow ":

$$\models (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\forall x)B(x)) \text{ and}$$
$$\not\models ((\forall x)A(x) \rightarrow (\forall x)B(x)) \rightarrow (\forall x)(A(x) \rightarrow B(x))$$

Example 2.8.

Prove that the universal and existential quantifiers do not commute.

We have to prove that the logical equivalence:

$$(\exists x)(\forall y)L(x, y) \equiv (\forall y)(\exists x)L(x, y) \text{ does not hold.}$$

We choose the interpretation $I = \langle D, m \rangle$, where:

- D is the set of all persons in the world
- $m(L) : D \times D \rightarrow \{T, F\}, m(L)(x, y) = "x loves y"$

Under the interpretation I , the formula $U_1 = (\exists x)(\forall y)L(x, y)$ has the meaning:

"There exists a person who loves all persons."

The formula $U_2 = (\forall y)(\exists x)L(x, y)$ has the meaning: *"All persons are loved by at least one person."*, under the same interpretation I .

These two natural language statements are not equivalent, so $U_1 \neq U_2$, but note that $U_1 \models U_2$.

Example 2.9.

Prove that the formula

$$U = (\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\exists x)(A(x) \wedge B(x))$$
 is not valid.

Let us consider the interpretation $I = \langle D, m \rangle$, where D is the set of all straight lines belonging to a plan P .

Let $d \in P$ be a constant object (line) from the domain of interpretation.

$$m(A) : D \rightarrow \{T, F\}, m(A)(x) = "x \perp d";$$

$$m(B) : D \rightarrow \{T, F\}, m(B)(x) = "x \parallel d";$$

$$v^I(U) = v^I((\exists x)A(x) \wedge (\exists x)B(x)) \rightarrow v^I((\exists x)(A(x) \wedge B(x))) =$$

$$= v^I((\exists x)A(x)) \wedge v^I((\exists x)B(x)) \rightarrow v^I((\exists x)(A(x) \wedge B(x))) =$$

$$= (\exists x)_{x \in D}(x \perp d) \wedge (\exists x)_{x \in D}(x \parallel d) \rightarrow (\exists x)_{x \in D}(x \perp d \wedge x \parallel d) =$$

$$= T \wedge T \rightarrow F = T \rightarrow F = F.$$

U is evaluated as false under the interpretation I , so I is an anti-model of U .

We conclude that U is not a valid formula, but it is consistent, having J as a model.

First-order Logic

$J = \langle D_1, m_1 \rangle$, where:

D_1 - the set of all persons living in Cluj-Napoca.

$m_1(A) : D_1 \rightarrow \{T, F\}, m_1(A)(x) : "x \text{ owns a car}"$;

$m_1(B) : D_1 \rightarrow \{T, F\}, m_1(B)(x) : "x \text{ has blue eyes}"$;

$$\begin{aligned} v^J(U) &= v^J((\exists x)A(x) \wedge (\exists x)B(x)) \rightarrow v^J((\exists x)(A(x) \wedge B(x))) = \\ &= v^J((\exists x)A(x)) \wedge v^J((\exists x)B(x)) \rightarrow v^J((\exists x)(A(x) \wedge B(x))) = \\ &= (\exists x)_{x \in D_1} "x \text{ owns a car}" \wedge (\exists x)_{x \in D_1} "x \text{ has blue eyes}" \rightarrow \\ &\quad \rightarrow (\exists x)_{x \in D_1} "x \text{ owns a car and } x \text{ has blue eyes}" \\ &= T \wedge T \rightarrow T = T \rightarrow T = T. \end{aligned}$$

U is evaluated as true under the interpretation J , J is a model of U .

Theorem 2.3. [58]

Soundness and completeness theorem states the equivalence between “*logical consequence*” and “*syntactic consequence*” concepts.

Let $U_1, \dots, U_{n-1}, U_n, V$ be first-order formulas.

1. completeness: if $U_1, \dots, U_{n-1}, U_n \models V$ then $U_1, \dots, U_{n-1}, U_n \vdash V$.
2. soundness: if $U_1, \dots, U_{n-1}, U_n \vdash V$ then $U_1, \dots, U_{n-1}, U_n \models V$.

A particular case of this theorem is the following result:

“A formula is a tautology if and only if it is a theorem in first-order logic.”

Theorem 2.4. (A. Church, 1936) [13]

The problem of the validity of a first-order formula is *undecidable*, but it is *semi-decidable*. If a procedure *Proc* is used to check the validity of a first-order formula U we have the following cases:

1. if U is a valid formula, then *Proc* ends with the corresponding answer.
2. if the formula U is not valid, then *Proc* ends with the corresponding answer or *Proc* may never stop.

The above result can be improved for fragments of first-order logic as follows:

Theorem 2.5. [17]

There is a decision procedure to check the validity of the formulas belonging to the following classes of formulas with the prefix of the prenex form:

1. $\forall^* \exists^* : (\forall x_1) \dots (\forall x_n) (\exists y_1) \dots (\exists y_m), m, n \geq 0$;
2. $\forall^* \exists \forall^* : (\forall x_1) \dots (\forall x_n) (\exists y) (\forall z_1) \dots (\forall z_m), m, n \geq 0$;
3. $\forall^* \exists \exists \forall^* : (\forall x_1) \dots (\forall x_n) (\exists y_1) (\exists y_2) (\forall z_1) \dots (\forall z_m), m, n \geq 0$.

A Computational Approach to Classical Logics and Circuits

The proof methods used in predicate logic:

- semantic tableaux method (semantic and refutation);
- resolution (syntactic and refutation);
- sequent/anti-sequent calculi (syntactic and direct);
- definition of deduction – axiomatic system (syntactic and direct)
- Herbrand-based procedure (syntactic and refutation)

2.5. Normal forms in first-order logic

The normal forms of predicate formulas are used as input data in proof methods such as *resolution* and *Herbrand-based procedure*.

Definition 2.7.

A predicate formula U is in *prenex normal form* if it has the form: $(Q_1x_1)\dots(Q_nx_n)M$, where $Q_i, i=1,\dots,n$ are quantifiers, and M is quantifier-free.

The sequence $(Q_1x_1)\dots(Q_nx_n)$ is called the *prefix of the formula* U , and M is called the *matrix of the formula* U . A predicate formula is in *conjunctive prenex normal form* if it is in prenex normal form and the matrix is in CNF.

Theorem 2.6. [20]

A predicate formula admits a logical equivalent conjunctive prenex normal form.

The prenex normal form is obtained by applying transformations which preserve the logical equivalence, according to the following algorithm:

Step 1: The connectives " \rightarrow " and " \leftrightarrow " are replaced using the connectives

" \neg , \wedge , \vee ".

Step 2: The bound variables are renamed such that they will be distinct.

Step 3: Application of infinitary DeMorgan's laws.

Step 4: The extraction of quantifiers in front of the formula laws are applied.

Step 5: The matrix is transformed into CNF using DeMorgan's laws and the distributive laws.

Remarks:

- After the Step 4 we obtain the prenex normal form which is not unique. If the formula obtained after the second step contains n distinct and independent groups of quantifiers, these groups can be extracted in an arbitrary order, therefore there exist $n!$ prenex normal forms, logically equivalent to the initial formula.
- A conjunctive prenex normal form is obtained after Step 5.

First-order Logic

Definition 2.8.

Let U be a first-order formula, and $U^P = (Q_1 x_1) \dots (Q_n x_n) M$ be one of its conjunctive prenex normal form.

1. A formula in **Skolem normal form**, denoted by U^S corresponds to U and it is obtained as follows:

For each existential quantifier Q_r from the prefix we apply the transformation:

- If Q_r is the left-most universal quantifier in the prefix, then we introduce a new constant a , and we replace in M all the occurrences of x_r by a . $(Q_r x_r)$ is deleted from the prefix.
- If Q_{s_1}, \dots, Q_{s_m} , $1 \leq s_1 < \dots < s_m < r$, are all the universal quantifiers at the left side of Q_r , then we introduce a new m -place function symbol, f , and we replace in M all occurrences of x_r by $f(x_{s_1}, \dots, x_{s_m})$. $(Q_r x_r)$ is deleted from the prefix.

The constants and functions used to replace the existential quantified variables are called **Skolem constants** and **Skolem functions**. The prefix of the formula U^S contains only universal quantifiers, and the matrix is in conjunctive normal form.

2. A formula in **clausal normal form** denoted by U^C corresponds to U and it is obtained by deleting the prefix of U^S .

Remarks:

The transformations used in the Skolemization process do not preserve the logical equivalence but preserve the inconsistency according to the following theorem.

Theorem 2.7.

Let U_1, U_2, \dots, U_n, V be first-order formulas.

1. V is inconsistent if and only if V^P is inconsistent, if and only if V^S is inconsistent, if and only if V^C is inconsistent.
2. The set $\{U_1, U_2, \dots, U_n\}$ is inconsistent if and only if the set $\{U_1^C, U_2^C, \dots, U_n^C\}$ is inconsistent.

Example 2.10.

Transform into prenex normal form and Skolem normal form the formula:

$$U = \neg((\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x)))$$

A Computational Approach to Classical Logics and Circuits

$$U = \neg((\forall x)(P(x) \rightarrow Q(x)) \stackrel{1}{\rightarrow} \stackrel{2}{\rightarrow} ((\forall x)P(x) \stackrel{3}{\rightarrow} (\forall x)Q(x)))$$

Step 1: replace the innermost " \rightarrow " connectives, denoted by 1 and 3

$$U \equiv \neg((\forall x)(\neg P(x) \vee Q(x)) \rightarrow (\neg(\forall x)P(x) \vee (\forall x)Q(x)))$$

Step 1: replace " \rightarrow " connective

$$U \equiv \neg(\neg((\forall x)(\neg P(x) \vee Q(x))) \vee (\neg(\forall x)P(x) \vee (\forall x)Q(x)))$$

Step 2: rename the bound variables such that they will be distinct

$$U \equiv \neg(\neg((\forall x)(\neg P(x) \vee Q(x))) \vee (\neg(\forall y)P(y) \vee (\forall z)Q(z)))$$

Step 3: apply DeMorgan's laws

$$U \equiv \neg(\neg((\forall x)(\neg P(x) \vee Q(x))) \vee (\neg(\forall y)P(y) \vee (\forall z)Q(z)))$$

Step 3: apply DeMorgan's laws

$$U \equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge \neg(\neg(\forall y)P(y) \vee (\forall z)Q(z))$$

Step 4: extract the quantifiers in front of the formula

$$U \equiv (\exists z)(\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(z)) = U_1 = U_1^P$$

$$U \equiv (\forall x)(\exists z)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(z)) = U_2 = U_2^P$$

$$U \equiv (\forall x)(\forall y)(\exists z)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(z)) = U_3 = U_3^P$$

U_1, U_2, U_3 are three prenex forms of the initial formula U .

Because the formula U contains 3 distinct and independent bound variables, there exists $3! = 6$ prenex normal forms logically equivalent to U .

Note that the matrix is in CNF, and we will not apply Step 5.

U_1^P, U_2^P, U_3^P are prenex normal forms.

After the Skolemization process, applied to the formulas U_1^P, U_2^P, U_3^P we obtain:

$$U_1^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a)), \quad [z \leftarrow a],$$

a is Skolem constant

$$U_2^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(f(x))), \quad [z \leftarrow f(x)],$$

f is a unary Skolem function

$$U_3^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(g(x, y))), \quad [z \leftarrow g(x, y)],$$

g is a binary Skolem function

The clausal normal forms are obtained by eliminating the universal quantifiers.

$$U_1^C = (\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a)$$

$$U_2^C = (\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(f(x))$$

$$U_3^C = (\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(g(x, y))$$

Example 2.11.

Transform into prenex normal form and Skolem normal form the formula:

$$U = (\exists x)(\forall y)P(x, y) \vee (\exists z)(\neg Q(z) \vee (\forall u)(\exists t)R(z, u, t))$$

$$U^P = (\exists x)(\forall y)(\exists z)(\forall u)(\exists t)(P(x, y) \vee \neg Q(z) \vee R(z, u, t))$$

$$U^S = (\forall y)(\forall u)(P(a, y) \vee \neg Q(f(y)) \vee R(f(y), u, g(y, u))), \text{ where:}$$

$[x \leftarrow a], [z \leftarrow f(y)], [t \leftarrow g(y, u)]$, a - Skolem constant,
 f, g - Skolem functions

$$U^c = P(a, y) \vee \neg Q(f(y)) \vee R(f(y), u, g(y, u))$$

2.6. Substitutions and unification

In this section we introduce *substitutions* and *unification*, used in predicate resolution (see chapter 5).

Definition 2.9.

A *substitution* is a mapping from the set of variables Var into the set of terms: **TERMS**. We denote by $\theta = [x_1 \leftarrow t_1, \dots, x_k \leftarrow t_k]$, a substitution, representing a finite set of replacements of variables, where x_1, \dots, x_k are distinct variables, t_1, \dots, t_k are terms, such that $\forall i = 1, \dots, k : t_i \neq x_i$ and x_i is not a subterm of t_i . $dom(\theta) = \{x_1, \dots, x_k\}$ is called the *domain* of θ .

We use Greek letters: $\phi, \delta, \phi, \eta, \theta, \lambda$ to stand for substitutions. The empty substitution is denoted by ε .

Definition 2.10.

The result of applying the substitution $\theta = [x_1 \leftarrow t_1, \dots, x_k \leftarrow t_k]$ to a formula is defined recursively as follows:

1. $\theta(x_i) = t_i, x_i \in dom(\theta)$;
2. $\theta(x) = x, x \notin dom(\theta)$;
3. $\theta(c) = c, c - \text{constant}$;
4. $\theta(f(t_1, \dots, t_n)) = f(\theta(t_1), \dots, \theta(t_n)), f \in F_n$;
5. $\theta(p(t_1, \dots, t_n)) = p(\theta(t_1), \dots, \theta(t_n)), p \in P_n$;
6. $\theta(\neg U) = \neg \theta(U)$;
7. $\theta(U \circ V) = \theta(U) \circ \theta(V), \circ \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$

Note: An instance $\theta(U)$ of U is obtained by replacing simultaneously each occurrence of x_i in U by t_i .

A Computational Approach to Classical Logics and Circuits

Definition 2.11.

The **composition** of two substitutions $\theta_1 = [x_1 \leftarrow t_1, \dots, x_k \leftarrow t_k]$ and $\theta_2 = [y_1 \leftarrow s_1, \dots, y_n \leftarrow s_n]$ is defined as:

$$\theta = \theta_1 \theta_2 = [x_i \leftarrow \theta_2(t_i) \mid x_i \in \text{dom}(\theta_1), x_i \neq \theta_2(t_i)] \cup [y_j \leftarrow s_j \mid y_j \in \text{dom}(\theta_2) \setminus \text{dom}(\theta_1)]$$

Properties: Let $\theta, \theta_1, \theta_2, \theta_3$ be substitutions.

- $\varepsilon\theta = \theta\varepsilon = \theta$; ε - empty substitution;
- $\theta_1(\theta_2\theta_3) = (\theta_1\theta_2)\theta_3$ associativity property;
- $\theta_1\theta_2 \neq \theta_2\theta_1$, the composition of two substitutions is not commutative

Example 2.12.

We prove that the composition of two substitutions is not commutative

$$\theta_1 = [x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u],$$

$$\theta_2 = [y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))]$$

$$\begin{aligned}\lambda_1 = \theta_1\theta_2 &= [x \leftarrow \theta_2(f(y)), y \leftarrow \theta_2(f(a)), z \leftarrow \theta_2(u)] \cup [u \leftarrow z, v \leftarrow f(f(a))] = \\ &= [x \leftarrow f(g(a)), y \leftarrow f(a), z \leftarrow z, u \leftarrow z, v \leftarrow f(f(a))] = \\ &= [x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))]\end{aligned}$$

$$\lambda_2 = \theta_2\theta_1 = [y \leftarrow g(a), v \leftarrow f(f(a)), x \leftarrow f(y), z \leftarrow u]$$

$\lambda_1 \neq \lambda_2$, so the composition of two substitutions is not commutative in general.

We compute for $U = p(u, v, x, y, z)$ the instances $\lambda_1(U)$ and $\lambda_2(U)$:

$$\lambda_1(U) = p(z, f(f(a)), f(g(a)), f(a), z)$$

$$\lambda_2(U) = p(u, f(f(a)), f(y), g(a), u)$$

Definition 2.12.

1. A substitution θ is a **unifier** of the terms t_1 and t_2 if $\theta(t_1) = \theta(t_2)$.
The term $\theta(t_1)$ is called the **common instance** of the unified terms.
2. A **unifier of a set** $\{U_1, \dots, U_n\}$ of formulas is a substitution θ such that $\theta(U_1) = \dots = \theta(U_n)$.

Example 2.13.

Let x, y be variables, a, b constants, f, g function symbols and P, Q predicate symbols

- the terms $f(x, x)$ and $f(a, b)$ cannot be unified because they do not have a common instance if a, b are distinct constants.
- $P(x)$ and $P(f(x))$ are not unifiable because x is a subterm of $f(x)$.
- the atoms $P(x)$ and $Q(a)$ are not unifiable because they do not have the same predicate symbol.
- the atoms $P(x, y, z)$ and $P(a, b)$ are not unifiable because the predicate symbols have different arity.

First-order Logic

- $t_1 = f(x, b)$, $t_2 = f(a, y)$ have a common instance $f(a, b) = \theta(t_1) = \theta(t_2)$, where $\theta = [x \leftarrow a, y \leftarrow b]$ is a unifier for these two terms.
- $t_1 = g(g(x))$ and $t_2 = g(y)$ have many unifiers:
 $\mu_1 = [y \leftarrow g(x)]$, the common instance: $g(g(x))$
 $\mu_2 = [x \leftarrow 5, y \leftarrow g(5)]$, the common instance: $g(g(5))$.
 μ_1 is more general than μ_2 .

Definition 2.13.

A **most general unifier (mgu)** is a unifier μ such that any other unifier θ can be obtained from μ by a further substitution λ , $\theta = \mu\lambda$.

Algorithm for computing the mgu of two literals [62]:

input: $l_1 = P_1(t_{1_1}, t_{1_2}, \dots, t_{1_n})$ and $l_2 = P_2(t_{2_1}, t_{2_2}, \dots, t_{2_k})$ two literals

output: $mgu(l_1, l_2)$ or the message “ l_1, l_2 are not unifiable”

begin

```

if ( $P_1 \neq P_2$ ) // the predicate symbols are different
    then write “ $l_1, l_2$  are not unifiable”; exit;
end_if
if ( $n \neq k$ )
    then write “ $l_1, l_2$  are not unifiable”; exit;
end_if
theta :=  $\varepsilon$ ; // initialization with empty substitution
while ( $\theta(l_1) \neq \theta(l_2)$ )
    find the terms corresponding to the outermost function symbols or variables
        that are different in  $\theta(l_1)$ ,  $\theta(l_2)$  and denote them by  $t_1$  and  $t_2$ .
    if (neither one of  $t_1$  and  $t_2$  is a variable or one is a subterm of the other one)
        then write “ $l_1$  and  $l_2$  are not unifiable”; exit;
    end_if
    if ( $t_1$  is a variable) //  $\lambda$  is the unifier of the terms  $t_1, t_2$  in the current iteration
        then  $\lambda := [t_1 \leftarrow t_2]$ ;
        else  $\lambda := [t_2 \leftarrow t_1]$ ;
    end_if
     $\theta := \theta\lambda$ ;
    if ( $\theta$  is not a substitution)
        then write “ $l_1$  and  $l_2$  are not unifiable”; exit;
    end_if
end_while
write “ $l_1$  and  $l_2$  are unifiable and  $\theta$  is  $mgu(l_1, l_2)$ ”
end
```

A Computational Approach to Classical Logics and Circuits

Remark:

The most general unifier of two literals may be not unique: if during an iteration, the terms that must be unified t_1 and t_2 are both variables we can replace t_1 by t_2 or t_2 by t_1 .

Example 2.14.

Find the most general unifier of the literals l_1 and l_2 .

$$l_1 = P(a, x, f(g(y))), \quad l_2 = P(y, f(z), f(z)),$$

where $x, y, z \in Var$, $a \in Const$, $f, g \in F_1$, $P \in P_3$.

The algorithm presented before is applied. In each iteration, the terms that are unified are underlined.

$$\theta := \varepsilon$$

$$\theta(l_1) = P(\underline{a}, x, f(g(y))) \quad \text{and} \quad \theta(l_2) = P(\underline{y}, f(z), f(z)),$$

first iteration:

$$\lambda := [y \leftarrow a];$$

$$\theta := \theta\lambda = [y \leftarrow a];$$

$$\theta(l_1) = P(a, \underline{x}, f(g(a))) \quad \text{and} \quad \theta(l_2) = P(a, f(\underline{z}), f(z))$$

second iteration:

$$\lambda := [x \leftarrow f(z)];$$

$$\theta := \theta\lambda = [y \leftarrow a][x \leftarrow f(z)] = [y \leftarrow a, x \leftarrow f(z)];$$

$$\theta(l_1) = P(a, f(z), f(\underline{g(a)})) \quad \text{and} \quad \theta(l_2) = P(a, f(z), f(\underline{z}))$$

third iteration:

$$\lambda := [z \leftarrow g(a)]$$

$$\theta := \theta\lambda =$$

$$= [y \leftarrow a, x \leftarrow f(z)][z \leftarrow g(a)] = [y \leftarrow a, x \leftarrow f(g(a)), z \leftarrow g(a)] = mgu(l_1, l_2)$$

l_1 and l_2 are unifiable and their common instance is:

$$\theta(l_1) = \theta(l_2) = p(a, f(g(a)), f(g(a))).$$

Example 2.15.

Check if the literals l_1 and l_2 are unifiable:

$$l_1 = Q(x, a, f(x, y)) \text{ and } l_2 = Q(b, y, f(z, c)),$$

where $x, y, z \in Var$, $a, b, c \in Const$, $f \in F_2$, $Q \in P_3$.

The algorithm for computing the mgu of two literals is applied:

$$\theta := \varepsilon;$$

$$\theta(l_1) = Q(\underline{x}, a, f(x, y)) \quad \text{and} \quad \theta(l_2) = Q(\underline{b}, y, f(z, c));$$

First-order Logic

first iteration:

$\lambda := [x \leftarrow b]$, unifier of the terms x (variable) and b (constant);

$\theta := \theta\lambda = [x \leftarrow b]$;

$\theta(l_1) = Q(b, \underline{a}, f(b, y))$ and $\theta(l_2) = Q(b, \underline{y}, f(z, c))$;

second iteration:

$\lambda = [y \leftarrow a]$, unifier of the terms y (variable) and a (constant)

$\theta := \theta\lambda$

$\theta := [x \leftarrow b][y \leftarrow a] = [x \leftarrow b, y \leftarrow a]$,

$\theta(l_1) = Q(b, a, f(\underline{b}, a))$ and $\theta(l_2) = Q(b, a, f(\underline{z}, c))$;

third iteration:

$\lambda = [z \leftarrow b]$, unifier of the terms z (variable) and b (constant)

$\theta := \theta\lambda$

$\theta := [x \leftarrow b, y \leftarrow a][z \leftarrow b] = [x \leftarrow b, y \leftarrow a, z \leftarrow b]$

$\theta(l_1) = Q(b, a, f(b, \underline{a}))$ and $\theta(l_2) = Q(b, a, f(b, \underline{c}))$;

fourth iteration:

The terms a and c , which are distinct constants, are not unifiable, so the terms $f(b, a)$ and $f(z, c)$ corresponding to the third argument of the literals are not unifiable.

The conclusion is that the *literals* l_1 and l_2 are not unifiable.

2.7. Herbrand – based procedure

Herbrand-based procedure [12] is a refutation proof method used to solve the decision problem in first-order logic.

Definition 2.14.

1. A **ground term** or a **ground atom** are free of variables.
2. A formula is called **ground formula** if it does not contain any variable or quantifier.
3. A^g is a **ground instance** of the quantifier-free formula A , if A^g is obtained by replacing all free variables from A by ground terms.

Example 2.16.

- $a, f(a), g(f(a), b)$ – ground terms, where:
 - a, b - constants, f, g - function symbols
- $P(a), Q(f(a), b)$ – ground atoms, where:
 - a, b - constants, f - function symbol, P, Q - predicate symbols
- $A^g = P(a) \wedge \neg Q(b, f(a))$ is a ground instance of the formula $A(x, y, z) = P(x) \wedge \neg Q(y, f(z))$, $[x \leftarrow a, y \leftarrow b, z \leftarrow a]$.

A Computational Approach to Classical Logics and Circuits

Definition 2.15. [44]

Let $S = \{A_1, A_2, \dots, A_n\}$ be a set of first-order quantifier-free formulas.

1. The **Herbrand universe** of S , denoted by H_S , is the set of all ground terms built using the constants and the function symbols from S as follows:

$$H_S = \bigcup_{i \geq 0} H_i, \text{ where:}$$

$H_0 = C =$ the set of all constants of S ;

if $C = \emptyset$ then $H_0 = \{a\}$, a is a new constant;

$$H_{i+1} = H_i \cup \{f(t_1, \dots, t_k) \mid t_1, \dots, t_k \in H_i, f \in F_k, f - \text{function symbol of } S\}.$$

2. The **Herbrand base** of S , denoted by BH_S , is the set of all ground atoms built using the predicate symbols of S and the ground terms of H_S .

$$BH_S = \{P(t_1, \dots, t_k) \mid t_1, \dots, t_k \in H_S, P \in P_k, P - \text{predicate symbol of } S\}.$$

3. The **Herbrand system** of S , denoted by SH_S , is the set of all ground instances of the formulas from S , obtained by replacing the free variables by ground terms of H_S .

Example 2.17.

1. $S_1 = \{P(a) \vee \neg Q(x), R(y)\}$

$H_{S_1} = \{a\}$ - the Herbrand universe is finite.

$BH_{S_1} = \{P(a), Q(a), R(a)\}$ - the Herbrand base

$SH_{S_1} = \{P(a) \vee \neg Q(a), R(a)\}$ - the Herbrand system

2. $S_2 = \{P(f(x)), \neg Q(z, y)\}$

$H_{S_2} = \{c, f(c), f(f(c)), \dots\}$; c - constant, the Herbrand universe is infinite

$BH_{S_2} = \{P(f(c)), Q(c, c), P(f(f(c))), Q(c, f(c)), Q(f(c), c), Q(f(c), f(c)), \dots\}$

- the Herbrand base is infinite

$SH_{S_2} = \{P(f(c)), \neg Q(c, c), P(f(f(c))), \neg Q(c, f(c)), \neg Q(f(c), c), \neg Q(f(c), f(c)), \dots\}$

- the Herbrand system is infinite

Theorem 2.8. Herbrand's theorem[12]

Let S be a set of first-order quantifier-free formulas. S admits a model if and only if the set SH_S (the Herbrand system of S) admits a model.

Theorem 2.9. [12]

A set S of first-order quantifier-free formulas is inconsistent if and only if the set SH_S (the Herbrand system of S) is inconsistent if and only if there is a finite inconsistent subset of SH_S .

First-order Logic

Remark:

The problem of checking the inconsistency of a set of first-order formulas was reduced to checking the inconsistency of a set (finite or infinite) of propositional formulas according to:

S inconsistent if and only if the Herbrand system SH_{S_1} is inconsistent,

where $S_1 = S^c$, S^c is the set of clausal normal forms of the formulas from S .

Herbrand's theorem suggests a refutation proof procedure based on the following theoretical result:

$U_1, U_2, \dots, U_n \vdash V$ if and only if

$S = \{U_1^c, U_2^c, \dots, U_n^c, (\neg V)^c\}$ is inconsistent if and only if

SH_S (Herbrand system of S) is inconsistent if and only if

there is a finite inconsistent subset of SH_S .

Herbrand-based algorithm:

input: U_1, U_2, \dots, U_n, V - first-order formulas

output: the message $U_1, U_2, \dots, U_n \vdash V$ or $U_1, U_2, \dots, U_n \not\vdash V$

begin

 build $S = \{U_1^c, U_2^c, \dots, U_n^c, (\neg V)^c\}$

 build SH_S ;

if SH_S is finite

then

if (SH_S is inconsistent)

then write " $U_1, U_2, \dots, U_n \vdash V$ "; exit;

else write " $U_1, U_2, \dots, U_n \not\vdash V$ "; exit;

end_if

else // $SH_S = \{Z_1, Z_2, \dots, Z_m, \dots\}$ is infinite

$k := 1$;

while ($Z_1 \wedge Z_2 \wedge \dots \wedge Z_k$ is consistent) (**)

$k := k + 1$;

end_while

write " $U_1, U_2, \dots, U_n \not\vdash V$ "; exit;

end_if

end

A Computational Approach to Classical Logics and Circuits

This is a semi-decision procedure which never stops if the set SH_S is an infinite consistent set (**). For implementation, the loop “while” must contain a restriction for the iterations number, to avoid the infinite loop. If we exit the loop because the maximum number of iterations was reached, we cannot decide if SH_S is inconsistent or not, and thus we cannot decide if $U_1, U_2, \dots, U_n \vdash V$ or $U_1, U_2, \dots, U_n \not\vdash V$.

Example 2.18.

Using the Herbrand-based algorithm check the validity of the predicate formula:

$$U = (\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x)).$$

Let us denote $A = \neg U$ and we transform A into clausal normal form.

$$\begin{aligned} A = \neg U &\equiv \neg((\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x))) \equiv \\ &\equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge \neg((\forall x)P(x) \rightarrow (\forall x)Q(x)) \equiv \\ &\equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge (\forall x)P(x) \wedge \neg(\forall x)Q(x) \equiv \\ &\equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge (\forall x)P(x) \wedge (\exists x)\neg Q(x) \equiv \\ &\equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge (\forall y)P(y) \wedge (\exists z)\neg Q(z) \equiv \end{aligned}$$

We consider two prenex normal forms:

$$A_1^P = (\exists z)(\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a))$$

$$A_2^P = (\forall x)(\exists z)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a))$$

The corresponding Skolem and clausal forms are as follows:

$$A_1^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a)), \text{ } a - \text{Skolem constant}$$

$$A_1^C = (\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a)$$

$$A_2^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(f(x))), \text{ } f - \text{unary Skolem function}$$

$$A_2^C = (\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(f(x))$$

We consider the following two sets of clauses:

$$S_1 = \{\neg P(x) \vee Q(x), P(y), \neg Q(a)\} \text{ and}$$

$$S_2 = \{\neg P(x) \vee Q(x), P(y), \neg Q(f(x))\}$$

“ U is valid if and only if SH_S is inconsistent”, where S is S_1 or S_2 .

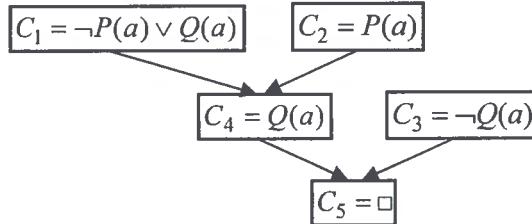
S_1 is used in the Herbrand-based algorithm:

$$H_{S_1} = \{a\}, SH_{S_1} = \{C_1 = \neg P(a) \vee Q(a), C_2 = P(a), C_3 = \neg Q(a)\}$$

The Herbrand universe and Herbrand system are finite.

First-order Logic

We derive the empty clause using propositional resolution (see Chapter 5) as follows:



SH_{S_1} is an inconsistent set, therefore A is inconsistent and U is a valid formula.

S_2 is used in the Herbrand-based algorithm:

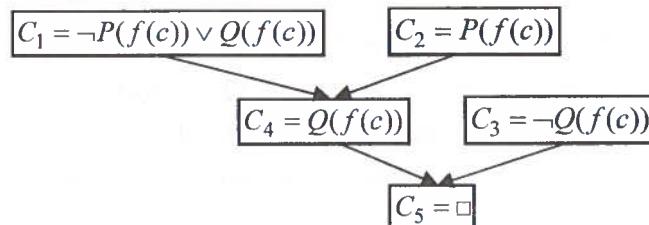
$$H_{S_2} = \{c, f(c), f(f(c)), \dots\}$$

$$SH_{S_2} = \{\neg P(c) \vee Q(c), P(c), \underline{\neg Q(f(c))}, \underline{\neg P(f(c)) \vee Q(f(c))}, \underline{P(f(c))}, \underline{\neg Q(f(f(c))))}, \dots\}$$

The Herbrand universe and Herbrand system are infinite.

We consider the finite set of clauses:

$W = \{P(f(c)), \neg Q(f(c)), \neg P(f(c)) \vee Q(f(c))\}$ which is a subset of SH_{S_2} . From W we can derive \square as follows:



Thus, W is an inconsistent set, and SH_{S_2} is inconsistent because it has an inconsistent subset. The conclusion is that A is an inconsistent formula and U is a tautology.

If we want to avoid working with an infinite set of clauses, caused by the existence of function symbols, we must choose the Skolem form containing only Skolem constants.

Example 2.19.

Check the consistency/inconsistency of the following set of first-order quantifier-free formulas:

$$S = \{P(x) \rightarrow Q(x) \vee R(x), Q(y) \rightarrow R(y), R(a), \neg P(a)\}.$$

We transform the formulas into clausal forms:

$$S = \{\neg P(x) \vee Q(x) \vee R(x), \neg Q(y) \vee R(y), R(a), \neg P(a)\}$$

$$H_S = \{a\} - \text{Herbrand universe}$$

A Computational Approach to Classical Logics and Circuits

$$SH_S = \{C_1 = \neg P(a) \vee Q(a) \vee R(a), C_2 = \neg Q(a) \vee R(a), C_3 = R(a), C_4 = \neg P(a)\}$$

- Herbrand system contains propositional clauses.

We try to simplify SH_S (see Theorem 5.6):

$\neg P(a)$ is a pure literal and we eliminate the clauses (C_1 and C_4) containing it.

$R(a)$ is also a pure literal and we eliminate C_2 and C_3 .

After applying these transformations on SH_S , we obtain $SH'_S = \emptyset$, and thus SH_S is a consistent set, therefore S is a consistent set of clauses.

2.8. Exercises

Exercise 2.1.

Transform the following sentences from natural language into predicate formulas. Explain the syntactic elements used in the predicate formulas: variables, constants, functions symbols, predicate symbols.

1. In a plane if a line x is perpendicular to a constant line d then all the lines parallel to x are perpendicular to d .
2. In a plane there are lines parallel to a constant line d and there are lines perpendicular to d .
3. If x is an integer divisible by 10, it can be decomposed in two factors such that one is divisible by 2 and the other one is divisible by 5, and x can be written as a sum of 2 even numbers.
4. Every positive number can be written as a product of two positive numbers and as a product of two negative numbers.
5. If x and y are positive prime numbers, $x \neq y$, $x \neq 2$ and $y \neq 2$, their sum and difference are even numbers and their product is an odd number.
6. For every positive integer x , if x is a square of an integer, then there exists an integer y such that $(y+1)*(y-1) = x-1$.
7. For every positive integer x , if x is not a prime, then there exists a prime y such that y divides x and y is smaller than x .
8. The sum of two even numbers is an even number and their product is divisible by 4.

Exercise 2.2.

Transform the following statements from natural language into predicate formulas choosing the appropriate constants, function symbols and predicate symbols:

1. Every student who makes good grades is brilliant or studies.
2. Some of John's colleagues like to draw and some like to dance.
3. CS students like either algebra or logic, all of them like Java but only Bill likes history.

First-order Logic

4. All Mary's relatives live in Cluj-Napoca, only her cousin John lives in Bucharest.
5. Anyone who owns a rabbit hates anything that chases any rabbit.
6. All birds have wings but only penguins do not fly.
7. Santa has some reindeer with a red nose, then every child loves Santa.
8. Every investor who bought something that falls is not happy.
9. Anyone who has any cats will not have any mice.
10. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves.
11. Caterpillars and snails like to eat some plants.
12. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants.

Exercise 2.3.

Using the definition of deduction in predicate logic prove:

1. $(\forall y)(\forall z)(P(y) \wedge Q(z)) \vdash (\forall x)(P(x) \wedge Q(x))$;
2. $(\forall y)(P(y) \vee Q(y)), \neg(\forall z)P(z) \vdash (\exists x)Q(x)$;
3. $(\exists y)P(y) \rightarrow (\forall z)Q(z) \vdash (\forall x)(P(x) \rightarrow Q(x))$;
4. $(\forall y)(P(y) \rightarrow Q(y)), (\forall z)P(z) \vdash (\forall x)Q(x)$;
5. $(\forall y)P(y) \vdash (\forall x)(Q(x) \rightarrow P(x))$;
6. $(\forall y)P(y) \vdash (\forall x)(Q(x) \vee P(x))$;
7. $(\forall y)P(y) \vdash (\exists x)P(x)$;
8. $(\forall y)(\forall z)(P(y) \rightarrow Q(z)) \vdash (\forall x)(P(x) \rightarrow Q(x))$.

Exercise 2.4.

Using the given interpretations evaluate the following formulas:

1. $U = (\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\forall x)(A(x) \vee B(x))$

Interpretation $I = \langle D, m \rangle$, where:

D = the set of all straight lines of a plan P

Let $d \in P$, a constant straight line belonging to the interpretation domain

$m(A) : D \rightarrow \{T, F\}, m(A)(x) : "x \perp d"; \quad m(B) : D \rightarrow \{T, F\}, m(B)(x) : "x \parallel d";$

2. $U = (\exists x)(P(x) \wedge Q(x)) \rightarrow (\exists x)P(x) \vee Q(12)$

Interpretation $I = \langle D, m \rangle$ where:

$D = \mathbb{N}$ (the set of natural numbers)

$m(P) : \mathbb{N} \rightarrow \{T, F\}, m(P)(x) : "x : 5"; \quad m(Q) : \mathbb{N} \rightarrow \{T, F\}, m(Q)(x) : "x : 7";$

3. $U(z) = (\exists x)(\forall y)(\exists z)P(f(x, y), z)$

Interpretation $I = \langle D, m \rangle$, where:

$D = \mathbb{Z}$ (the set of integer numbers),

A Computational Approach to Classical Logics and Circuits

$m(f) : \mathbf{Z}^2 \rightarrow \mathbf{Z}$, $m(f)(x, y) = (x + y)^2$ and

$m(P) : \mathbf{Z}^2 \rightarrow \{T, F\}$, $m(P)(x, y) : "x > y"$;

4. $U = (\forall x)(\exists y)P(x, y) \rightarrow (\exists y)(\forall x)P(x, y)$

Interpretation $I = \langle D, m \rangle$, where:

$D =$ the set of all triangles,

$m(P) : D^2 \rightarrow \{T, F\}$, $m(P)(x, y) : "Area(x) \leq Area(y)"$;

5. $U = (\forall x)(\exists y)(\exists z)P(x, f(g(y), g(z)))$

Interpretation $I = \langle D, m \rangle$, where:

$D = \mathbf{N}$ (the set of natural numbers)

$m(f) : \mathbf{N}^2 \rightarrow \mathbf{N}$, $m(f)(x, y) = x + y$ and

$m(g) : \mathbf{N} \rightarrow \mathbf{N}$, $m(g)(x) = x^2$ and

$m(P) : \mathbf{N}^2 \rightarrow \{T, F\}$, $m(P)(x, y) : "x = y"$;

6. $U = ((\forall x)A(x) \leftrightarrow (\forall x)B(x)) \rightarrow (\forall x)(A(x) \leftrightarrow B(x))$

Interpretation $I = \langle D, m \rangle$, where:

$D =$ the set of all persons from Cluj county

$m(A) : D \rightarrow \{T, F\}$, $m(A)(x) : "the person x has a driver license"$;

$m(B) : D \rightarrow \{T, F\}$, $m(B)(x) : "the person x owns a car"$.

7. $U(z) = (\exists x)(\forall y)(\forall z)P(x, f(y, z))$

Interpretation $I = \langle D, m \rangle$, where:

$D = \mathbf{Z}$ (the set of integer numbers),

$m(f) : \mathbf{Z}^2 \rightarrow \mathbf{Z}$, $m(f)(x, y) = (x + y)^2$ and

$m(P) : \mathbf{Z}^2 \rightarrow \{T, F\}$, $m(P)(x, y) : "x < y"$;

8. $U = ((\forall x)A(x) \rightarrow (\exists x)B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\forall x)B(x))$

Interpretation $I = \langle D, m \rangle$, where:

$D =$ the set of all persons from Romania

$m(A) : D \rightarrow \{T, F\}$, $m(A)(x) : "the person x lives in a city"$;

$m(B) : D \rightarrow \{T, F\}$, $m(B)(x) : "the person x has a job"$.

Exercise 2.5.

Prove that the following formulas are not valid by finding anti-models for them.

1. $U_1 = ((\exists x)P(x) \rightarrow (\exists x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x))$

2. $U_2 = (\exists x)(P(x) \rightarrow Q(x)) \rightarrow ((\exists x)P(x) \rightarrow (\exists x)Q(x))$;

First-order Logic

3. $U_3 = ((\forall x)P(x) \rightarrow (\forall x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x));$
4. $U_4 = (\exists x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)(P(x) \wedge Q(x));$
5. $U_5 = (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall x)P(x) \vee (\forall x)Q(x);$
6. $U_6 = ((\forall x)P(x) \rightarrow (\exists x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x));$
7. $U_7 = ((\exists x)P(x) \rightarrow (\exists x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x));$
8. $U_8 = (\exists x)P(x) \wedge (\exists x)Q(x) \rightarrow (\forall x)(P(x) \wedge Q(x)).$

Exercise 2.6.

Choose an arbitrary interpretation for the formula $U_j, j \in \{1, 2, \dots, 8\}$ and prove that it is a model of $U_j, j \in \{1, 2, \dots, 8\}$.

1. $U_1 = (\forall x)(A(x) \leftrightarrow B(x)) \rightarrow ((\forall x)A(x) \leftrightarrow (\forall x)B(x));$
2. $U_2 = (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\forall x)B(x));$
3. $U_3 = (\forall x)(A(x) \leftrightarrow B(x)) \rightarrow ((\exists x)A(x) \leftrightarrow (\exists x)B(x));$
4. $U_4 = (\exists x)(A(x) \rightarrow B(x)) \leftrightarrow ((\forall x)A(x) \rightarrow (\exists x)B(x));$
5. $U_5 = ((\exists x)A(x) \rightarrow (\forall x)B(x)) \rightarrow (\forall x)(A(x) \rightarrow B(x));$
6. $U_6 = (\forall x)(A(x) \vee B(x)) \rightarrow ((\forall x)A(x) \vee (\exists x)B(x));$
7. $U_7 = (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\exists x)B(x));$
8. $U_8 = (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\exists x)B(x)).$

All the formulas from Exercise 2.6 are tautologies.

Exercise 2.7.

Transform the following formulas into prenex, Skolem and clausal normal forms.

1. $U_1 = (\exists x)(\neg(\exists y)p(y) \rightarrow (\forall y)(q(y) \rightarrow r(x)));$
2. $U_2 = (\exists x)((\exists y)p(y) \rightarrow \neg(\forall y)(q(y) \rightarrow r(x)));$
3. $U_3 = (\forall x)(\neg(\exists y)p(y) \rightarrow (\forall y)(q(y) \rightarrow r(x)));$
4. $U_4 = (\forall x)((\exists y)p(y) \rightarrow \neg(\forall y)(q(y) \rightarrow r(x)));$
5. $U_5 = (\exists x)((\forall y)p(y) \rightarrow \neg(\exists y)(q(y) \rightarrow r(x)));$
6. $U_6 = (\forall x)(\neg(\forall y)p(y) \rightarrow (\exists y)(q(y) \rightarrow r(x)));$
7. $U_7 = (\forall x)((\forall y)p(y) \rightarrow \neg(\exists y)(q(y) \rightarrow r(x)));$
8. $U_8 = (\exists x)(\neg(\forall y)p(y) \rightarrow (\exists y)(q(y) \rightarrow r(x))).$

Exercise 2.8.

Transform the following formulas into prenex, Skolem and clausal normal forms.

1. $U_1 = (\forall x)(\forall y)((\exists z)p(z) \wedge (\exists u)(q(x, u) \rightarrow (\exists z)q(y, z)));$
2. $U_2 = (\exists x)(\forall y)((\exists z)p(z) \wedge (\exists u)(q(x, u) \rightarrow (\exists z)q(y, z)));$

A Computational Approach to Classical Logics and Circuits

3. $U_3 = (\forall x)(\exists y)((\exists z)p(z) \wedge (\exists u)(q(x,u) \rightarrow (\exists z)q(y,z))) ;$
4. $U_4 = (\exists x)(\exists y)((\exists z)p(z) \wedge (\forall u)(q(x,u) \rightarrow (\exists z)q(y,z))) ;$
5. $U_5 = (\forall x)(\exists y)((\exists z)p(z) \wedge (\forall u)(q(x,u) \rightarrow (\exists z)q(y,z))) ;$
6. $U_6 = (\forall x)(\forall y)((\exists z)p(z) \wedge (\forall u)(q(x,u) \rightarrow (\exists z)q(y,z))) ;$
7. $U_7 = (\forall x)(\forall y)((\exists z)p(z) \wedge (\exists u)(q(x,u) \rightarrow (\forall z)q(y,z))) ;$
8. $U_8 = (\exists x)(\forall y)((\exists z)p(z) \wedge (\forall u)(q(x,u) \rightarrow (\exists z)q(y,z))) .$

Exercise 2.9.

Are the literals from the following pairs unifiable? If yes, find their most general unifier. $x, y, z \in Var$, $a, b \in Const$, $f, g \in F_1$, $h \in F_2$, $P \in P_3$.

1. $P(a, x, g(g(y)))$ and $P(y, f(z), f(z))$;
 $P(x, g(f(a)), f(x))$ and $P(f(y), z, y)$;
 $P(a, x, g(g(y)))$ and $P(z, h(z, u), g(u))$;
2. $P(a, x, f(g(y)))$ and $P(y, f(z), f(z))$;
 $P(x, g(f(a)), f(b))$ and $P(f(y), z, z)$;
 $P(a, x, f(g(y)))$ and $P(z, h(z, u), f(b))$;
3. $P(a, f(x), g(h(y)))$ and $P(y, f(z), g(z))$;
 $P(x, g(f(a)), h(x, y))$ and $P(f(z), g(z), y)$;
 $P(g(y), x, f(g(y)))$ and $P(z, h(z, u), f(u))$;
4. $P(a, g(x), f(g(y)))$ and $P(y, z, f(z))$;
 $P(b, g(f(a)), z)$ and $P(f(y), z, g(y))$;
 $P(a, h(x, b), f(g(y)))$ and $P(z, h(z, u), f(u))$;
5. $P(a, x, g(f(y)))$ and $P(f(z), z, g(x))$;
 $P(a, x, g(f(y)))$ and $P(x, y, g(f(b)))$;
 $P(a, h(x, u), g(f(z)))$ and $P(y, h(y, f(z)), g(x))$;
6. $P(a, y, g(f(z)))$ and $P(z, f(z), x)$;
 $P(y, f(x), z)$ and $P(y, f(y), f(y))$;
 $P(h(x, y), x, y)$ and $P(h(y, x), f(z), z)$;
7. $P(a, x, g(f(y)))$ and $P(f(y), z, x)$;
 $P(x, a, g(b))$ and $P(f(y), f(y), g(x))$;
 $P(h(x, a), f(z), z)$ and $P(h(f(y), x), f(x), a)$;

8. $P(a, x, g(f(y)))$ and $P(f(y), f(z), g(z))$;
 $P(x, g(f(a)), x)$ and $P(f(y), z, h(y, f(y)))$;
 $P(a, h(x, u), f(g(y)))$ and $P(z, h(z, u), g(u))$.

Exercise 2.10.

Using the Herbrand-based algorithm check whether the following sets of formulas are inconsistent:

1. $S_1 = \{P(x) \rightarrow Q(x) \vee R(x), Q(y) \rightarrow R(y), P(a) \wedge \neg R(a)\}$;
2. $S_2 = \{\neg P(a) \rightarrow Q(x) \vee R(x), Q(y) \rightarrow R(y), \neg P(x) \wedge \neg R(a)\}$;
3. $S_3 = \{P(x) \rightarrow Q(x) \wedge R(x), Q(y) \rightarrow R(y), P(a) \wedge \neg R(a)\}$;
4. $S_4 = \{P(x) \wedge Q(x) \rightarrow R(x), \neg Q(y) \rightarrow R(y), P(a) \wedge \neg R(a)\}$;
5. $S_5 = \{P(a) \wedge Q(x) \rightarrow R(x), \neg Q(y) \rightarrow R(y), P(x) \wedge \neg R(a)\}$;
6. $S_6 = \{P(x) \vee Q(x) \rightarrow R(x), \neg Q(y) \rightarrow R(y), P(a) \wedge \neg R(a)\}$;
7. $S_7 = \{P(a) \vee Q(x) \rightarrow R(x), \neg Q(y) \rightarrow R(y), P(x) \wedge \neg R(a)\}$;
8. $S_8 = \{P(x) \wedge \neg Q(x) \rightarrow R(x), Q(y) \rightarrow R(y), P(a) \wedge \neg R(a)\}$.

3. SEMANTIC TABLEAUX METHOD

The semantic tableaux method is a refutation proof method introduced by Raymond Smullyan [58] for classical logics. It is an efficient method, based on semantic considerations, easily adapted to nonstandard logics (modal, temporal, many-valued, nonmonotonic) [34, 35, 49, 56]. Its basic aim is to decide the consistency (satisfiability) and to find all the models of a formula.

3.1. A Classical Approach to the Semantic Tableaux Method

Based on semantic considerations, a predicate formula belongs to one of the following four classes:

- α - class contains conjunctive formulas:

$$A \wedge B, \quad \neg(A \vee B) \equiv \neg A \wedge \neg B, \quad \neg(A \rightarrow B) \equiv A \wedge \neg B$$

A conjunctive formula is consistent (satisfiable) only if both of its component subformulas are satisfied.

- β - class contains disjunctive formulas:

$$A \vee B, \quad \neg(A \wedge B) \equiv \neg A \vee \neg B, \quad A \rightarrow B \equiv \neg A \vee B$$

A disjunctive formula is satisfied if at least one of its component subformulas is satisfiable.

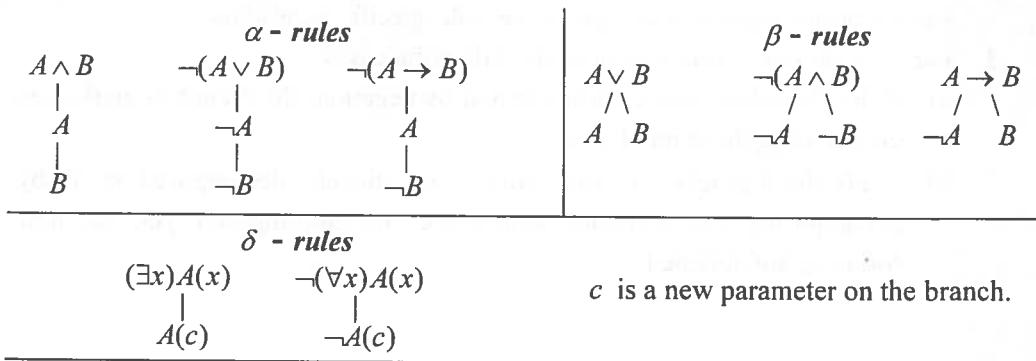
- γ - class contains universal formulas:

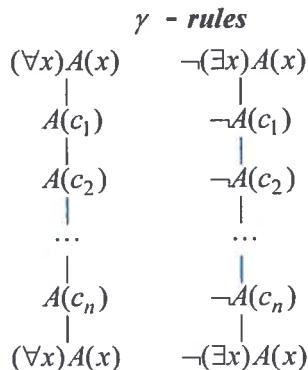
$$(\forall x)A(x), \quad \neg(\exists x)A(x) \equiv (\forall x)\neg A(x)$$

- δ - class contains existential formulas:

$$(\exists x)A(x), \quad \neg(\forall x)A(x) \equiv (\exists x)\neg A(x)$$

According to these classes there are four *decomposition rules*, symbolized graphically as follows:





c_1, c_2, \dots, c_n are all the parameters (constants) on that branch and they are used for the instantiation of A . If there is no constant on that branch, a new constant is introduced and used for instantiation.

Remarks:

- α -rule decomposes a conjunctive formula by adding on the same branch its component subformulas.
This rule can be generalized for conjunctive formulas having more than two components: all its component subformulas are added on the same branch.
- β -rule decomposes a disjunctive formula by adding two new successors on two new branches.
- δ -rule introduces a new constant used for instantiation and the corresponding instance is added on the same branch.
- γ -rule adds on the same branch all the instances of the universal formula using for instantiation all the constants on that branch and also adds a copy of the decomposed universal formula. In case of new constants introduced in the decomposition process, this copy will be used for new instantiations.

Definition 3.1. The construction of a semantic tableau

To a propositional/predicate formula U we can associate a **semantic tableau**, which is a binary tree having formulas in its nodes and it is built as follows:

1. the root of the tree is labeled with the initial formula;
2. every branch of the tree which contains a formula will be extended with a subtree according to the decomposition rule specific to its class;
3. the extension of a branch stops in the following cases:
 - a) if that branch contains a formula and its negation; the branch is marked as closed using the symbol \otimes ;
 - b) if all the formulas on that branch are already decomposed or if by decomposing the formulas which are not decomposed yet, no new formulas are obtained.

A Computational Approach to Classical Logics and Circuits

Definition 3.2.

1. A **branch** of the semantic tableau is called **closed** (marked by \otimes) if it contains a formula and its negation. Otherwise the **branch** is called **open** (marked by the symbol \odot).
2. A **branch** is called **complete** if it is closed or all the formulas on that branch are already decomposed.
3. A **semantic tableau** is called **closed** if all its branches are closed.
4. If a semantic tableau has at least one open branch, the **tableau** is called **open**.
5. A **semantic tableau** is called **complete** if all its branches are complete.

Remarks:

- A closed branch symbolizes inconsistency among the formulas on that branch.
- The set of formulas belonging to a complete and open branch is consistent, meaning that it has a model.

The process of building a semantic tableau is not deterministic because at some steps there is a choice of the branch to work with and a choice of which formula to be decomposed. Thus, to a formula we can associate more than one semantic tableau and all of them are logically equivalent.

Recommendations to obtain the simplest semantic tableau (binary tree), with fewest branches:

- apply α - rules and δ - rules which keep one branch before β - rules which make a branching;
- apply δ - rules (introduction of new constants) before γ - rules which use all the constants on that branch.

All the formulas belonging to a branch are connected by conjunction and all the branches are connected by disjunction. A semantic tableau is the disjunction of all its branches and it is a graphical representation of a disjunctive normal form (DNF) of the formula from the root node.

A consistent (satisfiable) formula has associated a complete and open semantic tableau. Each open branch provides a partial model of the formula, which covers one or more models (complete models).

If from the semantic tableau of the formula $U(p_1, p_2, \dots, p_n)$, 2^n distinct models are obtained, then the formula U is a tautology.

An inconsistent formula has associated a closed semantic tableau. All the branches of the semantic tableau are closed, meaning that there is no model of that formula.

The anti-models of a formula U , are the models of $\neg U$ and are provided by the open branches of the semantic tableau associated to $\neg U$.

Semantic Tableaux Method

From the previous assertions we conclude that it is easier to check if a formula U is a tautology, building the semantic tableau for $\neg U$ and checking if it is closed or not, than finding all the models of U from its semantic tableau.

Finding the models of a formula:

- For a *propositional formula* U , all its models are provided by all the open branches of the semantic tableau associated to U . Assigning the truth value T to all the literals belonging to an open branch, a partial/complete model of U is obtained. If a propositional variable is not on that branch, we can assign to it either T or F , obtaining more models.
- An open branch of the semantic tableau associated to the *predicate formula* U provides a partial/complete model of U with the minimum number of elements in the domain of interpretation. Such a model $I = \langle D, m \rangle$ is a ‘generic’ one and it is built as follows:
 - the domain of interpretation D contains all the constants on that branch;
 - the value T is assigned to all the literals on that branch:
 - if $c_1, \dots, c_n \in D$, $p \in P_n$ (P is an n -ary predicate) and
$$\begin{cases} \text{a) } P(c_1, \dots, c_n) \text{ belongs to the branch then } m(P)(c_1, \dots, c_n) = T \\ \text{b) } \neg P(c_1, \dots, c_n) \text{ belongs to the branch then } m(P)(c_1, \dots, c_n) = F \end{cases}$$
 - based on this generic model, more concrete models can be obtained.

The semantic tableaux method is used as a refutation proof method to solve the decision problems in propositional/predicate logic, according to the following theorems.

Theorem 3.1. Soundness and completeness of the semantic tableaux method
A propositional/predicate formula V is a theorem (tautology) if and only if there is a closed semantic tableau associated to $\neg V$.

Theorem 3.2.

Let U_1, U_2, \dots, U_n, V be propositional/predicate formulas.

$U_1, U_2, \dots, U_n \vdash V$ (equivalent to $U_1, U_2, \dots, U_n \models V$) if and only if there is a closed semantic tableau associated to the formula $U_1 \wedge U_2 \wedge \dots \wedge U_n \wedge \neg V$.

In propositional logic, which is decidable, the semantic tableau associated to a formula is always finite, therefore we can always decide if a formula is a tautology or not.

In predicate logic the semantic tableau can be infinite, caused by the combination of γ -rules and δ -rules.

A Computational Approach to Classical Logics and Circuits

The predicate logic is not decidable, it is only semi-decidable:

- if the semantic tableau associated to the formula $\neg V$ is finite, we can decide if the formula V is a tautology (closed tableau for $\neg V$) or not (open tableau for $\neg V$).
- if the semantic tableau of the formula $\neg V$ is infinite we cannot decide upon the validity of V .

Example 3.1.

Build two different semantic tableaux for the propositional formula:

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

Tableau 1

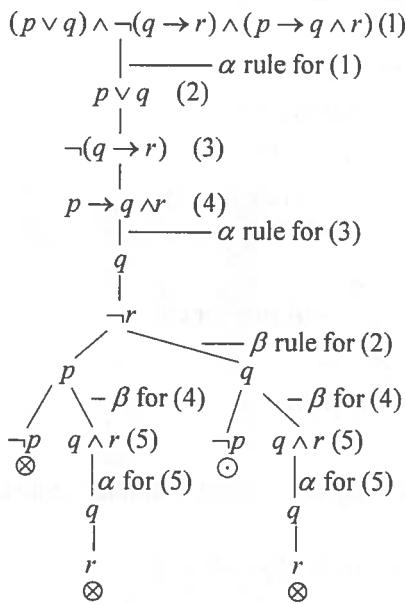
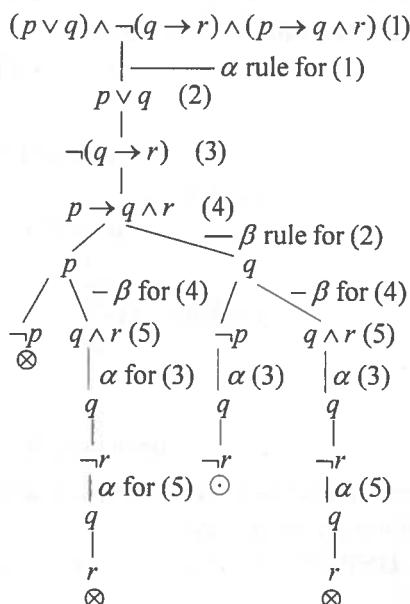


Tableau 2



Note that Tableau 1 is simpler (with fewer nodes on the branches) than Tableau 2, due to the fact that all the conjunctive subformulas were decomposed before the disjunctive subformulas (α -rules applied before β -rules).

These two semantic tableaux are equivalent and they represent graphically disjunctive normal forms logically equivalent to U .

From Tableau 1 we obtain:

$$\begin{aligned} DNF(U) = & (q \wedge \neg r \wedge \underline{p} \wedge \underline{\neg p}) \vee (q \wedge \neg r \wedge p \wedge q \wedge \underline{r}) \vee \\ & \vee (q \wedge \neg r \wedge q \wedge \neg p) \vee (q \wedge \neg r \wedge q \wedge q \wedge \underline{r}) \end{aligned}$$

From Tableau 2 we obtain:

$$\begin{aligned} DNF(U) = & (\underline{p} \wedge \underline{\neg p}) \vee (p \wedge q \wedge \neg r \wedge q \wedge \underline{r}) \vee \\ & \vee (q \wedge \neg p \wedge q \wedge \neg r) \vee (q \wedge q \wedge \neg r \wedge q \wedge \underline{r}) \end{aligned}$$

Semantic Tableaux Method

A simplified disjunctive normal form of U , obtained by eliminating the inconsistent cubes (they contain a pair of opposite literals - underlined) corresponding to the closed branches, is:

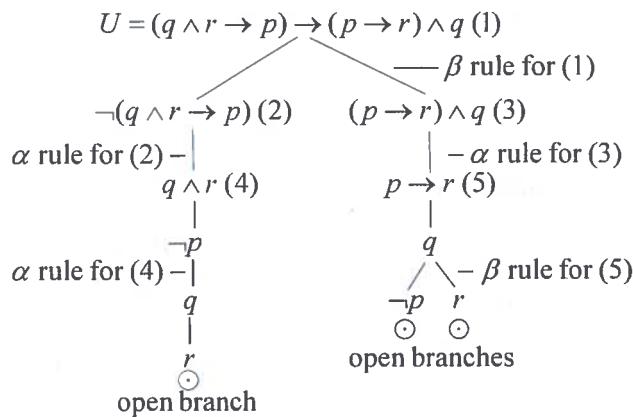
$$\text{DNF}(U) = \neg p \wedge q \wedge \neg r.$$

This cube corresponds to the only open branch of both semantic tableaux and provides the unique model i of the contingent formula U :

$$i : \{p, q, r\} \rightarrow \{T, F\}, i(p) = F, i(q) = T, i(r) = F \text{ and } i(U) = T$$

Example 3.2.

Build a semantic tableau for $U = (q \wedge r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q$ and decide what kind of formula (consistent, inconsistent, contingent, tautology) is U . In case of consistency find all its models.



The semantic tableau is complete and open, having three open branches and thus the formula is consistent.

$$\text{DNF}(U) = (\neg p \wedge q \wedge r) \vee (q \wedge \neg p) \vee (q \wedge r) \equiv (q \wedge \neg p) \vee (q \wedge r)$$

The simplified form of DNF was obtained by applying the absorption law. The left-most branch, corresponding to the first cube, is covered by the other two branches.

The branch represented by the cube $q \wedge \neg p$ provides a partial model (the propositional variable r is missing) that covers two complete models of U :

$$i_1 : \{p, q, r\} \rightarrow \{T, F\}, i_1(p) = F, i_1(q) = T, i_1(r) = T$$

$$i_2 : \{p, q, r\} \rightarrow \{T, F\}, i_2(p) = F, i_2(q) = T, i_2(r) = F$$

The branch represented by the cube $q \wedge r$ provides the following models of U :

$$i_3 : \{p, q, r\} \rightarrow \{T, F\}, i_3(p) = T, i_3(q) = T, i_3(r) = T$$

$$i_4 : \{p, q, r\} \rightarrow \{T, F\}, i_4(p) = F, i_4(q) = T, i_4(r) = T$$

Because $i_1 = i_4$, the formula U has three models: i_1 , i_2 and i_3 .

$$i_1(U) = i_2(U) = i_3(U) = T.$$

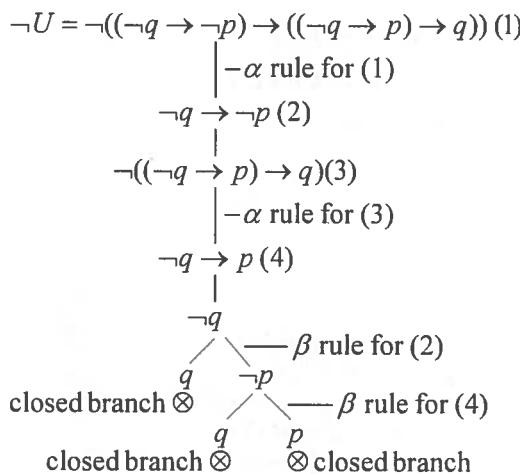
A Computational Approach to Classical Logics and Circuits

Having at least one model, U is a consistent formula, but it is not a tautology (does not have 8 models), therefore U is a contingent formula.

Example 3.3.

Using the semantic tableaux method, prove that the formula $U = (\neg q \rightarrow \neg p) \rightarrow ((\neg q \rightarrow p) \rightarrow q)$ is a tautology.

Theorem 3.1 is used, applying the semantic tableaux method as a refutation proof method. We build a semantic tableau for $\neg U$.



All three branches of the semantic tableau are closed, containing pairs of opposite literals: $(\neg q, q)$, $(\neg q, q)$, $(\neg p, p)$, therefore the formula $\neg U$ has no models because it is an inconsistent formula.

According to the theorem of soundness and completeness of this method we conclude that U is a tautology.

The semantic tableau corresponds to

$$\text{DNF}(\neg U) = (q \wedge \neg q) \vee (q \wedge \neg p \wedge \neg q) \vee (p \wedge \neg p \wedge \neg q).$$

Each cube represents a branch of the binary tree.

All the cubes are inconsistent, so $\neg U$ is an inconsistent formula and thus by contradiction U is a tautology.

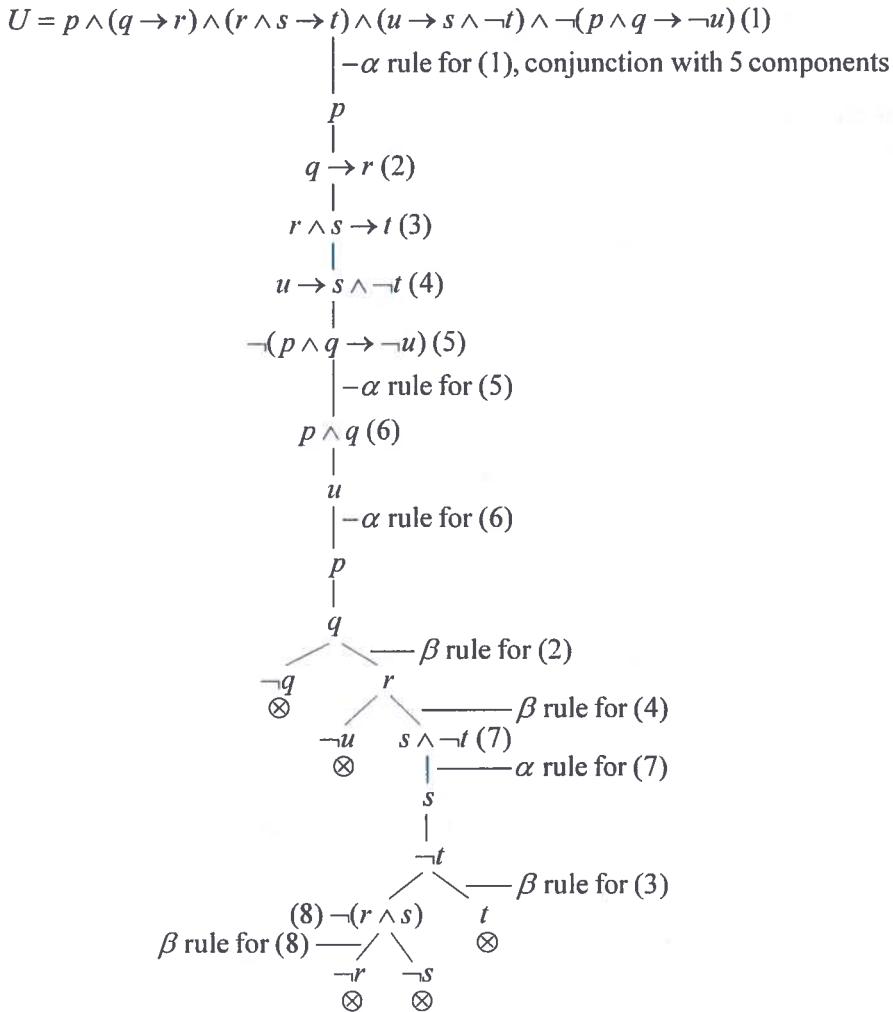
Example 3.4.

Using the semantic tableaux method check whether the following logical consequence holds or not.

$$p \wedge (q \rightarrow r), r \wedge s \rightarrow t, u \rightarrow s \wedge \neg t \models p \wedge q \rightarrow \neg u.$$

We apply Theorem 3.2 and we build the semantic tableau associated to the formula U obtained as the conjunction of all the premises and the negation of the conclusion.

Semantic Tableaux Method



The semantic tableau associated to U is closed (with five closed branches), therefore U is an inconsistent formula and according to Theorem 3.2 we have proved by contradiction that *the following logical consequence holds:*

$$p \wedge (q \rightarrow r), r \wedge s \rightarrow t, u \rightarrow s \wedge \neg t \models p \wedge q \rightarrow \neg u.$$

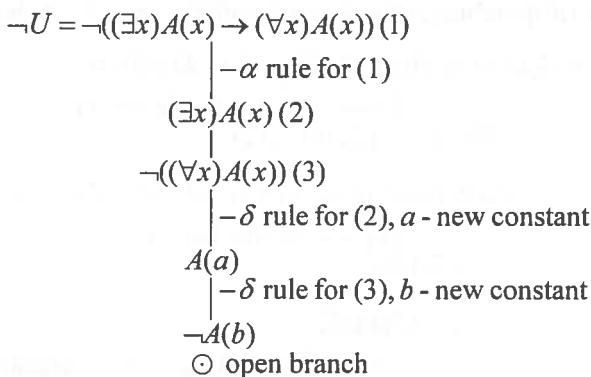
Example 3.5.

Check if the formula $U = (\exists x)A(x) \rightarrow (\forall x)A(x)$ is a tautology in first-order logic.

We prove by contradiction, so we negate what we have to prove.

The semantic tableau associated to the formula $\neg U$ is represented by the following binary tree.

A Computational Approach to Classical Logics and Circuits



The semantic tableau is complete and open, therefore $\neg U$ is consistent having at least one model which is an anti-model of U , so the formula U is not a tautology.

A generic model of $\neg U$ that is an anti-model of U , is the interpretation $I = \langle D, m \rangle$ where $D = \{a, b\}$ is the domain of interpretation and contains all the constants belonging to the open branch.

To the predicate symbol A we assign the unary predicate $m(A)$ according to the literals belonging to the open branch as follows:

$m(A):\{a,b\} \rightarrow \{T,F\}$, $m(A)(a) = T$ and $m(A)(b) = F$.

$$\nu^I(\neg U) = T \text{ and } \nu^I(U) = F$$

Example 3.6.

Check the validity of the predicate formula:

$$U = (\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\exists x)(A(x) \wedge B(x))$$

The semantic tableau associated to $\neg U$ is complete (there are no formulas such that by decomposing them new formulas which do not exist on that branch are obtained) and open. There are two closed branches and one open branch.

The open branch indicates that the formula $\neg U$ is consistent, thus U is not valid. The model of $\neg U$ provided by the open branch is an interpretation which evaluates the formula $\neg U$ as true. A model of $\neg U$ is an anti-model of U and it is obtained from the open branch as follows:

$$I = \langle D, m \rangle,$$

$D = \{a, b\}$ - domain containing all the constants from the open branch

$m(A) : D \rightarrow \{T, F\}$ and $m(B) : D \rightarrow \{T, F\}$

$m(A)(a) = T$ because $A(a)$ belongs to the branch

$m(A)(b) = F$ because $\neg A(b)$ belongs to the branch

$m(B)(g) \equiv F$ because $\neg B(g)$ belongs to the branch

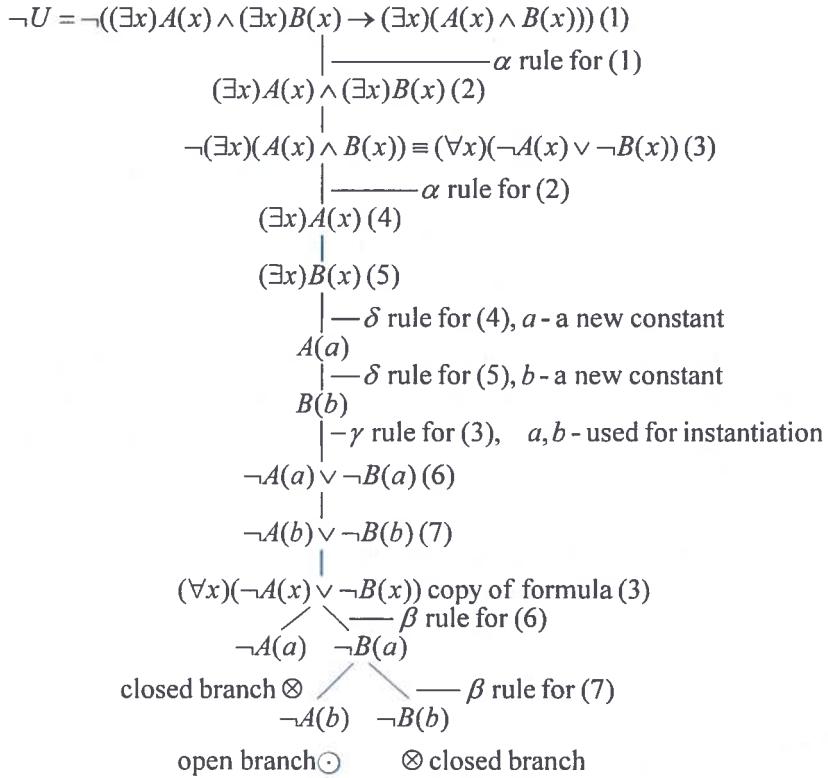
$m(B)(b) \equiv T$ because $B(b)$ belongs to the branch

↳ [View Details](#)

$\vee (\neg U) = T$ and $\vee (U) = F$

Semantic Tableaux Method

The binary tree corresponding to the semantic tableau of $\neg U$ is depicted below:



The copy of the formula (3), by decomposition, does not generate new formulas, because no new constants were introduced from the previous instantiation of (3). An open branch provides a model with the minimum number of elements (the constants on that branch) in the domain of interpretation.

I is a ‘generic’ anti-model of U and based on it, a concrete anti-model I_1 can be obtained: $I_1 = < D_1, m_1 >$, $D_1 = \{4,5\}$ - domain

$$m_1(A) : \{4,5\} \rightarrow \{T, F\}, m_1(A)(x) = 'x \text{ is an even number}',$$

$$m_1(A)(4) = T, m_1(A)(5) = F$$

$$m_1(B) : \{4,5\} \rightarrow \{T, F\}, m_1(B)(x) = 'x \text{ is an odd number}',$$

$$m_1(B)(4) = F, m_1(B)(5) = T$$

$$\nu^{I_1}(\neg U) = T \text{ and } \nu^{I_1}(U) = F$$

The evaluation of the formula U under the interpretation I_1 is:

$$\begin{aligned}
 \nu^{I_1}(U) &= (\exists x)_{x \in \{4,5\}} 'x \text{ is even}' \wedge (\exists x)_{x \in \{4,5\}} 'x \text{ is odd}' \rightarrow \\
 &\rightarrow ((\exists x)_{x \in \{4,5\}} 'x \text{ is even}' \wedge (\exists x)_{x \in \{4,5\}} 'x \text{ is odd}') = T \wedge T \rightarrow F = F
 \end{aligned}$$

A Computational Approach to Classical Logics and Circuits

Example 3.7.

Check the distributivity property of the universal quantifier over implication:

$$(\forall x)(P(x) \rightarrow Q(x)) \equiv (\forall x)P(x) \rightarrow (\forall x)Q(x)$$

We have that:

$(\forall x)(P(x) \rightarrow Q(x)) \equiv (\forall x)P(x) \rightarrow (\forall x)Q(x)$ if and only if

$\models (\forall x)(P(x) \rightarrow Q(x)) \leftrightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x))$ if and only if

$\models U_1$ and $\models U_2$, where:

$$U_1 = (\forall x)(P(x) \rightarrow O(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)O(x)) \text{ and}$$

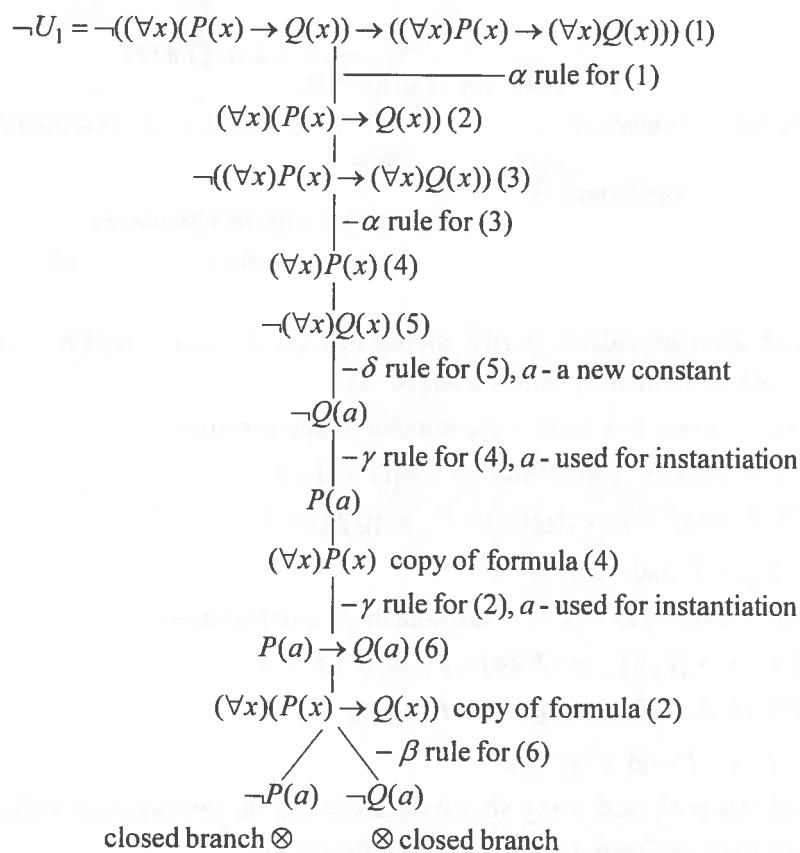
$$U_2 = ((\forall x)P(x) \rightarrow (\forall x)O(x)) \rightarrow (\forall x)(P(x) \rightarrow O(x)).$$

We shall prove that U_1 is a tautology and U_2 is not valid building the semantic tableaux of $\neg U_1$ and $\neg U_2$.

The following theoretic result is used:

$\models U$ if and only if $\neg U$ has a closed semantic tableau.

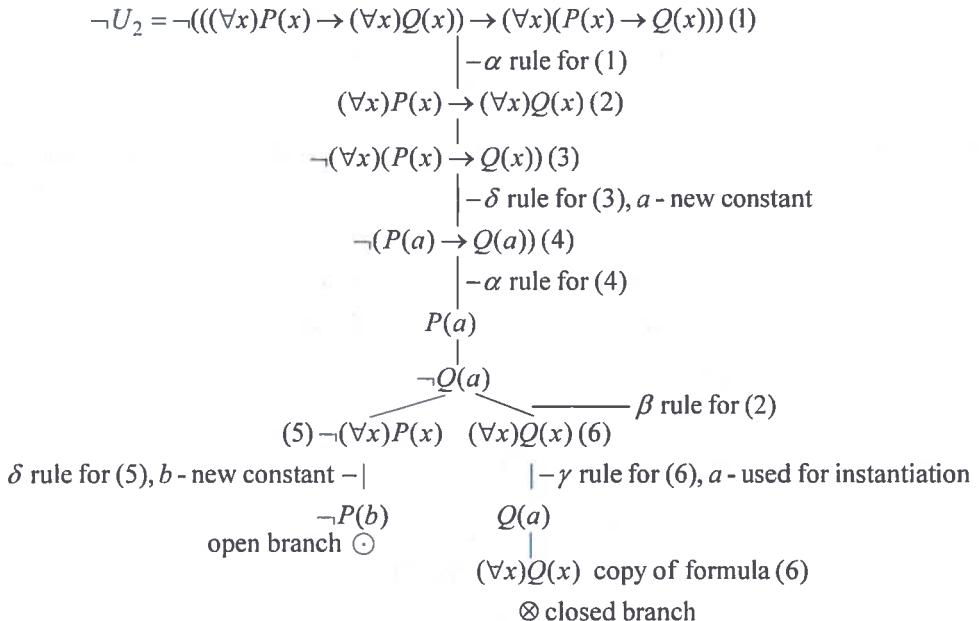
The semantic tableau associated to $\neg U_1$ is represented graphically as follows:



Semantic Tableaux Method

The semantic tableau of $\neg U_1$ is closed, with two closed branches containing pairs of opposite literals: $(P(a), \neg P(a))$ and $(Q(a), \neg Q(a))$, therefore $\neg U_1$ is inconsistent and U_1 is valid.

The semantic tableau of $\neg U_2$, depicted below, is finite, complete and open, with one open branch and one closed branch, so $\neg U_2$ is consistent and U_2 is not a tautolog.



The open branch provides a partial model that covers two complete models of $\neg U_2$: I_1 and I_2 , which are anti-models of U_2 .

$I_1 = < D, m_1 >$, where $D = \{a, b\}$ – the domain of interpretation

$$\begin{aligned}
m_1(P) : D \rightarrow \{T, F\}, \quad m_1(P)(a) = T, \quad m_1(P)(b) = F, \\
m_1(Q) : D \rightarrow \{T, F\}, \quad m_1(Q)(a) = F, \quad m_1(Q)(b) = T \\
v^{I_1}(\neg U_2) = T \text{ and } v^{I_1}(U_2) = F.
\end{aligned}$$

$I_2 = < D, m_2 >$, where $D = \{a, b\}$ – the domain of interpretation

$$\begin{aligned}
m_2(P) : D \rightarrow \{T, F\}, \quad m_2(P)(a) = T, \quad m_2(P)(b) = F, \\
m_2(Q) : D \rightarrow \{T, F\}, \quad m_2(Q)(a) = F, \quad m_2(Q)(b) = F \\
v^{I_2}(\neg U_2) = T \text{ and } v^{I_2}(U_2) = F.
\end{aligned}$$

We proved that $\models U_1$ and $\not\models U_2$ so, we conclude that *the universal quantifier is not distributive over implication, it is only semi-distributive*.

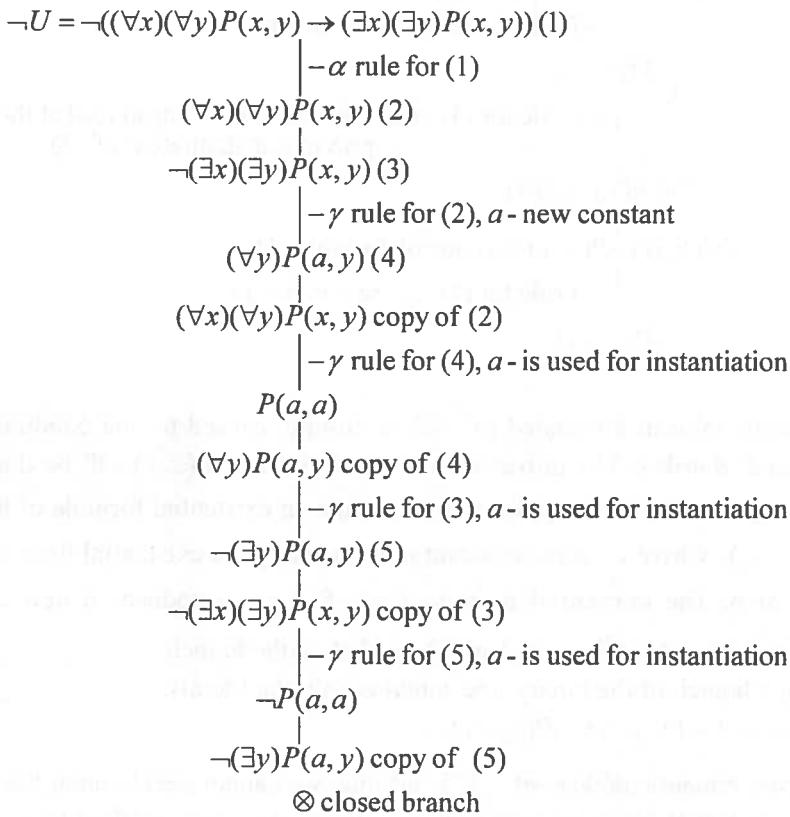
A Computational Approach to Classical Logics and Circuits

Example 3.8.

Check the validity of the closed predicate formula:

$$U = (\forall x)(\forall y)P(x, y) \rightarrow (\exists x)(\exists y)P(x, y).$$

We negate the formula U and the semantic tableau associated to $\neg U$ is depicted below.



After applying α -rule, the formulas (2) and (3) are added on the branch. Both these formulas are universal formulas, we do not have yet constants to be used in instantiations, so γ -rule applied to (2) introduces a new constant a . All further applications of γ -rule in the decomposition process use this constant.

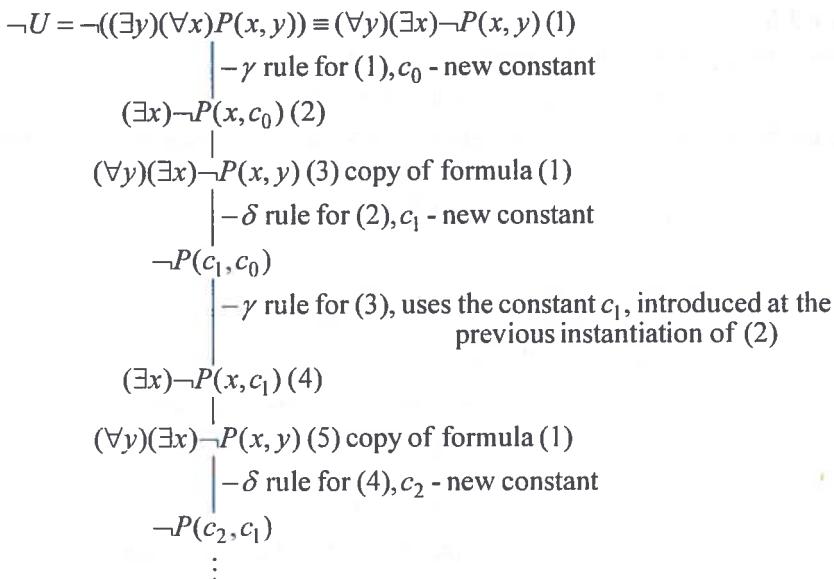
The semantic tableau of $\neg U$ is closed ($P(a,a)$ and $\neg P(a,a)$ are opposite literals), therefore U is a valid formula according to Theorem 3.1.

Example 3.9.

Let us consider the closed formula $U = (\exists y)(\forall x)P(x, y)$. We want to check the validity of U .

Using the ‘reductio ad absurdum’ principle we negate U in order to obtain a contradiction.

Semantic Tableaux Method



The semantic tableau associated to $\neg U$ is infinite, caused by the combination of γ -rules and δ -rules. The universal formula $(\forall y)(\exists x)\neg P(x, y)$ will be duplicated and it will generate at each application of γ -rule an existential formula of the form $(\exists x)\neg P(x, c_i)$, where c_i is the constant introduced by the existential formula at the previous step. The existential formula $(\exists x)\neg P(x, c_i)$ introduces a new constant c_{i+1} and the formula $\neg P(c_{i+1}, c_i)$ will be added on the branch.

The unique branch of the binary tree contains only the literals:

$$\neg P(c_1, c_0), \neg P(c_2, c_1), \neg P(c_3, c_2), \dots$$

Because the semantic tableau of $\neg U$ is infinite, we cannot decide upon the validity of U . Note that there is no possibility to obtain two opposite literals in order to close the tableau.

We prove that U is not valid (tautology) by finding an anti-model of it.

A well known interpretation which falsifies U is $I = \langle D, m \rangle$, where the domain $D = \mathbb{N}$ is the set of all natural numbers and

$$m(P) : \mathbb{N} \times \mathbb{N} \rightarrow \{T, F\}, m(P)(x, y) : "x < y".$$

This interpretation assigns to the binary predicate P the binary relation " $<$ ". I is an anti-model of U .

Under the interpretation I , the formula $(\exists y)(\forall x)P(x,y)$ is translated as: "The biggest natural number exists", which is obvious false, and thus U is not a tautology.

A Computational Approach to Classical Logics and Circuits

3.2. A New Approach to the Semantic Tableaux Method

This section presents a new approach to the semantic tableaux method used at the implementation level [56].

Definition 3.3.

A **branch** is a set of literals connected by conjunction and the reunion of the branches is the **semantic tableau** (a set of sets of literals).

Definition 3.4.

The **function TP**, which associates a semantic tableau to a formula, is defined as a mapping between sets of formulas and sets of sets of formulas. It is based on the decomposition rules: α , β , δ , γ as follows:

M is a set of formulas.

- $TP(M) = \{M\}$, if M is a set of literals.
- $TP(M) = TP(M' \cup \{p\})$, if $\neg\neg p \in M$ and $M' = M \setminus \{\neg\neg p\}$
- decomposition of conjunctive formulas (α -rule)
$$TP(M) = TP(M' \cup \{p\} \cup \{q\}), \text{ if } p \wedge q \in M \text{ and } M' = M \setminus \{p \wedge q\}$$
$$TP(M) = TP(M' \cup \{\neg p\} \cup \{\neg q\}), \text{ if } \neg(p \vee q) \in M \text{ and } M' = M \setminus \{\neg(p \vee q)\}$$
$$TP(M) = TP(M' \cup \{p\} \cup \{\neg q\}), \text{ if } \neg(p \rightarrow q) \in M \text{ and } M' = M \setminus \{\neg(p \rightarrow q)\}$$
- decomposition of disjunctive formulas (β -rule)
$$TP(M) = TP(M' \cup \{p\}) \cup TP(M' \cup \{q\}), \text{ if } p \vee q \in M \text{ and } M' = M \setminus \{p \vee q\}$$
$$TP(M) = TP(M' \cup \{\neg p\}) \cup TP(M' \cup \{\neg q\}), \text{ if } \neg(p \wedge q) \in M \text{ and } M' = M \setminus \{\neg(p \wedge q)\}$$
$$TP(M) = TP(M' \cup \{\neg p\}) \cup TP(M' \cup \{q\}), \text{ if } p \rightarrow q \in M \text{ and } M' = M \setminus \{p \rightarrow q\}$$
- decomposition of existential formulas (δ -rule)
$$TP(M) = TP(M' \cup \{A(c)\}), \text{ if } (\exists x)A(x) \in M, \text{ where } c - \text{new constant and } M' = M \setminus \{(\exists x)A(x)\}$$
$$TP(M) = TP(M' \cup \{\neg A(c)\}), \text{ if } \neg(\forall x)A(x) \in M, \text{ where } c - \text{new constant and } M' = M \setminus \{\neg(\forall x)A(x)\}$$
- decomposition of universal formulas (γ -rule)
$$TP(M) = TP(M \cup \{A(c) \mid c \in D\}), \text{ if } (\forall x)A(x) \in M$$
$$TP(M) = TP(M \cup \{\neg A(c) \mid c \in D\}), \text{ if } \neg(\exists x)A(x) \in M$$
 - where D is the domain of interpretation containing all the constants previously introduced. If $D = \emptyset$, then a new constant c is introduced and we have that $D = \{c\}$.

Semantic Tableaux Method

The semantic tableau $\cup_{i=1}^m \{\cup_{k=1}^{n_i} \{l_{i_k}\}\}$, where $l_{i_k}, i = \overline{1, m}, k = \overline{1, n_i}$ are literals, corresponds to a formula with the disjunctive normal form $\vee_{i=1}^m (\wedge_{k=1}^{n_i} l_{i_k})$. It has m branches and corresponds to the disjunction of its branches. The i -branch of the tableau is the set of literals: $\cup_{k=1}^{n_i} \{l_{i_k}\}$ and represents a cube, the conjunction of its literals.

Definition 3.5.

1. If a branch contains a literal and its negation, we say that the **branch** is *closed*, otherwise the **branch** is *open*.
2. If all the branches of a tableau are closed, the **tableau** is *closed*, otherwise the **tableau** is *open*.

Theorem 3.3. Soundness and completeness of TP

A propositional/predicate formula V is a tautology (theorem) if and only if $TP(\{\neg V\})$ is closed.

Theorem 3.4.

Let U_1, U_2, \dots, U_n, V be propositional/predicate formulas.

$U_1, U_2, \dots, U_n \models V$ if and only if $TP(\{U_1, U_2, \dots, U_n, \neg V\})$ is closed.

Remarks

- If the **tableau** associated to a set of formulas is *open* then the **set of formulas** is *consistent*. Each open branch provides a partial/complete model of the set of formulas.
- If the **tableau** associated to a set of formulas is *closed* then the **set of formulas** is *inconsistent*

If the resulted semantic tableau is too complex we transform it into a *simplified form* by eliminating the closed branches (inconsistent sets) and the branches (sets) that contain another branch (set) from the tableau.

Definition 3.6.

Let M be a set of formulas. The simplified form of the semantic tableau $TP(M)$, denoted by $TS(M)$, is obtained as follows:

$$TS(M) = TP(M) \setminus \{X \mid X \in TP(M), X \text{ is closed or } \exists Y \in TP(M), \emptyset \neq Y \subset X, Y \neq X\}$$

Remarks:

1. $TS(M)$ is empty or contains only open branches.
2. Using the simplified form of a semantic tableau, the theoretic results from Theorem 3.3 and Theorem 3.4 can be written as follows:

A Computational Approach to Classical Logics and Circuits

Let U_1, U_2, \dots, U_n, V be propositional/predicate formulas.

a) V is a tautology (theorem) if and only if $TS(\{\neg V\}) = \emptyset$.

b) $U_1, U_2, \dots, U_n \models V$ if and only if $TS(\{U_1, U_2, \dots, U_n, \neg V\}) = \emptyset$.

Example 3.10.

Check whether the following logical consequence holds using TP function.

$$p \rightarrow q, \neg(q \rightarrow r) \rightarrow \neg p \models p \rightarrow r .$$

We apply Theorem 3.4 and we begin with the set $M = \{p \rightarrow q, \neg(q \rightarrow r) \rightarrow \neg p, \neg(p \rightarrow r)\}$ containing the two hypotheses and the negation of the conclusion of the initial logical consequence:

$$T_1 = TP(\{p \rightarrow q, \neg(q \rightarrow r) \rightarrow \neg p, \neg(p \rightarrow r)\})$$

- decomposition of the conjunctive formula $\neg(p \rightarrow r)$:

$$T_1 = TP(\{p \rightarrow q, \neg(q \rightarrow r) \rightarrow \neg p, p, \neg r\})$$

- decomposition of the disjunctive formula $p \rightarrow q$

$$T_1 = TP(\{\neg(q \rightarrow r) \rightarrow \neg p, p, \neg r, \neg p\}) \cup TP(\{\neg(q \rightarrow r) \rightarrow \neg p, p, \neg r, q\})$$

- simplification: elimination of the closed subtableau:

$$TP(\{\neg(q \rightarrow r) \rightarrow \neg p, p, \underline{\neg r}, \underline{\neg p}\}), \text{ contains a pair of opposite literals: } (p, \underline{\neg p})$$

We obtain: $T_2 = TP(\{\neg(q \rightarrow r) \rightarrow \neg p, p, \neg r, q\})$

- decomposition of the disjunctive formula $\neg(q \rightarrow r) \rightarrow \neg p$

$$T_2 = TP(\{p, \neg r, q, q \rightarrow r\}) \cup TP(\{p, \neg r, q, \neg p\})$$

- simplification: elimination of the subtableau $TP(\{p, \neg r, q, \underline{\neg p}\})$ which is closed (contains a pair of opposite literals: $(p, \underline{\neg p})$)

We obtain: $T_3 = TP(\{p, \neg r, q, q \rightarrow r\})$

- decomposition of the disjunctive formula $q \rightarrow r$

$$T_3 = TP(\{p, \neg r, q, \neg q\}) \cup TP(\{p, \neg r, q, r\}) = \{\{p, \neg r, \underline{q}, \underline{\neg q}\}, \{p, \underline{\neg r}, q, \underline{r}\}\}$$

- simplification: elimination of those two subtableau which are closed (the pairs of opposite literals are underlined),

$$T_4 = \emptyset$$

After applying decomposition rules and simplifications on the initial set M we obtained $TS(M) = T_4 = \emptyset$, thus the logical consequence:

$$p \rightarrow q, \neg(q \rightarrow r) \rightarrow \neg p \models p \rightarrow r \text{ holds.}$$

Example 3.11.

Using TP function check whether $U = (p \rightarrow q) \wedge p \wedge \neg(q \rightarrow r)$ is a consistent formula. If so, find the models of U .

Semantic Tableaux Method

We build: $T_1 = TP(\{(p \rightarrow q) \wedge p \wedge \neg(q \rightarrow r)\})$

- decomposition of the conjunctive formula $(p \rightarrow q) \wedge p \wedge \neg(q \rightarrow r)$

$$T_1 = TP(\{p \rightarrow q, p, \neg(q \rightarrow r)\})$$

- decomposition of the conjunctive formula $\neg(q \rightarrow r)$

$$T_1 = TP(\{p \rightarrow q, p, q, \neg r\})$$

- decomposition of the disjunctive formula $p \rightarrow q$

$$T_1 = TP(\{p, q, \neg r, \neg p\}) \cup TP(\{p, q, \neg r, q\})$$

- simplification: elimination of the subtableau $TP(\{\underline{p}, \underline{q}, \neg r, \neg p\})$ which is closed (contains a pair of opposite literals: underlined)

$$T_2 = TP(\{p, q, \neg r, q\}) = \{\{p, q, \neg r\}\}$$

The semantic tableau $T_2 = TS(\{(p \rightarrow q) \wedge p \wedge \neg(q \rightarrow r)\})$ is open because it contains an open branch, so U is a consistent formula.

According to the above semantic tableau the reduced disjunctive normal form of U is: $DNF(U) = p \wedge q \wedge \neg r$.

U has a unique model:

$$i: F_p \rightarrow \{T, F\} \text{ with } i(p) = T, i(q) = T, i(r) = F \text{ and } i(U) = T.$$

Example 3.12.

Check the distributivity property of the existential quantifier (\exists) over implication (\rightarrow) using TP function.

The property is expressed by the following logical equivalence relation:

$$((\exists x)A(x) \rightarrow (\exists x)B(x)) \equiv (\exists x)(A(x) \rightarrow B(x)).$$

The problem is reduced to checking the validity of the formulas: U_1 and U_2 , where:

$$U_1 = ((\exists x)A(x) \rightarrow (\exists x)B(x)) \rightarrow (\exists x)(A(x) \rightarrow B(x))$$

$$U_2 = (\exists x)(A(x) \rightarrow B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\exists x)B(x))$$

We build $TP(\{\neg U_1\})$:

$$T_1 = TP(\{\neg U_1\}) = TP(\{\neg((\exists x)A(x) \rightarrow (\exists x)B(x)) \rightarrow (\exists x)(A(x) \rightarrow B(x)))\})$$

- decomposition of the conjunctive formula: $\neg U_1$:

$$T_1 = TP(\{(\exists x)A(x) \rightarrow (\exists x)B(x), \neg(\exists x)(A(x) \rightarrow B(x))\})$$

- decomposition of the disjunctive formula: $(\exists x)A(x) \rightarrow (\exists x)B(x)$:

$$T_1 = TP(\{\neg(\exists x)A(x), (\forall x)\neg(A(x) \rightarrow B(x))\}) \cup$$

$$TP(\{(\exists x)B(x), (\forall x)\neg(A(x) \rightarrow B(x))\})$$

A Computational Approach to Classical Logics and Circuits

- decomposition of the existential formula $(\exists x)B(x)$, b - a new constant, on the second branch and decomposition of the universal formula $\neg(\exists x)A(x) \equiv (\forall x)\neg A(x)$, a new constant a is introduced and used for instantiation, because there are no constants on the first branch:

$$T_1 = TP(\{\neg A(a), \neg(\exists x)A(x), (\forall x)\neg(A(x) \rightarrow B(x))\}) \cup \\ \cup TP(\{B(b), (\forall x)\neg(A(x) \rightarrow B(x))\})$$

- decomposition of the universal formula $(\forall x)\neg(A(x) \rightarrow B(x))$ on both branches, with the appropriate instantiations on each branch:

$$T_1 = TP(\{\neg A(a), (\forall x)\neg A(x), \neg(A(a) \rightarrow B(a)), (\forall x)\neg(A(x) \rightarrow B(x))\}) \cup \\ \cup TP(\{B(b), \neg(A(b) \rightarrow B(b)), (\forall x)\neg(A(x) \rightarrow B(x))\})$$

- decomposition of the conjunctive formulas on both branches:

$$T_1 = TP(\{\neg A(a), (\forall x)\neg A(x), A(a), \neg B(a), (\forall x)\neg(A(x) \rightarrow B(x))\}) \cup \\ \cup TP(\{B(b), A(b), \neg B(b), (\forall x)\neg(A(x) \rightarrow B(x))\}) = \\ = \{\underline{\neg A(a)}, \underline{A(a)}, \underline{\neg B(a)}, \{B(b), A(b), \underline{\neg B(b)}\}\}$$

T_1 is a closed semantic tableau because it has two closed branches, each branch contains a pair of opposite literals (underlined). Therefore the formula $\neg U_1$ is inconsistent and U_1 is valid.

We shall prove that U_2 is not valid because $TP(\neg U_2)$ is a complete and open semantic tableau.

$$TP(\{\neg U_2\}) = TP(\{\neg((\exists x)(A(x) \rightarrow B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\exists x)B(x)))\}) = \\ = TP(\{(\exists x)(A(x) \rightarrow B(x)), \neg((\exists x)A(x) \rightarrow (\exists x)B(x))\})$$

δ -rule is applied, a new constant a is introduced

$$= TP(\{A(a) \rightarrow B(a), \neg((\exists x)A(x) \rightarrow (\exists x)B(x))\}) = \\ = TP(\{A(a) \rightarrow B(a), (\exists x)A(x), \neg(\exists x)B(x)\})$$

γ -rule is applied, the constants a and b are used for instantiation

$$= TP(\{A(a) \rightarrow B(a), A(b), \neg(\exists x)B(x)\}) =$$

δ -rule is applied, a new constant b is introduced

$$= TP(\{A(a) \rightarrow B(a), A(b), \neg B(a), \neg B(b), \neg(\exists x)B(x)\}) =$$

β -rule is applied

$$= TP(\{\neg A(a), A(b), \neg B(a), \neg B(b), \neg(\exists x)B(x)\}) \cup \\ \cup TP(\{B(a), A(b), \neg B(a), \neg B(b), \neg(\exists x)B(x)\}) = \\ = \{\underline{\neg A(a)}, A(b), \neg B(a), \neg B(b), \neg(\exists x)B(x)\}, \\ \{B(a), \underline{A(b)}, \underline{\neg B(a)}, \neg B(b), \neg(\exists x)B(x)\}\}$$

The final semantic tableau is complete and open, having one complete open branch (the first branch) and one closed branch ($\{B(a), \neg B(a)\}$) is the pair of opposite

Semantic Tableaux Method

literals on the second branch). The formula $\neg U_2$ is not inconsistent, so U_2 is not valid.

We conclude that *the existential quantifier is only semi-distributive over implication:*

$$\models ((\exists x)A(x) \rightarrow (\exists x)B(x)) \rightarrow (\exists x)(A(x) \rightarrow B(x)) \text{ and}$$

$$\not\models (\exists x)(A(x) \rightarrow B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\exists x)B(x))$$

3.3. An Automated Theorem Prover for Propositional Logic, based on the Semantic Tableaux Method

The aim of the proposed propositional theorem prover is to check the consistency or inconsistency of a propositional formula/set of formulas and to provide a model in case of consistency.

This theorem prover is based on a modified version of the semantic tableaux method and it is described in paper [35]. We will use the same representation for a tableau as the function TP : a set of sets of literals, but the construction of the tableau is different.

A disadvantage of the definition of TP function described in the previous section is the fact that for a formula we can build many semantic tableaux, depending on the order of the application of the decomposition rules, and because of this it is hard to obtain a minimal form of the tableau.

The new idea is to construct the semantic tableau of a propositional formula from its postfix form. Traversing the postfix form from left to right, using a stack mechanism to memorize partial semantic tableaux (corresponding to the subformulas of the formula), and applying operations to the tableaux, the construction of the semantic tableau is very simple and efficient.

Data structures used to represent and manipulate propositional formulas

Postfix and prefix notations of arithmetic expressions are used internally by computers to manipulate and evaluate these expressions.

infix notation: $9 + (8 - 2 * 3 * 5) / 6 + 7 + 4 * 5$

- operators are in between the corresponding operands;
- parentheses are necessary to impose the order of operations;

postfix notation: $9 8 2 3 * 5 * - 6 / + 7 + 4 5 * +$

- operators are after the corresponding operands, no parentheses are needed;

prefix notation: $++ + 9 / - 8 ** 2 3 5 6 7 * 4 5$

- operators are before the corresponding operands, no parentheses are needed;
All three notations have the same order of the operands.

Traversing the postfix notation from left to right, using a stack mechanism to memorize partial results the expression is evaluated.

A Computational Approach to Classical Logics and Circuits

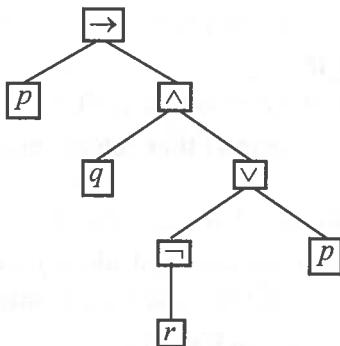
In the same manner, propositional formulas can be represented in prefix and postfix notations or as a binary tree corresponding to the decomposition of the formula in subformulas.

formula: $p \rightarrow q \wedge (\neg r \vee p)$ - infix notation

postfix notation: $p\ q\ r\ \neg\ p\ \vee\ \wedge\ \rightarrow$

prefix notation: $\rightarrow\ p\ \wedge\ q\ \vee\ \neg\ r\ p$

formula tree:



The root node contains the main connective of the formula. Each leaf node is labelled with a propositional variable and the other nodes contain connectives.

All these special representations help us to manipulate more efficiently the propositional formulas at the implementation level.

The following subalgorithm transforms a propositional formula into its postfix notation using a stack mechanism.

Subalgorithm *postfix_notation(formula, postfix)*

input parameter: *formula* – initial propositional formula,

output parameter: *postfix* – formula in postfix notation

// stack used to memorize connectives and open parentheses

// *formula, postfix* are represented as vectors

begin

j := 0

for *i* = 1, length(*formula*)

if (*formula*[*i*] is a propositional_variable)

then *j* := *j* + 1;

postfix[*j*] := *formula*[*i*];

else

if (*formula*[*i*] = '(')

then push(*stack*, '(')

else

Semantic Tableaux Method

```

if (formula[i]=='')
    then // extract from stack and add to postfix all the connectives
          // until '('
        do
          x := pop(stack)
          if (x is a connective)
              then j := j + 1;
                  postfix[j] := x;
          end_if
          until (stack is empty or x=='')
          if (stack is empty) then return("error")
          end_if
          else // formula[i] is a connective
              // extract from stack and add to postfix all the connectives with
                  // a higher or equal priority than the connective formula[i]
              oper := top_stack(stack);
                  // oper is the connective from the top of stack
              while (stack is not empty and pr(oper) >= pr(formula[i]))
                  // pr(x) returns the priority of the connective x
                  oper := pop(stack)
                  j := j + 1;
                  postfix[j] := oper;
                  oper := top_stack(stack)
              end_while
              push(stack, formula[i])
        end_if
      end_if
    end_for
// all the remaining connectives from stack are extracted and added to postfix
while (stack is not empty)
    oper := pop(stack)
    j := j + 1;
    postfix[j] := oper;
end_while
end

```

The formula tree can be built from the postfix form of the formula, using a stack mechanism to memorize nodes addresses according to the subalgorithm below.

A Computational Approach to Classical Logics and Circuits

```
Subalgorithm build_tree_from_postfix(postfix, adr_tree)
input parameter: postfix – formula in postfix notation
output parameter: adr_tree – the address of the root tree
// stack_addr is used to memorize nodes addresses
begin
    for i = 1, length(postfix)
        if (postfix[i] is a propositional_variable)
            then
                adr_node := build_node (postfix[i], NULL, NULL);
                push(stack_addr, adr_node);
            else // postfix[i] is a connective
                adr_right := pop(stack_addr);
                if (postfix[i] = '¬') // the negation connective
                    then adr_left := NULL;
                    else adr_left := pop(stack_addr);
                end_if
                adr_node := build_node (postfix[i], adr_left, adr_right)
                push(stack_addr, adr_node);
            end_if
        end_for
        adr_tree := pop(stack_addr);
    end
```

3.4. Considerations for implementation

A semantic tableau is represented as a set of sets of literals. The logical operations (connectives) of a formula are transformed into corresponding operations on semantic tableaux according to the definitions below [35].

Definition 3.7.

Let us denote by $Tsem(U)$ and $Tsem(V)$ the semantic tableaux associated to the propositional formulas U and V .

$$Tsem(l) = \{\{l\}\}, \text{ where } 'l' \text{ is a propositional literal}$$

$$Tsem(\neg U) = \neg Tsem(U),$$

$$Tsem(U \wedge V) = Tsem(U) \wedge Tsem(V),$$

$$Tsem(U \vee V) = Tsem(U) \vee Tsem(V),$$

$$Tsem(U \rightarrow V) = Tsem(U) \rightarrow Tsem(V),$$

$$Tsem(U \leftrightarrow V) = Tsem(U) \leftrightarrow Tsem(V).$$

Semantic Tableaux Method

The simplification of a semantic tableau, in order to obtain a minimal form, corresponds to the elimination of closed sets of literals, and sets containing other sets from the tableau. An empty simplified semantic tableau corresponds to a closed tableau. If we do not apply the simplification, then the final semantic tableau corresponds to the disjunctive normal form of the initial formula.

Definition 3.8.

Let $T_1 = \bigcup_{i=1}^N \{\bigcup_{k=1}^{n_i} \{a_{ik}\}\}$ and $T_2 = \bigcup_{j=1}^M \{\bigcup_{k=1}^{m_j} \{b_{jk}\}\}$ be two semantic tableaux represented using TP function as sets of sets of literals. The following relations provide the application of the logical operations: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, on semantic tableaux:

$$\neg T_1 = \{\{\neg x_1, \dots, \neg x_N\} \mid x_i \in \bigcup_{k=1}^{n_i} \{a_{ik}\}, i = 1, \dots, N\}$$

$$T_1 \vee T_2 = \{\bigcup_{k=1}^{n_i} \{a_{ik}\} \mid i = 1, \dots, N\} \cup \{\bigcup_{k=1}^{m_j} \{b_{jk}\} \mid j = 1, \dots, M\}$$

$$T_1 \wedge T_2 = \{\bigcup_{k=1}^{n_i} \{a_{ik}\} \cup \bigcup_{k=1}^{m_j} \{b_{jk}\} \mid i = 1, \dots, N, j = 1, \dots, M\}$$

$$T_1 \rightarrow T_2 = \neg T_1 \vee T_2$$

$$T_1 \leftrightarrow T_2 = (T_1 \rightarrow T_2) \wedge (T_2 \rightarrow T_1)$$

The above operations permit the construction of the semantic tableau of a formula from the semantic tableaux of its subformulas.

Example 3.13.

Let $T_1 = \{\{p, \neg q, r\}, \{\neg p\}, \{q, \neg r\}\}$ and $T_2 = \{\{s, r\}, \{t\}\}$ be two semantic tableaux represented using TP function. We apply the previous definition.

$$\neg T_1 = \{\{\neg p, p, \neg q\}, \{\neg p, p, r\}, \{q, p, \neg q\}, \{q, p, r\}, \{\neg r, p, \neg q\}, \{\neg r, p, r\}\}$$

The simplified form of T_1 obtained by eliminating the inconsistent sets (closed branches containing a pair of opposite literals) is $TS = \{\{q, p, r\}, \{\neg r, p, \neg q\}\}$

$$T_1 \vee T_2 = \{\{p, \neg q, r\}, \{\neg p\}, \{q, \neg r\}, \{s, r\}, \{t\}\}$$

$$T_1 \wedge T_2 = \{\{p, \neg q, r, s, r\}, \{p, \neg q, r, t\}, \{\neg p, s, r\}, \{\neg p, t\}, \{q, \neg r, s, r\}, \{q, \neg r, t\}\}$$

The simplified form of $T_1 \wedge T_2$ is

$$TS = \{\{p, \neg q, r, s\}, \{p, \neg q, r, t\}, \{\neg p, s, r\}, \{\neg p, t\}, \{q, \neg r, t\}\}$$

$$\begin{aligned} T_1 \rightarrow T_2 = \neg T_1 \vee T_2 &= \{\{q, p, r\}, \{\neg r, p, \neg q\}\} \vee \{\{s, r\}, \{t\}\} = \\ &= \{\{q, p, r\}, \{\neg r, p, \neg q\}, \{s, r\}, \{t\}\} \end{aligned}$$

A Computational Approach to Classical Logics and Circuits

Example 3.14.

The formula $U = \neg(p \wedge q) \vee r \wedge \neg s$ has its postfix notation: $p\ q\ \wedge\ \neg\ r\ s\ \neg\ \wedge\ \vee$.

The semantic tableau associated to $U : Tsem(U)$ is built step by step traversing the postfix form from left to right and using a stack which memorizes the partial semantic tableaux (see the next algorithm).

Symbol		
from postfix form	postfix notation: $p\ q\ \wedge\ \neg\ r\ s\ \neg\ \wedge\ \vee$	stack
'p'	$T_1 = Tsem(p) = \{\{p\}\}$	(T_1)
'q'	$T_2 = Tsem(q) = \{\{q\}\}$	(T_2, T_1)
' \wedge '	$T_3 = T_1 \wedge T_2 = \{\{p\}\} \wedge \{\{q\}\} = \{\{p, q\}\}$	(T_3)
' \neg '	$T_4 = \neg T_3 = \neg \{\{p, q\}\} = \{\{\neg p\}, \{\neg q\}\}$	(T_4)
'r'	$T_5 = Tsem(r) = \{\{r\}\}$	(T_5, T_4)
's'	$T_6 = Tsem(s) = \{\{s\}\}$	(T_6, T_5, T_4)
' \neg '	$T_7 = \neg T_6 = \neg \{\{s\}\} = \{\{\neg s\}\}$	(T_7, T_5, T_4)
' \wedge '	$T_8 = T_5 \wedge T_7 = \{\{r\}\} \wedge \{\{\neg s\}\} = \{\{r, \neg s\}\}$	(T_8, T_4)
' \vee '	$T_9 = T_4 \vee T_8 = \{\{\neg p\}, \{\neg q\}\} \vee \{\{r, \neg s\}\}$ $= \{\{\neg p\}, \{\neg q\}, \{r, \neg s\}\}$	(T_9)

$$Tsem(U) = T_9 = \{\{\neg p\}, \{\neg q\}, \{r, \neg s\}\}$$

$$\text{DNF}(U) = \neg p \vee \neg q \vee (r \wedge \neg s)$$

This new algorithm, used to build a semantic tableau for a propositional formula from its postfix notation, is more efficient than the algorithms which use the decomposition rules.

Subalgorithm construction_semantic_tableau (postfix, tab)

input parameter: **postfix** – postfix notation of the initial formula,

output parameter: **tab** – the semantic tableau

// stack is used to memorize the partial semantic tableaux

begin

for $i = 0$, length(postfix)

if (*postfix*[*i*] is a propositional_variable)

then

t := $Tsem(postfix[i])$ // $Tsem(postfix[i]) := \{\{postfix[i]\}\}$

 push(stack, *t*) // the semantic tableau *t* is added to the stack

else

Semantic Tableaux Method

```
if ( postfix[i] is “¬”)
then
    t := pop(stack);
    tl := ¬t ;
    t := simplify(tl);
    push(stack, t)
else // postfix[i] is one of the binary connectives { ∧, ∨, →, ↔ }
    tr := pop(stack) // tr is the tableau corresponding to the right
        // subformula (operand) of the connective postfix[i]
    tl := pop(stack) // tl is the tableau corresponding to the left
        // subformula (operand) of the connective postfix[i]
    t := simplify ( tl postfix[i] tr ) // tableau for tl postfix[i] tr is built
        and simplified
    push(stack, t) // the simplified tableau is added to the stack
end_if
end_if
end_for
tab := pop (stack)
// the tableau extracted from the stack is the final semantic tableau
// corresponding to the initial formula provided by its postfix notation
end
```

The following algorithm implements the semantic tableaux proof method and can be used to solve three tasks:

1. to check the validity of a propositional formula V ;
2. to check the consistency and find the models of a propositional formula V ;
3. to check if a propositional formula V is a logical consequence of the set of hypotheses: U_1, \dots, U_n .

Algorithm Semantic_tableaux_method

input: V, U_1, U_2, \dots, U_n - well-formed propositional formulas

$opt \in \{1, 2, 3\}$ - option for the task to be solved

output: a message depending on opt

begin

if ($opt = 1$) **then** //**checking_tautology**

postfix_notation($\neg V$, postfix)

construction_semantic_tableau(postfix, tab)

if (tab is closed) **then**

write " $\neg V$ is inconsistent and V is a tautology"

A Computational Approach to Classical Logics and Circuits

```
    else
        write "¬V is consistent and V is not a tautology"
    end_if
end_if
if (opt = 2) then //checking_consistent_formula
    postfix_notation(V, postfix)
    construction_semantic_tableau(postfix, tab)
    if (tab is closed) then write "V is inconsistent"
        else
            write "V is consistent"
            constructFormulaModels(tab)
    end_if
end_if
if (opt = 3) then //checking_logical_consequence
    C = U1 ∧ U2 ∧ ... ∧ Un ∧ ¬V
    postfix_notation(C, postfix)
    construction_semantic_tableau(postfix, tab)
    if (tab is closed) then
        write "V is a logical consequence of U1, U2, ..., Un"
        else write "V is not a logical consequence of U1, U2, ..., Un"
    end_if
end_if
end
```

3.5. Exercises

Exercise 3.1.

Using the semantic tableaux method (the binary tree and/or the function TP) decide what kind (consistent, inconsistent, valid) of a formula is $U_j, j \in \{1, 2, \dots, 8\}$.

If $U_j, j \in \{1, 2, \dots, 8\}$ is consistent, find all its models.

1. $U_1 = (p \wedge q) \vee (\neg p \wedge \neg r) \rightarrow (q \leftrightarrow r)$;
2. $U_2 = (p \vee q \rightarrow r) \rightarrow (p \vee r \rightarrow q)$;
3. $U_3 = (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r) \wedge q$;
4. $U_4 = (q \vee r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q$;
5. $U_5 = (r \vee q) \vee (p \rightarrow \neg r) \rightarrow (p \leftrightarrow q)$;
6. $U_6 = (r \wedge q) \vee (\neg p \vee \neg r) \rightarrow (p \leftrightarrow q)$;
7. $U_7 = (q \wedge r \rightarrow p) \rightarrow (p \rightarrow r) \wedge q$;
8. $U_8 = (p \wedge r) \vee (\neg p \wedge \neg r) \rightarrow (q \leftrightarrow r)$.

Semantic Tableaux Method

Exercise 3.2.

Prove that the following formulas are tautologies using the semantic tableaux method:

1. permutation of the premises law: $(p \rightarrow (q \rightarrow r)) \rightarrow (q \rightarrow (p \rightarrow r))$;
2. separation of the premises law: $(p \wedge q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$;
3. distribution of ' \rightarrow ' over ' \vee ': $(p \rightarrow q \vee r) \leftrightarrow (p \rightarrow q) \vee (p \rightarrow r)$;
4. distribution of ' \vee ' over ' \leftrightarrow ': $(p \vee (q \leftrightarrow r)) \leftrightarrow ((p \vee q) \leftrightarrow (p \vee r))$;
5. reunion of the premises law: $(p \rightarrow (q \rightarrow r)) \rightarrow (p \wedge q \rightarrow r)$;
6. distribution of implication: $(p \rightarrow (q \rightarrow r)) \leftrightarrow (p \rightarrow q) \rightarrow (p \rightarrow r)$;
7. distribution of ' \rightarrow ' over ' \wedge ': $(p \rightarrow q \wedge r) \leftrightarrow (p \rightarrow q) \wedge (p \rightarrow r)$;
8. distribution of ' \rightarrow ' over ' \leftrightarrow ': $(p \rightarrow (q \leftrightarrow r)) \leftrightarrow ((p \rightarrow q) \leftrightarrow (p \rightarrow r))$.

Exercise 3.3.

Using the semantic tableaux method, decide if the following logical consequences hold or not. If a logical consequence does not hold find an anti-model of it.

1. $p \rightarrow (\neg q \vee r \wedge s), p, \neg s \models \neg q$
2. $\neg p \rightarrow (\neg q \rightarrow r), r \vee q \models (\neg p \rightarrow q) \vee r$
3. $p \rightarrow (q \vee r \wedge s), p, \neg r \models q$
4. $p \rightarrow q, r \rightarrow t, p \wedge r \models q \wedge t$
5. $p \wedge (q \rightarrow r), q \vee r \models p \rightarrow (q \rightarrow r)$
6. $p \rightarrow q \models (r \rightarrow t) \rightarrow (p \wedge r \rightarrow q \wedge t)$
7. $p \wedge (q \rightarrow r), q \vee r \models p \rightarrow (q \rightarrow r)$
8. $p \rightarrow q \vee r \models (p \rightarrow q) \vee (p \rightarrow r)$

Exercise 3.4.

Write all the anti-models of the propositional formulas U_1, \dots, U_8 using the semantic tableaux method.

1. $U_1 = (p \vee q) \wedge \neg r \rightarrow p \wedge q \wedge r$;
2. $U_2 = q \wedge \neg p \wedge r \rightarrow \neg p \vee \neg(q \wedge r)$;
3. $U_3 = p \rightarrow (q \wedge r) \vee q \wedge \neg p$;
4. $U_4 = \neg p \vee (\neg q \vee r) \rightarrow q \vee \neg p \vee r$;
5. $U_5 = \neg p \vee (\neg q \vee \neg r) \rightarrow q \wedge \neg p$;
6. $U_6 = \neg p \vee (\neg q \wedge \neg r) \rightarrow q \wedge \neg p \wedge r$;
7. $U_7 = \neg p \vee \neg(q \wedge r) \rightarrow q \wedge \neg p$;
8. $U_8 = \neg(\neg p \vee q) \vee r \rightarrow \neg p \vee (\neg q \vee r)$.

A Computational Approach to Classical Logics and Circuits

Exercise 3.5.

Using the semantic tableaux method, prove the following properties in predicate logic:

1. ' \exists ' is semi-distributive over ' \wedge ':

$$\models (\exists x)(A(x) \wedge B(x)) \rightarrow (\exists x)A(x) \wedge (\exists x)B(x) \text{ and}$$

$$\not\models (\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\exists x)(A(x) \wedge B(x))$$

2. ' \forall ' is semi-distributive over ' \vee ':

$$\models (\forall x)A(x) \vee (\forall x)B(x) \rightarrow (\forall x)(A(x) \vee B(x)) \text{ and}$$

$$\not\models (\forall x)(A(x) \vee B(x)) \rightarrow (\forall x)A(x) \vee (\forall x)B(x)$$

3. ' \exists ' is semi-distributive over ' \rightarrow ':

$$\models ((\exists x)A(x) \rightarrow (\exists x)B(x)) \rightarrow (\exists x)(A(x) \rightarrow B(x)) \text{ and}$$

$$\not\models (\exists x)(A(x) \rightarrow B(x)) \rightarrow ((\exists x)A(x) \rightarrow (\exists x)B(x))$$

4. ' \forall ' is semi-distributive over ' \rightarrow ':

$$\models (\forall x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\forall x)B(x)) \text{ and}$$

$$\not\models ((\forall x)A(x) \rightarrow (\forall x)B(x)) \rightarrow (\forall x)(A(x) \rightarrow B(x))$$

5. $\models (\exists x)(A(x) \rightarrow B(x)) \rightarrow ((\forall x)A(x) \rightarrow (\exists x)B(x))$ and

$$\not\models ((\exists x)A(x) \rightarrow (\exists x)B(x)) \rightarrow (\forall x)(A(x) \rightarrow B(x))$$

6. ' \exists ' is distributive over ' \vee '

$$\models (\exists x)(A(x) \vee B(x)) \leftrightarrow (\exists x)A(x) \vee (\exists x)B(x)$$

7. ' \forall ' is distributive over ' \wedge '

$$\models (\forall x)(A(x) \wedge B(x)) \leftrightarrow (\forall x)A(x) \wedge (\forall x)B(x)$$

8. $\vdash (\exists x)(P(x) \rightarrow Q(x)) \leftrightarrow ((\forall x)P(x) \rightarrow (\exists x)Q(x))$

Exercise 3.6.

Check the validity of the following first-order formulas using the semantic tableaux method:

1. $U_1 = (\forall x)(\forall y)P(x, y) \leftrightarrow (\exists x)(\forall y)P(x, y) ;$
2. $U_2 = (\exists x)(\forall y)P(x, y) \leftrightarrow (\forall y)(\exists x)P(x, y) ;$
3. $U_3 = (\forall y)(\exists x)P(x, y) \leftrightarrow (\exists y)(\exists x)P(x, y) ;$
4. $U_4 = (\forall x)(\forall y)P(x, y) \leftrightarrow (\forall y)(\forall x)P(x, y) ;$
5. $U_5 = (\forall y)(\forall x)P(x, y) \leftrightarrow (\forall x)(\exists y)P(x, y) ;$
6. $U_6 = (\exists y)(\exists x)P(x, y) \leftrightarrow (\exists x)(\forall y)P(x, y) ;$
7. $U_7 = (\exists y)(\exists x)P(x, y) \leftrightarrow (\forall x)(\exists y)P(x, y) ;$
8. $U_8 = (\exists y)(\exists x)P(x, y) \leftrightarrow (\exists x)(\exists y)P(x, y) .$

Semantic Tableaux Method

Exercise 3.7.

Using the semantic tableaux method check whether the following logical consequences hold.

1. $(\forall x)(P(x) \rightarrow Q(x)), (\forall x)P(x) \models (\forall x)Q(x);$
2. $(\forall x)(\forall y)(Q(x, y) \rightarrow P(x, y)), (\forall z)Q(z, z) \models (\forall x)P(x, x);$
3. $P(a), (\forall x)(P(x) \rightarrow P(f(x))) \models (\forall x)P(x);$
4. $(\exists x)(\forall y)(P(x, y) \rightarrow R(x)), (\forall x)(\forall y)P(x, y) \models (\exists z)R(z);$
5. $(\forall x)(P(x) \rightarrow Q(x)), (\exists x)P(x) \models (\exists x)Q(x);$
6. $(\forall x)(\exists y)(P(x, y) \rightarrow Q(x, y)), (\exists z)P(z, z) \models (\forall x)Q(x, x);$
7. $(\exists x)(\forall y)(Q(x, y) \rightarrow P(x, y)), (\forall z)Q(z, z) \models (\exists x)P(x, x);$
8. $(\forall x)(\forall y)(P(x, y) \rightarrow P(y, x)) \models (\forall x)P(x, x).$

4. SEQUENT AND ANTI-SEQUENT CALCULI

In this chapter two complementary axiomatic systems: the *sequent* and *anti-sequent calculi*, for propositional and predicate logic are presented. They are used to check the validity/derivability and non-validity/non-derivability respectively in classical logics. We used as references the following papers: [1, 3, 5, 43, 45].

4.1. The sequent calculus

The sequent calculus axiomatic system, proposed by Gentzen in [22], is an improvement of the natural deduction system. It formalizes propositional/predicate logic and provides a direct and syntactic proof method used to check the validity and derivability of a propositional/predicate formula.

Definition 4.1. (syntax)

A *sequent* has the form: $X \Rightarrow Y$, where X and Y are finite sets of propositional/predicate formulas. X is called *antecedent* and Y is called *consequent*.

In the sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$

1. the symbol ‘ \Rightarrow ’ has the meaning of ‘provable’;
2. U_1, U_2, \dots, U_n are called hypotheses, assumptions;
3. V_1, V_2, \dots, V_m are formulas to be proved.

Definition 4.2.

A *basic sequent* has one of the forms:

1. $X, U \Rightarrow Y, U$; where the same formula, U , appears in both antecedent and consequent
2. $X \Rightarrow T$ (*true* is provable from anything).
3. $F \Rightarrow Y$ (anything is provable from *false*)

Definition 4.3. (semantics)

The sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$ is *true* if and only if the formula $U_1 \wedge U_2 \wedge \dots \wedge U_n \rightarrow V_1 \vee V_2 \vee \dots \vee V_m$ is valid (tautology), meaning that from the conjunction of the hypotheses at least one of the formulas from the consequent can be proved.

Sequent and Anti-Sequent Calculi

Remarks:

- From the semantic point of view, the formulas from the antecedent are connected by conjunction and the formulas from the consequent are connected by disjunction.
- All basic sequents are true.

Definition 4.4.

Sequent calculus – an axiomatic (formal) system of predicate logic:

$\text{Seq} = (\Sigma_{\text{Seq}}, F_{\text{Seq}}, A_{\text{Seq}}, R_{\text{Seq}})$, where:

1. $\Sigma_{\text{Seq}} = \Sigma_{\text{Pr}} \cup \{\Rightarrow\}$ is the vocabulary;
2. $F_{\text{Seq}} = \{X \Rightarrow Y \mid X, Y \subset F_{\text{Pr}}\}$ is the set of all sequents;
3. A_{Seq} is the set of axioms and contains all the basic sequents;
4. $R_{\text{Seq}} = \{\neg_l, \neg_r, \wedge_l, \wedge_r, \vee_l, \vee_r, \rightarrow_l, \rightarrow_r, \forall_l, \forall_r, \exists_l, \exists_r\}$ is the set of inference rules.

Table 1. Sequent calculus – inference/reduction rules

connective/ quantifier	Introduction into antecedent (left - side rules)	Introduction into consequent (right - side rules)
\neg	$(\neg_l) \frac{X \Rightarrow Y, V}{X, \neg V \Rightarrow Y}$	$(\neg_r) \frac{U, X \Rightarrow Y}{X \Rightarrow Y, \neg U}$
\wedge	$(\wedge_l) \frac{X, U, V \Rightarrow Y}{X, U \wedge V \Rightarrow Y}$	$(\wedge_r) \frac{X \Rightarrow U, Y \quad X \Rightarrow V, Y}{X \Rightarrow U \wedge V, Y}$
\vee	$(\vee_l) \frac{X, U \Rightarrow Y \quad X, V \Rightarrow Y}{X, U \vee V \Rightarrow Y}$	$(\vee_r) \frac{X \Rightarrow U, V, Y}{X \Rightarrow U \vee V, Y}$
\rightarrow	$(\rightarrow_l) \frac{X \Rightarrow U, Y \quad X, V \Rightarrow Y}{X, U \rightarrow V \Rightarrow Y}$	$(\rightarrow_r) \frac{X, U \Rightarrow V, Y}{X \Rightarrow U \rightarrow V, Y}$
\forall	$(\forall_l) \frac{X, U[x \leftarrow t], (\forall x)U(x) \Rightarrow Y}{X, (\forall x)U(x) \Rightarrow Y}$	$(\forall_r) \frac{X \Rightarrow V(x), Y}{X \Rightarrow (\forall x)V(x), Y}$
\exists	$(\exists_l) \frac{X, U(x) \Rightarrow Y}{X, (\exists x)U(x) \Rightarrow Y}$	$(\exists_r) \frac{X \Rightarrow V[x \leftarrow t], (\exists x)V(x), Y}{X \Rightarrow (\exists x)V(x), Y}$

In the above rules, U, V are formulas and X, Y are finite sets of formulas, even empty sets, which remain unaffected by the application of rules. t is an appropriate term (variable, constant) used in existential and universal instantiations.

The rules have one or two **premises** (above the horizontal line) and a **conclusion** (below the horizontal line).

A Computational Approach to Classical Logics and Circuits

Dual rules:

- (\wedge_l) and (\vee_r) ;
- (\vee_l) and; (\wedge_r)
- (\forall_l) and (\exists_r) ;
- (\exists_l) and. (\forall_r)

If we apply the rules from the premises to the conclusion, they are called *inference rules*. The rules are called *reduction rules* if they are applied backwards, from the conclusion to the premises.

Note: Usually, the sequent calculus rules are applied as reduction rules.

Remarks:

- The application of the reduction rules (\exists_l) and (\forall_r) realizes the transformation of the corresponding quantified variable from a bound variable into a free one (the quantifier is eliminated).
- The result of applying (\forall_l) and (\exists_r) rules is the instantiation of the universal/existential formula with appropriate terms (constants, variables) and the duplication of the universal/existential formula for further instantiations.

Recommendations:

- use first the rules with one premise;
- use the rules (\forall_r) , (\exists_l) , which transform the bound variables into free ones, before the rules (\forall_l) , (\exists_r) which instantiate the corresponding formulas with appropriate terms.

The *sequent calculus method* can be described as follows:

- We simplify (reduce) the initial sequent by successive applications of the reduction rules in order to obtain basic sequents.
- We build a binary tree (up-side down) representing the reduction process:
 - The root node (at the bottom) is labeled with the initial sequent. It is important to rename the bound variables such that they will be distinct in the initial sequent.
 - A sequent from a current node is reduced by applying the corresponding reduction rule that has it (the sequent) as conclusion. We add in the tree one or two successors of the current node and these are labeled with the premises of the applied rule.
 - The construction of the tree stops when all the leaf nodes are labeled with basic sequents (overlined) or with sequents that cannot be reduced any more.

Remarks:

- A branch having a leaf node labeled with a basic sequent is a closed branch and symbolizes validity.

Sequent and Anti-Sequent Calculi

- A branch having a leaf node labeled with a non-basic sequent that cannot be reduced any more is an open branch and symbolizes inconsistency.

Theorem 4.1. Soundness and completeness of sequent calculus

The sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$ is true if and only if it can be reduced to basic sequents using reduction rules.

or

The sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$ is true if and only if it can be derived from basic sequents using inference rules.

The following theoretic results provide a direct proof method used to solve the two decision problems (to check the validity / derivability) in classical logics.

Theorem 4.2.

Let V be a propositional/predicate formula.

$\vdash V$ (V is a theorem) if and only if

$\models V$ (V is a tautology) if and only if

the sequent $\Rightarrow V$ is true (it can be reduced to basic sequents).

Theorem 4.3.

Let U_1, U_2, \dots, U_n, V be propositional/predicate formulas.

$U_1, U_2, \dots, U_n \vdash V$ if and only if

$U_1, U_2, \dots, U_n \models V$ if and only if

the sequent $U_1, U_2, \dots, U_n \Rightarrow V$ is true (it can be reduced to basic sequents).

From the above theorems we have that the following meta-symbols are equivalent:

- \vdash (syntactic consequence, deductibility, derivability)
- \models (logical consequence)
- \Rightarrow (provability)

Propositional logic is decidable, therefore the following statements are true:

1. The binary tree is always finite.
2. If the initial sequent $\Rightarrow V$ is reduced to basic sequents, the propositional formula V is valid.
3. If the initial sequent $\Rightarrow V$ cannot be reduced to basic sequents, the propositional formula V is not valid. Its anti-models are provided by the non-basic sequents from the leaf nodes, assigning the truth value T to all the literals from the antecedent and the truth value F to all the literals from the consequent.

A Computational Approach to Classical Logics and Circuits

The *semi-decidability of predicate logic* is expressed as follows:

1. If the binary tree which represents the reduction process of the sequent $\Rightarrow V$ is finite, we can decide if the formula V is a tautology (all the leaf nodes contain basic sequents) or not (at least one leaf node contains a non-basic sequent).
2. If the binary tree which represents the reduction process of the sequent $\Rightarrow V$ is infinite, we cannot decide if the formula V is valid or not.

Example 4.1.

Prove that the syllogism rule: $V = (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$ is a theorem using the sequent calculus method.

We build the up-side down binary tree corresponding to the reduction process of the initial sequent: $\Rightarrow (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$

$$\frac{\frac{\frac{\frac{p, \underline{q} \rightarrow r \Rightarrow r, \underline{p}}{p, \underline{q} \rightarrow r \Rightarrow r, q \Rightarrow r} (\rightarrow_l) \text{ for } p \rightarrow q}{p, p \rightarrow q, q \rightarrow r \Rightarrow r} (\rightarrow_r)}{p \rightarrow q, q \rightarrow r \Rightarrow p \rightarrow r} (\rightarrow_r)}{p \rightarrow q \Rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_r)$$

$$\frac{p \rightarrow q \Rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)}{\Rightarrow (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))} (\rightarrow_r)$$

All three leaf nodes contain basic sequents (the literals belonging to both antecedent and consequent are underlined), therefore the initial sequent was reduced to basic sequents. Thus, $\Rightarrow V$ is a true sequent and according to Theorem 4.2, the formula $V = (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$ is a theorem.

Example 4.2.

Using the sequent calculus determine whether the following logical consequence: $\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \models \neg p$ holds or not.

According to Theorem 4.3 we have to check if the initial sequent: $\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \Rightarrow \neg p$ is true or not.

$$\frac{\frac{\frac{\frac{\frac{\neg p \vee q, \underline{r}, p \Rightarrow \underline{r}}{\neg p \vee q, r \vee \neg q, p \Rightarrow r} (\vee_l) \text{ for } r \vee \neg q}{\neg p \vee q, r \vee \neg q, \underline{p} \Rightarrow \underline{p}} (\neg_l)}{\neg p \vee q, r \vee \neg q, \underline{p} \Rightarrow p} (\neg_r)}{\neg p \vee q, r \vee \neg q, p \Rightarrow p \wedge r} (\wedge_r)}{\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \Rightarrow \neg p} (\neg_l)$$

The above binary tree corresponding to the reduction of the initial sequent has four leaf nodes containing basic sequents (overlined). Therefore the initial sequent is true and the logical consequence: $\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \models \neg p$ holds.

Sequent and Anti-Sequent Calculi

Example 4.3.

Prove that the formula $U = \neg(p \wedge q) \rightarrow \neg p \wedge \neg q$ is not valid.

The reduction of the initial sequent $\Rightarrow \neg(p \wedge q) \rightarrow \neg p \wedge \neg q$ is graphically symbolized as follows:

$$\begin{array}{c}
 \frac{\frac{\frac{p \Rightarrow p}{\Rightarrow p, \neg p}(\neg_r) \quad q \Rightarrow p}{\Rightarrow p, \neg p, q}(\wedge_r)}{\Rightarrow p, \neg p \wedge \neg q} \\
 \frac{\frac{p \Rightarrow q}{\Rightarrow q, \neg p}(\neg_r) \quad \frac{q \Rightarrow q}{\Rightarrow q, \neg q}(\neg_r)}{\Rightarrow q, \neg p \wedge \neg q}(\wedge_r) \text{ for } p \wedge q \\
 \frac{\Rightarrow p \wedge q, \neg p \wedge \neg q}{\neg(p \wedge q) \Rightarrow \neg p \wedge \neg q}(\neg_l) \\
 \frac{}{\Rightarrow \neg(p \wedge q) \rightarrow \neg p \wedge \neg q}(\rightarrow_r)
 \end{array}$$

The up-side down binary tree has two leaf nodes containing basic sequents ($p \Rightarrow p$ and $q \Rightarrow q$) and two leaf nodes with non-basic sequents ($q \Rightarrow p$ and $p \Rightarrow q$), therefore $\Rightarrow U$ is not a true sequent and U is not a valid formula.

The non-basic sequents provide the anti-models of the formula U assigning the truth value T to all the literals from the antecedent and F to all the literals from the consequent.

- the sequent $q \Rightarrow p$ provides the anti-model:

$$i_1 : \{p, q\} \rightarrow \{T, F\}, i_1(p) = F, i_1(q) = T, \text{ so } i_1(U) = F$$

- the sequent $p \Rightarrow q$ provides the anti-model:

$$i_2 : \{p, q\} \rightarrow \{T, F\}, i_2(p) = T, i_2(q) = F, \text{ so } i_2(U) = F$$

We conclude that the contingent formula U has two anti-models: i_1 , i_2 and two models corresponding to the other two interpretations.

Example 4.4.

Determine if we can apply a property of distributivity on the left-hand side of the ' \rightarrow ' connective over the ' \wedge ' connective.

This property is expressed as: $(p \rightarrow q \wedge r) \equiv (p \rightarrow q) \wedge (p \rightarrow r)$ equivalent to:

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

We replace ' \leftrightarrow ' by the conjunction of the ' \rightarrow ' in both directions.

$$\models ((p \rightarrow q \wedge r) \rightarrow (p \rightarrow q) \wedge (p \rightarrow r)) \wedge ((p \rightarrow q) \wedge (p \rightarrow r) \rightarrow (p \rightarrow q \wedge r))$$

and we have to check the validity of the formulas:

$$U_1 = (p \rightarrow q \wedge r) \rightarrow (p \rightarrow q) \wedge (p \rightarrow r) \text{ and}$$

$$U_2 = (p \rightarrow q) \wedge (p \rightarrow r) \rightarrow (p \rightarrow q \wedge r).$$

The binary tree corresponding to the reduction of the initial sequent $\Rightarrow (p \rightarrow q \wedge r) \rightarrow (p \rightarrow q) \wedge (p \rightarrow r)$ for U_1 is depicted below:

A Computational Approach to Classical Logics and Circuits

$$\begin{array}{c}
 \frac{\underline{p} \Rightarrow q, \underline{p}}{p \rightarrow q \wedge r, p \Rightarrow q} (\rightarrow_r) \quad \frac{\underline{p}, \underline{q}, \underline{r} \Rightarrow \underline{q}}{p, q \wedge r \Rightarrow q} (\wedge_l) \\
 \frac{p \rightarrow q \wedge r, p \Rightarrow q}{p \rightarrow q \wedge r \Rightarrow p \rightarrow q} (\rightarrow_l) \quad \frac{\underline{p} \Rightarrow r, \underline{p}}{p \rightarrow q \wedge r, p \Rightarrow r} (\rightarrow_l) \\
 \frac{p \rightarrow q \wedge r \Rightarrow p \rightarrow q}{p \rightarrow q \wedge r \Rightarrow (p \rightarrow q) \wedge (p \rightarrow r)} (\wedge_r) \\
 \frac{p \rightarrow q \wedge r \Rightarrow (p \rightarrow q) \wedge (p \rightarrow r)}{\Rightarrow (p \rightarrow q \wedge r) \rightarrow (p \rightarrow q) \wedge (p \rightarrow r)} (\rightarrow_r)
 \end{array}$$

The initial sequent was reduced to four basic sequents: $\underline{p} \Rightarrow q, \underline{p}$; $\underline{p}, \underline{q}, \underline{r} \Rightarrow \underline{q}$; $\underline{p} \Rightarrow r, \underline{p}$ and $\underline{p}, \underline{q}, \underline{r} \Rightarrow \underline{r}$, therefore U_1 is a valid formula.

Similarly, we can prove that the formula U_2 is valid, therefore the distributivity on the left of ' \rightarrow ' over ' \wedge ' is a valid law.

Example 4.5.

Prove the validity of the predicate formula $U = (\forall x)P(x) \rightarrow P(a) \wedge P(b)$.

The reduction tree for the initial sequent $\Rightarrow U$ is the following:

$$\begin{array}{c}
 \frac{(\forall x)P(x), \underline{P(a)} \Rightarrow \underline{P(a)}}{(\forall x)P(x) \Rightarrow P(a)} (\forall_l)[x \leftarrow a] \quad \frac{(\forall x)P(x), \underline{P(b)} \Rightarrow \underline{P(b)}}{(\forall x)P(x) \Rightarrow P(b)} (\forall_l)[x \leftarrow b] \\
 \frac{(\forall x)P(x) \Rightarrow P(a) \quad (\forall x)P(x) \Rightarrow P(b)}{(\forall x)P(x) \Rightarrow P(a) \wedge P(b)} (\wedge_r) \\
 \frac{}{\Rightarrow (\forall x)P(x) \rightarrow P(a) \wedge P(b)} (\rightarrow_r)
 \end{array}$$

The universal quantified formulas on both branches were instantiated with appropriate terms: the constant a and the constant b , respectively. The copies of these formulas are not needed further because basic sequents were obtained.

The initial sequent was reduced to two basic sequents, and thus U is a tautology.

Example 4.6.

Determine if the predicate formula $(\forall x)P(x) \vee (\forall x)Q(x) \rightarrow (\forall x)(P(x) \vee Q(x))$ is a tautology.

It is important to rename the bound variables such that they are distinct in the initial sequent: $\Rightarrow (\forall x)P(x) \vee (\forall y)Q(y) \rightarrow (\forall z)(P(z) \vee Q(z))$.

We build the up-side down binary tree corresponding to the reduction process.

- The application of the reduction rule (\forall_r) realizes the transformation of z from a bound variable into a free one (the universal quantifier is eliminated).
- The result of applying the rule (\forall_l) is the instantiation of the universal formula with appropriate terms (in this example x, y are replaced by z) and the duplication of the universal formula for further instantiations.

Sequent and Anti-Sequent Calculi

$$\begin{array}{c}
 (\forall_I), [x \leftarrow z] \frac{(\forall x)P(x), P(z) \Rightarrow P(z), Q(z)}{(\forall x)P(x) \Rightarrow P(z), Q(z)} \quad \frac{(\forall y)Q(y), Q(z) \Rightarrow P(z), Q(z)}{(\forall y)Q(y) \Rightarrow P(z), Q(z)} (\forall_I), [y \leftarrow z] \\
 \frac{}{(\forall x)P(x) \vee (\forall y)Q(y) \Rightarrow P(z), Q(z)} (\vee_I) \\
 \frac{(\forall x)P(x) \vee (\forall y)Q(y) \Rightarrow P(z), Q(z)}{(\forall x)P(x) \vee (\forall y)Q(y) \Rightarrow P(z) \vee Q(z)} (\vee_r) \\
 \frac{(\forall x)P(x) \vee (\forall y)Q(y) \Rightarrow P(z) \vee Q(z)}{(\forall x)P(x) \vee (\forall y)Q(y) \Rightarrow (\forall z)(P(z) \vee Q(z))} (\forall_r) \\
 \frac{(\forall x)P(x) \vee (\forall y)Q(y) \Rightarrow (\forall z)(P(z) \vee Q(z))}{\Rightarrow (\forall x)P(x) \vee (\forall y)Q(y) \rightarrow (\forall z)(P(z) \vee Q(z))} (\rightarrow_r)
 \end{array}$$

The reduction process is a finite one. Both leaf nodes of the reduction tree contain basic sequents, therefore the initial formula is a tautology.

Example 4.7.

Determine if there is a property of distributivity of the existential quantifier over conjunction.

The formula $U = (\exists x)P(x) \wedge (\exists x)Q(x) \leftrightarrow (\exists x)(P(x) \wedge Q(x))$ must be a tautology.

We prove that the formula $U = U_1 \wedge U_2$ is not valid, proving that $U_1 = (\exists x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)(P(x) \wedge Q(x))$ is not a tautology.

The reverse implication: $U_2 = (\exists x)(P(x) \wedge Q(x)) \rightarrow (\exists x)P(x) \wedge (\exists x)Q(x)$ is a tautology.

For building the initial sequents corresponding to U_1 and U_2 we have to rename the bound variables such that they are distinct in the initial sequents.

The reduction tree of the initial sequent

$\Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y) \rightarrow (\exists z)(P(z) \wedge Q(z))$ corresponding to U_1 is as follows:

$$\begin{array}{c}
 \frac{P(x), \underline{Q(y)} \Rightarrow Q(x), \underline{Q(y)}, A \quad P(x), Q(y) \Rightarrow Q(x), P(y), A}{P(x), Q(y) \Rightarrow Q(x), P(y) \wedge Q(y), (\exists z)(P(z) \wedge Q(z))} (\wedge_r) (\exists_r)[z \leftarrow y] \\
 \frac{P(x), Q(y) \Rightarrow P(x), A \quad P(x), Q(y) \Rightarrow Q(x), (\exists z)(P(z) \wedge Q(z))}{P(x), Q(y) \Rightarrow P(x) \wedge Q(x), (\exists z)(P(z) \wedge Q(z))} (\wedge_r) \\
 \frac{P(x), Q(y) \Rightarrow P(x) \wedge Q(x), (\exists z)(P(z) \wedge Q(z))}{P(x), Q(y) \Rightarrow (\exists z)(P(z) \wedge Q(z))} (\exists_r)[z \leftarrow x] \\
 \frac{P(x), Q(y) \Rightarrow (\exists z)(P(z) \wedge Q(z))}{(\exists x)P(x), (\exists y)Q(y) \Rightarrow (\exists z)(P(z) \wedge Q(z))} (\exists_l) \\
 \frac{(\exists x)P(x), (\exists y)Q(y) \Rightarrow (\exists z)(P(z) \wedge Q(z))}{(\exists x)P(x) \wedge (\exists y)Q(y) \Rightarrow (\exists z)(P(z) \wedge Q(z))} (\wedge_l) \\
 \frac{(\exists x)P(x) \wedge (\exists y)Q(y) \Rightarrow (\exists z)(P(z) \wedge Q(z))}{\Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y) \rightarrow (\exists z)(P(z) \wedge Q(z))} (\rightarrow_r)
 \end{array}$$

We denote by A the existential formula $(\exists z)(P(z) \wedge Q(z))$.

A Computational Approach to Classical Logics and Circuits

In the first application of the rule (\exists_r) for A we use the substitution $[z \leftarrow x]$, and in the second application the substitution $[z \leftarrow y]$ is used. The finite binary tree has two basic sequents (overlined) and one non-basic sequent.

The node which contains the sequent $P(x), Q(y) \Rightarrow Q(x), P(y), A$ is a leaf node because this sequent cannot be reduced any more (all the appropriate instantiations were applied).

The sequent $\Rightarrow U_1$ cannot be reduced to basic sequents, therefore U_1 is not valid.

Now we build the reduction tree of the initial sequent:

$\Rightarrow (\exists z)(P(z) \wedge Q(z)) \rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)$, corresponding to the formula U_2 :

$$\begin{array}{c}
 \dots \dots \dots \dots \dots \\
 (\exists_r), [x \leftarrow z] \frac{\begin{array}{c} P(z), Q(z) \Rightarrow P(z) \\ \hline P(z), Q(z) \Rightarrow (\exists x)P(x) \end{array}}{P(z), Q(z) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)} \quad (\exists_r), [y \leftarrow z] \frac{\begin{array}{c} P(z), Q(z) \Rightarrow Q(z) \\ \hline P(z), Q(z) \Rightarrow (\exists y)Q(y) \end{array}}{P(z), Q(z) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)} (\wedge_r) \\
 \frac{P(z), Q(z) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)}{P(z) \wedge Q(z) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)} (\wedge_l) \\
 \frac{P(z) \wedge Q(z) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)}{(\exists z)(P(z) \wedge Q(z)) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)} (\exists_l) \\
 \frac{(\exists z)(P(z) \wedge Q(z)) \Rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)}{\Rightarrow (\exists z)(P(z) \wedge Q(z)) \rightarrow (\exists x)P(x) \wedge (\exists y)Q(y)} (\rightarrow_r)
 \end{array}$$

The sequent $\Rightarrow U_2$ was reduced to two basic sequents, therefore U_2 is a valid formula. As a conclusion, the formula U is not a tautology and thus the property of distributivity of the existential quantifier over conjunction does not hold.

Due to the fact that U_2 is a tautology, we have the property of semi-distributivity of the existential quantifier over conjunction.

Example 4.8.

Determine if the following predicate formula:

$U = (\exists x)(\forall y)(P(x, y) \rightarrow Q(x)) \rightarrow ((\forall x)(\forall y)P(x, y) \rightarrow (\exists s)Q(s))$ is valid.

The initial sequent, obtained after renaming the bound variables,
 $\Rightarrow (\exists x)(\forall y)(P(x, y) \rightarrow Q(x)) \rightarrow ((\forall z)(\forall t)P(z, t) \rightarrow (\exists s)Q(s))$
is reduced according to the following binary tree.

We use the notations: $A = (\exists s)Q(s)$, $B(x) = (\forall y)(P(x, y) \rightarrow Q(x))$, $C = (\forall z)(\forall t)P(z, t)$, $D(x) = (\forall t)P(x, t)$ for the copies of the existential and universal formulas instantiated during the reduction process.

The reduction tree is a finite one and has two leaf nodes labeled with basic sequents (overlined), therefore the initial sequent is a true one and the formula U is a tautology.

Sequent and Anti-Sequent Calculi

$$\begin{array}{c}
 \frac{P(x, x), B(x), C, D(x) \Rightarrow P(x, x), Q(x), A \quad P(x, x), \underline{Q}(x), B(x), C, D(x) \Rightarrow \underline{Q}(x), A}{P(x, x) \rightarrow Q(x), P(x, x), B(x), C, D(x) \Rightarrow Q(x), A} (\rightarrow_l) \\
 \frac{P(x, x) \rightarrow Q(x), P(x, x), B(x), C, D(x) \Rightarrow Q(x), A \quad (\forall t)[t \leftarrow x]}{P(x, x) \rightarrow Q(x), B(x), (\forall t)P(x, t), C \Rightarrow Q(x), A} (\forall_l)[y \leftarrow x, z \leftarrow x] \\
 \frac{P(x, x) \rightarrow Q(x), B(x), (\forall t)P(x, t), C \Rightarrow Q(x), A \quad (\forall t)[y \leftarrow x, z \leftarrow x]}{(\forall y)(P(x, y) \rightarrow Q(x)), (\forall z)(\forall t)P(z, t) \Rightarrow Q(x), A} (\exists_r)[s \leftarrow x] \\
 \frac{(\forall y)(P(x, y) \rightarrow Q(x)), (\forall z)(\forall t)P(z, t) \Rightarrow Q(x), A \quad (\exists_r)[s \leftarrow x]}{(\forall y)(P(x, y) \rightarrow Q(x)), (\forall z)(\forall t)P(z, t) \Rightarrow (\exists s)Q(s)} (\exists_l) \\
 \frac{(\forall y)(P(x, y) \rightarrow Q(x)), (\forall z)(\forall t)P(z, t) \Rightarrow (\exists s)Q(s) \quad (\exists_l)}{(\exists x)(\forall y)(P(x, y) \rightarrow Q(x)), (\forall z)(\forall t)P(z, t) \Rightarrow (\exists s)Q(s)} (\rightarrow_r) \\
 \frac{(\exists x)(\forall y)(P(x, y) \rightarrow Q(x)) \Rightarrow (\forall z)(\forall t)P(z, t) \rightarrow (\exists s)Q(s) \quad (\rightarrow_r)}{\Rightarrow (\exists x)(\forall y)(P(x, y) \rightarrow Q(x)) \rightarrow ((\forall z)(\forall t)P(z, t) \rightarrow (\exists s)Q(s))} (\rightarrow_r)
 \end{array}$$

4.2. The anti-sequent calculus

The anti-sequent calculus axiomatic system was introduced in paper [5] as a complementary system of sequent calculus and the corresponding proof method is used to check the non-validity and non-derivability in propositional/predicate logic. This method was then adapted for non-monotonic logics [36].

Definition 4.5. (syntax)

An *anti-sequent* has the form $X \not\Rightarrow Y$, where X (*antecedent*) and Y (*consequent*) are finite sets of propositional/predicate formulas. The meta-symbol $\not\Rightarrow$ means *non-provable*.

An anti-sequent $X \not\Rightarrow Y$ is called a *basic anti-sequent* if all the formulas of X and Y are atomic formulas and $X \cap Y = \emptyset$.

Definition 4.6. (semantics)

Let $U_1, U_2, \dots, U_n, V_1, V_2, \dots, V_m$ be propositional/predicate formulas.

The *anti-sequent* $U_1, U_2, \dots, U_n \not\Rightarrow V_1, V_2, \dots, V_m$ is true if there is a model I of the set $\{U_1, U_2, \dots, U_n\}$ of formulas which is an anti-model of the set $\{V_1, V_2, \dots, V_m\}$.

Remark: The basic anti-sequents are true.

Definition 4.7. Anti-sequent calculus – axiomatic (formal) system

$Seq^c = (\Sigma_{Seq}^c, F_{Seq}^c, A_{Seq}^c, R_{Seq}^c)$, where:

1. $\Sigma_{Seq}^c = \Sigma_{Pr} \cup \{\not\Rightarrow\}$ is the vocabulary
2. $F_{Seq}^c = \{X \not\Rightarrow Y \mid X, Y \subset F_{Pr}\}$ is the set of all anti-sequents
3. A_{Seq}^c is the set of axioms and contains all the basic anti-sequents.
4. $R_{Seq}^c = \{\neg_l^c, \neg_r^c, \wedge_l^c, \wedge_r^c, \wedge_{r_1}^c, \wedge_{r_2}^c, \vee_l^c, \vee_{l_1}^c, \vee_{l_2}^c, \vee_r^c, \vee_{r_1}^c, \rightarrow_l^c, \rightarrow_{l_1}^c, \rightarrow_{l_2}^c, \rightarrow_r^c, \rightarrow_{r_1}^c, \exists_l^c, \exists_r^c\}$ is the set of inference/reduction rules

A Computational Approach to Classical Logics and Circuits

Table 2. Anti-sequent calculus – inference /reduction rules

Introduction into antecedent (left - side rules)	Introduction into consequent (right - side rules)
$(\neg_l^c) \frac{X \Rightarrow U, Y}{X, \neg U \Rightarrow Y}$	$(\neg_r^c) \frac{X, V \Rightarrow Y}{X \Rightarrow \neg V, Y}$
$(\wedge_l^c) \frac{X, U, V \Rightarrow Y}{X, U \wedge V \Rightarrow Y}$	$(\wedge_{r_1}^c) \frac{X \Rightarrow U, Y}{X \Rightarrow U \wedge V, Y}$ $(\wedge_{r_2}^c) \frac{X \Rightarrow V, Y}{X \Rightarrow U \wedge V, Y}$
$(\vee_{l_1}^c) \frac{X, U \Rightarrow Y}{X, U \vee V \Rightarrow Y}$ $(\vee_{l_2}^c) \frac{X, V \Rightarrow Y}{X, U \vee V \Rightarrow Y}$	$(\vee_r^c) \frac{X \Rightarrow U, V, Y}{X \Rightarrow U \vee V, Y}$
$(\rightarrow_{l_1}^c) \frac{X \Rightarrow U, Y}{X, U \rightarrow V \Rightarrow Y}$ $(\rightarrow_{l_2}^c) \frac{X, V \Rightarrow Y}{X, U \rightarrow V \Rightarrow Y}$	$(\rightarrow_r^c) \frac{X, U \Rightarrow V, Y}{X \Rightarrow U \rightarrow V, Y}$
$(\forall_l^c) \frac{\{X, U[x \leftarrow t] \Rightarrow Y \mid t - \text{term}\}}{X, (\forall x)U(x) \Rightarrow Y}$	$(\forall_r^c) \frac{X \Rightarrow V(x), Y}{X \Rightarrow (\forall x)V(x), Y}$
$(\exists_l^c) \frac{X, U(x) \Rightarrow Y}{X, (\exists x)U(x) \Rightarrow Y}$	$(\exists_r^c) \frac{\{X \Rightarrow V[x \leftarrow t], Y \mid t - \text{term}\}}{X \Rightarrow (\exists x)V(x), Y}$

In the reduction rules, U, V are formulas and X, Y are finite sets of formulas, even empty sets, which remain unaffected by the application of rules. t is an appropriate term (variable, constant) used in existential and universal instantiations.

The difference between *Table 1* and *Table 2* consists in splitting the rules with two premises from the sequent calculus into pairs of rules in the anti-sequent calculus. Thus, the exhaustive search in the sequent calculus becomes non-determinism in the anti-sequent calculus and the reduction process is a linear one.

Theorem 4.4. (complementarity of sequent and anti-sequent calculi)

The anti-sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$ is true

if and only if

the sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$ is not true.

Theorem 4.5.

A propositional/predicate formula V is not a theorem (not valid) if and only if the anti-sequent $\Rightarrow V$ is true.

Sequent and Anti-Sequent Calculi

According to Theorem 4.3, *derivability is expressed in sequent calculus* as follows:

$U_1 \wedge U_2 \wedge \dots \wedge U_n \vdash V_1 \vee V_2 \vee \dots \vee V_m$ if and only if

the sequent $U_1, U_2, \dots, U_n \Rightarrow V_1, V_2, \dots, V_m$ is true,

meaning that from the conjunction of the hypotheses at least one of the formulas from the consequent can be proved (derived).

Theorem 4.6.

The *non-derivability* is expressed in *anti-sequent calculus* as follows:

$U_1 \wedge U_2 \wedge \dots \wedge U_n \not\vdash V_1 \vee V_2 \vee \dots \vee V_m$ if and only if

the anti-sequent $U_1, U_2, \dots, U_n \not\Rightarrow V_1, V_2, \dots, V_m$ is true,

meaning that from the conjunction of the hypotheses none of the formulas from consequent can be proved (derived).

The anti-sequent calculus proof method:

We simplify (reduce) the initial anti-sequent by successive applications of the reduction rules in order to obtain an axiom (basic anti-sequent). The conclusion about the validity/derivability is based on the following:

1. *If the anti-sequent $\not\Rightarrow V$ is not true* because it cannot be reduced to a basic anti-sequent, in all the possible ways of applying anti-sequent rules, *then the formula V is valid*.

2. *If the anti-sequent $\not\Rightarrow V$ is true* because it was reduced to the basic anti-sequent $p_1, \dots, p_k \not\Rightarrow q_1, \dots, q_j$ (where $p_1, \dots, p_k, q_1, \dots, q_j$ are literals), *then the formula V is not valid*.

There is an interpretation, I , called anti-model, which evaluates the formula V as false, $I(V)=F$. A partial anti-model of V is obtained as follows:

$I(p_1)=\dots=I(p_k)=T$ and $I(q_1)=\dots=I(q_j)=F$.

3. *If the anti-sequent $U_1, U_2, \dots, U_n \not\Rightarrow V_1, V_2, \dots, V_m$ is not true* because it cannot be reduced to a basic anti-sequent, in all the possible ways of applying anti-sequent rules, *then the syntactic consequence: $U_1, U_2, \dots, U_n \vdash V_1 \vee V_2 \vee \dots \vee V_m$ holds*.

4. *If the anti-sequent $U_1, U_2, \dots, U_n \not\Rightarrow V_1, V_2, \dots, V_m$ is true* because it was reduced to the basic anti-sequent $p_1, \dots, p_k \not\Rightarrow q_1, \dots, q_j$ (where $p_1, \dots, p_k, q_1, \dots, q_j$ are literals), *then none of the formulas V_1, V_2, \dots, V_m is derivable (provable) from the set $\{U_1, U_2, \dots, U_n\}$ of hypotheses*.

A Computational Approach to Classical Logics and Circuits

There is an interpretation I , which is a model for the hypotheses, $I(\{U_1, U_2, \dots, U_n\}) = T$ but it falsifies each of the formulas from the consequent: $I(V_1) = \dots = I(V_m) = F$.

A partial anti-model for the derivation $U_1 \wedge U_2 \wedge \dots \wedge U_n \vdash V_1 \vee V_2 \vee \dots \vee V_m$ is obtained as follows: $I(p_1) = \dots = I(p_k) = T$ and $I(q_1) = \dots = I(q_j) = F$.

Example 4.9.

Prove that the formula $U = \neg p \vee \neg q \rightarrow \neg(p \vee q)$ is not valid.

Using the sequent calculus:

The reduction tree of the initial sequent: $\Rightarrow \neg p \vee \neg q \rightarrow \neg(p \vee q)$ has two leaf nodes which contain basic sequents and two leaf nodes containing non-basic sequents. Therefore, the initial sequent is not true and thus *the formula U is not valid*.

$$\frac{\begin{array}{c} \frac{\overline{p \Rightarrow p}}{p, \neg p \Rightarrow (\neg_l)} & \frac{p \Rightarrow q}{p, \neg q \Rightarrow (\neg_l)} \\ p, \neg p \vee \neg q \Rightarrow & \end{array}}{p \vee q, \neg p \vee \neg q \Rightarrow} \frac{\begin{array}{c} \frac{\overline{q \Rightarrow q}}{q, \neg q \Rightarrow (\neg_l)} & \frac{q \Rightarrow p}{q, \neg p \Rightarrow (\neg_l)} \\ q, \neg p \vee \neg q \Rightarrow & \end{array}}{q, \neg p \vee \neg q \Rightarrow (\vee_l) \text{ for } p \vee q} \frac{\neg p \vee \neg q \Rightarrow \neg(p \vee q) (\neg_r)}{\Rightarrow \neg p \vee \neg q \rightarrow \neg(p \vee q) (\rightarrow_r)}$$

The non-basic sequents $p \Rightarrow q$ and $q \Rightarrow p$ provide two anti-models of U :

$$i_1 : \{p, q\} \rightarrow \{T, F\}, i_1(p) = T, i_1(q) = F, \quad i_1(U) = F \text{ and}$$

$$i_2 : \{p, q\} \rightarrow \{T, F\}, i_2(p) = F, i_2(q) = T, \quad i_2(U) = F.$$

Using the anti-sequent calculus:

$$\frac{\begin{array}{c} \frac{q \not\Rightarrow p}{q, \neg p \not\Rightarrow (\neg_l^c)} \\ q, \neg p \vee \neg q \not\Rightarrow (\vee_{l_1}^c) \end{array}}{p \vee q, \neg p \vee \neg q \not\Rightarrow (\vee_{l_2}^c) \text{ for } p \vee q} \frac{\neg p \vee \neg q \not\Rightarrow \neg(p \vee q) (\neg_r^c)}{\not\Rightarrow \neg p \vee \neg q \rightarrow \neg(p \vee q) (\rightarrow_r^c)}$$

The initial anti-sequent $\not\Rightarrow \neg p \vee \neg q \rightarrow \neg(p \vee q)$ is true because it was reduced to the basic anti-sequent $q \not\Rightarrow p$, and thus *the formula U is not valid*.

Note the exhaustive search in sequent calculus and the linear reduction process in anti-sequent calculus.

From the basic anti-sequent $q \not\Rightarrow p$, the anti-model i_2 of U is obtained as follows:

$$i_2 : \{p, q\} \rightarrow \{T, F\}, i_2(q) = T, i_2(p) = F;$$

$$i_2(U) = i_2(\neg p \vee \neg q \rightarrow \neg(p \vee q)) = \neg F \vee \neg T \rightarrow \neg(F \vee T) = T \vee F \rightarrow \neg T = F$$

Sequent and Anti-Sequent Calculi

Example 4.10.

Using the anti-sequent calculus determine if the following deduction holds:

$$\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \not\Rightarrow p .$$

An up-side down binary tree corresponding to the reduction process of the initial anti-sequent: $\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \not\Rightarrow p$ is represented graphically as follows:

$$\frac{\frac{\frac{q, r \not\Rightarrow p}{q, r \vee \neg q \not\Rightarrow p} (\vee_{l_1}^c)}{\frac{\neg p \vee q, r \vee \neg q \not\Rightarrow p}{\frac{\neg p \vee q, r \vee \neg q \not\Rightarrow p, p \wedge r}{\frac{\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \not\Rightarrow p}} (\neg_l^c)}} (\wedge_{r_1}^c)} (\vee_{l_2}^c) \text{ for } \neg p \vee q$$

The initial anti-sequent was reduced to a basic anti-sequent $q, r \not\Rightarrow p$, which provides an anti-model for the derivation(deduction). The truth value T is assigned to all the literals from the antecedent and F is assigned to all the literals from the consequent.

The anti-model $i: \{p, q, r\} \rightarrow \{T, F\}$, $i(p) = F, i(q) = T, i(r) = T$ satisfies the hypotheses of the deduction: $i(\neg p \vee q) = T$, $i(r \vee \neg q) = T$, $i(\neg(p \wedge r)) = T$ and falsifies the conclusion: $i(p) = F$.

We conclude that *the initial deduction does not hold*: $\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \not\vdash p$

Another version of the non-deterministic reduction process for the same initial anti-sequent does not end in a basic anti-sequent.

$$\frac{\frac{\frac{q, r \not\Rightarrow p, r}{q, r \vee \neg q \not\Rightarrow p, r} (\vee_{l_1}^c)}{\frac{\neg p \vee q, r \vee \neg q \not\Rightarrow p, r}{\frac{\neg p \vee q, r \vee \neg q \not\Rightarrow p, p \wedge r}{\frac{\neg p \vee q, r \vee \neg q, \neg(p \wedge r) \not\Rightarrow p}{(\neg_l^c)}} (\wedge_{r_2}^c)}} (\vee_{l_2}^c) \text{ for } \neg p \vee q}{(\neg_l^c)}$$

Example 4.11.

Check the validity of the propositional formula $U = p \vee (q \wedge r) \rightarrow (p \vee q) \wedge (p \vee r)$ using the sequent and anti-sequent calculi.

Using the sequent calculus:

The initial sequent $\Rightarrow U$ is reduced to four basic sequents as we see in the reduction tree depicted below.

According to the soundness and completeness theorem, U is a valid formula.

A Computational Approach to Classical Logics and Circuits

$$\begin{array}{c}
 \dfrac{\vdots \vdots}{(\vee_r) \dfrac{\dfrac{p \Rightarrow p, q}{p \Rightarrow p \vee q}}{p \Rightarrow (p \vee q) \wedge (p \vee r)}} \quad (\vee_r) \dfrac{\vdots \vdots}{\dfrac{p \Rightarrow p, r}{p \Rightarrow p \vee r}} \\
 \dfrac{\dfrac{\dfrac{\dfrac{q, r \Rightarrow p, \underline{q}}{q, r \Rightarrow p \vee q} (\vee_r)}{q, r \Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_r)}{q \wedge r \Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_l)}{p \vee (q \wedge r) \Rightarrow (p \vee q) \wedge (p \vee r)} (\vee_l) \\
 \dfrac{\dfrac{\dfrac{\dfrac{q, r \Rightarrow p, \underline{r}}{q, r \Rightarrow p \vee r} (\vee_r)}{q, r \Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_r)}{q \wedge r \Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_l)}{p \vee (q \wedge r) \Rightarrow (p \vee q) \wedge (p \vee r)} (\rightarrow_r)
 \end{array}$$

Using the anti-sequent calculus:

There are four possible linear reduction processes for the initial anti-sequent $\Rightarrow U$, caused by four possible combinations of the application of $(\vee_{l_1}^c \text{ or } \vee_{l_2}^c)$ and $(\wedge_{r_1}^c \text{ or } \wedge_{r_2}^c)$ rules in the second and the third steps.

Process 1:

$$\frac{\frac{\frac{p \not\Rightarrow p, q}{p \not\Rightarrow p \vee q} (\vee_r^c)}{p \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_{r_1}^c)}{p \vee (q \wedge r) \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\rightarrow_r^c)$$

Process 2:

$$\frac{\frac{p \not\Rightarrow p, r}{p \not\Rightarrow p \vee r} (\vee_r^c) \quad \frac{p \not\Rightarrow (p \vee q) \wedge (p \vee r)}{p \vee (q \wedge r) \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_{r_2}^c) (\vee_{l_1}^c)}{\not\Rightarrow p \vee (q \wedge r) \rightarrow (p \vee q) \wedge (p \vee r)} (\rightarrow_r^c)$$

Process 3:

$$\frac{\frac{\frac{q, r \not\Rightarrow p, q}{q \wedge r \not\Rightarrow p \vee q} (\vee_r^c, \wedge_l^c)}{q \wedge r \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_r^c)}{p \vee (q \wedge r) \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\rightarrow_r^c)$$

Process 4:

$$\frac{\frac{q, r \not\Rightarrow p, r}{q \wedge r \not\Rightarrow p \vee r} (\vee_r^c, \wedge_l^c) \quad \frac{}{q \wedge r \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\wedge_{r_2}^c)}{p \vee (q \wedge r) \not\Rightarrow (p \vee q) \wedge (p \vee r)} (\vee_{l_2}^c) \quad \not\Rightarrow p \vee (q \wedge r) \rightarrow (p \vee q) \wedge (p \vee r) (\rightarrow_r^c)}$$

Sequent and Anti-Sequent Calculi

The anti-sequent $\not\Rightarrow U$ is not true because it was not reduced to a basic anti-sequent in none of the above four linear reduction processes, therefore U is a valid formula.

Example 4.12.

Apply the sequent calculus and anti-sequent calculus to prove the non-validity of the predicate formula:

$$U = (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall x)P(x) \vee (\forall x)Q(x).$$

For both methods we need to rename the bound variables such that they are distinct in the formula:

$$U \equiv (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall y)P(y) \vee (\forall z)Q(z).$$

Using the **sequent calculus**:

The initial sequent: $\Rightarrow (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall y)P(y) \vee (\forall z)Q(z)$ is reduced according to the following up-side down binary tree.

We use the notation: $A = (\forall x)(P(x) \vee Q(x))$.

$$\begin{array}{c}
 \dots \\
 \dfrac{\underline{Q(y), Q(z)}, A \Rightarrow P(y), \underline{Q(z)} \quad Q(y), P(z), A \Rightarrow P(y), Q(z)}{Q(y), P(z) \vee Q(z), A \Rightarrow P(y), Q(z)} (\vee_l) \\
 \dots \quad \dfrac{Q(y), P(z) \vee Q(z), A \Rightarrow P(y), Q(z)}{P(y), A \Rightarrow P(y), Q(z) \quad Q(y), (\forall x)(P(x) \vee Q(x)) \Rightarrow P(y), Q(z)} (\forall_l)[x \leftarrow z] \\
 \dfrac{P(y), A \Rightarrow P(y), Q(z) \quad Q(y), (\forall x)(P(x) \vee Q(x)) \Rightarrow P(y), Q(z)}{P(y) \vee Q(y), A \Rightarrow P(y), Q(z)} (\vee_l) \\
 \dfrac{P(y) \vee Q(y), A \Rightarrow P(y), Q(z)}{(\forall x)(P(x) \vee Q(x)) \Rightarrow P(y), Q(z)} (\forall_l)[x \leftarrow y] \\
 \dfrac{(\forall x)(P(x) \vee Q(x)) \Rightarrow P(y), Q(z)}{(\forall x)(P(x) \vee Q(x)) \Rightarrow (\forall y)P(y), (\forall z)Q(z)} (\forall_r) \\
 \dfrac{(\forall x)(P(x) \vee Q(x)) \Rightarrow (\forall y)P(y), (\forall z)Q(z)}{(\forall x)(P(x) \vee Q(x)) \Rightarrow (\forall y)P(y) \vee (\forall z)Q(z)} (\rightarrow_r) \\
 \Rightarrow (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall y)P(y) \vee (\forall z)Q(z)
 \end{array}$$

There are two leaf nodes that contain basic sequents (overlined) and one leaf node containing the sequent $\underline{Q(y), P(z)} \Rightarrow P(y), Q(z), A$. This sequent cannot be reduced any more (all the appropriate instantiations ($[x \leftarrow y], [x \leftarrow z]$) for the universal formula $(\forall x)(P(x) \vee Q(x))$ were done).

The sequent $\Rightarrow U$ cannot be reduced to basic sequents, therefore U is not a valid formula.

Using the **anti-sequent calculus**:

The reduction tree of the initial anti-sequent:

$\not\Rightarrow (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall y)P(y) \vee (\forall z)Q(z)$ is depicted below.

A Computational Approach to Classical Logics and Circuits

$$\begin{array}{c}
 \frac{\{P(t) \vee Q(t) \Rightarrow P(y), Q(z) \mid t - \text{term of the language}\}}{(\forall_i^c)} \\
 \frac{(\forall x)(P(x) \vee Q(x)) \Rightarrow P(y), Q(z)}{(\forall_r^c)} \\
 \frac{(\forall x)(P(x) \vee Q(x)) \Rightarrow (\forall y)P(y), (\forall y)Q(z)}{(\vee_r^c)} \\
 \frac{(\forall x)(P(x) \vee Q(x)) \Rightarrow (\forall y)P(y) \vee (\forall y)Q(z)}{(\rightarrow_r^c)} \\
 \Rightarrow (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall y)P(y) \vee (\forall z)Q(z)
 \end{array}$$

There is an infinite number of premises and we have to prove that all of them can be reduced to basic anti-sequents.

The anti-sequent $P(t) \vee Q(t) \Rightarrow P(y), Q(z)$ is true if either $P(t) \Rightarrow P(y), Q(z)$ or $Q(t) \Rightarrow P(y), Q(z)$ are true anti-sequents.

If $t \neq y$, $S_1 : P(t) \Rightarrow P(y), Q(z)$ is a basic anti-sequent.

If $t = y$ the anti-sequent $Q(t) \Rightarrow P(y), Q(z)$ becomes:

$S_2 : Q(y) \Rightarrow P(y), Q(z)$, which is a basic anti-sequent.

We have proved that all the premises are true anti-sequents, therefore the initial anti-sequent ($\Rightarrow U$): $\Rightarrow (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall y)P(y) \vee (\forall z)Q(z)$ is true and thus *the formula U is not valid*.

From the basic anti-sequents S_1 and S_2 we can obtain an infinite number of anti-models of U . Such a generic anti-model is:

$I = < D, m >$ with

$\{y, z\} \subseteq D$ - the domain of interpretation, $|D| \geq 2$ and

$m(P) : D \rightarrow \{T, F\}$, $m(Q) : D \rightarrow \{T, F\}$

- $m(P)(y) = F$ (the atoms from the consequent of S_1 and S_2 must be evaluated as *false*)
- $m(P)(t) = T, \forall t \in D \setminus \{y\}$ (the atom from the antecedent of $S_1 : P(t), t \neq y$, must be evaluated as *true*)
- $m(Q)(z) = F$ (the atoms from the consequent of S_1 and S_2 must be evaluated as *false*)
- $m(Q)(y) = T$ (the atom from the antecedent of $S_2 : Q(y)$, must be evaluated as *true*)
- for $m(Q)(t), \forall t \in D \setminus \{y, z\}$, there is no restriction for the truth value assigned to it.

Note that an anti-model of U must have at least 2 distinct elements in the domain of interpretation.

Sequent and Anti-Sequent Calculi

Let us consider the interpretation $I_1 = \langle D_1, m_1 \rangle$, where:

$D_1 = \{9, 4\}$ is the domain of interpretation;

$m_1(P) : \{9, 4\} \rightarrow \{T, F\}, m_1(P)(x) : "x:2"; m_1(P)(9) = F, m_1(P)(4) = T$

$m_1(Q) : \{9, 4\} \rightarrow \{T, F\}, m_1(Q)(x) : "x:3"; m_1(Q)(9) = T, m_1(Q)(4) = F$

For evaluating the formula U under the interpretation I_1 , with the finite domain $D_1 = \{9, 4\}$, the universally quantified subformulas are replaced by the conjunction of their instances for $x = 4$ and $x = 9$.

$$\begin{aligned}\nu^{I_1}(U) &= \nu^{I_1}((\forall x)(P(x) \vee Q(x))) \rightarrow \nu^{I_1}((\forall x)P(x) \vee (\forall x)Q(x)) = \\ &= \nu^{I_1}((\forall x)(P(x) \vee Q(x))) \rightarrow \nu^{I_1}((\forall x)P(x)) \vee \nu^{I_1}((\forall x)Q(x)) = \\ &= (4:2 \vee 4:3) \wedge (9:2 \vee 9:3) \rightarrow (4:2 \wedge 9:2) \vee (4:3 \wedge 9:3) = \\ &= (T \vee F) \wedge (F \vee T) \rightarrow (T \wedge F) \vee (F \wedge T) = T \wedge T \rightarrow F \vee F = T \rightarrow F = F\end{aligned}$$

I_1 evaluates the formula U as false, so I_1 is an anti-model of U and thus U is not a valid formula.

Conclusions:

- Both sequent and anti-sequent proof methods can be used to check the validity/derivability and non-validity/non-derivability in classical logics.
- It is more efficient to check validity/derivability using sequent calculus and non-validity/non-derivability using anti-sequent calculus.
- The basic anti-sequents or the non-basic sequents from the leaf nodes of the reduction tree provide anti-models of the initial formula or derivation (deduction).
- The semantic tableau method and the sequent calculus method are dual proof methods:

Semantic tableaux method	Sequent calculus method
- semantic and refutation method	- syntactic and direct method
- graphical representation: a binary tree	- graphical representation: an up-side down binary tree
- models are built	- anti-models are built
- a closed branch represents inconsistency	- a closed branch represents validity

A Computational Approach to Classical Logics and Circuits

4.3. Exercises

Exercise 4.1.

Using the sequent calculus method prove the validity of the following formulas:

1. the left-distributivity of ' \rightarrow ' over ' \wedge ':

$$U_1 = (p \rightarrow q \wedge r) \leftrightarrow (p \rightarrow q) \wedge (p \rightarrow r);$$

2. the "cut" law:

$$U_2 = (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r);$$

3. the reunion and separation of the premises law:

$$U_3 = (p \rightarrow (q \rightarrow r)) \leftrightarrow (p \wedge q \rightarrow r);$$

4. the left-semi-distributivity of ' \wedge ' over ' \rightarrow ':

$$U_4 = p \wedge (q \rightarrow r) \rightarrow ((p \wedge q) \rightarrow (p \wedge r));$$

5. the permutation of the premises law:

$$U_5 = (p \rightarrow (q \rightarrow r)) \leftrightarrow (q \rightarrow (p \rightarrow r));$$

6. the left-distributivity of ' \rightarrow ' over ' \vee ':

$$U_6 = (p \rightarrow q \vee r) \leftrightarrow (p \rightarrow q) \vee (p \rightarrow r);$$

7. the left-distributivity of ' \vee ' over ' \rightarrow ':

$$U_7 = p \vee (q \rightarrow r) \leftrightarrow (p \vee q) \rightarrow (p \vee r);$$

8. the third axiom (A_3) of propositional logic:

$$U_8 = (p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p).$$

Exercise 4.2.

Using the sequent/anti-sequent calculus method determine if the following relations hold or not.

1. $\neg p \rightarrow (\neg q \rightarrow r) \models (\neg p \rightarrow q) \vee r;$

2. $p \vee (q \rightarrow r), q \wedge r \models p \rightarrow (q \rightarrow r);$

3. $p \wedge (q \rightarrow r), q \vee r \models p \rightarrow (q \rightarrow r);$

4. $p \rightarrow r, q \rightarrow t, p \wedge q \models r \wedge t;$

5. $p \rightarrow (q \vee r \wedge s), p, \neg r \models q \vee r;$

6. $p \rightarrow (\neg q \vee r \wedge s), p, \neg s \models \neg q \vee s;$

7. $p \rightarrow q, r \rightarrow t, p \wedge r \models q \wedge t;$

8. $p \rightarrow (q \vee r \wedge s), p, \neg r \models q \wedge p.$

Exercise 4.3.

Prove the non-validity of the following formulas using the sequent/anti-sequent calculus. Find an anti-model for each of the following formulas:

1. $U_1 = (p \wedge q \rightarrow r) \wedge \neg(p \rightarrow (q \rightarrow r));$

2. $U_2 = (p \rightarrow (q \rightarrow r)) \wedge \neg(p \rightarrow (q \rightarrow r));$

Sequent and Anti-Sequent Calculi

3. $U_3 = (p \rightarrow (q \rightarrow r)) \wedge p \wedge q \wedge \neg r ;$
4. $U_4 = (p \rightarrow r) \wedge \neg((\neg p \rightarrow q) \rightarrow (\neg q \rightarrow r)) ;$
5. $U_5 = (p \rightarrow (q \vee r)) \wedge \neg(p \rightarrow q) \wedge \neg(p \rightarrow r) ;$
6. $U_6 = (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \wedge (p \wedge \neg r) ;$
7. $U_7 = (\neg p \vee q) \wedge \neg(\neg q \rightarrow \neg p) ;$
8. $U_8 = (p \rightarrow (q \rightarrow r)) \wedge \neg((p \rightarrow q) \rightarrow (p \rightarrow r)) .$

Exercise 4.4.

Using the anti-sequent calculus method prove that the following deductions do not hold. Find an anti-model for each of them.

1. $p \wedge r \vee \neg p \wedge \neg r \vdash q \leftrightarrow r ;$
2. $q \wedge r \rightarrow p \vdash (p \rightarrow r) \wedge q ;$
3. $r \wedge q \vee \neg p \vee \neg r \vdash p \leftrightarrow q ;$
4. $r \vee q \vee (p \rightarrow \neg r) \vdash p \leftrightarrow q ;$
5. $q \vee r \rightarrow p \vdash (p \rightarrow r) \wedge q ;$
6. $p \wedge q \rightarrow r \vdash (p \rightarrow r) \wedge q ;$
7. $p \vee q \rightarrow r \vdash (p \vee r) \rightarrow q ;$
8. $p \wedge q \vee \neg p \wedge \neg r \vdash q \leftrightarrow r .$

Exercise 4.5.

Prove the following properties using the sequent calculus method.

1. ‘ \forall ’ is semi-distributive over ‘ \vee ’:
 $\models (\forall x)P(x) \vee (\forall x)Q(x) \rightarrow (\forall x)(P(x) \vee Q(x))$ and
 $\not\models (\forall x)(P(x) \vee Q(x)) \rightarrow (\forall x)P(x) \vee (\forall x)Q(x)$
2. ‘ \forall ’ is distributive over ‘ \wedge ’
 $\models (\forall x)(P(x) \wedge Q(x)) \leftrightarrow (\forall x)P(x) \wedge (\forall x)Q(x)$
3. ‘ \exists ’ is semi-distributive over ‘ \wedge ’:
 $\models (\exists x)(P(x) \wedge Q(x)) \rightarrow (\exists x)P(x) \wedge (\exists x)Q(x)$ and
 $\not\models (\exists x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)(P(x) \wedge Q(x))$
4. ‘ \forall ’ is semi-distributive over ‘ \rightarrow ’:
 $\models (\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x))$ and
 $\not\models ((\forall x)P(x) \rightarrow (\forall x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x))$
5. ‘ \exists ’ is semi-distributive over ‘ \rightarrow ’:
 $\models ((\exists x)P(x) \rightarrow (\exists x)Q(x)) \rightarrow (\exists x)(P(x) \rightarrow Q(x))$ and
 $\not\models (\exists x)(P(x) \rightarrow Q(x)) \rightarrow ((\exists x)P(x) \rightarrow (\exists x)Q(x))$
6. ‘ \forall ’ is distributive over ‘ \wedge ’:
 $\models (\forall x)P(x) \wedge (\forall x)Q(x) \leftrightarrow (\forall x)(P(x) \wedge Q(x))$

A Computational Approach to Classical Logics and Circuits

7. $\models (\exists x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\exists x)Q(x))$ and
 $\not\models ((\exists x)P(x) \rightarrow (\exists x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x))$
8. $\models (\exists x)(P(x) \wedge Q(x)) \rightarrow (\exists x)(P(x) \rightarrow Q(x))$
 $\not\models (\exists x)(P(x) \wedge Q(x)) \rightarrow (\forall x)P(x) \vee (\forall x)Q(x)$

Exercise 4.6.

Using the sequent/anti-sequent calculus methods determine if the following deductions hold or not.

1. $P(a), (\forall x)(P(x) \rightarrow P(f(x))) \vdash (\forall x)P(x);$
2. $(\forall x)(P(x) \rightarrow Q(x)), (\forall x)P(x) \vdash (\forall x)Q(x);$
3. $(\forall x)(\forall y)(Q(x, y) \rightarrow P(x, y)), (\forall z)Q(z, z) \vdash (\forall x)P(x, x);$
4. $(\exists x)(\forall y)(P(x, y) \rightarrow R(x)), (\forall x)(\forall y)P(x, y) \vdash (\exists z)R(z);$
5. $(\forall x)(P(x) \rightarrow Q(x)), (\exists x)P(x) \vdash (\exists x)Q(x);$
6. $(\exists x)(\forall y)(Q(x, y) \rightarrow P(x, y)), (\forall z)Q(z, z) \vdash (\exists x)P(x, x);$
7. $(\forall x)(\forall y)(P(x, y) \rightarrow R(x)), (\exists x)(\exists y)P(x, y) \vdash (\exists z)R(z);$
8. $(\forall x)(\forall y)(P(x, y) \rightarrow P(y, x)) \vdash (\forall x)P(x, x).$

5. RESOLUTION PROOF METHOD

In 1965 J.A. Robinson proposed resolution [50], an efficient and easily to be implemented proof method. The primary purpose of the resolution method is to check the consistency/inconsistency of a set of clauses. Resolution is used as a refutation method (proof by contradiction) to solve the decision problems of classical logics. This method was successfully adapted for non-classical logics (multivalent logics, modal logics, temporal logics and non-monotonic logics). The papers [1, 2, 12, 16, 18, 20, 21, 31, 32, 43, 45, 47, 52, 59, 60, 61, 62, 63] were used as bibliographic references.

5.1. Resolution method for propositional logic

The resolution, as a syntactic proof method, is introduced by the *formal (axiomatic) system*: $\text{Res} = (\Sigma_{\text{Res}}, F_{\text{Res}}, A_{\text{Res}}, R_{\text{Res}})$ as was proposed in paper [62]:

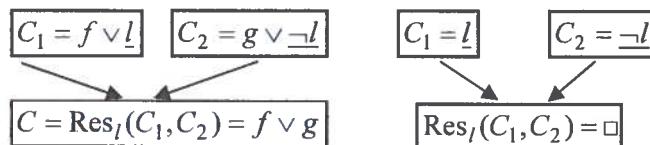
- $\Sigma_{\text{Res}} = \Sigma_P - \{\rightarrow, \leftrightarrow, \wedge\}$ is the alphabet;
- $F_{\text{Res}} \cup \{\square\}$ is the set of well formed formulas;
 - F_{Res} is the set of all clauses built using the alphabet Σ_{Res} ;
 - \square is the empty clause which does not contain any literal and it symbolizes inconsistency;
- $A_{\text{Res}} = \emptyset$ is the set of axioms ;
- $R_{\text{Res}} = \{res\}$ is the set of inference rules containing the *resolution rule (res)*:
 $f \vee l, g \vee \neg l \vdash_{\text{res}} f \vee g$, where l is a literal and $f, g \in F_{\text{Res}}$.

Definition 5.1.

The clauses $C_1 = f \vee l$ and $C_2 = g \vee \neg l$ are called *clashing clauses* and they *resolve upon the literal l*. We use the notation: $C = \text{Res}_l(C_1, C_2) = f \vee g$, where C is called the *resolvent* of the *parent clauses* C_1 and C_2 .

In the particular case: $C_1 = l$ and $C_2 = \neg l$, $\text{Res}_l(C_1, C_2) = \square$ (empty clause) which is inconsistent.

Graphical representation:



A Computational Approach to Classical Logics and Circuits

Resolution as an inference rule is a generalization of the rules: *modus ponens*, *modus tollens* and the *syllogism rule*.

Modus ponens rule: $p, p \rightarrow q \vdash_{mp} q$ can be written in the equivalent form:

$p, \neg p \vee q \vdash_{mp} q$, which is a particularization of the resolution rule.

Modus tollens rule (A_3 axiom):

$\vdash (p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$, apply the reverse of the theorem of deduction

$p \rightarrow q \vdash \neg q \rightarrow \neg p$, apply the reverse of the theorem of deduction

$p \rightarrow q, \neg q \vdash \neg p$, the implication is written as a disjunction

$\neg p \vee q, \neg q \vdash \neg p$ ---- a particularization of the resolution rule.

Syllogism rule:

$\vdash (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$, apply the reverse of the theorem of

deduction

$p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)$, apply the reverse of the theorem of deduction

$p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$, the implications are written as disjunctions

$\neg p \vee q, \neg q \vee r \vdash \neg p \vee r$ ---- a particularization of the resolution rule

The resolution method consists of successive applications of the resolution rule upon an initial set of clauses, enriched in this process with new clauses (resolvents), in order to derive the empty clause. If the empty clause is derived, then the initial set of clauses is consistent, otherwise the initial set of clauses is inconsistent.

Algorithm propositional_resolution:

input: S – a set of propositional clauses

output: message “ S is consistent” or “ S is inconsistent”

begin

$S_0 := S$; $i := 0$;

do

 choose two clashing clauses: $C_1, C_2 \in S_i$;

$C := \text{Res}(C_1, C_2)$; $S_{i+1} := S_i \cup \{C\}$;

if ($C = \square$) **then** write “ S is inconsistent”; **exit**;

else $i := i + 1$;

end_if

until ($S_i = S_{i-1}$) // no new clauses can be derived

write “ S is consistent”;

end

Resolution Proof Method

Theorem 5.1. Soundness theorem [50]

If the empty clause is derived from the set S of propositional clauses using the resolution algorithm ($S \vdash_{\text{Res}} \square$), then S is an inconsistent set.

Theorem 5.2. Completeness theorem [50]

If the set S of propositional clauses is inconsistent, then the empty clause can be derived from S using the resolution algorithm ($S \vdash_{\text{Res}} \square$).

Theorem 5.3. Soundness and completeness theorem [50]

A set S of propositional clauses is inconsistent if and only if $S \vdash_{\text{Res}} \square$.

The resolution method is used as a refutation proof method (proof by contradiction) to solve the decision problems in propositional logic, according to the following theorems.

Theorem 5.4.

A propositional formula U is a tautology (theorem) if and only if the empty clause can be derived from the conjunctive normal form of $\neg U$, using the resolution algorithm.

U is a tautology (theorem) if and only if $\text{CNF}(\neg U) \vdash_{\text{Res}} \square$.

Theorem 5.5.

Let U_1, U_2, \dots, U_n, V be propositional formulas.

$U_1, U_2, \dots, U_n \models V$ if and only if $U_1, U_2, \dots, U_n \vdash V$ if and only if $\text{CNF}(U_1 \wedge U_2 \wedge \dots \wedge U_n \wedge \neg V) \vdash_{\text{Res}} \square$.

Example 5.1.

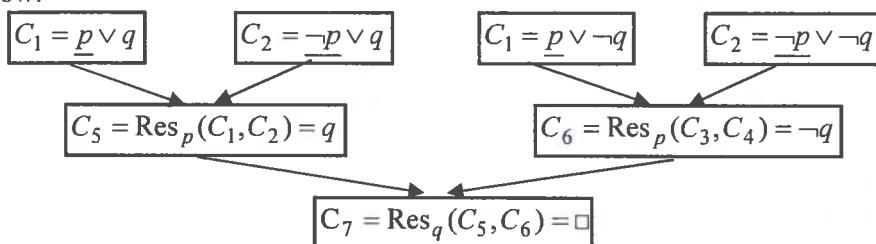
Using the resolution method prove that the following set of clauses is inconsistent.

$$S = \{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$$

We denote the propositional clauses as follows:

$$C_1 = p \vee q, C_2 = \neg p \vee q, C_3 = p \vee \neg q, C_4 = \neg p \vee \neg q.$$

The process of deriving the empty clause is symbolized by the binary tree depicted below:



$S \vdash_{\text{Res}} \square$ and according to the soundness theorem we conclude that the set S of propositional clauses is inconsistent.

A Computational Approach to Classical Logics and Circuits

The following theorem, inspired from Davis-Putman procedure [15], introduces a set of transformations used to simplify an initial set of clauses, preserving its consistency/inconsistency.

Theorem 5.6.

A set S of propositional clauses can be simplified, preserving its consistency/inconsistency, by applying the following transformations:

1. *Delete the clauses that are tautologies* (they contain at least one pair of opposite literals).

example: $\neg p \vee \neg q \vee r \vee p$ is a tautology because $\neg p$, p are opposite literals

2. *Delete the clauses subsumed by other clauses of S .*

C_1 is subsumed by C_2 or C_2 subsumes C_1 if there exists a clause C_3 such that $C_1 = C_2 \vee C_3$.

example: $C_1 = t \vee \neg q \vee r \vee p$, $C_2 = r \vee p$,

C_1 is subsumed by C_2 , C_2 subsumes C_1 and $C_3 = t \vee \neg q$

This transformation is based on the absorption property:

$$C_2 \wedge C_1 = C_2 \wedge (C_2 \vee C_3) \equiv C_2,$$

C_1 will be deleted from the set of clauses containing both C_1 and C_2 .

3. *Delete every clause which contains a pure literal.*

A **pure literal** is a literal that appears in a clause of S , but its negation does not appear in any clause of S .

example: $S = \{C_1 = \neg p \vee \neg q \vee r, C_2 = r \vee p, C_3 = \neg p \vee \neg r\}$

$\neg q$ is a pure literal in S and C_1 will be deleted.

4. *Let $C = l$ be a unit clause of S . Delete every clause containing l and delete $\neg l$ from every remaining clause.*

example: $S = \{C_1 = \neg p \vee \neg q \vee r, C_2 = r \vee p, C_3 = \neg p \vee \neg r, C_4 = q\}$

C_4 is a unit clause and is eliminated, the literal $\neg q$ is deleted from C_1 ,

$S' = \{C_1' = \neg p \vee r, C_2 = r \vee p, C_3 = \neg p \vee \neg r\}$ is the simplified set.

Remarks:

- If \emptyset is obtained after applying all the above transformations on a set S of clauses, then S is a consistent set.
- S is an inconsistent set of clauses if the empty clause (\square) is obtained after applying transformation 4.

Example 5.2.

Using the previous transformations, simplify the following set of propositional clauses: $S = \{C_1 = \neg p \vee p \vee r, C_2 = r \vee \neg r, C_3 = p \vee q, C_4 = p \vee q \vee r\}$.

Resolution Proof Method

- C_4 is subsumed by C_3 so C_4 will be eliminated.
- C_1 and C_2 are tautologies and they will be eliminated.
- only C_3 remains but it contains two pure literals: p and q and C_3 will be eliminated too.

After we applied the transformations, S becomes $S' = \emptyset$ and thus S is consistent.

Example 5.3.

Simplify the set $S = \{C_1 = \neg q, C_2 = q, C_3 = p \vee q \vee r\}$ of propositional clauses.

- C_3 is subsumed by C_2 and it is eliminated.
- If we apply transformation 4, C_2 is a unit clause, C_2 is eliminated and C_1 becomes $C_1' = \square$ after the literal $\neg q$ is deleted.

After simplification, S becomes $S' = \{\square\}$ and we conclude that S is an inconsistent set.

5.2. Strategies for propositional resolution

The strategies used in the resolution process assure that all the possible clauses to be derived are generated and they also try to avoid the derivation of redundant and irrelevant clauses in order to obtain the empty clause.

1. The **level saturation strategy** [52] generates levels of resolvents corresponding to the exploration of the whole search space which contains all the possible resolvents:
 - the initial level contains the initial set of clauses;
 - we compute all the resolvents on a level (a resolvent has as one parent a clause from the level generated in the last iteration and the other parent belongs to one of the previous levels), we add them to the current level, and then we compute the next level.
 - we continue until we obtain the empty clause or the last level is empty (no more resolvents can be derived).

Algorithm level-saturation-strategy:

input: S - the initial set of clauses;

output: message “ S is inconsistent” or “ S is consistent”

begin

// we generate the sequence S^0, S^1, \dots, S^k containing sets of clauses (levels)

$S^0 := S; k := 0$; // initial level

do

{ $k := k + 1$;

A Computational Approach to Classical Logics and Circuits

```

 $S^k := \{\text{Res}(C_1, C_2) \mid C_1 \in S^0 \cup S^1 \cup \dots \cup S^{k-1}, C_2 \in S^{k-1}\};$ 
// we eliminate the resolvents obtained in the current level but
// which appear already in the previous levels
 $S^k := S^k \setminus (S^0 \cup S^1 \cup \dots \cup S^{k-1});$ 
} until ( $\square \in S^k$  or  $S^k = \emptyset$ ).
if ( $\square \in S^k$ ) then write "S - inconsistent"
else // no more resolvents can be derived and  $\square \notin S^k$ 
    write "S - consistent" // no more resolvents can be derived
end_if
end

```

2. The **deletion strategy** [15]: the resolvents that are tautologies or are subsumed by other clauses in the set S of clauses are eliminated and they will not be used further in the resolution process because they produce redundant clauses.

Example 5.4.

Using the level saturation strategy combined with the deletion strategy check the consistency/inconsistency of the set $S = \{p \vee q, \neg p \vee q \vee \neg r, \neg q \vee r\}$ of clauses.

We generate the sequence S^0, S^1, S^2, \dots representing the levels of resolvents:

$$S^0 = S = \{C_1 = p \vee q, C_2 = \neg p \vee q \vee \neg r, C_3 = \neg q \vee r\} \text{ - initial set of clauses}$$

The first level of resolvents is generated:

$$C_4 = \text{Res}_p(C_1, C_2) = q \vee \neg r$$

$$C_5 = \text{Res}_q(C_1, C_3) = p \vee r$$

$$C_6 = \text{Res}_q(C_2, C_3) = \neg p \vee \neg r \vee r \text{ - tautology}$$

$$C_7 = \text{Res}_r(C_2, C_3) = \neg p \vee q \vee \neg q \text{ - tautology}$$

C_6 and C_7 will not be included in the first level.

$$S^1 = \{C_4 = q \vee \neg r, C_5 = p \vee r\}$$

The second level of resolvents is generated:

$$C_8 = \text{Res}_r(C_4, C_5) = q \vee p = C_1$$

$$C_9 = \text{Res}_q(C_4, C_3) = r \vee \neg r \text{ - tautology}$$

$$C_{10} = \text{Res}_r(C_4, C_3) = q \vee \neg q \text{ - tautology}$$

$$C_{11} = \text{Res}_p(C_5, C_2) = q \vee r \vee \neg r \text{ - tautology}$$

$$C_{12} = \text{Res}_r(C_5, C_2) = \neg p \vee p \vee q \text{ - tautology}$$

Resolution Proof Method

The clauses $C_9, C_{10}, C_{11}, C_{12}$, are tautologies, and C_8 belongs to the initial set of clauses as C_1 , so they will not be introduced in the second level.

$S^2 = \emptyset$ with the meaning that no more resolvents can be generated.

The empty clause cannot be derived from S and thus S is a consistent set.

3. *The set-of-support strategy* [65] avoids resolving two clauses belonging to a consistent subset of the initial set of clauses, because the resolvents derived from a consistent set are irrelevant in the process of deriving \square .

This strategy was inspired by the fact that usually, the set of premises (hypotheses, facts) of a deduction is consistent, and resolving two clauses from this set will not help us to derive the empty clause (inconsistency).

Definition 5.2. [62]

Let S be a set of clauses. A subset Y of S is called the *support set of S* , if the set $S \setminus Y$ is consistent. The *set-of-support resolution* is the resolution of two clauses that are not both from the set $S \setminus Y$. The resolvents generated during the resolution process are added to Y .

Example 5.5.

Using the set-of-support strategy, prove the following deduction:

$$p \rightarrow (q \rightarrow r), r \wedge s \rightarrow t, u \rightarrow s \wedge \neg t \vdash p \wedge q \rightarrow \neg u.$$

We denote the hypotheses and the conclusion of the deduction as follows:

$$U_1 = p \rightarrow (q \rightarrow r),$$

$$U_2 = r \wedge s \rightarrow t,$$

$$U_3 = u \rightarrow s \wedge \neg t, \quad V = p \wedge q \rightarrow \neg u$$

Applying the normalization algorithm we transform the above formulas into their conjunctive normal forms.

$$\text{CNF}(U_1) = \neg p \vee \neg q \vee r, \quad C_1 = \neg p \vee \neg q \vee r$$

$$\text{CNF}(U_2) = \neg r \vee \neg s \vee t, \quad C_2 = \neg r \vee \neg s \vee t$$

$$\text{CNF}(U_3) = \neg u \vee (s \wedge \neg t) = (\neg u \vee s) \wedge (\neg u \vee \neg t), \text{ provides the clauses:}$$

$$C_3 = \neg u \vee s \text{ and } C_4 = \neg u \vee \neg t$$

$$\text{CNF}(\neg V) = p \wedge q \wedge r \text{ provides the clauses: } C_5 = p, \quad C_6 = q, \quad C_7 = u$$

The set-of-support strategy is applied to the set of clauses:

$$S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$$

We choose $Y = \{C_5, C_6, C_7\}$ as a support set of S , corresponding to $\neg V$ (the negation of the conclusion of the deduction). $S \setminus Y = \{C_1, C_2, C_3, C_4\}$ is a consistent set corresponding to the premises of the deduction.

A Computational Approach to Classical Logics and Circuits

This strategy avoids to resolve two clauses belonging to the consistent set $S \setminus Y$. C_1 and C_2 can resolve upon the literal r , but they belong to $S \setminus Y$, so in this strategy we will not resolve them.

Using the set-of-support strategy, we derive the empty clause from the set of clauses:

$$S = \{ C_1 = \neg p \vee \neg q \vee r, C_2 = \neg r \vee \neg s \vee t, C_3 = \neg u \vee s, C_4 = \neg u \vee \neg t, C_5 = p, \\ C_6 = q, C_7 = u \}.$$

The resolvents are as follows:

$$C_8 = \text{Res}_p(C_1, C_5) = \neg q \vee r$$

$$C_9 = \text{Res}_q(C_8, C_6) = r$$

$$C_{10} = \text{Res}_r(C_9, C_2) = \neg s \vee t$$

$$C_{11} = \text{Res}_u(C_7, C_3) = s$$

$$C_{12} = \text{Res}_u(C_7, C_4) = \neg t$$

$$C_{13} = \text{Res}_s(C_{10}, C_{11}) = t$$

$$C_{14} = \text{Res}_s(C_{13}, C_{12}) = \square.$$

We conclude that S is an inconsistent set and *the deduction holds*.

In order to make the resolution process more efficient, the **refinements of resolution** (*lock resolution, linear resolution, semantic resolution*) impose restrictions on the clashing clauses.

All the refinements and strategies of resolution preserve the soundness and the completeness properties. The question is: are these properties preserved when we combine refinements and strategies?

All the combinations of these refinements and strategies preserve the soundness property. The completeness is not preserved in some combinations.

Incompleteness: the initial set of clauses is inconsistent, but the empty clause cannot be derived because there are too many restrictions imposed by those strategies and refinements.

general resolution + deletion strategy is sound and complete

general resolution + set-of-support strategy is sound and complete

general resolution + deletion strategy + set-of-support strategy = sound and complete

5.3. Lock resolution

This refinement of resolution, introduced by R.S. Boyer in 1971 [10], is very efficient and easily to be implemented.

- Each occurrence of a literal from a set of clauses is arbitrarily indexed with an integer.
- **Restriction:** the literals resolved upon must have the lowest indices in their clauses.
- The literals from resolvents inherit the indices from their parent clauses. If the parent clauses have a common literal, in the resolvent this literal will have the lowest index from the inherited indices.
- At the implementation level we must combine lock resolution with the level saturation strategy in order to check all the possible ways of deriving \square .

Theorem 5.7. Soundness [10]

Let S be a set of clauses having each literal *arbitrarily* indexed with an integer. If from S the empty clause can be derived using lock resolution ($S \vdash_{\text{Res}}^{\text{lock}} \square$), then S is inconsistent.

Theorem 5.8. Completeness [10]

Let S be a set of clauses having each literal *arbitrarily* indexed with an integer. If S is inconsistent, then there is a lock derivation of the empty clause from S ($S \vdash_{\text{Res}}^{\text{lock}} \square$).

The *completeness* property is not preserved if lock resolution is combined with the deletion strategy or with the set-of-support strategy.

The level saturation strategy is a safe method that can be used to prove that a set of clauses is consistent, since it assures the fact that all the possible clauses to be derived are obtained. This process can be shortened by using the combination of level saturation strategy with lock resolution.

Remarks:

1. In order to prove the consistency of a set S of clauses, lock resolution must be combined with the level saturation strategy:

*if $S^k = \emptyset$ (the last level of lock resolvents is empty)
then the set S is consistent.*

2. In order to prove inconsistency we do not need to use a strategy. It is enough to find a lock derivation of \square to prove the inconsistency of a set S of clauses. If we combine lock resolution with the level saturation strategy the decision is as follows:

*if $\square \in S^k$ (\square was derived as a lock resolvent in the level k)
then the set S is inconsistent.*

A Computational Approach to Classical Logics and Circuits

Example 5.6.

Check the consistency/inconsistency of the set S of propositional clauses.

$$S = \{\neg r, p \vee \neg q, r \vee \neg p, \neg q\}$$

a) The literals from the clauses are indexed as follows:

$$C_1 =_{(1)} \neg r, C_2 =_{(3)} p \vee_{(2)} \neg q, C_3 =_{(4)} r \vee_{(5)} \neg p, C_4 =_{(6)} q.$$

We use lock resolution + the level saturation strategy.

$$S^0 = S$$

The first level of resolvents is obtained as follows:

$$S^1 = \{\text{Res}^{lock}(C_i, C_j) \mid C_i \in S^0, C_j \in S^0\}$$

$$C_5 = \text{Res}_r^{lock}(C_1, C_3) =_{(5)} \neg p$$

$$C_6 = \text{Res}_q^{lock}(C_2, C_4) =_{(3)} p$$

$$S^1 = \{C_5, C_6\}$$

We generate the second level of resolvents:

$$S^2 = \{\text{Res}^{lock}(C_i, C_j) \mid C_i \in S^1, C_j \in S^0 \cup S^1\}$$

$$C_7 = \text{Res}_p^{lock}(C_5, C_6) = \square$$

$$S^2 = \{C_7\}$$

$\square \in S^2$, the empty clause was derived from S , therefore S is an inconsistent set.

b) We use another indexing of the literals from clauses:

$$C_1 =_{(1)} \neg r, C_2 =_{(2)} p \vee_{(3)} \neg q, C_3 =_{(5)} r \vee_{(4)} \neg p, C_4 =_{(6)} q$$

and we do not apply the level saturation strategy.

The empty clause is derived from S as follows:

$$C_5 = \text{Res}_p^{lock}(C_2, C_3) =_{(3)} \neg q \vee_{(5)} r$$

$$C_6 = \text{Res}_q^{lock}(C_4, C_5) =_{(5)} r$$

$$C_7 = \text{Res}_r^{lock}(C_4, C_5) = \square.$$

Example 5.7.

Using lock resolution and the level saturation strategy check the consistency/inconsistency of the following set of clauses:

$$S = \{p \vee q, p \vee \neg q \vee \neg r, \neg p \vee r\}.$$

The literals from the clauses are indexed as follows:

$$C_1 =_{(1)} p \vee_{(2)} q, C_2 =_{(3)} p \vee_{(4)} \neg q \vee_{(5)} \neg r, C_3 =_{(7)} \neg p \vee_{(6)} r,$$

$$S^0 = S = \{C_1, C_2, C_3\} - \text{the initial set of clauses}$$

Resolution Proof Method

No lock resolvents can be generated ($S^1 = \emptyset$), the empty clause cannot be derived from S and we conclude that *the set S is consistent*.

Another indexing of literals of S will give the same result regarding consistency, but the resolution process will be longer, having more levels.

Let us consider the following indexing of the literals:

$$C_1 = {}_{(1)}p \vee {}_{(2)}q, \quad C_2 = {}_{(4)}p \vee {}_{(3)}\neg q \vee {}_{(5)}\neg r, \quad C_3 = {}_{(6)}\neg p \vee {}_{(7)}r,$$

$$S^0 = S = \{C_1, C_2, C_3\} \text{ - the initial set of clauses}$$

The first level of resolvents is generated: $S^1 = \{C_4\}$

$$C_4 = \text{Res}_p(C_1, C_3) = {}_{(2)}q \vee {}_{(7)}r$$

The second level of resolvents is generated: $S^2 = \{C_5\}$

$$C_5 = \text{Res}_q(C_4, C_2) = {}_{(4)}p \vee {}_{(5)}\neg r \vee {}_{(7)}r \equiv T \text{ (contains a pair of opposite literals)}$$

The third level of resolvents is generated: $S^3 = \{C_6\}$

$$C_6 = \text{Res}_p(C_5, C_3) = {}_{(5)}\neg r \vee {}_{(7)}r \equiv T$$

The fourth level of resolvents is the empty set, $S^4 = \emptyset$, since there is no clause where r is the literal with the lowest index in its clause.

The empty clause cannot be derived from S and thus S is a consistent set.

Example 5.8.

In this example we shall prove that *lock resolution + the deletion strategy is not complete*.

The set $S = \{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$ is inconsistent and we have the following *lock refutation* from S .

We index the literals of S as follows:

$$C_1 = {}_{(1)}p \vee {}_{(2)}q, \quad C_2 = {}_{(3)}\neg p \vee {}_{(4)}q, \quad C_3 = {}_{(5)}p \vee {}_{(6)}\neg q, \quad C_4 = {}_{(7)}\neg p \vee {}_{(8)}\neg q$$

The lock resolvents are obtained:

$$C_5 = \text{Res}_p^{lock}(C_1, C_2) = {}_{(2)}q,$$

$$C_6 = \text{Res}_p^{lock}(C_3, C_4) = {}_{(6)}\neg q,$$

$$C_7 = \text{Res}_p^{lock}(C_5, C_6) = \square$$

Another indexing of the literals of S is used and we combine lock resolution with the deletion strategy:

$$C_1 = {}_{(2)}p \vee {}_{(1)}q, \quad C_2 = {}_{(3)}\neg p \vee {}_{(4)}q, \quad C_3 = {}_{(5)}p \vee {}_{(6)}\neg q, \quad C_4 = {}_{(8)}\neg p \vee {}_{(7)}\neg q$$

Using the restriction imposed by lock resolution we can derive only the resolvents C_5 and C_6 which are tautologies:

A Computational Approach to Classical Logics and Circuits

$$C_5 = \text{Res}_p^{\text{lock}}(C_1, C_4) =_{(2)} p \vee_{(8)} \neg p \text{ and}$$

$$C_6 = \text{Res}_p^{\text{lock}}(C_2, C_3) =_{(4)} q \vee_{(6)} \neg q$$

If we apply also the deletion strategy, and we delete these tautologies, then we cannot continue the derivation process to obtain \square .

Note that even if S is inconsistent, the empty clause (\square) cannot be derived because there are too many restrictions imposed by lock resolution and the deletion strategy.

Therefore, *lock resolution + the deletion strategy is not complete*.

Continuing the lock resolution process but using also the resolvents C_5 and C_6 we obtain: $C_7 = \text{Res}_p^{\text{lock}}(C_2, C_5) =_{(4)} q \vee_{(8)} \neg p$

The clauses C_7 and C_2 are similar but not identical, the literals order (provided by the indices) differs in these clauses: $\neg p$ is the literal to resolve upon from C_2 , and q is the literal to resolve upon form C_7 . Thus, the role of a tautology as a parent clause is to modify the literals order in the other parent clause.

$$C_1 =_{(2)} p \vee_{(1)} q, C_2 =_{(3)} \neg p \vee_{(4)} q, C_3 =_{(5)} p \vee_{(6)} \neg q, C_4 =_{(8)} \neg p \vee_{(7)} \neg q$$

$$C_5 =_{(2)} p \vee_{(8)} \neg p, C_6 =_{(4)} q \vee_{(6)} \neg q, C_7 =_{(4)} q \vee_{(8)} \neg p$$

$$C_8 = \text{Res}_q^{\text{lock}}(C_7, C_4) =_{(8)} \neg p,$$

$$C_9 = \text{Res}_p^{\text{lock}}(C_3, C_8) =_{(6)} \neg q,$$

$$C_{10} = \text{Res}_q^{\text{lock}}(C_1, C_9) =_{(2)} p,$$

$$C_{11} = \text{Res}_p^{\text{lock}}(C_{10}, C_8) = \square.$$

As a conclusion, in this second indexing the empty clause was derived from S with the help of tautologies.

Example 5.9.

In this example we shall prove that *lock resolution + the set-of-support strategy is not complete*.

Using lock resolution prove the deduction:

$$p \rightarrow (q \rightarrow r), r \wedge s \rightarrow t, u \rightarrow s \wedge \neg t \vdash p \wedge q \rightarrow \neg u.$$

The premises (hypotheses) of the deduction are:

$$U_1 = p \rightarrow (q \rightarrow r), U_2 = r \wedge s \rightarrow t, U_3 = u \rightarrow s \wedge \neg t$$

The conclusion of the deduction is: $V = p \wedge q \rightarrow \neg u$

Resolution Proof Method

The resolution is applied to the set of clauses S , obtained from the conjunctive normal forms of the hypotheses and the negation of the conclusion.

$$\text{CNF}(U_1) = \neg p \vee \neg q \vee r,$$

$$C_1 = \neg p \vee \neg q \vee r$$

$$\text{CNF}(U_2) = \neg r \vee \neg s \vee t,$$

$$C_2 = \neg r \vee \neg s \vee t$$

$$\text{CNF}(U_3) = (\neg u \vee s) \wedge (\neg u \vee \neg t),$$

$$C_3 = \neg u \vee s, C_4 = \neg u \vee \neg t$$

$$\text{CNF}(\neg V) = p \wedge q \wedge r,$$

$$C_5 = p, C_6 = q, C_7 = u$$

$$S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$$

Using the set-of-support strategy we choose the support set $Y = \{C_5, C_6, C_7\}$ corresponding to the conclusion of the deduction.

$S \setminus Y = \{C_1, C_2, C_3, C_4\}$ is a consistent set of clauses corresponding to the premises (hypotheses) of the deduction.

We index the literals from the clauses as follows:

$$C_1 =_{(3)} \neg p \vee_{(2)} \neg q \vee_{(1)} r,$$

$$C_2 =_{(6)} \neg r \vee_{(5)} \neg s \vee_{(4)} t,$$

$$C_3 =_{(8)} \neg u \vee_{(7)} s,$$

$$C_4 =_{(10)} \neg u \vee_{(9)} \neg t,$$

$$C_5 =_{(11)} p, \quad C_6 =_{(12)} q, \quad C_7 =_{(13)} u$$

Note that the restriction imposed by lock resolution (the literals of lowest indices in their clauses resolve) combined with the set-of-support strategy (avoids to resolve two clauses belonging to the consistent set $S \setminus Y = \{C_1, C_2, C_3, C_4\}$) will block the resolution process and \square cannot be derived.

If we do not use the set-of-support strategy we can derive \square .

$$C_8 = \text{Res}_t^{lock}(C_2, C_4) =_{(6)} \neg r \vee_{(5)} \neg s \vee_{(10)} \neg u$$

$$C_9 = \text{Res}_s^{lock}(C_3, C_8) =_{(6)} \neg r \vee_{(8)} \neg u$$

$$C_{10} = \text{Res}_r^{lock}(C_1, C_9) =_{(3)} \neg p \vee_{(2)} \neg q \vee_{(8)} \neg u$$

$$C_{11} = \text{Res}_q^{lock}(C_6, C_{10}) =_{(3)} \neg p \vee_{(8)} \neg u$$

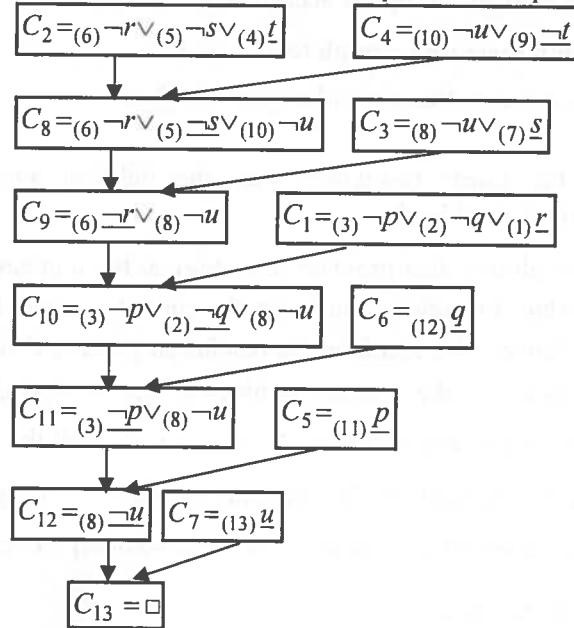
$$C_{12} = \text{Res}_p^{lock}(C_5, C_{11}) =_{(8)} \neg u$$

$$C_{13} = \text{Res}_u^{lock}(C_7, C_{12}) = \square$$

We conclude that the set $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$ is inconsistent and the deduction $p \rightarrow (q \rightarrow r)$, $r \wedge s \rightarrow t$, $u \rightarrow s \wedge \neg t \vdash p \wedge q \rightarrow \neg u$ holds.

A Computational Approach to Classical Logics and Circuits

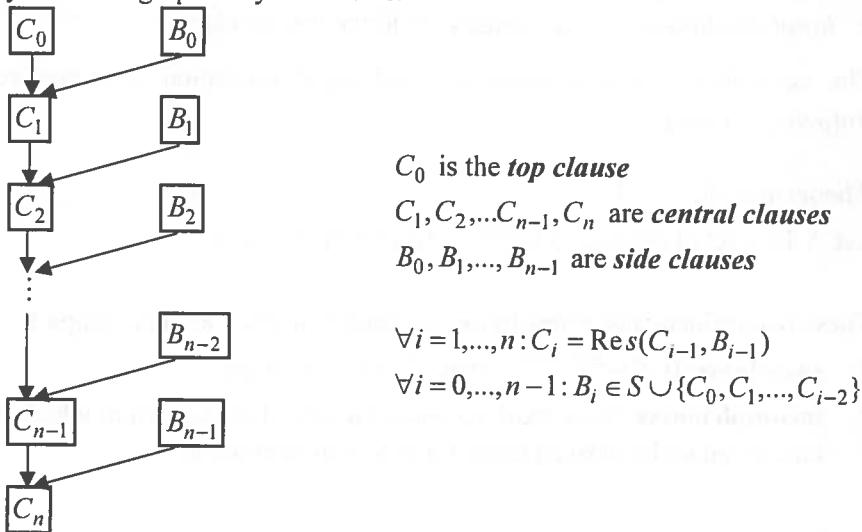
The binary tree corresponding to the refutation process is depicted below.



5.4. Linear resolution

Proposed by D.W. Loveland in 1970 [30], linear resolution is a very efficient refinement of resolution. The resolution process is a linear one: at each step one of the parent clauses is the resolvent derived at the previous step.

For a set S of clauses, a *linear deduction* of C_n from S with $C_0 \in S$ as the top clause is symbolized graphically as follows:



Resolution Proof Method

Linear resolution is sound and complete according to the theorem:

Theorem 5.9. Soundness and completeness [30]

The set S of clauses is inconsistent if and only if $S \vdash_{\text{Res}}^{\text{lin}} \square$.

We can combine the linear resolution with the deletion strategy and the completeness property is preserved.

This refinement of resolution also provides a strategy at the implementation level: **backtracking** algorithm. In each iteration, for the current central clause there are more possible side clauses. We continue the resolution process choosing one side clause. If in the iteration i the process is blocked (the central clause C_i is a tautology or it is an existing central clause: $C_i = C_j, j < i$) or all the side clauses of C_i were used, then we go back to the previous iteration ($i-1$) and we choose another possible side clause for C_{i-1} to continue the resolution process.

The algorithm stops in two cases:

- the empty clause was derived and the conclusion is that S is inconsistent.
- for the top clause all the possible side clauses were used, but the empty clause was not derived, then we conclude that the set S is consistent.

The consistency of a set of clauses is proved after a complete search without the derivation of the empty clause.

Special cases of linear resolution [11]:

- **unit resolution**: the central clauses have at least a unit clause as a parent clause.
- **input resolution**: all side clauses are initial (input) clauses.

The equivalence of unit resolution and input resolution is expressed by the following theorem.

Theorem 5.10. [11]

Let S be a set of clauses. $S \vdash_{\text{Res}}^{\text{input}} \square$ if and only if $S \vdash_{\text{Res}}^{\text{unit}} \square$.

These two refinements of resolution are sound, but they are not complete:

1. **soundness**: If $S \vdash_{\text{Res}}^{\text{input/unit}} \square$ then S is inconsistent;
2. **incompleteness**: there exist inconsistent sets of clauses from which the empty clause cannot be derived using input or unit resolution.

A Computational Approach to Classical Logics and Circuits

Example 5.10.

The set $S = \{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$ is inconsistent (see Example 5.1), but there is no unit refutation from S because there is no unit clause in S . According to Theorem 5.10 there is no input refutation from S . This is an example of incompleteness of unit/input resolution.

Definition 5.3.

A clause is called a *positive clause* if it contains only positive literals. A clause is called a *negative clause* if it contains only negative literals. A clause is called *Horn clause* if it contains exactly one positive literals, all the other literals are negative.

Theorem 5.11.

The input resolution is complete on a set of Horn clauses with a negative top clause. (PROLOG).

Example 5.11. knowledge base

From the hypotheses:

$$H_1 : U_1 \wedge U_2 \wedge \dots \wedge U_n \rightarrow V$$

$$H_2 : X_1 \wedge X_2 \wedge \dots \wedge X_l \rightarrow Y$$

...

$$H_j : W_1 \wedge W_2 \wedge \dots \wedge W_m \rightarrow R$$

is $C = Z_1 \wedge Z_2 \wedge \dots \wedge Z_m$ deducible?

To prove by refutation using resolution we have to transform into clausal normal forms the hypotheses and the negation of the conclusion.

A formula of type: $U_1 \wedge U_2 \wedge \dots \wedge U_n \rightarrow V$ provides a Horn clause:

$$\neg U_1 \vee \neg U_2 \vee \dots \vee \neg U_n \vee V$$

The hypothesis H_i provides the Horn clause $C_i, i = 1, 2, \dots, j$.

The negation of C is $\neg(Z_1 \wedge Z_2 \wedge \dots \wedge Z_m)$ and provides a negative clause:

$$C_{j+1} = \neg Z_1 \vee \neg Z_2 \vee \dots \vee \neg Z_m$$

We have to apply the input resolution to the set $S = \{C_1, C_2, \dots, C_j, C_{j+1}\}$ with the top clause C_{j+1} .

According to Theorem 5.11 we have that:

$$H_1, H_2, \dots, H_j \vdash C \text{ if and only if } S \vdash_{\text{Res}}^{\text{input}} \square.$$

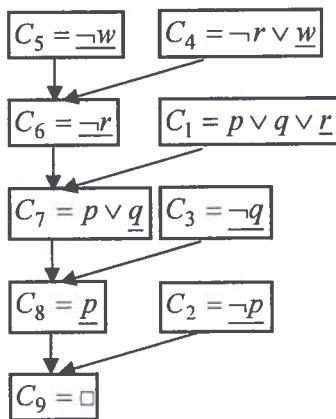
Example 5.12.

Using linear resolution prove the inconsistency of the set S of clauses: $S = \{C_1, C_2, C_3, C_4, C_5\}$.

Resolution Proof Method

$$C_1 = p \vee q \vee r, C_2 = \neg p, C_3 = \neg q, C_4 = \neg r \vee w, C_5 = \neg w$$

The derivation tree is as follows:



Top clause: C_5 , side clauses: C_4, C_1, C_3, C_2 , central clauses: C_6, C_7, C_8, C_9 .

This linear resolution is also an *input* resolution and a *unit* resolution

$S \vdash_{\text{Res}}^{\text{lin}} \square$, therefore S is an inconsistent set.

Example 5.13.

Check the consistency/inconsistency of the set S of clauses using linear resolution.

$$S = \{C_1 = p \vee q, C_2 = \neg p \vee q, C_3 = \neg p \vee \neg q\}$$

We begin the linear search of the derivation of the empty clause using the backtracking strategy:

First iteration:

C_1 is the top clause and it has 3 possible side clauses: C_2, C_3, C_3 .

Note that C_1 and C_3 can resolve in two ways: with p and q respectively as the literal resolved upon.

I) C_2 is used as a side clause for C_1

Second iteration:

$$C_4 = \text{Res}_p(C_1, C_2) = q \text{ -- central clause having one possible side clause: } C_3$$

Third iteration:

$$C_5 = \text{Res}_q(C_4, C_3) = \neg p \text{ -- central clause having one possible side clause: } C_1$$

Fourth iteration:

$$C_6 = \text{Res}_p(C_5, C_1) = q = C_4 \text{ (an existing clause)}$$

The search process is blocked, otherwise it will be an infinite loop.

A Computational Approach to Classical Logics and Circuits

We go back to the previous iteration (*third iteration*): all the possible side clauses of C_5 were used.

We go back to the previous iteration (*second iteration*): all the possible side clauses of C_4 were used.

We go back to the previous iteration (*first iteration*).

- II) C_3 (with $\neg p$ as the literal resolved upon) is used as a side clause for C_1

Second iteration:

$$C_7 = \text{Res}_p(C_1, C_3) = q \vee \neg q \equiv T \text{ (tautology)}$$

The search process is blocked, because a tautology cannot help to derive \square (inconsistency).

We go back to the previous iteration (*first iteration*).

- III) C_3 (with $\neg q$ as the literal resolved upon) is used as a side clause for C_1

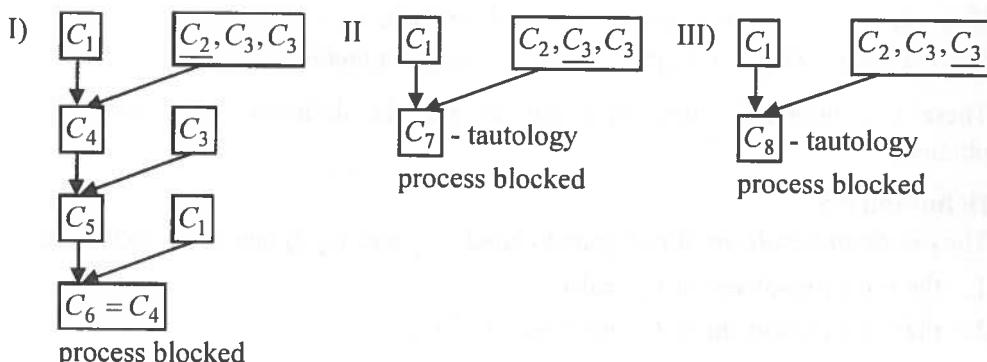
$$C_8 = \text{Res}_q(C_1, C_3) = p \vee \neg p \equiv T \text{ (tautology)}$$

The search process is blocked.

We go back to the previous iteration (*first iteration*).

All the possible side clauses of C_1 were used. A complete linear derivation search was performed, but the empty clause was not derived, so S is a consistent set.

The graphical representation of the resolution process, following the backtracking strategy is depicted below.



Resolution Proof Method

5.5. Resolution in first-order (predicate) logic

We present the ***predicate resolution*** as a formal (axiomatic) system according to the approach from the paper [63]:

$\text{Res}^{\text{Pr}} = (\Sigma_{\text{Res}}^{\text{Pr}}, F_{\text{Res}}^{\text{Pr}}, A_{\text{Res}}^{\text{Pr}}, R_{\text{Res}}^{\text{Pr}})$, where:

- $\Sigma_{\text{Res}}^{\text{Pr}} = \Sigma_{\text{Pr}} - \{\rightarrow, \leftrightarrow, \wedge, \exists, \forall\}$ is the alphabet;
- $F_{\text{Res}}^{\text{Pr}} \cup \{\square\}$ is the set of well formed formulas;
 - $F_{\text{Res}}^{\text{Pr}}$ is the set of all clauses built using the alphabet $\Sigma_{\text{Res}}^{\text{Pr}}$;
 - \square is empty clause which does not contain any literal and symbolizes inconsistency;
- $A_{\text{Res}}^{\text{Pr}} = \emptyset$ is the set of axioms;
- $R_{\text{Res}}^{\text{Pr}} = \{res^{\text{Pr}}, fact\}$ is the set of inference rules containing the ***resolution rule*** (res^{Pr}) and the ***factoring rule*** ($fact$).

$f \vee l_1, g \vee \neg l_2 \vdash_{res^{\text{Pr}}} \lambda(f) \vee \lambda(g)$, where $\lambda = mgu(l_1, l_2)$, $f, g \in F_{\text{Res}}^{\text{Pr}}$.

$l_1 \vee l_2 \vee \dots \vee l_k \vee l_{k+1} \vee \dots \vee l_n \vdash_{fact} l_1 \vee l_{k+1} \vee \dots \vee l_n$, where $\lambda = mgu(l_1, l_2, \dots, l_k)$

Definition 5.4.

The predicate clauses $C_1 = f \vee l_1$ and $C_2 = g \vee \neg l_2$, without common free variables, are called ***clashing clauses*** if the literals l_1 and l_2 are unifiable: there exists $\lambda = mgu(l_1, l_2)$.

The ***binary resolvent*** of C_1 and C_2 is $C = \text{Res}_{\lambda}^{\text{Pr}}(C_1, C_2) = \lambda(f) \vee \lambda(g)$.

If $C = l_1 \vee l_2 \vee \dots \vee l_k \vee l_{k+1} \vee \dots \vee l_n$ and $\lambda = mgu(l_1, l_2, \dots, l_k)$,

$Fact(C) = \lambda(l_1) \vee \lambda(l_{k+1}) \vee \dots \vee \lambda(l_n)$ is called a ***factor*** of C .

These two inference rules are combined and the definition of a resolvent is obtained.

Definition 5.5.

The ***predicate resolvent*** of two parent clauses C_1 and C_2 is one of the following:

1. the binary resolvent of C_1 and C_2 ;
2. the binary resolvent of C_1 and a factor of C_2 ;
3. the binary resolvent of a factor of C_1 and C_2 ;
4. the binary resolvent of a factor of C_1 and a factor of C_2 .

Example 5.14.

a) $C_1 = P(f(x), g(y)) \vee Q(x, y)$ and $C_2 = \neg P(f(f(a)), g(z)) \vee Q(f(a), g(z))$

$$\text{Res}_{[x \leftarrow f(a), y \leftarrow z]}^{\text{Pr}}(C_1, C_2) = Q(f(a), g(z)) \vee Q(f(a), z)$$

A Computational Approach to Classical Logics and Circuits

b) $C = P(x, g(y), z) \vee P(a, g(b), a) \vee Q(x, z)$

$$l_1 = P(x, g(y), z), l_2 = P(a, g(b), a)$$

$$\text{Fact}(C) = \lambda(C), \text{ where } \lambda = [x \leftarrow a, y \leftarrow b, z \leftarrow a] = \text{mgu}(l_1, l_2).$$

$\text{Fact}(C) = P(a, g(b), a) \vee Q(a, a)$ is a factor of C

Theorem 5.12.

Let U_1, U_2, \dots, U_n, V be first-order formulas.

1. V is a theorem if and only if $(\neg V)^c \vdash_{\text{Res}}^{\text{Pr}} \square$.

2. $U_1, U_2, \dots, U_n \vdash V$ if and only if $\{U_1^c, U_2^c, \dots, U_n^c, (\neg V)^c\} \vdash_{\text{Res}}^{\text{Pr}} \square$.

All the refinements and strategies for propositional logic can be used in predicate logic. It is recommended to rename the free variables in the initial set of clauses, such that they will be distinct in different clauses.

Algorithm predicate_resolution:

input: U_1, U_2, \dots, U_n, V - first-order formulas.

output: message: " $U_1, U_2, \dots, U_n \vdash V$ " or " $U_1, U_2, \dots, U_n \not\vdash V$ " or

"we cannot decide if $U_1, U_2, \dots, U_n \vdash V$ or $U_1, U_2, \dots, U_n \not\vdash V$ "

begin

build the set of clauses: $S := \{U_1^c, U_2^c, \dots, U_n^c, (\neg V)^c\}$;

do{

select l_1, l_2, C_1, C_2 such that:

C_1, C_2 are clauses or factors of clauses of S ;

$l_1 \in C_1$, and $\neg l_2 \in C_2$;

if (l_1 and l_2 are unifiable with $\theta := \text{mgu}(l_1, l_2)$) **then**

$C := \text{Res}_{\theta}^{\text{Pr}}(C_1, C_2)$;

if ($C = \square$) **then** write " $U_1, U_2, \dots, U_n \vdash V$ "; **exit**;

else $S := S \cup \{C\}$;

end_if

end_if

}until (no new resolvents can be derived **or** a predefined quantity of effort was done)

if (no new resolvents can be derived) **then** write " $U_1, U_2, \dots, U_n \not\vdash V$ ";

else write "we cannot decide if $U_1, U_2, \dots, U_n \vdash V$ or $U_1, U_2, \dots, U_n \not\vdash V$ "

end_if

end

The resolution algorithm for predicate logic is a semi-decision procedure.

Resolution Proof Method

Theorem 5.13. (Church 1936 [13])

The problem of validity of a first-order formula is *undecidable*, but it is *semi-decidable*. If a procedure *Proc* is used to check the validity of a first-order formula U we have the following cases:

1. if the formula U is valid, then *Proc* ends with the corresponding answer.
2. if the formula U is not valid, then *Proc* ends with the corresponding answer or *Proc* may never stop.

Example 5.15.

Check if $U \vdash V$, where $U = (\forall x)(P(x) \rightarrow Q(x))$ and $V = (\forall x)P(x) \rightarrow (\forall x)Q(x)$.

We transform the formulas U and $\neg V$ into clausal normal forms:

- Prenex normal forms:

$$U^P = (\forall x)(\neg P(x) \vee Q(x))$$

$$\begin{aligned} (\neg V)^P &= \neg((\forall z)P(z) \rightarrow (\forall y)Q(y)) \equiv (\forall z)P(z) \wedge (\exists y)\neg Q(y) \equiv \\ &\equiv (\forall z)(\exists y)(P(z) \wedge \neg Q(y)) \end{aligned}$$

- Skolem normal forms:

$$U^S = (\forall x)(\neg P(x) \vee Q(x)),$$

$$(\neg V)^S = (\forall z)(P(z) \wedge \neg Q(f(z))), f \text{ is a unary Skolem function}$$

- Clausal normal forms:

$$U^C = \neg P(x) \vee Q(x), \quad (\neg V)^C = P(z) \wedge \neg Q(f(z))$$

We consider the following set of clauses: $S = \{C_1, C_2, C_3\}$. The free variables are renamed such that they are distinct in the set S .

$$C_1 = \neg P(x) \vee Q(x), \quad C_2 = P(y), \quad C_3 = \neg Q(f(z))$$

We derive the empty clause using predicate resolution as follows:

$$C_1, C_2 \vdash_{res}^{[y \leftarrow x]} C_4 = Q(x)$$

$$C_4, C_5 \vdash_{res}^{[x \leftarrow f(z)]} \square$$

According to Theorem 5.12 we proved that $U \vdash V$.

Example 5.16.

Using linear resolution prove that $S = \{C_1, C_2, C_3, C_4\}$ is an inconsistent set of clauses:

$$C_1 = P(x, f(x), e), \quad C_2 = \neg R(x) \vee \neg R(y) \vee \neg P(x, f(y), z) \vee R(z),$$

$$C_3 = R(a), \quad C_4 = \neg R(e)$$

Constants: a, e , function symbols: f , predicate symbols: P, R

A Computational Approach to Classical Logics and Circuits

For linear resolution we choose the top clause: C_4

$$C_2 \vdash_{fact}^{[y \leftarrow x]} C_5 = \neg R(x) \vee \neg P(x, f(x), z) \vee R(z), C_5 \text{ is a factor of } C_2$$

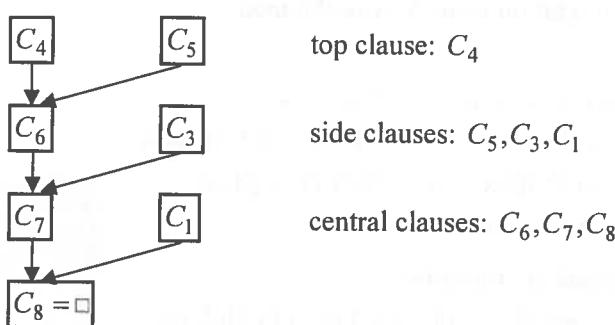
$$C_4, C_5 \vdash_{res}^{[z \leftarrow e]} C_6 = \neg R(x) \vee \neg P(x, f(x), e), \text{ central clause}$$

$$C_6, C_3 \vdash_{res}^{[x \leftarrow a]} C_7 = \neg P(a, f(a), e), \text{ central clause}$$

$$C_7, C_1 \vdash_{res}^{[x \leftarrow a]} C_8 = \square$$

From the set S we derived the empty clause, therefore S is inconsistent.

The refutation tree is depicted below.



Example 5.17.

Check the inconsistency of the set S of clauses using lock resolution.

$$S = \{\neg P(x) \vee Q(x) \vee R(x), \neg Q(y) \vee R(y), P(a), \neg R(a)\}$$

a) We index the literals as follows:

$$C_1 =_{(3)} \neg P(x) \vee_{(2)} Q(x) \vee_{(1)} R(x)$$

$$C_2 =_{(5)} \neg Q(y) \vee_{(4)} R(y)$$

$$C_3 =_{(6)} P(a), C_4 =_{(7)} \neg R(a)$$

The following resolvents are obtained:

$$C_5 = \text{Res}_{[x \leftarrow a]}^{\text{Pr}}(C_1, C_4) =_{(3)} \neg P(a) \vee_{(2)} Q(a)$$

$$C_6 = \text{Res}_{[y \leftarrow a]}^{\text{Pr}}(C_2, C_4) =_{(5)} \neg Q(a)$$

$$C_7 = \text{Res}^{\text{Pr}}(C_5, C_6) =_{(3)} \neg P(a)$$

$$C_8 = \text{Res}^{\text{Pr}}(C_6, C_7) = \square$$

$S \vdash_{\text{Res}}^{\text{Pr}} \square$ and thus S is inconsistent.

Resolution Proof Method

- b) another indexing of the literals:

$$C_1 =_{(2)} \neg P(x) \vee_{(1)} Q(x) \vee_{(3)} R(x)$$

$$C_2 =_{(4)} \neg Q(y) \vee_{(5)} R(y)$$

$$C_3 =_{(6)} P(a) \quad C_4 =_{(7)} \neg R(a)$$

The following resolvents are derived:

$$C_5 = \text{Res}_{[y \leftarrow x]}^{\text{Pr}}(C_1, C_2) =_{(2)} \neg P(x) \vee_{(3)} R(x)$$

$$C_6 = \text{Res}_{[x \leftarrow a]}^{\text{Pr}}(C_3, C_5) =_{(3)} R(a)$$

$$C_7 = \text{Res}^{\text{Pr}}(C_4, C_6) = \square$$

Another logic refutation from S was obtained.

Example 5.18.

Prove the semidistributivity of „ \forall ” over „ \rightarrow ” :

$$\vdash (\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x)) \text{ and}$$

$$\nvdash ((\forall x)P(x) \rightarrow (\forall x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x))$$

by applying predicate resolution.

We consider the predicate formulas.

$$U_1 = (\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x))$$

$$U_2 = ((\forall x)P(x) \rightarrow (\forall x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x))$$

According to Theorem 5.12:

- $\vdash U_1$ if and only if $(\neg U_1)^c \vdash_{\text{Res}}^{\text{Pr}} \square$ and
- $\nvdash U_2$ if and only if $(\neg U_2)^c \nvdash_{\text{Res}}^{\text{Pr}} \square$.

We apply syntactic transformations in order to obtain the prenex, Skolem and clausal normal forms of the formulas $\neg U_1$ and $\neg U_2$.

$$\neg U_1 = \neg((\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x)))$$

- the logical equivalence $\neg(A \rightarrow B) \equiv A \wedge \neg B$ is applied

$$\equiv (\forall x)(P(x) \rightarrow Q(x)) \wedge \neg((\forall x)P(x) \rightarrow (\forall x)Q(x)) \equiv$$

$$\equiv (\forall x)(P(x) \rightarrow Q(x)) \wedge (\forall x)P(x) \wedge \neg(\forall x)Q(x)$$

- infinitary DeMorgan's law: $\neg(\forall x)A(x) \equiv (\exists x)\neg A(x)$ is applied

$$\equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge (\forall x)P(x) \wedge (\exists x)\neg Q(x)$$

- rename the bound variables

$$\equiv (\forall x)(\neg P(x) \vee Q(x)) \wedge (\forall y)P(y) \wedge (\exists z)\neg Q(z)$$

- extraction of the quantifiers in front of the formula, we extract first the existential quantifier

$$\equiv (\exists z)(\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(z)) = (\neg U_1)^p - \text{prenex form}$$

A Computational Approach to Classical Logics and Circuits

$$(\neg U_1)^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a)) \text{ - Skolem normal form}$$

$[z \leftarrow a]$, a - Skolem constant

$$(\neg U_1)^c = (\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a) \text{ - clausal normal form}$$

The set of clauses used in predicate resolution is:

$$S_1 = \{C_1 = \neg P(x) \vee Q(x), C_2 = P(y), C_3 = \neg Q(a)\}$$

The following resolvents are derived:

$$C_4 = \text{Res}_{[x \leftarrow a]}^{\text{Pr}}(C_1, C_3) = \neg P(a)$$

$$C_5 = \text{Res}_{[y \leftarrow a]}^{\text{Pr}}(C_2, C_4) = \square$$

We proved: $(\neg U_1)^c \vdash_{\text{Res}}^{\text{Pr}} \square$, thus $\vdash (\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x))$.

$$\neg U_2 = \neg((\forall x)P(x) \rightarrow (\forall x)Q(x)) \rightarrow (\forall x)(P(x) \rightarrow Q(x))$$

- the logical equivalence $\neg(A \rightarrow B) \equiv A \wedge \neg B$ is applied
 $\equiv ((\forall x)P(x) \rightarrow (\forall x)Q(x)) \wedge \neg(\forall x)(P(x) \rightarrow Q(x))$
- the logical equivalence $A \rightarrow B \equiv \neg A \vee B$ is applied
 $\equiv (\neg(\forall x)P(x) \vee (\forall x)Q(x)) \wedge \neg(\forall x)(P(x) \rightarrow Q(x))$
- infinitary DeMorgan's law: $\neg(\forall x)A(x) \equiv (\exists x)\neg A(x)$ and
 $\neg(A \rightarrow B) \equiv A \wedge \neg B$ are applied
 $\equiv ((\exists x)\neg P(x) \vee (\forall x)Q(x)) \wedge (\exists x)(P(x) \wedge \neg Q(x))$
- rename the bound variables
 $\equiv ((\exists x)\neg P(x) \vee (\forall y)Q(y)) \wedge (\exists z)(P(z) \wedge \neg Q(z))$
- the quantifiers $(\exists x), (\forall y), (\exists z)$ are independent: none of them is within the scope of the other one, so we can extract them in any order. To obtain the simplest Skolem form (with fewest Skolem functions) we extract first the existential quantifiers.

$$\equiv (\exists z)(\exists x)(\forall y)((\neg P(x) \vee Q(y)) \wedge P(z) \wedge \neg Q(z)) = (\neg U_2)^P \text{ - prenex form}$$

$$(\neg U_1)^S = (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a)) \text{ - Skolem normal form}$$

$[z \leftarrow a, x \leftarrow b]$, a, b - Skolem constants

$$(\neg U_2)^c = (\neg P(b) \vee Q(y)) \wedge P(a) \wedge \neg Q(a) \text{ - clausal normal form}$$

The set of clauses used in predicate resolution is:

$$S_2 = \{C_1' = \neg P(b) \vee Q(y), C_2' = P(a), C_3' = \neg Q(a)\}$$

Resolution Proof Method

The only resolvent that can be derived is:

$$C_4' = \text{Res}_{[y \leftarrow a]}^{\text{Pr}}(C_1', C_3') = \neg P(b)$$

The literals $P(a), P(b)$ are not unifiable because a, b are distinct constants, so the clauses $C_2' = P(a), C_4' = \neg P(b)$ do not resolve and thus \square cannot be derived.

We proved: $(\neg U_2)^c \not\vdash_{\text{Res}}^{\text{Pr}} \square$, so $\not\vdash (\forall x)(P(x) \rightarrow Q(x)) \rightarrow ((\forall x)P(x) \rightarrow (\forall x)Q(x))$

U_1 is a theorem, but U_2 is not a theorem, therefore we conclude that the quantifier ' \forall ' is not distributive over the connective ' \rightarrow ', it is only semi-distributive.

Example 5.19.

Using predicate resolution check the validity of the formula:

$$U = (\forall y)(\exists x)\neg(P(x, y) \leftrightarrow \neg P(x, x))$$

We apply Theorem 5.12 and we prove by contradiction that U is valid.

$$\begin{aligned} \neg U &= \neg((\forall y)(\exists x)\neg(P(x, y) \leftrightarrow \neg P(x, x))) \equiv (\exists y)(\forall x)(P(x, y) \leftrightarrow \neg P(x, x)) \equiv \\ &\equiv (\exists y)(\forall x)((\neg P(x, y) \vee \neg P(x, x)) \wedge (P(x, x) \vee P(x, y))) - \text{prenex normal form} \\ (\neg U)^S &= (\forall x)(\neg P(x, a) \vee \neg P(x, x)) \wedge (P(x, x) \vee P(x, a)) - \text{Skolem normal form}, \\ &\quad [x \leftarrow a], \text{ } a - \text{Skolem constant} \end{aligned}$$

$$(\neg U)^c = (\neg P(x, a) \vee \neg P(x, x)) \wedge (P(x, x) \vee P(x, a)) - \text{clausal normal form}$$

From the above clausal normal form we consider the clauses:

$$C_1 = \neg P(x, a) \vee \neg P(x, x) \text{ and}$$

$$C_2 = P(x, x) \vee P(x, a)$$

Resolving these two clauses we can derive only tautologies.

It is important to apply the factoring rule for C_1 and C_2 and try to resolve their factors.

$$C_1 \vdash_{\text{fact}}^{[x \leftarrow a]} C_3 = \neg P(a, a), C_3 \text{ is a factor of } C_1.$$

$$C_2 \vdash_{\text{fact}}^{[x \leftarrow a]} C_4 = P(a, a), C_4 \text{ is a factor of } C_2.$$

$$C_3, C_4 \vdash_{\text{res}} \square$$

$$(\neg U)^c \vdash_{\text{Res}}^{\text{Pr}} \square, \text{ therefore } \neg U \text{ is inconsistent and } U \text{ is a valid formula.}$$

5.6. Semantic resolution

Semantic resolution, proposed by J.R.Slagle in 1965 [57], is a refinement of general resolution. In order to reduce the number of irrelevant and redundant clauses in the resolution process, this refinement of resolution:

- divides the set of clauses into two subsets using an interpretation (semantic aspect), to avoid resolving clauses within the same subset;
- uses the ordering of propositional/predicate symbols to impose a restriction on the literals resolved upon: the literal resolved upon from the parent clauses contains the largest symbol.

We suppose that the initial set of clauses, S , was simplified: tautologies, subsumed clauses and the clauses which contain pure literals were deleted. S is divided into the subsets: S_E and S_N using an interpretation I such that all the clauses from S_E are falsified by I and all the clauses from S_N are satisfied by I :

$$S = S_E \cup S_N.$$

There always exists at least one interpretation such that S_E and S_N are non-empty sets.

The basic idea is to avoid resolving two clauses which are both falsified or both satisfied by the same interpretation, because the resolvent is irrelevant in the process of deriving the empty clause.

The ordering of the propositional/predicate symbols and the restriction on the literal resolved upon cut down the number of useless clauses generated in the resolution process.

Another important idea is to unify all the variants of generating the same resolvent, where the order of using the parent clauses is immaterial.

Example 5.20.

Let $S = \{E_1 = p, E_2 = q, E_3 = r, N = \neg p \vee \neg q \vee \neg r\}$ be a set of propositional clauses.

There are six possible derivations of the empty clause, with the only difference being the order of using the clauses E_1, E_2, E_3 as parent clauses. We present in the following two such derivations of \square .

Version 1:

$$R_2 = \text{Res}(N, E_1) = \neg q \vee \neg r ;$$

$$R_3 = \text{Res}(R_2, E_2) = \neg r ;$$

$$R_4 = \text{Res}(R_3, E_3) = \square ;$$

Version 2:

$$R'_2 = \text{Res}(N, E_2) = \neg p \vee \neg r ;$$

$$R'_3 = \text{Res}(R'_2, E_3) = \neg p ;$$

$$R'_4 = \text{Res}(R'_3, E_1) = \square ;$$

Resolution Proof Method

To avoid all these versions being considered distinct derivations of the empty clause we shall unify them.

The combination of all the ideas presented informally above is formalized in the following definition.

Definition 5.6.

Let I be an interpretation and P an ordering of the propositional/ predicate symbols. A finite set of clauses: $\{E_1, E_2, \dots, E_q, N\}$, $q \geq 1$, is called a **semantic clash** with respect to P and I (or **PI-clash**) if and only if E_1, E_2, \dots, E_q (called **electrons**) and N (called **nucleus**) satisfy the following conditions:

1. E_1, E_2, \dots, E_q are falsified by I .
3. Let $R_1 = N$ and $R_{i+1} = \text{Res}_P(R_i, E_i)$, $i = 1, \dots, q$, where the literal resolved upon in E_i contains the largest symbol in that clause. The resolvents R_1, R_2, \dots, R_q , are all satisfied by I .
2. R_{q+1} is falsified by I . $R_{q+1} = \text{Res}_{PI}(N, E_1, E_2, \dots, E_q)$ is called a **PI-resolvent** of the **PI-clash** $\{E_1, E_2, \dots, E_q, N\}$.

Remarks:

- The order of the electrons is immaterial.
- In semantic resolution we can use any ordering of propositional/predicate symbols and any interpretation.
- The resolvents R_1, R_2, \dots, R_q , are all satisfied by I . We do not keep them because they will not be used further in the resolution process.
- All the **PI-resolvents** are falsified by the interpretation I , so they are added to the set of electrons.

Example 5.21.

$S = \{p, q, r, \neg p \vee \neg q \vee \neg r\}$ is a set of propositional clauses, I is an interpretation, $I : \{p, q, r\} \rightarrow \{T, F\}$, $I(p) = F$, $I(q) = F$, $I(r) = F$, and $P : p > q > r$ is an ordering of the propositional variables.

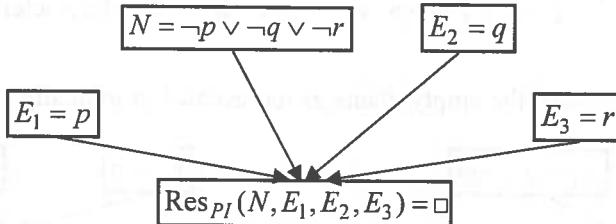
The set S is a **PI-clash**:

- $S_E = \{E_1 = p, E_2 = q, E_3 = r\}$, the electrons are falsified by I
- $S_N = \{N = \neg p \vee \neg q \vee \neg r\}$, the nucleus is satisfied by I
- $R_1 = N = \neg p \vee \neg q \vee \neg r$ is satisfied by I
- $R_2 = \text{Res}_P(R_1, E_1) = \neg q \vee \neg r$ is satisfied by I

A Computational Approach to Classical Logics and Circuits

- $R_3 = \text{Res}_P(R_2, E_2) = \neg r$ is satisfied by I
- $R_4 = \text{Res}_P(R_3, E_3) = \square = \text{Res}_{PI}(N, E_1, E_2, E_3)$ is falsified by I .

We represent graphically the PI -clash as follows:



Example 5.22.

$S = \{E_1 = p \vee r, E_2 = q \vee r, N = \neg p \vee \neg q \vee r\}$ is a set of clauses, I is an interpretation, $I : \{p, q, r\} \rightarrow \{T, F\}$, $I(p) = F$, $I(q) = F$, $I(r) = F$, and $P : r > p > q$ is an ordering of the propositional symbols.

S is divided by the interpretation I as follows:

- $S_E = \{E_1 = p \vee \underline{r}, E_2 = q \vee \underline{r}\}$, the electrons are falsified by I
- $S_N = \{N = \neg p \vee \neg q \vee r\}$, the nucleus is satisfied by I

The underlined symbols are the largest in the electrons, according to the ordering P . No electrons can resolve with the nucleus, so S is not a PI -clash.

Definition 5.7.

Let S be a set of clauses, I an interpretation of S and P an ordering of the propositional/predicate symbols of S . A deduction from S is called a **PI -deduction** if and only if each clause in the deduction is either a clause of S or a PI -resolvent.

Theorem 5.14. Soundness and completeness [57]

Let S be set of clauses, I an interpretation of S and P an ordering of the propositional/predicate symbols of S . S is inconsistent if and only if there exists a PI -deduction of the empty clause from S .

Remark:

- The completeness property is preserved if this refinement of resolution is combined with the deletion strategy: because no tautologies are generated we only delete the subsumed clauses.

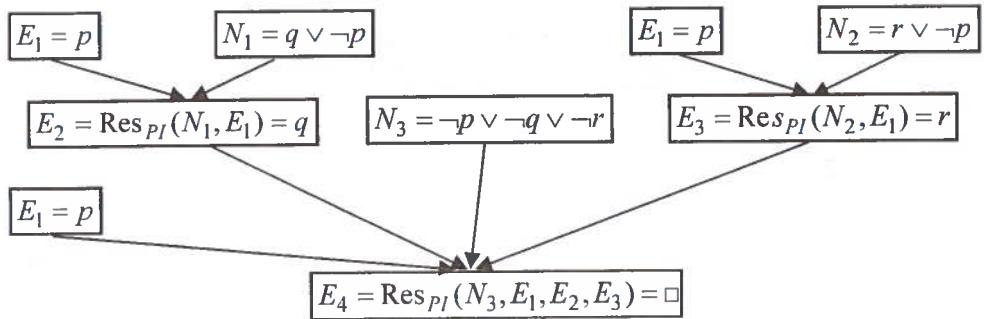
Example 5.23.

Using the semantic resolution prove that the following set of clauses is inconsistent. $S = \{p, q \vee \neg p, r \vee \neg p, \neg p \vee \neg q \vee \neg r\}$

Resolution Proof Method

- We choose the interpretation $I_1 : \{p, q, r\} \rightarrow \{T, F\}$, $I_1(p) = F$, $I_1(q) = F$, $I_1(r) = F$, and the ordering $P_1 : p > q > r$
 $S_E = \{E_1 = p\}$, the electron is falsified by I_1
 $S_N = \{N_1 = q \vee \neg p, N_2 = r \vee \neg p, N_3 = \neg p \vee \neg q \vee \neg r\}$, the nuclei are satisfied by I_1

The PI-deduction of the empty clause is represented graphically as follows:



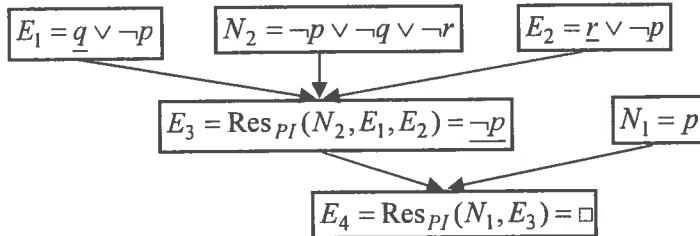
The PI-resolvents E_2, E_3 are falsified by the interpretation I_1 and they are used as electrons further in the semantic resolution process.

- We choose: $I_2 : \{p, q, r\} \rightarrow \{T, F\}$, $I_2(p) = T$, $I_2(q) = F$, $I_2(r) = F$
and $P_2 : q > r > p$

$S_E = \{E_1 = \underline{q} \vee \neg p, E_2 = \underline{r} \vee \neg p\}$, the electrons are falsified by I_2

$S_N = \{N_1 = p, N_2 = \neg p \vee \neg q \vee \neg r\}$, the nuclei are satisfied by I_2

The PI-deduction of the empty clause is represented as follows:



Special cases of semantic resolution are the *set-of-support strategy* [65] and *hyperresolution* (positive and negative) [51].

The set-of support strategy: avoids to resolve two clauses belonging to a consistent subset of the initial set of clauses, because the resolvents derived from a consistent set are irrelevant in the process of deriving the empty clause.

A Computational Approach to Classical Logics and Circuits

Hyperresolution is a semantic resolution with a particular interpretation that assigns the same truth value (F or T) to all the propositional variables.

Definition 5.8.

A *clause* is *positive* if it contains only non-negated atoms. A *clause* is *negative* if all its literals are negated atoms. A *clause* is called *mixed* if it is neither positive nor negative.

Definition 5.9.

A *positive hyperresolution* is a special case of *PI-resolution* in which the interpretation I assigns to all the propositional variables the truth value F . It is called positive hyperresolution because *all the electrons and PI-resolvents are positive clauses*.

Note: The negative and mixed clauses are nuclei.

Definition 5.10.

A *negative hyperresolution* is a special case of *PI-resolution* in which the interpretation I assigns to all the propositional variables the truth value T . It is called negative hyperresolution because *all the electrons and PI-resolvents are negative clauses*.

Note: The positive and mixed clauses are nuclei.

Often, the hypotheses of a deduction are represented by some positive and mixed clauses, and the negation of the conclusion is represented as a negative clause.

The *positive hyperresolution* corresponds to “**thinking forward**”: from the hypotheses the conclusion is derived and together with the negation of the conclusion the empty clause is obtained.

The *negative hyperresolution* corresponds to “**thinking backward**”: from the negation of the conclusion and the hypotheses the empty clause is derived.

Example 5.24.

Using positive and negative hyperresolution prove the following deduction:

$$q \vee r, q \rightarrow r, w \vdash r \wedge w.$$

We consider the set of clauses corresponding to the hypotheses and the negation of the conclusion: $S = \{q \vee r, \neg q \vee r, w, \neg r \vee \neg w\}$

Positive hyperresolution - thinking forward

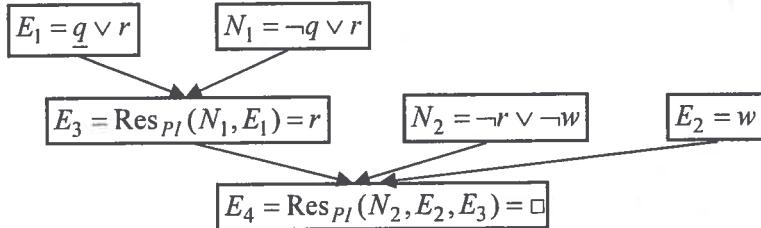
The interpretation is $I_1 : \{q, r, w\} \rightarrow \{T, F\}, I_1(q) = F, I_1(r) = F, I_1(w) = F$ and the ordering of the symbols is $P_1 : q > r > w$.

Resolution Proof Method

$S_E = \{E_1 = \underline{q} \vee r, E_2 = w\}$, the electrons are falsified by I_1

$S_N = \{N_1 = \neg q \vee r, N_2 = \neg r \vee \neg w\}$, the nuclei are satisfied by I_1

The PI -deduction of the empty clause is graphically represented as follows:



Negative hyperresolution- thinking backward

$S = \{q \vee r, \neg q \vee r, w, \neg r \vee \neg w\}$.

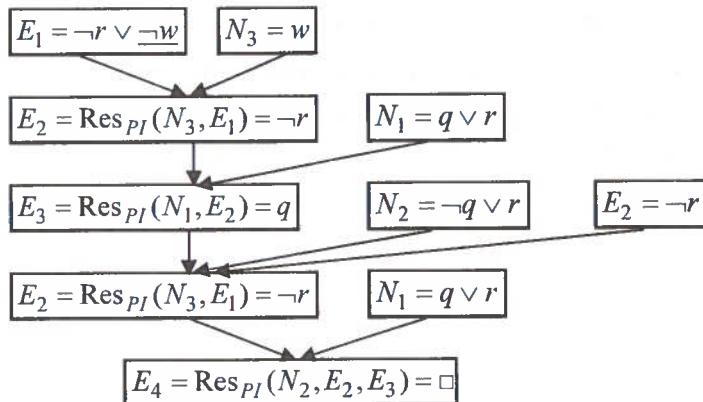
The interpretation is $I_2 : \{q, r, w\} \rightarrow \{T, F\}, I_2(q) = T, I_2(r) = T, I_2(w) = T$,

and the ordering of the propositional variables is $P_2 : w > q > r$

$S_E = \{E_1 = \neg r \vee \underline{\neg w}\}$ the electron is falsified by I_2

$S_N = \{N_1 = q \vee r, N_2 = \neg q \vee r, N_3 = w\}$ the nuclei are satisfied by I_2

The PI -deduction of the empty clause is



Positive hyperresolution can be simulated by lock resolution using the following rules for indexing the literals from the clauses [33]:

- **RP₁:** In the positive and negative clauses the indices of literals increase while the propositional symbols decrease in the ordering P of symbols.
- **RP₂:** In the mixed clauses the indices of literals with the negation sign are smaller than the indices of the literals without the negation sign. For the literals with the same sign, the rule for indexing is RP₁.

A Computational Approach to Classical Logics and Circuits

These rules block the resolution of two mixed clauses and the resolution of a negative clause and a mixed one. Therefore the resolution is permitted only to a positive clause (electron) and a negative or a mixed clause (nucleus). The literal resolved upon from the positive clause (electron) is the one which contains the largest propositional/predicate symbol.

Example 5.25.

$S = \{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$ is a set of clauses and

$P : p > q$ an ordering of the propositional symbols.

We shall simulate positive hyperresolution by lock resolution.

$$I : \{p, q\} \rightarrow \{T, F\}, \quad I(p) = F, \quad I(q) = F$$

The literals from the clauses are indexed according to the above rules as follows:

$E_1 =_{(1)} p \vee_{(2)} q$ is an electron, the literal resolved upon must be p .

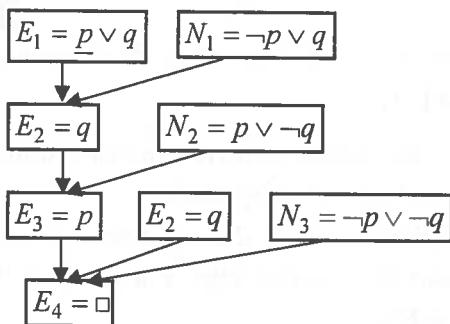
The nuclei are the clauses N_1, N_2, N_3 and we apply the rules RP_1 and RP_2 .

$$N_1 =_{(3)} \neg p \vee_{(4)} q,$$

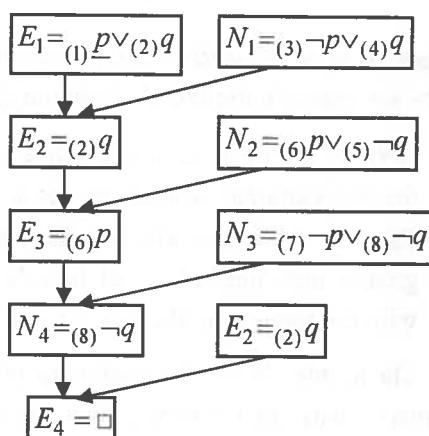
$$N_2 =_{(6)} p \vee_{(5)} \neg q,$$

$$N_3 =_{(7)} \neg p \vee_{(8)} \neg q$$

positive hyperresolution:



lock resolution:



Resolution Proof Method

Algorithm positive_hyperresolution_propositional_logic [33]

input: S - a set of clauses, P - an ordering of the propositional symbols;

output: message: " S is inconsistent" or " S is consistent"

begin

$E^0 :=$ all positive clauses from S indexed according to RP_1

$N^0 :=$ all non-positive (negative or mixed) clauses from S , indexed according to RP_1 and RP_2

$i := 0;$

// we will generate levels of electrons E^i and levels of nuclei N^i , $i > 0$

do

{

$i := i + 1$

$W := \{\text{Res}^{lock}(C_1, C_2) \mid C_1 \in E^{i-1}, C_2 \in N^{i-1}\}$

if ($\square \in W$) **then write** " S is inconsistent"; **exit**

end_if

$E^i := E^{i-1} \cup \{ \text{all positive clauses from } W \}$

$N^i := N^{i-1} \cup \{ \text{all non-positive clauses from } W \}$

} **until** ($E^i = E^{i-1}$ **and** $N^i = N^{i-1}$)

write " S is consistent"

end

Negative hyperresolution can be simulated by lock resolution using the following rules for indexing the literals from the clauses [33]:

- **RN₁:** In the positive and negative clauses the indices of literals increase while the propositional symbols decrease in the ordering P of symbols.
- **RN₂:** In the mixed clauses the indices of literals with the negation sign are greater than the indices of literals without the negation sign. For the literals with the same sign, the rule for indexing is **RN₁**.

These rules block the resolution of two mixed clauses and the resolution of a positive clause and a mixed one. Therefore the resolution is permitted only to a negative clause (electron) and a positive or mixed clause (nucleus). The literal resolved upon from the negative clause (electron) must contain the largest propositional/predicate symbol.

A Computational Approach to Classical Logics and Circuits

Example 5.26.

$S = \{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$ is a set of propositional clauses and

$P : q > p$ is an ordering of the propositional symbols.

We shall simulate negative hyperresolution by lock resolution.
 $I : \{p, q\} \rightarrow \{T, F\}, I(p) = T, I(q) = T$

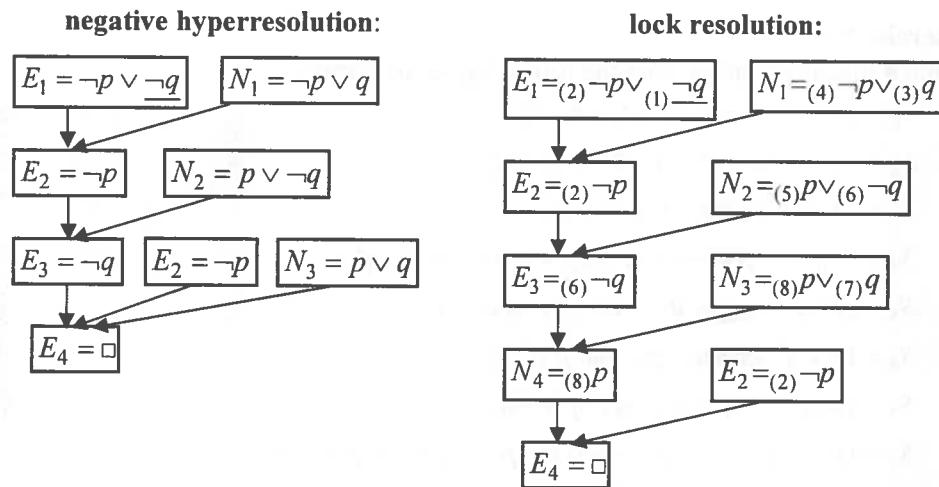
The literals from clauses are indexed according to the above rules as follows:

$E_1 =_{(2)} \neg p \vee_{(1)} \underline{\neg q}$ is an electron, the literal resolved upon must be $\neg q$.

The nuclei are the clauses N_1, N_2, N_3 :

$$N_1 =_{(4)} \neg p \vee_{(3)} q, N_2 =_{(5)} p \vee_{(6)} \neg q, N_3 =_{(8)} p \vee_{(7)} q$$

For comparison, the negative hyperresolution process and the lock resolution process are represented graphically in the following.



5.7. Exercises

Exercise 5.1.

Using general resolution prove that the following formulas are theorems.

1. $U_1 = (\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B);$
2. $U_2 = (B \rightarrow A) \wedge (C \rightarrow A) \rightarrow (B \wedge C \rightarrow A);$
3. $U_3 = (B \rightarrow A) \wedge (C \rightarrow A) \rightarrow (B \vee C \rightarrow A);$
4. $U_4 = (A \rightarrow C) \rightarrow ((\neg A \rightarrow B) \rightarrow (\neg B \rightarrow C));$
5. $U_5 = A \vee (B \rightarrow C) \rightarrow (A \vee B) \rightarrow (A \vee C);$
6. $U_6 = (A \rightarrow B) \rightarrow ((C \rightarrow A) \rightarrow (C \rightarrow B));$
7. $U_7 = (A \rightarrow B) \rightarrow ((\neg A \rightarrow C) \rightarrow (\neg B \rightarrow C));$
8. $U_8 = (A \rightarrow B \wedge C) \rightarrow (A \rightarrow B) \wedge (A \rightarrow C).$

Resolution Proof Method

Exercise 5.2.

Using lock resolution check the inconsistency of the following sets of clauses.
Choose two different indexings for the literals:

1. $S_1 = \{p \vee q, p \vee \neg q \vee r, p \vee \neg q \vee \neg r, \neg p \vee r, \neg p \vee \neg r\};$
2. $S_2 = \{\neg p \vee \neg q, \neg p \vee q \vee \neg r, p \vee \neg r, \neg p \vee r, p \vee r\};$
3. $S_3 = \{p \vee q, p \vee \neg q \vee \neg r, \neg p \vee \neg r, r, \neg p \vee r\};$
4. $S_4 = \{p \vee q, \neg p \vee q \vee \neg r, \neg p \vee q \vee r, \neg q \vee \neg r, \neg q \vee r\};$
5. $S_5 = \{p \vee \neg q, \neg p \vee \neg q \vee r, \neg p \vee q \vee r, p \vee q, \neg r\};$
6. $S_6 = \{p \vee q, \neg p \vee q \vee \neg r, \neg p \vee \neg q \vee \neg r, p \vee \neg q, r\};$
7. $S_7 = \{p \vee \neg q, \neg p \vee \neg q \vee r, \neg p \vee \neg q \vee \neg r, r \vee q, \neg r \vee q\};$
8. $S_8 = \{p \vee r, p \vee q \vee \neg r, \neg p \vee \neg q \vee r, \neg p \vee q \vee r, \neg r\}.$

Exercise 5.3.

Build a linear refutation from the following set of clauses:

1. $S_1 = \{p \vee q \vee r, \neg q \vee r, \neg r, \neg p \vee r\};$
2. $S_2 = \{p \vee \neg r, q \vee r, \neg q \vee r, \neg p \vee \neg r\};$
3. $S_3 = \{q \vee r, \neg p, \neg q \vee r, p \vee \neg r\};$
4. $S_4 = \{\neg p \vee q, p \vee \neg q \vee r, \neg r, p \vee q \vee r, \neg p \vee \neg q\};$
5. $S_5 = \{p \vee r, \neg q, p \vee q \vee \neg r, \neg p \vee \neg r, q \vee r\};$
6. $S_6 = \{p \vee q, \neg p \vee q, \neg p \vee \neg q, p \vee \neg q\};$
7. $S_7 = \{p, q \vee r, \neg p \vee q \vee \neg r, \neg p \vee \neg q\};$
8. $S_8 = \{p \vee \neg q \vee r, q, \neg p \vee \neg q \vee r, \neg p \vee \neg q \vee \neg r, p \vee \neg r\}.$

Exercise 5.4.

Prove the consistency of the following sets of clauses.

1. $S_1 = \{p \vee q \vee r, \neg q \vee r, \neg r, \neg p \vee r\};$
2. $S_2 = \{p \vee \neg r, q \vee r, \neg q \vee r, \neg p \vee \neg r\};$
3. $S_3 = \{q \vee r, \neg p, \neg q \vee r, p \vee \neg r\};$
4. $S_4 = \{\neg p \vee q, p \vee \neg q \vee r, \neg r, p \vee q \vee r, \neg p \vee \neg q\};$
5. $S_5 = \{p \vee r, \neg q, p \vee q \vee \neg r, \neg p \vee \neg r, q \vee r\};$
6. $S_6 = \{p \vee q, \neg p \vee q, \neg p \vee \neg q, p \vee \neg q\};$
7. $S_7 = \{p, q \vee r, \neg p \vee q \vee \neg r, \neg p \vee \neg q\};$
8. $S_8 = \{p \vee \neg q \vee r, q, \neg p \vee \neg q \vee r, \neg p \vee \neg q \vee \neg r, p \vee \neg r\}.$

A Computational Approach to Classical Logics and Circuits

Exercise 5.5.

Using the set-of-support strategy prove the following deductions:

1. $\neg(p \vee q) \rightarrow r, \neg p \vee q \vee r, \neg r \vdash q \wedge \neg r ;$
2. $p \vee \neg r, \neg q \rightarrow r, \neg q \vdash \neg(p \rightarrow q) ;$
3. $q \wedge r \rightarrow p, p \vee q, q \rightarrow r \vdash p ;$
4. $r \rightarrow p \vee q, \neg p \rightarrow r, \neg q \vdash p \wedge \neg q ;$
5. $\neg p \rightarrow q, (q \rightarrow r) \wedge \neg r \vdash p \wedge \neg r ;$
6. $q \rightarrow p, q \vee r, p \rightarrow r \vdash r ;$
7. $\neg p \rightarrow q \vee r, \neg q, p \rightarrow q \vdash \neg(p \vee q) \wedge r ;$
8. $r \rightarrow p, \neg p, q \rightarrow p \vee r \vdash \neg(\neg p \rightarrow q \vee r) .$

Exercise 5.6.

Prove the inconsistency of the following set of clauses using lock resolution.
Try two different indexings for the literals.

1. $S_1 = \{ \neg p(x) \vee q(x), p(a), \neg q(x) \vee \neg r(x), \neg w(a), r(y) \vee w(y) \} ;$
2. $S_2 = \{ p(x) \vee \neg q(x), \neg p(a) \vee r(x), q(x), w(z), \neg r(y) \vee \neg w(y) \} ;$
3. $S_3 = \{ p(x) \vee q(x) \vee r(x), \neg p(a), \neg q(x), \neg w(a), \neg r(y) \vee w(y) \} ;$
4. $S_4 = \{ p(x) \vee q(x), \neg p(x) \vee r(x), \neg q(y) \vee r(y), \neg r(x) \vee w(x), \neg w(f(z)) \}$
5. $S_5 = \{ p(x) \vee q(x), \neg p(a) \vee w(x), \neg q(y) \vee r(y), \neg r(x) \vee w(x), \neg w(a) \} ;$
6. $S_6 = \{ \neg p(x) \vee \neg q(x), p(z) \vee w(x), q(y) \vee w(y) \vee \neg r(y), \neg r(x) \vee \neg w(x), r(g(a,b)) \}$
7. $S_7 = \{ p(x) \vee q(x), \neg p(x), \neg q(f(a)) \vee r(z), \neg w(z), \neg r(y) \vee w(y) \} ;$
8. $S_8 = \{ \neg p(x) \vee q(x) \vee \neg r(x), p(f(b)), \neg q(x), \neg w(y), r(y) \vee w(y) \} .$

Exercise 5.7.

Prove the following deductions using linear resolution

1. $(\forall x)(\forall y)(p(y,x) \wedge q(x) \rightarrow q(y)), (\forall x)(\forall y)(r(y,x) \rightarrow q(y)), r(b,a),$
 $r(b,a), p(c,b) \vdash (\exists z)q(z) ;$
2. $(\forall x)(p(x) \rightarrow r(x)), (\forall y)(r(y) \rightarrow q(y)), p(a), p(b) \vdash (\exists z)q(z) ;$
3. $(\forall x)(\neg p(x) \wedge \neg q(x) \rightarrow r(x)), (\forall y)(r(y) \rightarrow w(y)), (\forall x)(w(x) \rightarrow p(x)),$
 $\neg p(a), \neg p(b), \neg w(c) \vdash (\exists z)q(z) ;$
4. $(\forall x)(p(x) \rightarrow r(x)), (\forall y)(r(y) \rightarrow q(y)), r(a), r(b), \neg r(c) \vdash (\exists z)q(z) ;$
5. $(\forall x)(\neg p(x) \wedge \neg q(x) \rightarrow r(x)), (\forall y)(r(y) \rightarrow w(y)), (\forall x)(w(x) \rightarrow p(x)),$
 $\neg p(a), \neg w(c) \vdash (\exists z)q(z) ;$

Resolution Proof Method

6. $(\forall x)(\forall y)(\neg p(y, x) \rightarrow q(y)), (\forall x)(\forall y)(r(y, x) \wedge q(x) \rightarrow q(y)),$
 $r(b, a), \neg p(a, b) \vdash (\exists z)q(z);$
7. $(\forall x)(p(x) \rightarrow r(x)), (\forall y)(p(y) \rightarrow q(y)), p(a), \neg r(c) \vdash (\exists z)q(z);$
8. $(\forall x)(p(x) \rightarrow r(x)), (\forall y)(p(y) \rightarrow q(y)), p(a), p(b), \neg p(c) \vdash (\exists z)q(z).$

Exercise 5.8.

Using a refinement of predicate resolution prove:

1. the semidistributivity of ' \forall ' over ' \vee ':
 $\vdash (\forall x)p(x) \vee (\forall x)q(x) \rightarrow (\forall x)(p(x) \vee q(x))$ and
 $\nvdash (\forall x)(p(x) \vee q(x)) \rightarrow (\forall x)p(x) \vee (\forall x)q(x)$
2. the semidistributivity of ' \exists ' over ' \wedge ':
 $\vdash (\exists x)(p(x) \wedge q(x)) \rightarrow (\exists x)p(x) \wedge (\exists x)q(x)$ and
 $\nvdash (\exists x)p(x) \wedge (\exists x)q(x) \rightarrow (\exists x)(p(x) \wedge q(x))$
3. $\vdash (\exists x)(p(x) \rightarrow q(x)) \leftrightarrow ((\forall x)p(x) \rightarrow (\exists x)q(x));$
4. the distributivity of ' \exists ' over ' \vee ':
 $\vdash (\exists x)(p(x) \vee q(x)) \leftrightarrow (\exists x)p(x) \vee (\exists x)q(x);$
5. $\vdash (\exists x)p(x) \vee (\exists x)(p(x) \wedge q(x)) \leftrightarrow (\exists x)p(x);$
6. the semidistributivity of ' \forall ' over ' \rightarrow ':
 $\vdash (\forall x)(p(x) \rightarrow q(x)) \rightarrow ((\forall x)p(x) \rightarrow (\forall x)q(x))$ and
 $\nvdash ((\forall x)p(x) \rightarrow (\forall x)q(x)) \rightarrow (\forall x)(p(x) \rightarrow q(x))$
7. $\vdash (\forall x)p(x) \wedge ((\forall x)p(x) \vee (\forall x)q(x)) \leftrightarrow (\forall x)p(x);$
8. the distributivity of ' \forall ' over ' \wedge ':
 $\vdash (\forall x)p(x) \wedge (\forall x)q(x) \leftrightarrow (\forall x)(p(x) \wedge q(x)).$

Exercise 5.9.

Check if the following formulas are theorems using lock resolution.

1. $U_1 = (\forall x)(\forall y)p(x, y) \leftrightarrow (\forall y)(\forall x)p(x, y);$
2. $U_2 = (\exists y)(\exists x)p(x, y) \leftrightarrow (\exists x)(\exists y)p(x, y);$
3. $U_3 = (\forall x)(\forall y)p(x, y) \leftrightarrow (\exists x)(\forall y)p(x, y);$
4. $U_4 = (\exists x)(\forall y)p(x, y) \leftrightarrow (\forall y)(\exists x)p(x, y);$
5. $U_5 = (\exists y)(\exists x)p(x, y) \leftrightarrow (\forall x)(\exists y)p(x, y);$
6. $U_6 = (\forall y)(\forall x)p(x, y) \leftrightarrow (\forall x)(\exists y)p(x, y);$
7. $U_7 = (\exists y)(\exists x)p(x, y) \leftrightarrow (\exists x)(\forall y)p(x, y);$
8. $U_8 = (\forall y)(\exists x)p(x, y) \leftrightarrow (\exists y)(\exists x)p(x, y).$

6. REASONING MODELING AND PROGRAM VERIFICATION

Propositional logic and predicate logic can be used to model simple types of human common-sense reasoning: decide if a statement (*conjecture, conclusion*) is derivable from a set of statements (*hypotheses*). The hypotheses and the conclusion are expressed in natural language. Since predicate logic allows reasoning about the objects of some universe and the relations among these objects, this formalism is also appropriate to model mathematical reasoning. The proof methods presented in the previous chapters are applied in numerous examples.

6.1. Common-sense reasoning modeling using propositional logic

Example 6.1. Party

Hypotheses:

- H_1 : Mary will go to the party if Lucy will go and George will not go.
- H_2 : If John will go to the party then Lucy will go too.
- H_3 : If John is in town he will go to the party.
- H_4 : George is sick and can't go to the party.
- H_5 : Yesterday John has returned in town from Paris.

Conclusion:

- C : Will Mary go to the party?

We have to check if the following deduction holds.

$$H_1, H_2, H_3, H_4, H_5 \vdash C$$

The following notations for the propositional variables are used:

- M – Mary will go to the party
- L – Lucy will go to the party
- G – George will go to the party
- J – John will go to the party
- Jt – John is in town

The hypotheses and the conclusion are transformed into propositional formulas as follows:

Reasoning modeling and program verification

$$\begin{array}{lll} H_1 : L \wedge \neg G \rightarrow M & H_2 : J \rightarrow L & H_3 : Jt \rightarrow J \\ H_4 : \neg G & H_5 : Jt & C : M \end{array}$$

The definition of the deduction (Definition 1.8) and the axiomatic system are used:

$$\begin{aligned} f_1 &= H_1 : L \wedge \neg G \rightarrow M \text{ (hypothesis)} \\ f_2 &= H_2 : J \rightarrow L \text{ (hypothesis)} \\ f_3 &= H_3 : Jt \rightarrow J \text{ (hypothesis)} \\ f_4 &= H_4 : \neg G \text{ (hypothesis)} \\ f_5 &= H_5 : Jt \text{ (hypothesis)} \\ f_5, f_3 &\vdash_{mp} J : f_6 \\ f_6, f_3 &\vdash_{mp} L : f_7 \\ f_4, f_7 &\vdash L \wedge \neg G : f_8 \text{ (conjunction in conclusions)} \\ f_8, f_1 &\vdash_{mp} M : f_9 = C \end{aligned}$$

Modus ponens inference rule was applied to derive the formulas f_6, f_7, f_9 .

The sequence of formulas: $(f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9)$ is the deduction of C from the hypotheses, therefore, based on the hypotheses, *Mary will go to the party*.

Example 6.2.

Hypotheses:

H_1 : If it is sunny, Diane and Alice go to the swimming pool.

H_2 : Ben goes to the swimming pool on Thursdays.

H_3 : It was sunny last Thursday.

Conclusion:

C : Did Diane meet Ben at the swimming pool last Thursday?

We have to check if the following deduction holds:

$H_1, H_2, H_3 \vdash C$

The resolution, a refutation proof method, is applied and according to the Theorem of soundness and completeness of resolution, we have to check if:

$\text{CNF}(H_1 \wedge H_2 \wedge H_3 \wedge \neg C) \vdash_{\text{Res}} \square$.

We use the following notations for the propositional variables:

D – Diane goes to the swimming pool

A – Alice goes to the swimming pool

B – Ben goes to the swimming pool

S – it is sunny

Th – it is Thursday

A Computational Approach to Classical Logics and Circuits

The following propositional formulas and their conjunctive normal forms correspond to the hypotheses and the negation of the conclusion:

$$H_1 : S \rightarrow D \wedge A \equiv \neg S \vee (D \wedge A) \equiv (\neg S \vee D) \wedge (\neg S \vee A) : C_1 \wedge C_2$$

$$H_2 : Th \rightarrow B \equiv \neg Th \vee B : C_3$$

$$H_3 : S \wedge Th : C_4 \wedge C_5$$

$$C : Th \wedge D \wedge B$$

$$\neg C : \neg(Th \wedge D \wedge B) \equiv \neg Th \vee \neg D \vee \neg B : C_6$$

The hypotheses and the negation of the conclusion were transformed into CNFs and the set of clauses $S = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ was obtained:

If we want to apply the set-of-support strategy, we have to avoid resolving two clauses belonging to the consistent subset of clauses: $\{C_1, C_2, C_3, C_4, C_5\}$ provided by the hypotheses. The support set of S is $Y = \{C_6\}$ and corresponds to the negation of the conclusion. All the resolvents are added to Y .

The refutation from S (the derivation of \square from S) is provided below:

$$C_7 = \text{Res}(C_5, C_6) = \neg D \vee \neg B$$

$$C_8 = \text{Res}(C_7, C_3) = \neg D \vee \neg Th$$

$$C_9 = \text{Res}(C_8, C_5) = \neg D$$

$$C_{10} = \text{Res}(C_9, C_1) = \neg S$$

$$C_{11} = \text{Res}(C_{10}, C_4) = \square$$

We have proved that $\text{CNF}(H_1 \wedge H_2 \wedge H_3 \wedge \neg C) \vdash_{\text{Res}} \square$, so C is deducible from the hypotheses, therefore: "*Diane met Ben at the swimming pool last Thursday*".

We shall prove the deduction $H_1, H_2, H_3 \vdash_{\text{Res}} C$ using another proof method, a direct method, the sequent calculus.

The up-side down binary tree corresponding to the reduction process of the initial sequent $H_1, H_2, H_3 \Rightarrow C$ is built.

The initial sequent was reduced to five basic sequents (overlined), therefore it is a true sequent and according to the theorem of soundness and completeness of this method we conclude that C is deducible from the hypotheses H_1, H_2, H_3 .

$$\begin{array}{c}
 \frac{\underline{D}, A, B, S, Th \Rightarrow D}{B, \underline{S}, Th \Rightarrow \underline{S}, D} (\wedge_l) \\
 \frac{B, \underline{S}, Th \Rightarrow \underline{S}, D}{D \wedge A, B, S, Th \Rightarrow D} (\rightarrow_l) \\
 \dots \\
 \frac{S \rightarrow D \wedge A, \underline{B}, S, Th \Rightarrow \underline{B}}{S \rightarrow D \wedge A, B, S, Th \Rightarrow D} (\wedge_r) \\
 \frac{S \rightarrow D \wedge A, S, Th \Rightarrow Th, D \wedge B}{S \rightarrow D \wedge A, B, S, Th \Rightarrow D \wedge B} (\rightarrow_l) \text{ for } Th \rightarrow B \\
 \dots \\
 \frac{S \rightarrow D \wedge A, Th \rightarrow B, S, Th \Rightarrow D \wedge B}{S \rightarrow D \wedge A, Th \rightarrow B, S, Th \Rightarrow Th} (\wedge_l, \wedge_r)
 \end{array}$$

Example 6.3.

A client describes the requirements of a software application:

- R_1 . If condition A is satisfied then condition B must also be satisfied.
- R_2 . If conditions B and C are satisfied, then D must also be satisfied.
- R_3 . If condition D is satisfied then condition A is not satisfied.
- R_4 . If condition C is satisfied then A must also be satisfied.
- R_5 . If A is satisfied then D or C are satisfied.
- R_6 . C is satisfied if neither B nor A are satisfied.
- R_7 . B is not satisfied if C is not satisfied.

Are these requirements simultaneously satisfiable?

In order to answer the question we have to check the consistency/inconsistency of $U = R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5 \wedge R_6 \wedge R_7$.
 The statements corresponding to the requirements are transformed into propositional formulas and further into their conjunctive normal forms:

A Computational Approach to Classical Logics and Circuits

$$R_1. A \rightarrow B \equiv \neg A \vee B: C_1$$

$$R_2. B \wedge C \rightarrow D \equiv \neg B \vee \neg C \vee D: C_2$$

$$R_3. D \rightarrow \neg A \equiv \neg D \vee \neg A: C_3$$

$$R_4. C \rightarrow A \equiv \neg C \vee A: C_4$$

$$R_5. A \rightarrow C \vee D \equiv \neg A \vee C \vee D: C_5$$

$$R_6. \neg A \wedge \neg B \rightarrow C \equiv A \vee B \vee C: C_6$$

$$R_7. \neg C \rightarrow \neg B \equiv C \vee \neg B: C_7$$

$$S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}.$$

We have to check the consistency/inconsistency of the set S of clauses corresponding to the conjunction of the requirements. The general resolution procedure is applied:

$$C_8 = \text{Res}(C_1, C_7) = C \vee \neg A$$

$$C_9 = \text{Res}(C_7, C_6) = C \vee A$$

$$C_{10} = \text{Res}(C_8, C_9) = C$$

$$C_{11} = \text{Res}(C_4, C_{10}) = A$$

$$C_{12} = \text{Res}(C_{11}, C_1) = B$$

$$C_{13} = \text{Res}(C_{11}, C_3) = \neg D$$

$$C_{14} = \text{Res}(C_{13}, C_2) = \neg B \vee \neg C$$

$$C_{15} = \text{Res}(C_{14}, C_{12}) = \neg C$$

$$C_{16} = \text{Res}(C_{10}, C_{15}) = \square$$

The empty clause was derived from S , so S is inconsistent and the requirements are contradictory.

Resolution as a proof method is very efficient compared to the truth table method as we shall see in the following.

To check the consistency/inconsistency of the requirements $R_1 - R_7$ we have to build the truth table of the propositional formula representing the conjunction of all the requirements: $U = R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5 \wedge R_6 \wedge R_7$.

Each requirement has a conjunctive normal form composed of one clause (see the normalization process presented before), thus:

$$U \equiv C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \wedge C_7.$$

The formula U has 4 propositional variables: A, B, C, D , so its truth table has $2^4 = 16$ rows, corresponding to all 16 interpretations which assign in all the possible ways the truth values $\{T, F\}$ to the variables A, B, C, D .

Reasoning modeling and program verification

	A	B	C	D	R_1	R_2	R_3	R_4	R_5	R_6	R_7	U
i_1	T	T	T	T	T	T	F	T	T	T	T	F
i_2	T	T	T	F	T	F	T	T	T	T	T	F
i_3	T	T	F	T	T	T	F	T	T	T	F	F
i_4	T	T	F	F	T	T	T	T	F	T	F	F
i_5	T	F	T	T	F	T	F	T	T	T	T	F
i_6	T	F	T	F	F	T	T	T	T	T	T	F
i_7	T	F	F	T	F	T	F	T	T	T	T	F
i_8	T	F	F	F	F	T	T	T	F	T	T	F
i_9	F	T	T	T	T	T	T	F	T	T	T	F
i_{10}	F	T	T	F	T	F	T	F	T	T	T	F
i_{11}	F	T	F	T	T	T	T	T	T	T	F	F
i_{12}	F	T	F	F	T	T	T	T	T	T	F	F
i_{13}	F	F	T	T	T	T	T	F	T	T	T	F
i_{14}	F	F	T	F	T	T	T	F	T	T	T	F
i_{15}	F	F	F	T	T	T	T	T	T	F	T	F
i_{16}	F	F	F	F	T	T	T	T	T	T	F	F

Each requirement is evaluated in all 16 interpretations and the column (truth table) of U is obtained as the conjunction of the columns of the requirements.

For each row (interpretation) there is at least one value F in the columns of $R_1 - R_7$, meaning that in each interpretation at least one requirement is evaluated as false, so U is evaluated as false in that interpretation.

The column of U contains only the truth value F , so U is evaluated as false in all 16 interpretations, therefore U is an inconsistent formula.

The conclusion is that *the requirements are contradictory, they cannot be satisfied simultaneously by the software application.*

Note that the truth table method is time consuming and it is not efficient at the implementation level.

Example 6.4.

Consider the following hypotheses:

H_1 . Mary will go to London this summer if both her friends Kate and Susan go.

H_2 . If Kate passes the English exam in May then she will go to London.

H_3 . Kate was in hospital from April until July and she didn't take the English exam.

H_4 . This summer Susan will go to London on a business trip.

and the conclusion:

C. Will Mary go to London this summer?

A Computational Approach to Classical Logics and Circuits

Using the semantic tableaux method check if the conclusion C is a logical consequence of the set of hypotheses: $\{H_1, H_2, H_3, H_4\}$.

We transform the hypotheses and the conclusion into propositional formulas:

$$H_1 : K \wedge S \rightarrow M$$

$$H_2 : KE \rightarrow K$$

$$H_3 : \neg KE$$

$$H_4 : S$$

$$C : M$$

where the notations for the propositional variables are as follows:

M – Mary will go to London

S – Susan will go to London

K – Kate will go to London

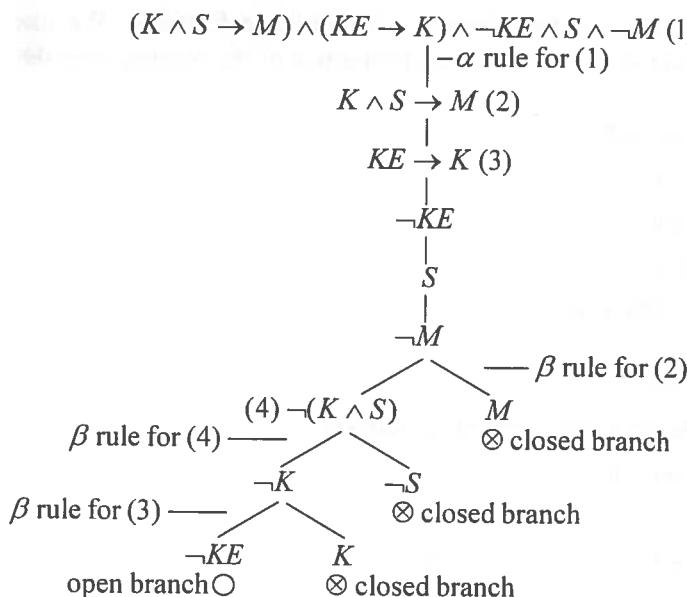
KE – Kate passed the English exam

The semantic tableau method is a refutation proof method, thus we have to negate the conclusion and use the Theorem of soundness and completeness:

$H_1, H_2, H_3, H_4 \vdash C$ if and only if

$H_1 \wedge H_2 \wedge H_3 \wedge H_4 \wedge \neg C$ has a closed semantic tableau.

The semantic tableau corresponding to the conjunction of the hypotheses and the negation of the conclusion is depicted below.



Reasoning modeling and program verification

We have obtained a complete and open tableau with one open branch (the left-most one) and three closed branches containing the following pairs of opposite literals: $(\neg K, K), (S, \neg S), (M, \neg M)$.

Therefore $H_1, H_2, H_3, H_4 \not\models C$ and based on the hypotheses we conclude that '*Mary will not go to London this summer*'.

6.2. Reasoning modeling using predicate logic

Example 6.5.

Hypotheses:

H_1 : All hummingbirds are richly colored.

H_2 : No large birds live on honey.

H_3 : Birds that do not live on honey are dull in color.

H_4 : *Piky* is a hummingbird.

Conclusions:

D_1 : All hummingbirds are small.

D_2 : *Piky* is a small bird and lives on honey.

We have to check if the following deductions hold or not.

$H_1, H_2, H_3 \vdash D_1$ and $H_1, H_2, H_3, H_4 \vdash D_2$.

The natural language sentences are transformed into predicate formulas. We use the following unary predicate symbols to express properties of the objects from the universe of birds.

$hb(x)$ – x is a hummingbird,

$rc(x)$ – x is richly colored

$sb(x)$ – x is a small bird

$lh(x)$ – x lives on honey

Piky is a constant of the universe.

$H_1 : (\forall x)(hb(x) \rightarrow rc(x))$

$H_2 : \neg(\exists x)(\neg sb(x) \wedge lh(x)) \equiv (\forall x)(\neg sb(x) \rightarrow \neg lh(x))$

$H_3 : (\forall x)(\neg lh(x) \rightarrow \neg rc(x))$

$H_4 : hb(Piky)$

$D_1 : (\forall x)(hb(x) \rightarrow sb(x))$

$D_2 : sb(Piky) \wedge lh(Piky)$

A Computational Approach to Classical Logics and Circuits

We prove the deduction $H_1, H_2, H_3 \vdash D_1$ using the sequent calculus method, a direct and syntactic proof method.

The reduction process of the initial sequent $H_1, H_2, H_3 \Rightarrow D_1$ is depicted below:

$$\begin{array}{c}
 \frac{\overline{rc(t), H_1, H_2, H_3, hb(t), lh(t) \Rightarrow lh(t), sb(t)}}{(\neg r)} \quad \frac{\overline{rc(t), H_1, H_2, H_3, hb(t) \Rightarrow rc(t), lh(t), sb(t)}}{rc(t), H_1, H_2, H_3, hb(t) \Rightarrow lh(t), sb(t)} \quad (\neg l) \\
 \frac{\overline{rc(t), C, H_1, H_2, H_3, hb(t) \Rightarrow sb(t)}}{rc(t), C, H_1, H_2, H_3, hb(t) \Rightarrow lh(t), sb(t)} \quad \frac{\overline{rc(t), C, H_1, H_2, H_3, hb(t) \Rightarrow lh(t), sb(t)}}{rc(t), \neg lh(t), C, H_1, H_2, H_3, hb(t) \Rightarrow sb(t)} \quad (\neg l) \\
 \frac{\overline{B, C, H_1, H_2, H_3, hb(t) \Rightarrow hb(t), sb(t)}}{B, C, H_1, H_2, H_3, hb(t) \Rightarrow hb(t), sb(t)} \quad \frac{\overline{rc(t), B, C, H_1, H_2, H_3, hb(t) \Rightarrow sb(t)}}{rc(t), B, C, H_1, H_2, H_3, hb(t) \Rightarrow sb(t)} \quad (\neg r) \\
 \frac{\overline{hb(t) \rightarrow rc(t), \neg sb(t) \rightarrow \neg lh(t), \neg lh(t) \rightarrow \neg rc(t), H_1, H_2, H_3 \Rightarrow hb(t) \rightarrow sb(t)}}{hb(t) \rightarrow rc(t), \neg sb(t) \rightarrow \neg lh(t), \neg lh(t) \rightarrow \neg rc(t), H_1, H_2, H_3 \Rightarrow hb(t) \rightarrow sb(t)} \quad (\neg r) \\
 \frac{(\forall x)(hb(x) \rightarrow rc(x)), (\forall y)(\neg sb(y) \rightarrow \neg lh(y)), (\forall z)(\neg lh(z) \rightarrow \neg rc(z)) \Rightarrow hb(t) \rightarrow sb(t)}{(\forall x)(hb(x) \rightarrow rc(x)), (\forall y)(\neg sb(y) \rightarrow \neg lh(y)), (\forall z)(\neg lh(z) \rightarrow \neg rc(z)) \Rightarrow hb(t) \rightarrow sb(t)} \quad (\forall_r) \\
 \end{array}$$

- the bound variables from the formulas H_1, H_2, H_3, D_1 were renamed such that they are distinct in the initial sequent.

$$\begin{aligned}
 H_1 &: (\forall x)(hb(x) \rightarrow rc(x)), \\
 H_2 &: (\forall y)(\neg sb(y) \rightarrow \neg lh(y)), \\
 H_3 &: (\forall z)(\neg lh(z) \rightarrow \neg rc(z)), \\
 D_1 &: (\forall t)(hb(t) \rightarrow sb(t))
 \end{aligned}$$

Reasoning modeling and program verification

- in the first step the universal quantified variable t from the consequent becomes a free variable: the rule (\forall_r) is applied;
- in the second step the rule (\forall_l) is applied three times. All three universal quantified formulas H_1, H_2, H_3 , from the antecedent, are instantiated using t and A, B, C open formulas are obtained. H_1, H_2, H_3 are duplicated (for further instantiations);
$$A = hb(t) \rightarrow rc(t),$$
$$B = \neg sb(t) \rightarrow \neg lh(t),$$
$$C = \neg lh(t) \rightarrow \neg rc(t).$$
- in the third step the rule (\rightarrow_r) is applied;
- in the following steps the formulas A, B, C are decomposed using the rule (\rightarrow_l) ;
- the complete reduction tree has four leaf nodes containing basic sequents.

The initial sequent $H_1, H_2, H_3 \Rightarrow D_1$ was reduced to four basic sequents, so it is a true sequent and the deduction $H_1, H_2, H_3 \vdash D_1$ holds.

Based on the hypotheses we conclude that:

'All hummingbirds are small'.

The deduction $H_1, H_2, H_3, H_4 \vdash D_2$ is proved by contradiction by applying the semantic tableaux method.

The semantic tableau of $H_1 \wedge H_2 \wedge H_3 \wedge H_4 \wedge \neg D_2$ is built.

The copies of the universal quantified formulas H_1, H_2, H_3 are not used further because no new constants were introduced after their instantiations.

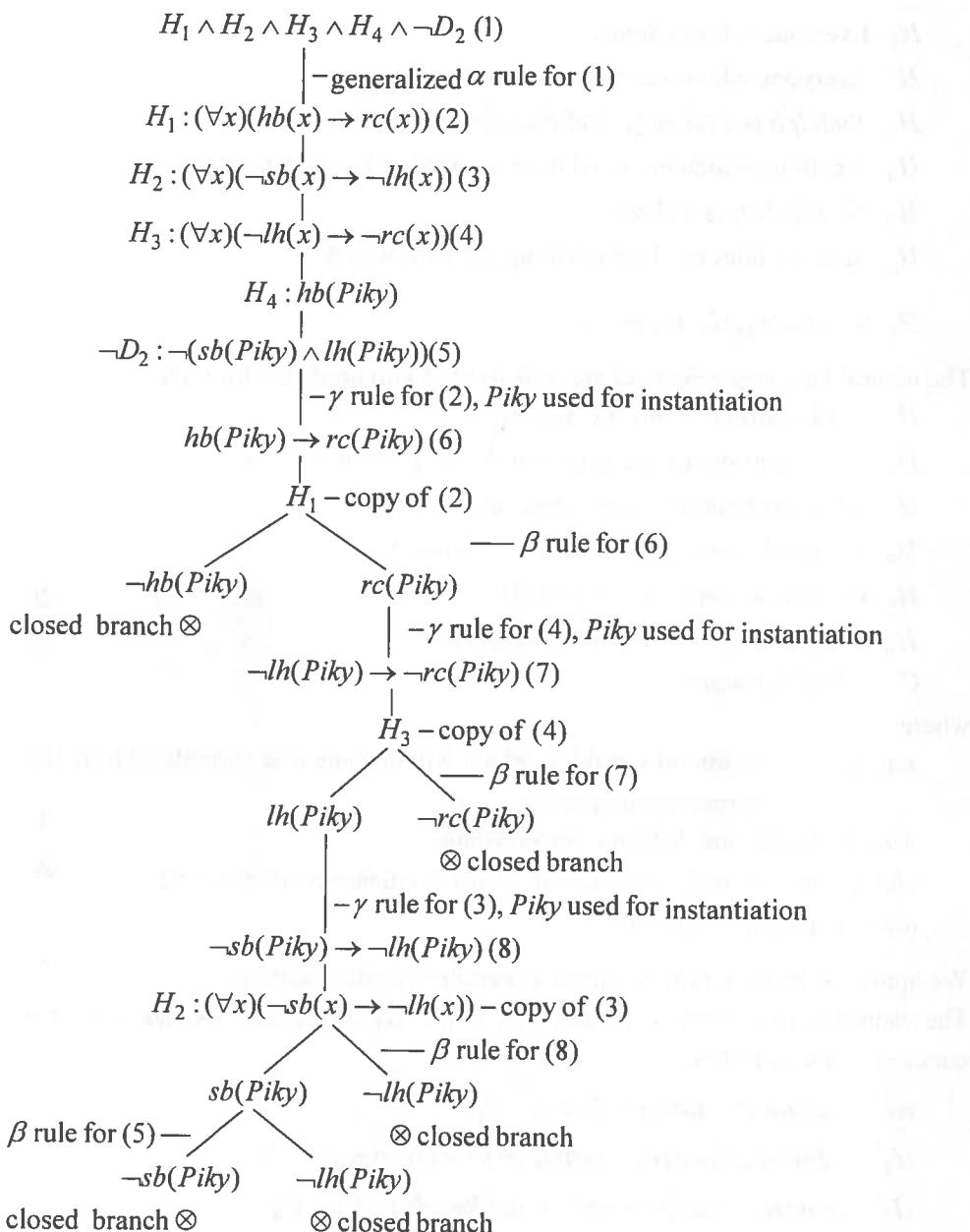
The semantic tableau is closed, having five closed branches. The set of the hypotheses is consistent, therefore there is a contradiction among the hypotheses and the negation of the conclusion.

According to the 'reduction ad absurdum' principle, the conclusion D_2 is derivable from the hypotheses H_1, H_2, H_3, H_4 .

Based on the hypotheses we conclude that:

'Piky is a small bird and lives on honey.'

A Computational Approach to Classical Logics and Circuits



Example 6.6.

Consider the following set of hypotheses $\{H_1, H_2, H_3, H_4, H_5, H_6\}$ and check the validity of the conclusion (C):

C. Scrooge is not a child.

Reasoning modeling and program verification

H_1 . Every child loves *Santa*.

H_2 . Everyone who loves *Santa* loves any reindeer.

H_3 . *Rudolph* is a reindeer, and *Rudolph* has a red nose.

H_4 . Anything which has a red nose is weird or is a clown.

H_5 . No reindeer is a clown.

H_6 . *Scrooge* does not love anything which is weird.

$H_1, H_2, H_3, H_4, H_5, H_6 \stackrel{?}{\vdash} C$.

The natural language sentences are transformed into predicate formulas:

$H_1 : (\forall x)(child(x) \rightarrow loves(x, Santa))$

$H_2 : (\forall x)(\forall y)(loves(x, Santa) \wedge reindeer(y) \rightarrow loves(x, y))$

$H_3 : reindeer(Rudolf) \wedge red_nose(Rudolf)$

$H_4 : (\forall z)(red_nose(z) \rightarrow weird(z) \vee clown(z))$

$H_5 : (\forall s)(reindeer(s) \rightarrow \neg clown(s))$

$H_6 : (\forall t)(weird(t) \rightarrow \neg loves(Scrooge, t))$

$C : \neg child(Scrooge)$

where:

x, u, y, z, s, t are bound variables, which will become free variables during the normalization process,

Rudolf, *Santa* and *Scrooge* are constants,

child, *reindeer*, *red _ nose*, *weird*, *clown* are unary predicates and

loves is a binary predicate

We apply a refutation proof method: general predicate resolution.

The clausal normal forms corresponding to the hypotheses and the negation of the conclusion are as follows:

$H_1^C : \neg child(x) \vee loves(x, Santa) = C_1$

$H_2^C : \neg loves(x, Santa) \vee \neg reindeer(y) \vee loves(x, y) = C_2$

$H_3^C : reindeer(Rudolf) \wedge red_nose(Rudolf) = C_3 \wedge C_4$

$H_4^C : \neg red_nose(z) \vee weird(z) \vee clown(z) = C_5$

$H_5^C : \neg reindeer(s) \vee \neg clown(s) = C_6$

$H_6^C : \neg weird(t) \vee \neg loves(Scrooge, t) = C_7$

$(\neg C)^C : child(Scrooge) = C_8$

A Computational Approach to Classical Logics and Circuits

To the set of clauses $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$, general predicate resolution is applied.

In the resolution process, the following resolvents are obtained:

$$C_9 = \text{Res}^{\text{Pr}}_{[x \leftarrow \text{Scrooge}]}(C_8, C_1) = \text{loves}(\text{Scrooge}, \text{Santa})$$

$$C_{10} = \text{Res}^{\text{Pr}}_{[s \leftarrow \text{Rudolf}]}(C_3, C_6) = \neg \text{clown}(\text{Rudolf})$$

$$C_{11} = \text{Res}^{\text{Pr}}_{[z \leftarrow \text{Rudolf}]}(C_4, C_5) = \text{weird}(\text{Rudolf}) \vee \text{clown}(\text{Rudolf})$$

$$C_{12} = \text{Res}^{\text{Pr}}(C_{10}, C_{11}) = \text{weird}(\text{Rudolf})$$

$$C_{13} = \text{Res}^{\text{Pr}}_{[t \leftarrow \text{Rudolf}]}(C_{12}, C_7) = \neg \text{loves}(\text{Scrooge}, \text{Rudolf})$$

$$C_{14} = \text{Res}^{\text{Pr}}_{[y \leftarrow \text{Rudolf}]}(C_2, C_3) = \neg \text{loves}(x, \text{Santa}) \vee \text{loves}(x, \text{Rudolf})$$

$$C_{15} = \text{Res}^{\text{Pr}}_{[x \leftarrow \text{Scrooge}]}(C_{13}, C_{14}) = \neg \text{loves}(\text{Scrooge}, \text{Santa})$$

$$C_{16} = \text{Res}^{\text{Pr}}(C_9, C_{15}) = \square$$

The most general unifier generated during the resolution process is the substitution:

$$[x \leftarrow \text{Scrooge}, y \leftarrow \text{Rudolf}, z \leftarrow \text{Rudolf}, s \leftarrow \text{Rudolf}, t \leftarrow \text{Rudolf}]$$

$S \vdash_{\text{Res}}^{\text{Pr}} \square$, therefore S is an inconsistent set and the deduction

$$H_1, H_2, H_3, H_4, H_5, H_6 \vdash C \text{ holds.}$$

From the hypotheses we conclude that '*Scrooge is not a child*'.

Example 6.7. Succession to the British throne

Hypotheses:

H_1 : If x is the king and y is his oldest son, then y can become the king.

H_2 : If x is the king and y defeats x , then y will become the king.

H_3 : *RichardIII* is the king.

H_4 : *HenryVII* defeated *RichardIII*.

H_5 : *HenryVIII* is *HenryVII*'s oldest son.

Conclusion:

C : Can *HenryVIII* become the king?

Check if the conclusion C is derivable from the set of hypotheses $\{H_1, H_2, H_3, H_4, H_5\}$ using a syntactic proof method.

We transform the hypotheses and the conclusion into predicate formulas using:

variables: x, y, z, t

constants: *RichardIII*, *HenryVII*, *HenryVIII*

predicate symbols: unary: *king*, binary: *oldest.son*, *defeat*

Reasoning modeling and program verification

$H_1 : (\forall x)(\forall y)(king(x) \wedge oldest_son(y, x) \rightarrow king(y))$

$H_2 : (\forall z)(\forall t)(king(z) \wedge defeat(t, z) \rightarrow king(t))$

$H_3 : king(RichardIII)$

$H_4 : defeat(HenryVII, RichardIII)$

$H_5 : oldest_son(HenryVIII, HenryVII)$

$C : king(HenryVIII)$

We prove the deduction $H_1, H_2, H_3, H_4, H_5 \vdash C$ using the axiomatic system of predicate logic and Definition 2.2.

The following sequence of predicate formulas: $(f_1, f_2, \dots, f_{13})$ is generated.

The inference rules used in the deduction process are **universal instantiation**: *univ_inst* and **modus ponens**: *mp*.

$f_1 = H_1 : (\forall x)(\forall y)(king(x) \wedge oldest_son(y, x) \rightarrow king(y))$

$f_2 = H_2 : (\forall z)(\forall t)(king(z) \wedge defeat(t, z) \rightarrow king(t))$

$f_3 = H_3 : king(RichardIII)$

$f_4 = H_4 : defeat(HenryVII, RichardIII)$

$f_5 = H_5 : oldest_son(HenryVIII, HenryVII)$

$f_2 \vdash_{univ_inst} (\forall t)(king(RichardIII) \wedge defeat(t, RichardIII) \rightarrow king(t)) : f_6,$

the universal variable z was instantiated using the constant *RichardIII*

$f_6 \vdash_{univ_inst} king(RichardIII) \wedge defeat(HenryVII, RichardIII) \rightarrow king(HenryVII) : f_7$

the universal variable t was instantiated using the constant *HenryVII*

$f_3 \wedge f_4 = king(RichardIII) \wedge defeat(HenryVII, RichardIII) : f_8$ (conjunction in conclusions)

$f_8, f_7 \vdash_{mp} king(HenryVII) : f_9$

$f_1 \vdash_{univ_inst} (\forall y)(king(HenryVII) \wedge oldest_son(y, HenryVII) \rightarrow king(y)) : f_{10},$

the universal variable x was instantiated using the constant *HenryVII*

$f_{10} \vdash_{univ_inst} king(HenryVII) \wedge oldest_son(HenryVIII, HenryVII) \rightarrow king(HenryVIII) : f_{11}$

the universal variable y was instantiated using the constant *HenryVIII*

$f_5 \wedge f_9 = king(HenryVII) \wedge oldest_son(HenryVIII, HenryVII) : f_{12}$ (conjunction in conclusions)

$f_{10}, f_{12} \vdash_{mp} king(HenryVIII) : f_{13}=C$

The sequence of formulas $(f_1, f_2, \dots, f_{13})$ is the deduction of C from the hypotheses H_1, H_2, H_3, H_4, H_5 , therefore based on the hypotheses we conclude that '*HenryVIII can become the king*'.

A Computational Approach to Classical Logics and Circuits

Example 6.8.

Hypotheses:

H_1 : Any Computer Science student likes *logic* and likes any programming language.

H_2 : Someone who likes *logic* is a Computer Science student or a Philosophy student.

H_3 : *Java* is a programming language.

H_4 : *John* doesn't like *Java* but he likes *logic*.

Conclusion:

C : *John* is a Philosophy student but he is not a Computer Science student.

The hypotheses and the conclusion are translated into first-order language.

Symbols used:

- x, y, z are variables;
- *logic*, *Java*, *John* are constants,
- CS and P are unary predicate symbols with the meanings:

$CS(x)$: ' x is a Computer Science student'

$P(x)$: ' x is a Philosophy student'

- pl is a unary predicate, $pl(x)$: ' x is a programming language'

- *likes* is a binary predicate, $likes(x, y)$: ' x likes y '

$$H_1 : (\forall x)(\forall y)(CS(x) \wedge pl(y) \rightarrow likes(x, logic) \wedge likes(x, y))$$

$$H_2 : (\forall z)(likes(z, logic) \rightarrow CS(z) \vee P(z))$$

$$H_3 : pl(Java)$$

$$H_4 : likes(John, logic) \wedge \neg likes(John, Java)$$

$$C : \neg CS(John) \wedge P(John)$$

We transform the hypotheses and the negation of the conclusion into clausal normal forms:

$$\begin{aligned} (H_1)^c &= \neg CS(x) \vee \neg pl(y) \vee likes(x, logic) \wedge likes(x, y) \equiv \\ &\equiv (\neg CS(x) \vee \neg pl(y) \vee likes(x, logic)) \wedge (\neg CS(x) \vee \neg pl(y) \vee likes(x, y)) = C_1 \wedge C_2 \\ C_1 &= \neg CS(x) \vee \neg pl(y) \vee likes(x, logic), \quad C_2 = \neg CS(x) \vee \neg pl(y) \vee likes(x, y) \end{aligned}$$

$$(H_2)^c = \neg likes(z, logic) \vee CS(z) \vee P(z) = C_3$$

$$(H_3)^c = pl(Java) = C_4$$

$$(H_4)^c = likes(John, logic) \wedge \neg likes(John, Java) = C_5 \wedge C_6$$

$$C_5 = likes(John, logic), \quad C_6 = \neg likes(John, Java)$$

$$(\neg C)^c = CS(John) \vee \neg P(John) = C_7$$

Reasoning modeling and program verification

We apply linear resolution to the set $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$ of clauses, with C_3 as the top clause.

$$C_8 = \text{Res}_{[z \leftarrow \text{John}]}^{\text{Pr}}(C_3, C_5) = CS(\text{John}) \vee P(\text{John})$$

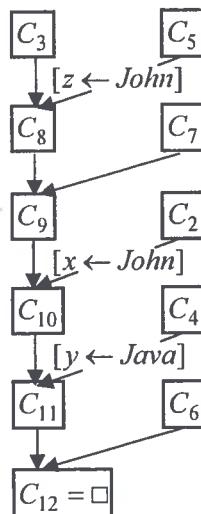
$$C_9 = \text{Res}(C_8, C_7) = CS(\text{John})$$

$$C_{10} = \text{Res}_{[x \leftarrow \text{John}]}^{\text{Pr}}(C_9, C_2) = \neg pl(y) \vee likes(\text{John}, y)$$

$$C_{11} = \text{Res}_{[y \leftarrow \text{Java}]}^{\text{Pr}}(C_{10}, C_4) = likes(\text{John}, \text{Java})$$

$$C_{12} = \text{Res}(C_{11}, C_6) = \square$$

The linear refutation process is represented graphically as follows:



$S \vdash_{\text{Res}}^{lm} \square$, therefore S is an inconsistent set and based on the hypotheses we conclude that:

'John is a Philosophy student but he is not a Computer Science student'.

Example 6.9. Reasoning modeling in geometry using predicate logic

The domain is the set of all the lines in a plane. We use variables: x, y, z to denote arbitrary objects (lines) and constants: d, d_1, d_2 to denote constant objects (lines).

Hypotheses:

H_1 : If x is perpendicular to y then x intersects y .

H_2 : If x is parallel to y then x doesn't intersect y .

H_3 : If x is perpendicular to y and z is perpendicular to y then x is parallel to z .

A Computational Approach to Classical Logics and Circuits

$H_4 : d_1$ is perpendicular to d .

$H_5 : d$ is perpendicular to d_2 .

Conclusion.

$C : d_2$ does not intersect d_1 .

Check if the conclusion C is derivable from the set of hypotheses $\{H_1, H_2, H_3, H_4, H_5\}$ using a syntactic proof method.

In order to translate the hypotheses and the conclusion into first-order formulas we use the binary predicate symbols with the same names as the corresponding geometric relations. Distinct names for the bound variables in the first-order formulas are used.

$H_1 : (\forall x_1)(\forall x_2)(\text{perpendicular}(x_1, x_2) \rightarrow \text{intersects}(x_1, x_2))$

$H_2 : (\forall x_3)(\forall x_4)(\text{parallel}(x_3, x_4) \rightarrow \neg \text{intersects}(x_3, x_4))$

$H_3 : (\forall x_5)(\forall x_6)(\forall x_7)(\text{perpendicular}(x_5, x_6) \wedge \text{perpendicular}(x_7, x_6) \rightarrow \text{parallel}(x_5, x_7))$

$H_4 : \text{perpendicular}(d_1, d)$

$H_5 : \text{perpendicular}(d, d_2)$

$C : \neg \text{intersects}(d_2, d_1)$

We have to add predicate formulas which express the properties of *symmetry* for the geometric relations: *parallel*, *perpendicular*, *intersects*.

$P_1 : (\forall x_8)(\forall x_9)(\text{parallel}(x_8, x_9) \rightarrow \text{parallel}(x_9, x_8))$

$P_2 : (\forall x_{10})(\forall x_{11})(\text{perpendicular}(x_{10}, x_{11}) \rightarrow \text{perpendicular}(x_{11}, x_{10}))$

$P_3 : (\forall x_{12})(\forall x_{13})(\text{intersects}(x_{12}, x_{13}) \rightarrow \text{intersects}(x_{13}, x_{12}))$

The properties of *reflexivity* for *parallel* and *intersects*, and *transitivity* for *parallel* must be also added:

$P_4 : (\forall x_{14})\text{parallel}(x_{14}, x_{14})$

$P_5 : (\forall x_{15})\text{intersects}(x_{15}, x_{15})$

$P_6 : (\forall x_{16})(\forall x_{17})(\forall x_{18})(\text{parallel}(x_{16}, x_{17}) \wedge \text{parallel}(x_{17}, x_{18}) \rightarrow \text{parallel}(x_{16}, x_{18}))$

The clausal normal forms of the hypotheses: H_1, H_2, H_3, H_4, H_5 , the negation of the conclusion C and the properties: $P_1, P_2, P_3, P_4, P_5, P_6$ are as follows:

$H_1^C : \neg \text{perpendicular}(x_1, x_2) \vee \text{intersects}(x_1, x_2) : C_1$

$H_2^C : \neg \text{parallel}(x_3, x_4) \vee \neg \text{intersects}(x_3, x_4) : C_2$

$H_3^C : \neg \text{perpendicular}(x_5, x_6) \vee \neg \text{perpendicular}(x_7, x_6) \vee \text{parallel}(x_5, x_7) : C_3$

$H_4^C : \text{perpendicular}(d_1, d) : C_4$

Reasoning modeling and program verification

$$H_5^C : \text{perpendicular}(d, d_2) : C_5$$

$$(\neg C)^C : \text{intersects}(d_2, d_1) : C_6$$

$$P_1^C : \neg \text{parallel}(x_8, x_9) \vee \text{parallel}(x_9, x_8) : C_7$$

$$P_2^C : \neg \text{perpendicular}(x_{10}, x_{11}) \vee \text{perpendicular}(x_{11}, x_{10}) : C_8$$

$$P_3^C : \neg \text{intersects}(x_{12}, x_{13}) \vee \text{intersects}(x_{13}, x_{12}) : C_9$$

$$P_4^C : \text{parallel}(x_{14}, x_{14}) : C_{10}$$

$$P_5^C : \text{intersects}(x_{15}, x_{15}) : C_{11}$$

$$P_6^C : \neg \text{parallel}(x_{16}, x_{17}) \vee \neg \text{parallel}(x_{17}, x_{18}) \vee \text{parallel}(x_{16}, x_{18}) : C_{12}$$

We apply general resolution to the set of clauses: $S = \{C_1, C_2, \dots, C_{12}\}$.

The following resolvents are obtained during the resolution process:

$$C_{13} = \text{Res}_{[x_{10} \leftarrow d, x_{11} \leftarrow d_2]}^{\text{Pr}}(C_5, C_8) = \text{perpendicular}(d_2, d)$$

$$C_{14} = \text{Res}_{[x_5 \leftarrow d_1, x_6 \leftarrow d]}^{\text{Pr}}(C_3, C_4) = \neg \text{perpendicular}(x_7, d) \vee \text{parallel}(d_1, x_7)$$

$$C_{15} = \text{Res}_{[x_7 \leftarrow d_2]}^{\text{Pr}}(C_{13}, C_{14}) = \text{parallel}(d_1, d_2)$$

$$C_{16} = \text{Res}_{[x_3 \leftarrow d_1, x_4 \leftarrow d_2]}^{\text{Pr}}(C_{15}, C_2) = \neg \text{intersects}(d_1, d_2)$$

$$C_{17} = \text{Res}_{[x_{13} \leftarrow d_1, x_{12} \leftarrow d_2]}^{\text{Pr}}(C_{16}, C_9) = \neg \text{intersects}(d_2, d_1)$$

$$C_{18} = \text{Res}(C_{17}, C_6) = \square$$

The empty clause was derived from the set S of clauses, so S is inconsistent and the deduction $H_1, H_2, H_3, H_4, H_5 \vdash C$ holds.

The conclusion ‘ d_2 does not intersect d_1 ’ is valid, based on the validity of the hypotheses.

Example 6.10. Mathematical reasoning modeling in algebra.

Prove: “If every element of a group G is its own inverse, then G is an Abelian group”.

We will introduce the axioms, H_1, H_2 which define the group, the hypothesis, H_3 : “every element of the group is its own inverse”, and the conclusion C : “the group is an Abelian group”.

Mathematical language:

$$H_1 : (\forall x)(\forall y)(\forall z)[(x * y) * z = x * (y * z)] \text{ - associativity}$$

$$H_2 : (\forall x)[x * e = e * x = x] \text{ - } e \text{ - neutral element}$$

A Computational Approach to Classical Logics and Circuits

$H_3 : (\forall x)[x^* x = e]$ - every element is its own inverse

$C : (\forall x)(\forall y)[x^* y = y^* x]$ - conclusion: G is an Abelian group

First-order logic language:

We use the ternary predicate symbol P , with the meaning: $P(x, y, z) : "x^* y = z"$.

' e ' is a constant.

The formulas U_1 and U_2 correspond to H_1 :

$$U_1 : (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)[P(x, y, u) \wedge P(u, z, w) \wedge P(y, z, v) \rightarrow P(x, v, w)]$$

$$U_2 : (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)[P(y, z, v) \wedge P(x, v, w) \wedge P(x, y, u) \rightarrow P(u, z, w)]$$

$$U_1^C : \neg P(x, y, u) \vee \neg P(u, z, w) \vee \neg P(y, z, v) \vee P(x, v, w) = C_1$$

$$U_2^C : \neg P(y, z, v) \vee \neg P(x, v, w) \vee \neg P(x, y, u) \vee P(u, z, w) = C_2$$

The formulas U_3 and U_4 correspond to H_2 :

$$U_3 : (\forall x)P(x, e, x), \quad U_3^C : P(s, e, s) = C_3$$

$$U_4 : (\forall x)P(e, x, x), \quad U_4^C : P(e, r, r) = C_4$$

The formula U_5 corresponds to H_3 :

$$U_5 : (\forall x)P(x, x, e), \quad U_5^C : P(l, l, e) = C_5$$

The formula U_6 corresponds to the conclusion:

$$U_6 : (\forall x)(\forall y)(\exists t)(P(x, y, t) \rightarrow P(y, x, t))$$

$$\begin{aligned} \neg U_6 &: \neg((\forall x)(\forall y)(\exists t)(P(x, y, t) \rightarrow P(y, x, t))) \equiv \\ &\equiv (\exists x)(\exists y)(\forall t)(P(x, y, t) \wedge \neg P(y, x, t)) \end{aligned}$$

$$(\neg U_6)^C : P(a, b, t) \wedge \neg P(b, a, t) = C_6 \wedge C_7 \quad a, b - \text{Skolem constants}$$

In the clauses we renamed some of the free variables.

Checking whether $H_1, H_2, H_3 \vdash C$ was reduced to checking the inconsistency of the set of clauses: $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$.

The refutation from S is presented below. At each application of the resolution rule, the literals resolved upon from the parent clauses are underlined.

$$C_1 = \underline{\neg P(x, y, u)} \vee \neg P(u, z, w) \vee \neg P(y, z, v) \vee P(x, v, w), \quad C_5 = \underline{P(l, l, e)}$$

$$\theta_1 = [x \leftarrow l, y \leftarrow l, u \leftarrow e] = \text{mgu}(P(x, y, u), P(l, l, e))$$

- $C_8 = \text{Res}_{\theta_1}^{Pr}(C_1, C_5) = \neg P(e, z, w) \vee \neg P(l, z, v) \vee P(l, v, w)$

$$C_2 = \neg P(y, z, v) \vee \neg P(x, v, w) \vee \underline{\neg P(x, y, u)} \vee P(u, z, w), \quad C_6 = \underline{P(a, b, t)}$$

$$\theta_2 = [x \leftarrow a, y \leftarrow b, u \leftarrow t] = \text{mgu}(P(a, b, t), P(x, y, u))$$

- $C_9 = \text{Res}_{\theta_2}^{Pr}(C_2, C_6) = \neg P(b, z, v) \vee \neg P(a, v, w) \vee P(t, z, w)$

Reasoning modeling and program verification

$$C_4 = \underline{P(e, r, r)},$$

$$C_8 = \underline{\neg P(e, z, w) \vee \neg P(l, z, v) \vee P(l, v, w)}$$

$$\theta_3 = [z \leftarrow r, w \leftarrow r] = \text{mgu}(P(e, z, w), P(e, r, r))$$

- $C_{10} = \text{Res}_{\theta_3}^{\text{Pr}}(C_4, C_8) = \neg P(l, r, v) \vee P(l, v, r)$

$$C_5 = \underline{P(l, l, e)},$$

$$C_9 = \underline{\neg P(b, z, v) \vee \neg P(a, v, w) \vee P(t, z, w)}$$

$$\theta_4 = [l \leftarrow b, z \leftarrow b, v \leftarrow e] = \text{mgu}(P(l, l, e), P(b, z, v))$$

- $C_{11} = \text{Res}_{\theta_4}^{\text{Pr}}(C_5, C_9) = \neg P(a, e, w) \vee P(t, b, w)$

$$C_3 = \underline{P(s, e, s)},$$

$$C_{11} = \underline{\neg P(a, e, w) \vee P(t, b, w)}$$

$$\theta_5 = [s \leftarrow a, w \leftarrow a] = \text{mgu}(P(s, e, s), P(a, e, w))$$

- $C_{12} = \text{Res}_{\theta_5}^{\text{Pr}}(C_3, C_{11}) = P(t, b, a),$

$$C_{10} = \underline{\neg P(l, r, v) \vee P(l, v, r)},$$

$$C_{12} = \underline{P(t, b, a)}$$

$$\theta_6 = [l \leftarrow t, r \leftarrow b, v \leftarrow a] = \text{mgu}(P(t, b, a), P(l, r, v))$$

- $C_{13} = \text{Res}_{\theta_6}^{\text{Pr}}(C_{10}, C_{12}) = P(t, a, b)$

$$C_2 = \underline{\neg P(y, z, v) \vee \neg P(x, v, w) \vee \neg P(x, y, u) \vee P(u, z, w)},$$

$$C_5 = \underline{P(l, l, e)}$$

$$\theta_7 = [y \leftarrow l, z \leftarrow l, v \leftarrow e] = \text{mgu}(P(l, l, e), P(y, z, v))$$

- $C_{14} = \text{Res}_{\theta_7}^{\text{Pr}}(C_2, C_5) = \neg P(x, e, w) \vee \neg P(x, l, u) \vee P(u, l, w)$

$$C_3 = \underline{P(s, e, s)},$$

$$C_{14} = \underline{\neg P(x, e, w) \vee \neg P(x, l, u) \vee P(u, l, w)}$$

$$\theta_8 = [x \leftarrow s, w \leftarrow s] = \text{mgu}(P(s, e, s), P(x, e, w))$$

- $C_{15} = \text{Res}_{\theta_8}^{\text{Pr}}(C_3, C_{14}) = \neg P(s, l, u) \vee P(u, l, s)$

$$C_{13} = \underline{P(t, a, b)},$$

$$C_{15} = \underline{\neg P(s, l, u) \vee P(u, l, s)}$$

$$\theta_9 = [s \leftarrow t, l \leftarrow a, u \leftarrow b] = \text{mgu}(P(t, a, b), P(s, l, u))$$

- $C_{16} = \text{Res}_{\theta_9}^{\text{Pr}}(C_{13}, C_{15}) = P(b, a, t)$

$$C_7 = \underline{\neg P(b, a, t)},$$

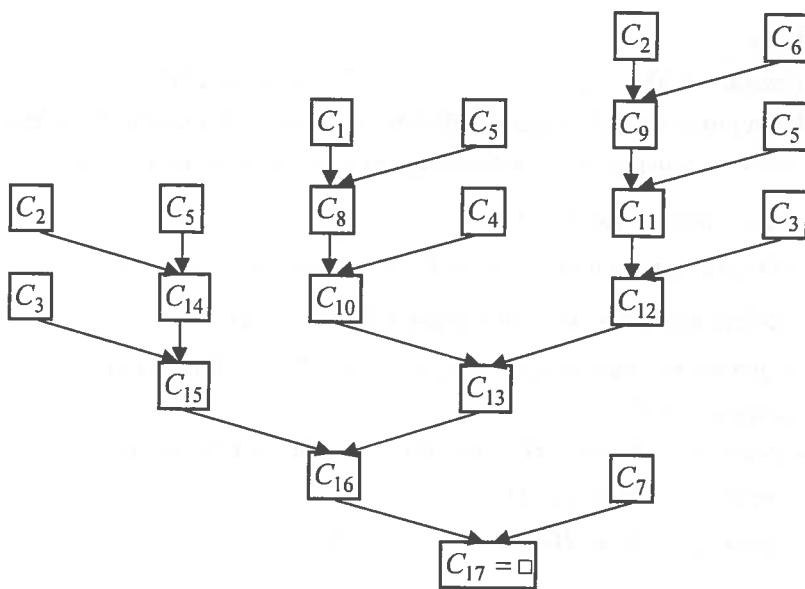
$$C_{16} = \underline{P(b, a, t)}$$

- $C_{17} = \text{Res}^{\text{Pr}}(C_{16}, C_7) = \square, \text{ so the set } S \text{ is inconsistent and } U_1, U_2, U_3, U_4, U_5 \vdash U_6.$

We conclude that the statement “*If every element of a group G is its own inverse, then G is an Abelian group*” is valid.

The resolution process is represented graphically by the following binary tree.

A Computational Approach to Classical Logics and Circuits



Example 6.11. Formalization of mathematical reasoning

Prove the statement: “*There exists an infinite number of primes*”.

We do not have hypotheses, we need mathematical axioms in order to prove the statement. The axioms in mathematical language, predicate logic language and their clausal forms are provided below.

$$a_1 : x \neq x ;$$

$$(\forall x) \neg \text{less}(x, x), \quad C_1 = \neg \text{less}(x, x)$$

$$a_2 : \text{if } x < y \text{ then } y \neq x$$

$$(\forall x)(\forall y)(\text{less}(x, y) \rightarrow \neg \text{less}(y, x)), \quad C_2 = \neg \text{less}(x, y) \vee \neg \text{less}(y, x)$$

$$a_3 : x \text{ divides } x$$

$$(\forall x) \text{divides}(x, x), \quad C_3 = \text{divides}(x, x)$$

$$a_4 : \text{if } x \text{ divides } y \text{ and } y \text{ divides } z, \text{ then } x \text{ divides } z$$

$$(\forall x)(\forall y)(\forall z)(\text{divides}(x, y) \wedge \text{divides}(y, z) \rightarrow \text{divides}(x, z)),$$

$$C_4 = \neg \text{divides}(x, y) \vee \neg \text{divides}(y, z) \vee \text{divides}(x, z))$$

$$a_4 : \text{if } x < y \text{ then } y \text{ does not divide } x$$

$$(\forall x)(\forall y)(\text{less}(x, y) \rightarrow \neg \text{divides}(y, x)), \quad C_5 = \neg \text{less}(x, y) \vee \neg \text{divides}(y, x)$$

$$a_6 : \text{if } y \text{ divides } F(x)=x!+1, \text{ then } x < y$$

$$(\forall x)(\forall y)(\text{divides}(y, F(x)) \rightarrow \text{less}(x, y)) \quad C_6 = \neg \text{divides}(y, F(x)) \vee \text{less}(x, y)$$

Reasoning modeling and program verification

$a_7 : x < F(x)$

$$(\forall x) \text{less}(x, F(x)), \quad C_7 = \text{less}(x, F(x))$$

a_8 : if x is not prime, then there is a y such that y divides x , y is prime and y is less than x .

$$U = (\forall x)(\neg \text{prime}(x) \rightarrow (\exists y)(\text{divides}(y, x) \wedge \text{prime}(y) \wedge \text{less}(y, x)))$$

The prenex normal form of U is:

$$U^P = (\forall x)(\exists y)(\text{prime}(x) \vee (\text{divides}(y, x) \wedge \text{prime}(y) \wedge \text{less}(y, x)))$$

The Skolem normal form without quantifiers of U is:

$$U^{Sq} = \text{prime}(x) \vee (\text{divides}(H(x), x) \wedge \text{prime}(H(x)) \wedge \text{less}(H(x), x))$$

The clauses provided:

$$C_8 = \text{prime}(x) \vee \text{divides}(H(x), x), \quad H(x) \text{ is a Skolem function}$$

$$C_9 = \text{prime}(x) \vee \text{prime}(H(x))$$

$$C_{10} = \text{prime}(x) \vee \text{less}(H(x), x))$$

The conclusion: "If x is a prime then there is a y such that y is a prime, x is less than y and $F(x)$ is not less than y ."

$$C = (\forall x)(\text{prime}(x) \rightarrow (\exists y)(\text{prime}(y) \wedge \text{less}(x, y) \wedge \neg \text{less}(F(x), y)))$$

$$\neg C = \neg (\forall x)(\text{prime}(x) \rightarrow (\exists y)(\text{prime}(y) \wedge \text{less}(x, y) \wedge \neg \text{less}(F(x), y))) \equiv$$

$$\equiv (\exists x)(\text{prime}(x) \wedge (\forall y)(\neg \text{prime}(y) \vee \neg \text{less}(x, y) \vee \text{less}(F(x), y))) \equiv$$

$$\equiv (\exists x)(\forall y)(\text{prime}(x) \wedge (\neg \text{prime}(y) \vee \neg \text{less}(x, y) \vee \text{less}(F(x), y)))$$

– prenex normal form

The clausal normal form of $\neg C$ is:

$$\text{prime}(a) \wedge (\neg \text{prime}(y) \vee \neg \text{less}(a, y) \vee \text{less}(F(a), y)) \text{ providing the clauses:}$$

$$C_{11} = \text{prime}(a),$$

$$C_{12} = \neg \text{prime}(y) \vee \neg \text{less}(a, y) \vee \text{less}(F(a), y)$$

We want to prove that the conclusion is derivable from the set of axioms:

$a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$.

Using resolution we prove that $S = \{C_1, C_2, \dots, C_{12}\}$ is an inconsistent set.

$$C_{12} = \neg \text{less}(a, y) \vee \text{less}(F(a), y) \vee \underline{\neg \text{prime}(y)},$$

$$C_9 = \underline{\text{prime}(x)} \vee \text{prime}(H(x)),$$

- $C_{13} = \text{Res}_{[y \leftarrow x]}^{\text{Pr}}(C_9, C_{12}) = \neg \text{less}(a, x) \vee \text{less}(F(a), x) \vee [\neg \text{prime}(x)] \vee \text{prime}(H(x))$

$$C_8 = \underline{\text{prime}(x)} \vee \text{divides}(H(x), x),$$

$$C_{12} = \underline{\neg \text{prime}(y)} \vee \neg \text{less}(a, y) \vee \text{less}(F(a), y))$$

- $C_{14} = \text{Res}_{[y \leftarrow x]}^{\text{Pr}}(C_8, C_{12}) = \text{divides}(H(x), x) \vee \neg \text{less}(a, x) \vee \text{less}(F(a), x)$

A Computational Approach to Classical Logics and Circuits

$$C_{13} = \text{prime}(H(x)) \vee \neg \text{less}(a, x) \vee \text{less}(F(a), x)$$

$$C_{12} = \neg prime(y) \vee \neg less(a, y) \vee less(F(a), y)$$

- $C_{15} = \text{Res}_{[y \leftarrow H(x)]}^{\text{Pr}}(C_{13}, C_{12}) = \neg \text{less}(a, x) \vee \underline{\neg \text{less}(a, H(x))} \vee \text{less}(F(a), x)$
 $\qquad\qquad\qquad \vee \text{less}(F(a), H(x))$
 $C_6 = \neg \text{divides}(u, F(v)) \vee \underline{\text{less}(v, u)}$
 - $C_{16} = \text{Res}_{[v \leftarrow a, u \leftarrow H(x)]}^{\text{Pr}}(C_6, C_{15}) = \neg \text{less}(a, x) \vee \text{less}(F(a), x) \vee \underline{\text{less}(F(a), H(x))}$
 $\qquad\qquad\qquad \vee \neg \text{divides}(H(x), F(a))$
 - $C_5 = \underline{\text{less}(u, v)} \vee \neg \text{divides}(v, u)$
 - $C_{17} = \text{Res}_{[u \leftarrow F(a), v \leftarrow H(x)]}^{\text{Pr}}(C_5, C_{16}) = \neg \text{less}(a, x) \vee \text{less}(F(a), x)$
 $\qquad\qquad\qquad \vee \underline{\neg \text{divides}(H(x), F(a))}$
 - $C_{14} = \underline{\text{divides}(H(x), x)} \vee \neg \text{less}(a, x) \vee \text{less}(F(a), x)$
 - $C_{18} = \text{Res}_{[x \leftarrow a]}^{\text{Pr}}(C_{17}, C_{14}) = \underline{\neg \text{less}(a, F(a))} \vee \text{less}(F(a), F(a))$
 $C_7 = \underline{\text{less}(x, F(x))}$
 - $C_{19} = \text{Res}(C_1, C_{18}) = \text{less}(F(a), F(a))$
 $C_1 = \neg \text{less}(x, x)$
 - $C_{20} = \text{Res}_{[x \leftarrow F(a)]}^{\text{Pr}}(C_1, C_{19}) = \square$

The empty clause was derived from the set of clauses $S=\{C_1, C_5, C_6, C_7, C_8, C_9, C_{12}\}$, so there is a contradiction among the formulas $\{U_1, U_2, \dots, U_8, \neg C\}$.

By applying the ‘reductio ad absurdum’ principle we conclude that C is derivable from the set of axioms and the statement:

“There exists an infinite number of primes” is true.

6.3. Program verification using predicate logic

We present an approach of the program verification task introduced in paper [62]. A program is described by a set of predicate formulas and then using the resolution method, the execution of the program is modeled.

Let P be a **program**: an algorithm described using a chart flow, pseudocode or a programming language.

For a program we have to solve these problems:

- the **stop problem**: having an input data \bar{x} , will the program stop?
- the **answer problem**: given the input data \bar{x} which is the output data \bar{z} of a program?
- the **correctness problem**: if a program stops, will the relations input data – output data be satisfied?
- the **equivalence problem**: having two programs, will they provide the same output data for the same input data?

A program uses three data vectors denoted by:

- $\bar{x} = (x_1, \dots, x_n)$ - the input vector (the variables are used only on the right side of an assignment)
- $\bar{y} = (y_1, \dots, y_k)$ - the work vector
- $\bar{z} = (z_1, \dots, z_m)$ - the output vector (the variables are used only on the left side of an assignment)

Definition 6.1.

To a program P we associate a graph $G = (X, U)$ as follows:

1. there is a unique initial vertex (without incoming arcs) called **Start**, denoted by S ;
2. there is a unique final vertex (without outgoing arcs) called **Halt**, denoted by H ;
3. each arc a that is not incident with H has associated a formula $P_a(\bar{x}, \bar{y})$ and an assignment $\bar{y} = f_a(\bar{x}, \bar{y})$
4. each arc a that is incident with H has associated a formula $P_a(\bar{x}, \bar{y})$ and an assignment $\bar{z} = f_a(\bar{x}, \bar{y})$
5. if a_1 and a_2 are two outgoing arcs from the same vertex $v \neq H$, only one of the formulas $P_{a_1}(\bar{x}, \bar{y})$ and $P_{a_2}(\bar{x}, \bar{y})$ is true.

A Computational Approach to Classical Logics and Circuits

Note that for a given input data \bar{x} , the vector \bar{y} is well defined and the formulas $P_a(\bar{x}, \bar{y})$ can be evaluated as true or false for each arc a .

Definition 6.2.

The **condition for accessing the node (vertex) v_i** from S is expressed by the predicate $q_i(\bar{x}, \bar{y})$ which must be true.

Note: $q_S(\bar{x}, \bar{y}) = T, \forall \bar{x}, \bar{y}$.

Definition 6.3.

The **stop condition** is the condition for accessing H from S and it is expressed by the predicate $q_H(\bar{x}, \bar{z})$, which must be true.

Definition 6.4.

The formula W_a is called the **description formula for the arc $a = (v_i, v_j)$** and has the form: $W_a : q_i(\bar{x}, \bar{y}) \wedge P_a(\bar{x}, \bar{y}) \rightarrow q_j(\bar{x}, f_a(\bar{x}, \bar{y}))$.

Note that W_a must be true if the arc a is traversed. This formula is in fact a clause:

$$W_a \equiv \neg q_i(\bar{x}, \bar{y}) \vee \neg P_a(\bar{x}, \bar{y}) \vee q_j(\bar{x}, f_a(\bar{x}, \bar{y})).$$

Definition 6.5.

Let a_1, \dots, a_r be all the arcs of the graph associated to the program P . The formula $U_P = W_{a_1} \wedge \dots \wedge W_{a_r}$ is called the **description formula** of P . We will also identify U_P with the set of all description clauses of P .

Theorem 6.1.

Let P be a program and U_P the set of all description clauses of P . U_P is a consistent set of clauses.

A program is characterized not only by U_P but also by U_I (a formula that defines the properties of input data) and U_r (the set of formulas representing the axioms for the predicates $P_a(\bar{x}, \bar{y})$ and the functions $f_a(\bar{x}, \bar{y})$).

Definition 6.6.

A program P is convergent for an input data \bar{x} if there is a path from S to H using \bar{x} . **A program P is convergent** if for all \bar{x} there is a path from S to H .

Reasoning modeling and program verification

Theorem 6.2.

Let P be a program and U_T a set of formulas obtained from U_P by eliminating all the literals containing the predicate $q_H(\bar{x}, \bar{z})$. P is convergent if and only if the set $U_T \cup U_I \cup U_r$ is inconsistent.

Definition 6.7.

A clause containing only the literal $q_H(\bar{x}, \bar{z})$ is called a **halt clause**.

Theorem 6.3.

Let P be a program and X the set of clauses corresponding to $U_P \cup U_I \cup U_r$. The program P is convergent if and only if there is a deduction of a halt clause from X using resolution.

In the following we will study the convergence of the following program according to the theory presented above. Predicate resolution is used to model the execution of the program.

Program P :

input: $\bar{x} = (x_1, x_2, n)$

output: $z := x_1^n$ (if $n \geq 2$) or $z := n * x_2$ (if $n \leq 2$)

begin

$y := 1;$

$i := 0;$

if ($n \geq 2$)

then

while ($i < n$)

$y := y * x_1;$

$i := i + 1;$

end_while

else

$y := n * x_2;$

end_if

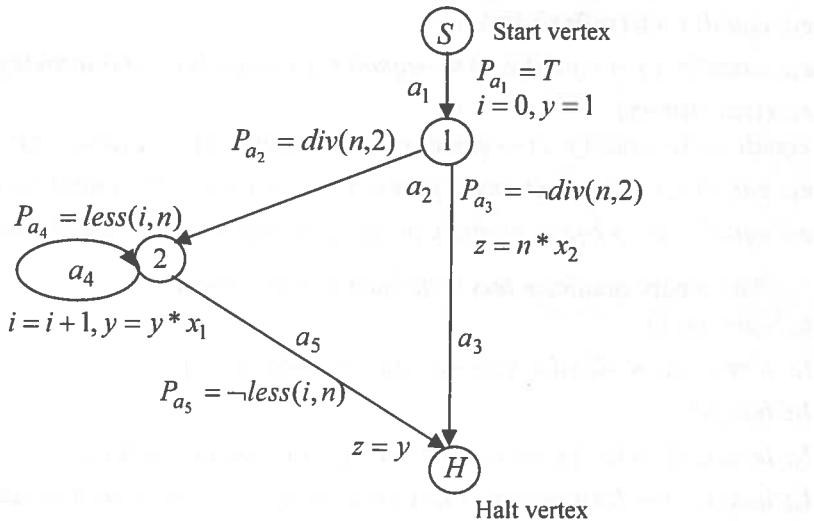
$z := y;$

end

- the input vector: $\bar{x} = (x_1, x_2, n);$
- the output vector: $\bar{z} = (z);$
- the work vector: $\bar{y} = (i, y)$

A Computational Approach to Classical Logics and Circuits

We associate the following graph to the program P :



The description formulas for the arcs are as follows:

$$\begin{aligned}
 W_{a_1} : q_S(x_1, x_2, n, i, y) \wedge P_{a_1} \rightarrow q_1(x_1, x_2, n, 0, 1) &\equiv T \wedge T \rightarrow q_1(x_1, x_2, n, 0, 1) \equiv \\
 &\equiv q_1(x_1, x_2, n, 0, 1) \\
 W_{a_2} : q_1(x_1, x_2, n, i, y) \wedge \text{div}(n, 2) \rightarrow q_2(x_1, x_2, n, i, y) \\
 W_{a_3} : q_1(x_1, x_2, n, i, y) \wedge \neg \text{div}(n, 2) \rightarrow q_H(x_1, x_2, n, n * x_2) \\
 W_{a_4} : q_2(x_1, x_2, n, i, y) \wedge \text{less}(i, n) \rightarrow q_2(x_1, x_2, n, i + 1, y * x_1) \\
 W_{a_5} : q_2(x_1, x_2, n, i, y) \wedge \neg \text{less}(i, n) \rightarrow q_H(x_1, x_2, n, y)
 \end{aligned}$$

The clausal normal forms of these formulas are:

$$\begin{aligned}
 C_1 &: q_1(x_1, x_2, n, 0, 1) \\
 C_2 &: \neg q_1(x_1, x_2, n, i, y) \vee \neg \text{div}(n, 2) \vee q_2(x_1, x_2, n, i, y) \\
 C_3 &: \neg q_1(x_1, x_2, n, i, y) \vee \text{div}(n, 2) \vee q_H(x_1, x_2, n, n * x_2) \\
 C_4 &: \neg q_2(x_1, x_2, n, i, y) \vee \neg \text{less}(i, n) \vee q_2(x_1, x_2, n, i + 1, y * x_1) \\
 C_5 &: \neg q_2(x_1, x_2, n, i, y) \vee \text{less}(i, n) \vee q_H(x_1, x_2, n, y)
 \end{aligned}$$

$$U_P = \{C_1, C_2, C_3, C_4, C_5\}$$

The axioms which define the predicates: *equal*, *less* and *div* for integers are expressed by predicate formulas having universal quantified variables.

We omit the quantifiers obtaining the free formulas presented below.

The functions: $\text{pred}(y) = y - 1$, $\text{succ}(y) = y + 1$ are used.

Reasoning modeling and program verification

The binary predicate *equal* is defined by the axioms:

$$e_1: \text{equal}(x, x) \text{ (reflexivity)}$$

$$e_2: \text{equal}(x, y) \rightarrow \text{equal}(y, x) \equiv \neg \text{equal}(x, y) \vee \text{equal}(y, x) \text{ (symmetry)}$$

$$e_3: \text{(transitivity)}$$

$$\text{equal}(x, y) \wedge \text{equal}(y, z) \rightarrow \text{equal}(x, z) \equiv \neg \text{equal}(x, y) \vee \neg \text{equal}(y, z) \vee \text{equal}(x, z)$$

$$e_4: \text{equal}(x, y) \rightarrow \text{equal}(\text{succ}(x), \text{succ}(y)) \equiv \neg \text{equal}(x, y) \vee \text{equal}(\text{succ}(x), \text{succ}(y))$$

$$e_5: \text{equal}(x, y) \rightarrow \text{equal}(\text{pred}(x), \text{pred}(y)) \equiv \neg \text{equal}(x, y) \vee \text{equal}(\text{pred}(x), \text{pred}(y))$$

The binary predicate *less* is defined by the axioms:

$$l_0: \neg \text{less}(x, x)$$

$$l_1: \text{less}(x, y) \rightarrow \neg \text{less}(y, x) \equiv \neg \text{less}(x, y) \vee \neg \text{less}(y, x)$$

$$l_2: \text{less}(0, 1)$$

$$l_3: \text{less}(x, y) \rightarrow \text{less}(x, \text{succ}(y)) \equiv \neg \text{less}(x, y) \vee \text{less}(x, \text{succ}(y))$$

$$l_4: \text{less}(x, y) \rightarrow \text{less}(\text{succ}(x), \text{succ}(y)) \equiv \neg \text{less}(x, y) \vee \text{less}(\text{succ}(x), \text{succ}(y))$$

$$l_5: \text{less}(x, y) \rightarrow \text{less}(\text{pred}(x), y) \equiv \neg \text{less}(x, y) \vee \text{less}(\text{pred}(x), y)$$

$$l_6: \text{less}(x, y) \rightarrow \text{less}(\text{pred}(x), \text{pred}(y)) \equiv \neg \text{less}(x, y) \vee \text{less}(\text{pred}(x), \text{pred}(y))$$

The predicate *div* (*divisibility by 2*) is defined by the axioms:

$$d_1: \text{div}(0, 2)$$

$$d_2: \text{div}(k, 2) \rightarrow \text{div}(k + 2, 2) \equiv \neg \text{div}(k, 2) \vee \text{div}(k + 2, 2)$$

$$d_3: \text{div}(k, 2) \rightarrow \text{div}(k - 2, 2) \equiv \neg \text{div}(k, 2) \vee \text{div}(k - 2, 2)$$

$$d_4: \text{div}(k, 2) \rightarrow \neg \text{div}(k + 1, 2) \equiv \neg \text{div}(k, 2) \vee \neg \text{div}(k + 1, 2)$$

$$d_5: \text{div}(k, 2) \rightarrow \neg \text{div}(k - 1, 2) \equiv \neg \text{div}(k, 2) \vee \neg \text{div}(k - 1, 2)$$

The set of formulas representing the axioms for the predicates and the functions used in the description of the program is

$$U_r = \{e_1, e_2, e_3, e_4, e_5, l_0, l_1, l_2, l_3, l_4, l_5, l_6, d_1, d_2, d_3, d_4, d_5\},$$

$$U_I = \emptyset$$

For $n = 3$ ($n \mid 2$) we prove the convergence of P . We follow the resolution process in order to derive a halt clause and obtain the output data. At each resolution step the corresponding unifier is written.

$$C_6 = \text{Res}_{[i \leftarrow 0, y \leftarrow 1]}^{\text{Pr}}(C_1, C_3) = \text{div}(3, 2) \vee q_H(x_1, x_2, 3, 3 * x_2)$$

$$d_6' = \text{Res}_{[k \leftarrow 0]}^{\text{Pr}}(d_1, d_2) = \text{div}(2, 2)$$

$$d_7' = \text{Res}_{[k \leftarrow 2]}^{\text{Pr}}(d_6', d_4) = \neg \text{div}(3, 2)$$

$$C_7 = \text{Res}(C_6, d_7') = q_H(x_1, x_2, 3, 3 * x_2) \text{ halt clause, the output data is } z = 3 * x_2$$

$U_P \cup U_r \vdash_{\text{Res}} C_7$, therefore the program P is convergent.

A Computational Approach to Classical Logics and Circuits

For $n = 4$ ($n:2$) we prove the convergence of P . In the while loop four iterations are executed and the resolution process models this execution as follows:

$$C_8 = \text{Res}_{[i \leftarrow 0, y \leftarrow 1]}^{\text{Pr}}(C_1, C_2) = \neg \text{div}(4, 2) \vee q_2(x_1, x_2, 4, 0, 1)$$

$$d_6' = \text{Res}_{[k \leftarrow 0]}^{\text{Pr}}(d_1, d_2) = \text{div}(2, 2)$$

$$d_8' = \text{Res}_{[k \leftarrow 2]}^{\text{Pr}}(d_6', d_2) = \text{div}(4, 2)$$

$$C_9^0 = \text{Res}(C_8, d_8') = q_2(x_1, x_2, 4, 0, 1)$$

$$C_{10}^0 = \text{Res}_{[i \leftarrow 0, y \leftarrow 1]}^{\text{Pr}}(C_9^0, C_4) = \neg \text{less}(0, 4) \vee q_2(x_1, x_2, 4, 1, x_1)$$

Using the axioms l_0, \dots, l_6 we derive $l_7' = \text{less}(0, 4)$ as follows:

$$c_1 = \text{Res}(l_2, l_3) = \text{less}(0, 2), c_3 = \text{Res}(c_1, l_3) = \text{less}(0, 3)$$

$$l_7' = \text{Res}(c_3, l_3) = \text{less}(0, 4)$$

Iteration 1:

$$C_9^1 = \text{Res}(C_{10}^0, l_7') = q_2(x_1, x_2, 4, 1, x_1)$$

$$C_{10}^1 = \text{Res}_{[i \leftarrow 1, y \leftarrow x_1]}^{\text{Pr}}(C_9^1, C_4) = \neg \text{less}(1, 4) \vee q_2(x_1, x_2, 4, 2, x_1 * x_1)$$

Using the axioms l_0, \dots, l_6 we derive $l_8' = \text{less}(1, 4)$

Iteration 2:

$$C_9^2 = \text{Res}(C_{10}^1, l_8') = q_2(x_1, x_2, 4, 2, x_1 * x_1)$$

$$C_{10}^2 = \text{Res}_{[i \leftarrow 2, y \leftarrow x_1 * x_1]}^{\text{Pr}}(C_9^2, C_4) = \neg \text{less}(2, 4) \vee q_2(x_1, x_2, 4, 3, x_1 * x_1 * x_1)$$

Using the axioms l_0, \dots, l_6 we derive $l_9' = \text{less}(2, 4)$

Iteration 3:

$$C_9^3 = \text{Res}(C_{10}^2, l_9') = q_2(x_1, x_2, 4, 3, x_1 * x_1 * x_1)$$

$$C_{10}^3 = \text{Res}_{[i \leftarrow 3, y \leftarrow x_1 * x_1 * x_1]}^{\text{Pr}}(C_9^3, C_4) = \neg \text{less}(3, 4) \vee q_2(x_1, x_2, 4, 4, x_1 * x_1 * x_1 * x_1)$$

Using the axioms l_0, \dots, l_6 we derive $l_{10}' = \text{less}(3, 4)$

Iteration 4:

$$C_9^4 = \text{Res}(C_{10}^3, l_{10}') = q_2(x_1, x_2, 4, 4, x_1 * x_1 * x_1 * x_1)$$

The exit of the loop:

$$C_{11} = \text{Res}(C_9^4, C_5) = \text{less}(4, 4) \vee q_H(x_1, x_2, 4, x_1 * x_1 * x_1 * x_1)$$

$$l_0 = \neg \text{less}(4, 4), [x \leftarrow 4]$$

$$C_{12} = \text{Res}(C_{11}, l_0) = q_H(x_1, x_2, 4, x_1 * x_1 * x_1 * x_1)$$

C_{12} is a halt clause and provides the output data: $z = x_1 * x_1 * x_1 * x_1$

$U_P \cup U_r \vdash_{\text{Res}} C_{12}$, therefore P is a convergent program.

7. BOOLEAN ALGEBRAS AND BOOLEAN FUNCTIONS

George Boole introduced Boolean algebras in 1854 in the paper [7]. In 1938 Claude Shannon proved that a two-valued binary Boolean algebra can describe the operations of two-valued electrical switching circuits. In modern times Boolean algebras and Boolean functions are indispensable in the design of computer chips and digital circuits.

In this chapter the theoretical concepts of Boolean algebras and Boolean functions are introduced. These concepts are used further in the simplification of Boolean functions. Three simplification methods: Veitch-Karnaugh diagrams method, Quine's method and Moisil's method are described and applied in numerous examples. The following papers [8, 14, 25, 26, 37, 53, 62] were used as bibliographic references.

7.1. Boolean algebras

Definition 7.1.

A **Boolean algebra** is a structure $(A, \wedge, \vee, \bar{}, 0, 1)$ described by the following axioms:

1. $|A| \geq 2$, A contains at least two elements: 0 and 1, $0 \neq 1$
2. \wedge, \vee are binary operations
3. $\bar{}$ is a unary operation
4. 0 is the zero element:
 $x \wedge 0 = 0 \wedge x = 0$ and $x \vee 0 = 0 \vee x = x, \forall x \in A$
5. 1 is the unity element:
 $x \wedge 1 = 1 \wedge x = x$ and $x \vee 1 = 1 \vee x = 1, \forall x \in A$
6. 0 is the first element, 1 is the last element
 $x \wedge \bar{x} = 0$ and $x \vee \bar{x} = 1, \forall x \in A$
7. double negation: $\bar{\bar{x}} = x, \forall x \in A$
8. commutativity:
 $x \wedge y = y \wedge x$ and $x \vee y = y \vee x, \forall x, y \in A$
9. associativity:
 $x \wedge (y \wedge z) = (x \wedge y) \wedge z (= x \wedge y \wedge z)$ and
 $x \vee (y \vee z) = (x \vee y) \vee z (= x \vee y \vee z), \forall x, y, z \in A$
10. distributivity:
 $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ and $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z), \forall x, y, z \in A$

A Computational Approach to Classical Logics and Circuits

11. idempotency:

$$x \wedge x = x \text{ and } x \vee x = x, \forall x \in A$$

12. De Morgan's laws:

$$\overline{x \wedge y} = \overline{x} \vee \overline{y} \text{ and } \overline{x \vee y} = \overline{x} \wedge \overline{y}, \forall x, y \in A$$

13. absorption laws:

$$x \wedge (x \vee y) = x \text{ and } x \vee (x \wedge y) = x, \forall x, y \in A$$

Note: A Boolean algebra is a complemented distributive lattice.

In the literature there are alternative symbols used for the binary operations of a Boolean algebra: (\wedge, \vee) or $(*, +)$. In this paper we use the pair of symbols: (\wedge, \vee) .

The **duality principle** in a Boolean algebra: „For any equality of two Boolean expressions, $U = V$, there is another equality, $U' = V'$ ”, obtained by interchanging the Boolean operations: \wedge, \vee and the Boolean values: 0, 1”.

Example 7.1.

The **binary Boolean algebra** is $B = (B_2 = \{0, 1\}, \wedge, \vee, \neg, 0, 1)$, where the truth tables for the Boolean operations are as follows:

\vee	0	1
0	0	1
1	1	1

\wedge	0	1
0	0	0
1	0	1

x	\bar{x}
0	1
1	0

Example 7.2.

The structure $(F_P, \wedge, \vee, \neg, F, T)$ is a Boolean algebra.

- F_P is the set of all well-formed propositional formulas;
- binary operations: \wedge (conjunction), \vee (disjunction);
- unary operation: \neg (negation);
- zero element: F , unity element: T , where $F(\text{false})$, $T(\text{true})$ are the truth values.

Propositional logic is a logical system that is intimately connected to Boolean algebra. Many syntactic concepts of Boolean algebra carry over to propositional logic with only minor changes in notation and terminology, while the semantics of propositional logic are defined via Boolean algebras in a way that the tautologies (theorems) of propositional logic correspond to equational theorems of Boolean algebra.

The semantics of propositional logic is based on truth assignments. The basic idea of a truth assignment is that the propositional variables are mapped to elements of a fixed Boolean algebra, and then the truth value of a propositional formula is the element of the Boolean algebra that is obtained by computing the value of the Boolean expression corresponding to the formula.

Boolean Algebras and Boolean Functions

Example 7.3.

The structure $(\mathcal{P}(X), \cap, \cup, C_X, \emptyset, X)$ is a Boolean algebra.

- X is a set and $\mathcal{P}(X)$ is the set of all subsets of X ;
- binary operations: \cap (intersection), \cup (reunion);
- unary operation: $C_X(A) = X \setminus A$ - complementary set of A with respect to X ;
- zero element: \emptyset and unity element: X .

The properties from Definition 7.1 can be easily verified.

7.2. Boolean functions

Definition 7.2.

Let $B = (B_2, \wedge, \vee, \neg, 0, 1)$ be the binary Boolean algebra, $B_2 = \{0, 1\}$ and $n \in \mathbb{N}^*$. A

Boolean function of n variables is a function $f : (B_2)^n \rightarrow B_2$ defined as follows:

1. the projection function: $P_i : B_2^n \rightarrow B_2$, $P_i(x_1, \dots, x_i, \dots, x_n) = x_i$, is a Boolean function.
2. if $f, g : B_2^n \rightarrow B_2$ are Boolean functions then $f \wedge g, f \vee g, \overline{f}$ are Boolean functions of n variables, where:

$$(f \wedge g)(x_1, \dots, x_n) = f(x_1, \dots, x_n) \wedge g(x_1, \dots, x_n);$$

$$(f \vee g)(x_1, \dots, x_n) = f(x_1, \dots, x_n) \vee g(x_1, \dots, x_n);$$

$$\overline{f}(x_1, \dots, x_n) = \overline{f(x_1, \dots, x_n)}.$$
3. any Boolean function can be obtained by the application of the rules 1 and 2.

Theorem 7.1.

$\forall n \in \mathbb{N}^*$, there exist 2^{2^n} Boolean functions of n variables.

Theorem 7.2.

The structure $(\mathbf{FB}(n), \wedge, \vee, \neg, f_0, f_{2^{2^n}-1})$ is a Boolean algebra, where **FB(n)** is the set of all Boolean functions of $n \in \mathbb{N}^*$ variables, where \wedge, \vee and \neg are defined in the above definition and $f_0(x_1, \dots, x_n) = 0$ and $f_{2^{2^n}-1}(x_1, \dots, x_n) = 1$ are the constant functions corresponding to *contradiction* and *tautology* respectively.

Example 7.4.

For $n = 1$, there exist $2^{2^1} = 4$ Boolean functions of one variable represented by their expressions and tables of (truth) values as follows:

x	$f_0(x) = 0$ <i>contradiction</i>	$f_1(x) = x$	$f_2(x) = \overline{x}$	$f_3(x) = 1$ <i>tautology</i>
0	0	0	1	1
1	0	1	0	1

A Computational Approach to Classical Logics and Circuits

Example 7.5.

The $2^2 = 2^4 = 16$ Boolean functions of 2 variables are represented in the table below.

x	y	$f_0(x, y)$	$f_1(x, y)$	$f_2(x, y)$	$f_3(x, y)$	$f_4(x, y)$	$f_5(x, y)$	$f_6(x, y)$	$f_7(x, y)$
		<i>contra-diction</i>	$x \wedge y$	$x \wedge \bar{y}$	x	$\bar{x} \wedge y$	y	$x \oplus y$	$x \vee y$
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1
1	1	0	1	0	1	0	1	0	1

x	y	$f_8(x, y)$	$f_9(x, y)$	$f_{10}(x, y)$	$f_{11}(x, y)$	$f_{12}(x, y)$	$f_{13}(x, y)$	$f_{14}(x, y)$	$f_{15}(x, y)$
		$x \downarrow y$	$x \leftrightarrow y$	\bar{y}	$y \rightarrow x$	\bar{x}	$x \rightarrow y$	$x \uparrow y$	<i>tautology</i>
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1
1	1	0	1	0	1	0	1	0	1

Each function has an inverse function obtained as its negation. The pairs of inverse functions are: $(f_0, f_{15}), (f_1, f_{14}), (f_2, f_{13}), (f_3, f_{12}), (f_4, f_{11}), (f_5, f_{10}), (f_6, f_9), (f_7, f_8)$.

We use the following symbols for operations:

- \downarrow (nor) – Pierce's function, $x \downarrow y = \overline{(x \vee y)} = \bar{x} \wedge \bar{y}$
- \uparrow (nand) – Schaffer's function, $x \uparrow y = \overline{(x \wedge y)} = \bar{x} \vee \bar{y}$
- \rightarrow (logical implication), $x \rightarrow y = \bar{x} \vee y$
- \leftrightarrow (logical equivalence), $x \leftrightarrow y = (x \rightarrow y) \wedge (y \rightarrow x) = (x \wedge y) \vee (\bar{x} \wedge \bar{y})$
- \oplus („xor”- exclusive ‘or’), $x \oplus y = \overline{(x \leftrightarrow y)} = (\bar{x} \wedge y) \vee (x \wedge \bar{y})$

For a uniform representation of the variables and their negations we introduce the following notation: $x^\alpha = \begin{cases} x, & \text{if } \alpha = 1 \\ \bar{x}, & \text{if } \alpha = 0 \end{cases}, \quad x \in \{0, 1\}$.

For $x, \alpha \in \{0, 1\}$ we have: $x^0 = \bar{x}, \quad x^1 = x$ and

$$0^0 = \bar{0} = 1; \quad 0^1 = 0; \quad 1^0 = \bar{1} = 0; \quad 1^1 = 1$$

$$x^\alpha = \begin{cases} 1, & \text{if } x = \alpha \\ 0, & \text{if } x \neq \alpha \end{cases}, \quad x, \alpha \in \{0, 1\}.$$

Boolean Algebras and Boolean Functions

7.3. Canonical forms of Boolean functions

All Boolean expressions, regardless of their form, can be transformed into two standard (canonical) forms: the *disjunction-of-conjunctions* form or the *conjunction-of-disjunctions* form.

In the $(*, +)$ notation of the Boolean operations, these forms are called *sum-of-products* (*SOP*) and *product-of-sums* (*POS*) respectively. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

Theorem 7.3.

A Boolean function $f : (B_2)^n \rightarrow B_2, n \in \mathbb{N}^*$ can be transformed into two equivalent forms:

1. *disjunctive canonical form (DCF): disjunction of conjunctions*

$$(1) f(x_1, \dots, x_n) = \bigvee_{(\alpha_1, \dots, \alpha_n) \in B_2^n} (f(\alpha_1, \dots, \alpha_n) \wedge x_1^{\alpha_1} \wedge \dots \wedge x_n^{\alpha_n})$$

2. *conjunctive canonical form (CCF): conjunction of disjunctions*

$$(2) f(x_1, \dots, x_n) = \bigwedge_{(\alpha_1, \dots, \alpha_n) \in B_2^n} (f(\alpha_1, \dots, \alpha_n) \vee x_1^{\overline{\alpha}_1} \vee \dots \vee x_n^{\overline{\alpha}_n})$$

Theorem 7.4.

A Boolean function $f : (B_2)^n \rightarrow B_2$ is unique determined by its values $f(\alpha_1, \dots, \alpha_n)$, where $(\alpha_1, \dots, \alpha_n) \in B_2^n$:

1. *disjunctive canonical form (DCF):*

$$(1) \Leftrightarrow (1') f(x_1, \dots, x_n) = \bigvee_{(\alpha_1, \dots, \alpha_n) \in B_2^n \text{ and } f(\alpha_1, \dots, \alpha_n) = 1} (x_1^{\alpha_1} \wedge \dots \wedge x_n^{\alpha_n})$$

2. *conjunctive canonical form (CCF):*

$$(2) \Leftrightarrow (2') f(x_1, \dots, x_n) = \bigwedge_{(\alpha_1, \dots, \alpha_n) \in B_2^n \text{ and } f(\alpha_1, \dots, \alpha_n) = 0} (x_1^{\overline{\alpha}_1} \vee \dots \vee x_n^{\overline{\alpha}_n})$$

Example 7.6.

For $n=2$, the disjunctive canonical forms of the functions f_8, f_{13}, f_6, f_{12} from Example 7.5 are built using formula (1):

$$f_8(x, y) = (1 \wedge x^0 \wedge y^0) \vee (0 \wedge x^0 \wedge y^1) \vee (0 \wedge x^1 \wedge y^0) \vee (0 \wedge x^1 \wedge y^1) = \bar{x} \wedge \bar{y}$$

$$\begin{aligned} f_{13}(x, y) &= (1 \wedge x^0 \wedge y^0) \vee (1 \wedge x^0 \wedge y^1) \vee (0 \wedge x^1 \wedge y^0) \vee (1 \wedge x^1 \wedge y^1) = \\ &= (\bar{x} \wedge \bar{y}) \vee (\bar{x} \wedge y) \vee (x \wedge y) \end{aligned}$$

$$f_6(x, y) = (0 \wedge x^0 \wedge y^0) \vee (1 \wedge x^0 \wedge y^1) \vee (1 \wedge x^1 \wedge y^0) \vee (0 \wedge x^1 \wedge y^1) = (\bar{x} \wedge y) \vee (x \wedge \bar{y})$$

$$\begin{aligned} f_{12}(x, y) &= (1 \wedge x^0 \wedge y^0) \vee (1 \wedge x^0 \wedge y^1) \vee (0 \wedge x^1 \wedge y^0) \vee (0 \wedge x^1 \wedge y^1) = \\ &= (\bar{x} \wedge \bar{y}) \vee (\bar{x} \wedge y) = \bar{x} \wedge (\bar{y} \vee y) = \bar{x} \end{aligned}$$

A Computational Approach to Classical Logics and Circuits

Note that the functions' indices, represented in binary using 4 binary digits, are the binary numbers obtained reading the corresponding column.

For $f_{12}(x, y)$, $12=1100_{(2)}$ and the column of the function's values is: $\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$.

Remarks:

1. The disjunctive canonical form is recommended if the function has more values 0 than values 1.
2. The conjunctive canonical form is recommended if the function has more values 1 than values 0.

Example 7.7.

Write the canonical forms of $f_9(x, y)$.

$$f_9(x, y) = (1 \wedge x^0 \wedge y^0) \vee (0 \wedge x^0 \wedge y^1) \vee (0 \wedge x^1 \wedge y^0) \vee (1 \wedge x^1 \wedge y^1) = (\bar{x} \wedge \bar{y}) \vee (x \wedge y) \quad \text{--- DCF}$$

and by applying distributivity we obtain:

$$\begin{aligned} &= ((\bar{x} \wedge \bar{y}) \vee x) \wedge ((\bar{x} \wedge \bar{y}) \vee y) = (\bar{x} \vee x) \wedge (\bar{y} \vee y) \wedge (\bar{x} \vee y) \wedge (\bar{y} \vee y) = \\ &= 1 \wedge (x \vee \bar{y}) \wedge (\bar{x} \wedge y) \wedge 1 = (x \vee \bar{y}) \wedge (\bar{x} \vee y) \quad \text{--- CCF} \end{aligned}$$

Example 7.8.

Write the corresponding canonical forms of f_2, f_5, f_3, f_0 , according to the previous remark:

Formula (1') will be used for f_2, f_5 and formula (2') will be used for f_3, f_0 :

$$f_2(x, y) = (x^1 \wedge y^0) = x \wedge \bar{y} \quad (\text{DCF})$$

$$f_5(x, y) = (x^0 \wedge y^1) \vee (x^1 \wedge y^1) = (\bar{x} \wedge y) \vee (x \wedge y) = (\bar{x} \vee x) \wedge y = 1 \wedge y = y \quad (\text{DCF})$$

$$f_3(x, y) = (x^{\bar{0}} \vee y^{\bar{0}}) \wedge (x^{\bar{0}} \vee y^{\bar{1}}) = (x \vee y) \wedge (x \vee \bar{y}) = x \wedge (y \vee \bar{y}) = x \wedge 1 = x \quad (\text{CCF})$$

$$\begin{aligned} f_0(x, y) &= (x^{\bar{0}} \vee y^{\bar{0}}) \wedge (x^{\bar{0}} \vee y^{\bar{1}}) \wedge (x^{\bar{1}} \vee y^{\bar{0}}) \wedge (x^{\bar{1}} \vee y^{\bar{1}}) \\ &= (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y}) = ((x \vee y) \wedge (x \vee \bar{y})) \wedge ((\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y})) \\ &= (x \vee (y \wedge \bar{y})) \vee (\bar{x} \vee (y \wedge \bar{y})) = (x \vee 0) \wedge (\bar{x} \vee 0) = x \wedge \bar{x} = 0 \quad (\text{CCF}) \end{aligned}$$

Remarks:

- The Boolean function f_0 does not have a disjunctive canonical form.
- The Boolean function $f_{2^{2^n}-1}$ does not have a conjunctive canonical form.

Boolean Algebras and Boolean Functions

Definition 7.3.

Let $f : (B_2)^n \rightarrow B_2$, $n \in N^*$ be a Boolean function of n variables.

1. A conjunction of variables is called a **monom**.
2. A monom which contains all n variables is called **canonical monom (conjunction)** or **minterm** of n variables and has the form:
 $x_1^{\alpha_1} \wedge \dots \wedge x_n^{\alpha_n}, \alpha_i \in B_2$.
3. A disjunction containing all n variables: $x_1^{\alpha_1} \vee \dots \vee x_n^{\alpha_n}, \alpha_i \in B_2$ is called **canonical disjunction** or **maxterm** of n variables.

Example 7.9.

For $n=3$:

- $x_1 \wedge x_2$ is a monom, but not a minterm (canonical monom);
- $x_1 \wedge \bar{x}_2 \wedge x_3$ is a minterm (canonical conjunction) of 3 variables;
- $\bar{x}_1 \vee \bar{x}_2 \vee x_3$ is a maxterm (canonical disjunction) of 3 variables.

Properties:

- $\forall n \in N^*$, there are 2^n maxterms, denoted by $M_0, M_1, \dots, M_{2^n-1}$ and 2^n minterms, denoted by $m_0, m_1, \dots, m_{2^n-1}$.
- A maxterm is a Boolean function which is 0 for only one argument.
- A minterm is a Boolean function which is 1 for only one argument.

Remarks:

- The index of the standard notation of a minterm of n variables is obtained by converting into decimal of the binary number composed of the digits representing the powers of all n variables from the minterm expression.
- The index of the standard notation of a maxterm of n variables is obtained by the conversion into decimal of the binary number composed of the duals of the digits, digits representing the powers of all n variables from the maxterm expression.

Example 7.10.

For $n=1$, there are $2^n = 2^1 = 2$ minterms and 2 maxterms:

x	$m_0 = M_1 = \bar{x}$	$m_1 = M_0 = x$
0	1	0
1	0	1

A Computational Approach to Classical Logics and Circuits

Example 7.11.

For $n=2$, there are 4 minterms and 4 maxterms having the following expressions and tables of values:

x	y	$m_0 = \overline{x} \wedge \overline{y}$	$m_1 = x \wedge \overline{y}$	$m_2 = \overline{x} \wedge y$	$m_3 = x \wedge y$	$M_0 = \overline{x} \vee \overline{y}$	$M_1 = x \vee \overline{y}$	$M_2 = \overline{x} \vee y$	$M_3 = x \vee y$
0	0	1	0	0	0	0	1	1	1
0	1	0	1	0	0	1	0	1	1
1	0	0	0	1	0	1	1	0	1
1	1	0	0	0	1	1	1	1	0

To be noted that each minterm/maxterm has a unique value 1/0 on its column.

$$m_0 = m_{00_{(2)}} = x^0 \wedge y^0 = \overline{x} \wedge \overline{y}, \quad m_3 = m_{11_{(2)}} = x^1 \wedge y^1 = x \wedge y,$$

$$M_1 = M_{01_{(2)}} = x^{\overline{0}} \vee y^{\overline{1}} = x \vee \overline{y}, \quad M_2 = M_{10_{(2)}} = x^{\overline{1}} \vee y^{\overline{0}} = \overline{x} \vee y.$$

Example 7.12.

For $n=3$, there are 8 minterms (m_0, m_1, \dots, m_7) and 8 maxterms (M_0, M_1, \dots, M_7).

$$m_0 = m_{000_{(2)}} = x^0 \wedge y^0 \wedge z^0 = \overline{x} \wedge \overline{y} \wedge \overline{z}, \quad M_0 = M_{000_{(2)}} = x^{\overline{0}} \vee y^{\overline{0}} \vee z^{\overline{0}} = x \vee y \vee z$$

$$m_1 = m_{001_{(2)}} = x^0 \wedge y^0 \wedge z^1 = \overline{x} \wedge \overline{y} \wedge z, \quad M_1 = M_{001_{(2)}} = x^{\overline{0}} \vee y^{\overline{0}} \vee z^{\overline{1}} = x \vee y \vee \overline{z}$$

$$m_2 = m_{010_{(2)}} = x^0 \wedge y^1 \wedge z^0 = \overline{x} \wedge y \wedge \overline{z}, \quad M_2 = M_{010_{(2)}} = x^{\overline{0}} \vee y^{\overline{1}} \vee z^{\overline{0}} = x \vee \overline{y} \vee z$$

$$m_3 = m_{011_{(2)}} = x^0 \wedge y^1 \wedge z^1 = \overline{x} \wedge y \wedge z, \quad M_3 = M_{011_{(2)}} = x^{\overline{0}} \vee y^{\overline{1}} \vee z^{\overline{1}} = x \vee \overline{y} \vee \overline{z}$$

$$m_4 = m_{100_{(2)}} = x^1 \wedge y^0 \wedge z^0 = x \wedge \overline{y} \wedge \overline{z}, \quad M_4 = M_{100_{(2)}} = x^{\overline{1}} \vee y^{\overline{0}} \vee z^{\overline{0}} = \overline{x} \vee y \vee z$$

$$m_5 = m_{101_{(2)}} = x^1 \wedge y^0 \wedge z^1 = x \wedge \overline{y} \wedge z, \quad M_5 = M_{101_{(2)}} = x^{\overline{1}} \vee y^{\overline{0}} \vee z^{\overline{1}} = \overline{x} \vee y \vee \overline{z}$$

$$m_6 = m_{110_{(2)}} = x^1 \wedge y^1 \wedge z^0 = x \wedge y \wedge \overline{z}, \quad M_6 = M_{110_{(2)}} = x^{\overline{1}} \vee y^{\overline{1}} \vee z^{\overline{0}} = \overline{x} \vee \overline{y} \vee z$$

$$m_7 = m_{111_{(2)}} = x^1 \wedge y^1 \wedge z^1 = x \wedge y \wedge z, \quad M_7 = M_{111_{(2)}} = x^{\overline{1}} \vee y^{\overline{1}} \vee z^{\overline{1}} = \overline{x} \vee \overline{y} \vee \overline{z}$$

Theorem 7.5.

1. The conjunction of two distinct minterms is 0.

$$m_i \wedge m_j = 0, \quad \forall i \neq j, \quad i, j \in \{0, \dots, 2^{n-1}\}.$$

2. The disjunction of two distinct maxterms is 1:

$$M_i \vee M_j = 1, \quad \forall i \neq j, \quad i, j \in \{0, \dots, 2^{n-1}\}.$$

3. A minterm and a maxterm with the same index are inverse functions.

$$M_i = \overline{m}_i, \quad \overline{M}_i = m_i, \quad \forall i \in \{0, \dots, 2^{n-1}\}.$$

Boolean Algebras and Boolean Functions

Example 7.13.

- $m_2 \wedge m_6 = (x^0 \wedge y^1 \wedge z^0) \wedge (x^1 \wedge y^1 \wedge z^0) = (\bar{x} \wedge y \wedge \bar{z}) \wedge (x \wedge y \wedge \bar{z}) =$
 $= (\bar{x} \wedge x) \wedge y \wedge \bar{z} = 0 \wedge y \wedge \bar{z} = 0$
- $M_6 \vee M_2 = (x^1 \vee y^1 \vee z^0) \vee (x^0 \vee y^1 \vee z^0) = (\bar{x} \vee \bar{y} \vee z) \vee (x \vee \bar{y} \vee z) =$
 $= (\bar{x} \vee x) \vee \bar{y} \vee z = 1 \vee \bar{y} \vee z = 1$
- $\overline{m_3} = \overline{x^0 \wedge y^1 \wedge z^1} = x^0 \vee y^1 \vee z^1 = x \vee \bar{y} \vee \bar{z} = M_3$

Remarks:

- The conjunctive canonical form, CCF, is the conjunction of the maxterms corresponding to the values 0 of the function.
- The disjunctive canonical form, DCF, is the disjunction of the minterms corresponding to the values 1 of the function.

Example 7.14.

A Boolean function of 3 variables: $f(x_1, x_2, x_3)$ is given below by its table of values. Write the canonical forms of f .

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	M_2	M_4	M_5	m_0	m_1	m_3	m_6	m_7
0	0	0	1	1	1	1	1	0	0	0	0
0	0	1	1	1	1	1	0	1	0	0	0
0	1	0	0	0	1	1	0	0	0	0	0
0	1	1	1	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0
1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	1	1	1	0	0	0	0	1

The last eight columns of the table represent the tables of values of the maxterms and the minterms used to write the canonical forms of f .

In the conjunctive canonical form, the maxterms: M_2 , M_4 and M_5 are used, and they correspond to the values 0 (on rows 3,5,6) of the function f :

$$CCF(f) = M_2 \wedge M_4 \wedge M_5 = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$

There are five minterms: m_0 , m_1 , m_3 , m_6 and m_7 corresponding to the values 1 (on rows 1, 2, 4, 7, 8) of the function and they are used to write DCF:

$$\begin{aligned} DCF(f) &= m_0 \vee m_1 \vee m_3 \vee m_6 \vee m_7 = \\ &= (\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge x_3) \vee (\bar{x}_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_1 \wedge x_2 \wedge x_3) \end{aligned}$$

A Computational Approach to Classical Logics and Circuits

7.4. Simplification of Boolean functions

Although the truth table representation of a Boolean function is unique, its algebraic expression may be of many different forms. Boolean functions may be simplified by algebraic means using the axioms from Definition 7.1, or by applying simplification methods. Simplification of Boolean functions is essential for the development of simple logic circuits, as we shall see in the next chapter.

In this section the theoretical concepts and a general simplification algorithm are introduced, then three simplification methods are presented and applied in examples.

To simplify a Boolean function means to obtain an equivalent expression of the function with a minimum number of occurrences of variables and connectives. As input data for the simplification process canonical forms are used. We present the simplification procedure based on the disjunctive canonical form.

Definition 7.4.

Let $f : (B_2)^n \rightarrow B_2$ be a Boolean function of n variables.

The set $S_f = \{(x_1, x_2, \dots, x_n) | f(x_1, x_2, \dots, x_n) = 1\}$, containing all the groups $(x_1, x_2, \dots, x_n) \in B_2^n$ for which f takes the value 1, is called the **support** of f .

Example 7.15.

For $n = 4$ variables, the support of the monom $m = x_1 \wedge \bar{x}_3$, is:
 $S_m = \{(1,1,0,1), (1,0,0,1), (1,1,0,0), (1,0,0,0)\}$

Definition 7.5.

The monom m is **smaller or equal** than the monom m' if the support of m is included or equal to the support of m' :

$$m \leq m' \text{ if } S_m \subseteq S_{m'}$$

Example 7.16.

Let $m = x_1 \wedge \bar{x}_2 \wedge x_4$ and $m' = x_1 \wedge \bar{x}_2$, be two monoms of $n = 4$ variables.

$$S_m = \{(1,0,0,1), (1,0,1,1)\}$$

$$S_{m'} = \{(1,0,0,0), (1,0,0,1), (1,0,1,0), (1,0,1,1)\}$$

$$S_m \subseteq S_{m'} \text{ and thus } m \leq m', x_1 \wedge \bar{x}_2 \wedge x_4 \leq x_1 \wedge \bar{x}_2.$$

Note that all the variables of the bigger monom m' occur in the smaller monom m .

Definition 7.6.

For a Boolean function of n variables let us consider the monoms:

$$m = x_{k_1}^{\alpha_{k_1}} \wedge \dots \wedge x_{k_j}^{\alpha_{k_j}} \wedge x_{k_i} \wedge x_{k_l}^{\alpha_{k_l}} \wedge \dots \wedge x_{k_s}^{\alpha_{k_s}} \text{ and}$$

Boolean Algebras and Boolean Functions

$$m' = x_{k_1}^{\alpha_{k_1}} \wedge \dots \wedge x_{k_j}^{\alpha_{k_j}} \wedge \bar{x}_{k_i} \wedge x_{k_l}^{\alpha_{k_l}} \wedge \dots \wedge x_{k_s}^{\alpha_{k_s}}, \text{ where :}$$

$k_1, \dots, k_j, k_i, k_l, \dots, k_s \in \{1, 2, \dots, n\}$ and $k_1 < \dots < k_j < k_i < k_l < \dots < k_s$,

1. m and m' are called **adjacent (neighbor) monoms** because they differ only by the power of the variable with the index " k_i ".
2. **the factorization of the monoms m and m'** is:

$$m \vee m' = x_{k_1}^{\alpha_{k_1}} \wedge \dots \wedge x_{k_j}^{\alpha_{k_j}} \wedge x_{k_l}^{\alpha_{k_l}} \wedge \dots \wedge x_{k_s}^{\alpha_{k_s}} \text{ obtained by eliminating the variable}$$

with the index " k_i ".

Adjacency (neighborhood) relation is defined by a single variable change.

The factorization is based on the application of the distributivity of „ \wedge ” over „ \vee ”. Applied in the particular case of the previous definition, where y is the common part of the monoms, $m = y \wedge x_{k_i}$ and $m' = y \wedge \bar{x}_{k_i}$, factorization provides:

$$(y \wedge x_{k_i}) \vee (y \wedge \bar{x}_{k_i}) = y \wedge (x_{k_i} \vee \bar{x}_{k_i}) = y \wedge 1 = y.$$

As a conclusion, the result of factorization is a monom containing the common part of m and m' , this monom is bigger than both m and m' and it covers them.

Note: In this chapter and the next one we shall use the concatenation of the variables to symbolize their conjunction. For example, the monom $m = x_1 \wedge \bar{x}_2 \wedge x_4$ is represented as $m = x_1 x_2 x_4$.

Definition 7.7. [62]

The set $M(f)$ is called the **set of the maximal monoms** for the function $f : (B_2)^n \rightarrow B_2$ if:

1. $\forall m \in M(f), m \in FB(n), m \leq f$ and
2. $\forall m \in M(f), \exists m' \in FB(n)$ such that $m < m' \leq f$.

Example 7.17.

$f(x, y) = (x \wedge y) \vee (\bar{x} \wedge y) = m_3 \vee m_1$. We have that:

- $m_3 < f$ and $m_1 < f$;
- $m_3 < y \leq f$ and $m_1 < y \leq f$;

So, $m_3, m_1 \notin M(f)$ and $M(f) = \{y\}$

Note: The maximal monoms are minterms or monoms obtained by applying factorizations.

A Computational Approach to Classical Logics and Circuits

Definition 7.8. [62]

The set $C(f)$ is called the *set of the central monoms* for the function $f : (B_2)^n \rightarrow B_2$ if:

1. $\forall m \in C(f), m \in M(f)$ and
2. $\forall m \in C(f), m \not\leq \vee_{m' \in M(f) \setminus \{m\}} m'$

Remarks:

- A maximal monom is a central monom for a function f , if the monom is not smaller than the disjunction of all the other maximal monoms of f .
- All the central monoms belong to all the disjunctive simplified forms of the function f .

Properties:

Let $f : (B_2)^n \rightarrow B_2$ be a Boolean function.

1. $M(f) \neq \emptyset$, but $C(f)$ can be empty.
2. $f(x_1, \dots, x_n) = \bigvee_{m \in M(f)} m$ but this is not always the simplest form of f .

Informally, to simplify a Boolean function given in disjunctive canonical form, means to cover all its minterms with a minimum number of maximal monoms and with a minimum number of overlaps.

The simplification process is formalized by the steps below. The steps 2, 3, 4 are specific to the applied simplification method.

1. The initial function f is transformed into its equivalent disjunctive canonical form $DCF(f)$.
2. Factorization process \Rightarrow the set of maximal monoms $M(f)$.
3. From the set of maximal monoms the central monoms are selected $\Rightarrow C(f)$
4. The case of the simplification algorithm (presented below) is identified and all the simplified forms are obtained.

The following algorithm ([62]) is a general one and it is used in all three simplification methods presented in the next sections:

- Veitch-Karnaugh diagrams method – a graphical method;
- Quine-Mc'Clusky's method – an analytical method;
- Moisil's method – an algebraic method which uses propositional logic.

Boolean Algebras and Boolean Functions

Algorithm Simplification:

input: f – a Boolean function in disjunctive canonical form

output: $f^s_1, f^s_2, \dots, f^s_k$ – all the disjunctive simplified forms of f

begin

 calculate $M(f)$ and $C(f)$

if $M(f) = C(f)$ **then**

$$f^s = \bigvee_{m \in M(f)} m ; \quad \text{STOP 1} // \text{case 1--- one solution}$$

else

if $C(f) \neq \emptyset$ **then** // f has central monoms

$$g = \bigvee_{m \in C(f)} m$$

$f^s_i = g \vee h_i, i = \overline{1, k}$, h_i is a disjunction of a minimum number of maximal monoms such that $S_{h_i} = S_f \setminus S_g$

STOP 2 // case 2 --- k solutions

else // f does not have central monoms

$f^s_i = h_i, i = \overline{1, k}$, h_i is a disjunction of a minimum number of maximal monoms such that $S_{h_i} = S_f$

STOP 3 // case 3 --- k solutions

end_if

end_if

end

7.5. Veitch-Karnaugh diagrams method

This method is based on a graphical representation of the disjunctive canonical form of the initial function. It is recommended to be used for Boolean functions of 2, 3 and 4 variables. For more than 4 variables the graphical representation using diagrams is not so intuitive.

For a function of n variables, the diagram (Veitch or Karnaugh) has 2^n cells, and each cell (intersection of a line and a column) corresponds to a minterm. The headers of the lines and columns represent:

- the variables simple or negated in Veitch diagram
- all the possible combinations of the powers (0,1) of the variables in Karnaugh diagram.

The minterms from the DCF are represented in the diagrams as follows:

A Computational Approach to Classical Logics and Circuits

No. var.	Veitch diagram	Karnaugh diagram
2	$ \begin{array}{c cc} x_1 & \bar{x}_1 \\ \hline x_2 & m_3 & m_1 \\ \hline \bar{x}_2 & m_2 & m_0 \end{array} $	$ \begin{array}{c cc} x_1 x_2 & 0 & 1 \\ \hline 0 & m_0 & m_1 \\ \hline 1 & m_2 & m_3 \end{array} $
3	$ \begin{array}{c cccc} x_1 & \bar{x}_1 \\ \hline x_2 & m_7 & m_6 & m_2 & m_3 \\ \hline \bar{x}_2 & m_5 & m_4 & m_0 & m_1 \\ \hline x_3 & \bar{x}_3 & \bar{x}_3 & x_3 & x_3 \end{array} $	$ \begin{array}{c cccc} x_1 x_2 & 00 & 01 & 11 & 10 \\ \hline 0 & m_0 & m_1 & m_3 & m_2 \\ \hline 1 & m_4 & m_5 & m_7 & m_6 \end{array} $
4	$ \begin{array}{c cc cc c} x_1 & \bar{x}_1 & & & & x_4 \\ \hline x_2 & m_{15} & m_{13} & m_5 & m_7 & x_4 \\ \hline & m_{14} & m_{12} & m_4 & m_6 & \bar{x}_4 \\ \hline \bar{x}_2 & m_{10} & m_8 & m_0 & m_2 & x_4 \\ \hline & m_{11} & m_9 & m_1 & m_3 & x_4 \\ \hline x_3 & \bar{x}_3 & \bar{x}_3 & x_3 & & \end{array} $	$ \begin{array}{c cccc} x_1 \bar{x}_2 \bar{x}_3 x_4 & 00 & 01 & 11 & 10 \\ \hline 00 & m_0 & m_1 & m_3 & m_2 \\ \hline 01 & m_4 & m_5 & m_7 & m_6 \\ \hline 11 & m_{12} & m_{13} & m_{15} & m_{14} \\ \hline 10 & m_8 & m_9 & m_{11} & m_{10} \end{array} $

The cells in a Veitch/Karnaugh diagram are arranged so that there is only a single variable change between neighbor (adjacent cells). Adjacency (graphical neighborhood) relation is defined by a single variable change.

The diagrams are circular, meaning that the first row/column and the last row/column are neighbors. Two minterms belonging to two neighbor cells (graphical neighborhood relation) are neighbors according to Definition 7.6 and they factorize.

For $n = 2$ variables:

- $m_0 = x_1 \bar{x}_2$ and $m_2 = x_1 \bar{x}_2$ are neighbors, they differ only by variable x_1 and they factorize: $m_0 \vee m_2 = \bar{x}_2$
- $m_2 = x_1 \bar{x}_2$ and $m_3 = x_1 x_2$ are neighbors, they differ only by variable x_2 and they factorize: $m_2 \vee m_3 = x_1$

For $n = 3$ variables:

- $m_5 = x_1 \bar{x}_2 x_3$ and $m_7 = x_1 x_2 x_3$ are neighbors, they differ only by variable x_2 and they factorize: $m_5 \vee m_7 = x_1 x_3$

Boolean Algebras and Boolean Functions

- $m_4 = x_1\bar{x}_2x_3$ and $m_6 = x_1x_2\bar{x}_3$ are neighbors, they differ only by variable x_2 and they factorize: $m_4 \vee m_6 = x_1\bar{x}_3$

For $n = 4$ variables:

- $m_{12} = \underline{x_1}x_2\underline{x_3}\bar{x_4}$ and $m_8 = x_1\underline{x_2}\underline{x_3}\bar{x_4}$ are neighbors, they differ only by variable x_2 and they factorize: $m_{12} \vee m_8 = \underline{x_1}\underline{x_3}\bar{x_4}$
 - $m_1 = x_1\underline{x_2}\underline{x_3}\bar{x_4}$ and $m_3 = \underline{x_1}\underline{x_2}\underline{x_3}\bar{x_4}$ are neighbors, they differ only by variable x_3 and they factorize: $m_1 \vee m_3 = \underline{x_1}\bar{x_2}\bar{x_4}$

The *simple factorization* operation between two neighbor minterms can be generalized to a *k-factorization*: 2^k neighbor minterms (cells) are used. The result of a *k-factorization* is a monom which contains the common variables of all 2^k minterms.

The **maximal monoms** are obtained from the diagram (Veitch/Karnaugh) by applying factorizations as follows: for a Boolean function of n variables we try first a *n -factorization*, we continue with *$n-1$ factorizations*, ..., *simple factorizations* and *0 -factorizations* (isolated minterms).

Examples of factorizations are presented in the following table.

Number of simplified variables	Veitch diagram	The maximal monom obtained after factorization																		
1 from 3 simple factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td style="text-align: center;">x_1</td> <td></td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td style="text-align: center;">m_7</td> <td></td> <td></td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td style="text-align: center;">m_5</td> <td></td> <td></td> </tr> <tr> <td></td> <td style="text-align: center;">x_3</td> <td style="text-align: center;">\bar{x}_3</td> <td style="text-align: center;">x_3</td> </tr> </table>		x_1		\bar{x}_1	x_2	m_7			\bar{x}_2	m_5				x_3	\bar{x}_3	x_3	$m_7 \vee m_5 = x_1 x_3$		
	x_1		\bar{x}_1																	
x_2	m_7																			
\bar{x}_2	m_5																			
	x_3	\bar{x}_3	x_3																	
1 from 3 simple factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td style="text-align: center;">x_1</td> <td></td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td style="text-align: center;">m_7</td> <td></td> <td style="text-align: center;">m_3</td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td style="text-align: center;">x_3</td> <td style="text-align: center;">\bar{x}_3</td> <td style="text-align: center;">x_3</td> </tr> </table>		x_1		\bar{x}_1	x_2	m_7		m_3	\bar{x}_2					x_3	\bar{x}_3	x_3	$m_7 \vee m_3 = x_2 x_3$		
	x_1		\bar{x}_1																	
x_2	m_7		m_3																	
\bar{x}_2																				
	x_3	\bar{x}_3	x_3																	
2 from 3 double factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td style="text-align: center;">x_1</td> <td></td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td></td> <td style="text-align: center;">m_6</td> <td style="text-align: center;">m_2</td> <td></td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td></td> <td style="text-align: center;">m_4</td> <td style="text-align: center;">m_0</td> <td></td> </tr> <tr> <td></td> <td style="text-align: center;">x_3</td> <td style="text-align: center;">\bar{x}_3</td> <td style="text-align: center;">x_3</td> </tr> </table>		x_1		\bar{x}_1	x_2		m_6	m_2		\bar{x}_2		m_4	m_0			x_3	\bar{x}_3	x_3	$m_0 \vee m_2 \vee m_4 \vee m_6 = \\ = \bar{x}_3$
	x_1		\bar{x}_1																	
x_2		m_6	m_2																	
\bar{x}_2		m_4	m_0																	
	x_3	\bar{x}_3	x_3																	

A Computational Approach to Classical Logics and Circuits

Number of simplified variables	Veitch diagram	The maximal monom obtained after factorization																				
2 from 3 double factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">x_1</td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td>m_7</td> <td></td> <td></td> <td>m_3</td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td>m_5</td> <td></td> <td></td> <td>m_1</td> </tr> <tr> <td style="text-align: center;">x_3</td> <td></td> <td style="text-align: center;">\bar{x}_3</td> <td></td> <td>x_3</td> </tr> </table>	x_1	\bar{x}_1	x_2	m_7			m_3	\bar{x}_2	m_5			m_1	x_3		\bar{x}_3		x_3	$m_1 \vee m_3 \vee m_5 \vee m_7 = \\ = x_3$			
x_1	\bar{x}_1																					
x_2	m_7			m_3																		
\bar{x}_2	m_5			m_1																		
x_3		\bar{x}_3		x_3																		
2 from 3 double factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">x_1</td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td>m_7</td> <td>m_6</td> <td>m_2</td> <td>m_3</td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td style="text-align: center;">x_3</td> <td></td> <td style="text-align: center;">\bar{x}_3</td> <td></td> <td>x_3</td> </tr> </table>	x_1	\bar{x}_1	x_2	m_7	m_6	m_2	m_3	\bar{x}_2					x_3		\bar{x}_3		x_3	$m_2 \vee m_3 \vee m_6 \vee m_7 = \\ = x_2$			
x_1	\bar{x}_1																					
x_2	m_7	m_6	m_2	m_3																		
\bar{x}_2																						
x_3		\bar{x}_3		x_3																		
1 from 4 simple factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">x_1</td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td>m_{15}</td> <td>m_{13}</td> <td></td> <td></td> <td>x_4</td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td></td> <td></td> <td></td> <td></td> <td>\bar{x}_4</td> </tr> <tr> <td style="text-align: center;">x_3</td> <td></td> <td style="text-align: center;">\bar{x}_3</td> <td></td> <td>x_3</td> <td>x_4</td> </tr> </table>	x_1	\bar{x}_1	x_2	m_{15}	m_{13}			x_4	\bar{x}_2					\bar{x}_4	x_3		\bar{x}_3		x_3	x_4	$m_{13} \vee m_{15} = x_1 x_2 x_4$
x_1	\bar{x}_1																					
x_2	m_{15}	m_{13}			x_4																	
\bar{x}_2					\bar{x}_4																	
x_3		\bar{x}_3		x_3	x_4																	
2 from 4 double factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">x_1</td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td>m_{15}</td> <td>m_{13}</td> <td></td> <td></td> <td>x_4</td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td>m_{14}</td> <td>m_{12}</td> <td></td> <td></td> <td>\bar{x}_4</td> </tr> <tr> <td style="text-align: center;">x_3</td> <td></td> <td style="text-align: center;">\bar{x}_3</td> <td></td> <td>x_3</td> <td>x_4</td> </tr> </table>	x_1	\bar{x}_1	x_2	m_{15}	m_{13}			x_4	\bar{x}_2	m_{14}	m_{12}			\bar{x}_4	x_3		\bar{x}_3		x_3	x_4	$m_{12} \vee m_{13} \vee m_{14} \vee m_{15} = \\ = x_1 x_2$
x_1	\bar{x}_1																					
x_2	m_{15}	m_{13}			x_4																	
\bar{x}_2	m_{14}	m_{12}			\bar{x}_4																	
x_3		\bar{x}_3		x_3	x_4																	
2 from 4 double factorization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">x_1</td> <td style="text-align: center;">\bar{x}_1</td> </tr> <tr> <td style="text-align: center;">x_2</td> <td></td> <td>m_{13}</td> <td>m_5</td> <td></td> <td>x_4</td> </tr> <tr> <td style="text-align: center;">\bar{x}_2</td> <td></td> <td></td> <td></td> <td></td> <td>\bar{x}_4</td> </tr> <tr> <td style="text-align: center;">x_3</td> <td></td> <td>m_9</td> <td>m_1</td> <td></td> <td>x_4</td> </tr> </table>	x_1	\bar{x}_1	x_2		m_{13}	m_5		x_4	\bar{x}_2					\bar{x}_4	x_3		m_9	m_1		x_4	$m_1 \vee m_5 \vee m_9 \vee m_{13} = \\ = \bar{x}_3 x_4$
x_1	\bar{x}_1																					
x_2		m_{13}	m_5		x_4																	
\bar{x}_2					\bar{x}_4																	
x_3		m_9	m_1		x_4																	

Boolean Algebras and Boolean Functions

Number of simplified variables	Veitch diagram	The maximal monom obtained after factorization
2 from 4 double factorization		$m_3 \vee m_7 \vee m_{11} \vee m_{15} = x_3 x_4$
2 from 4 double factorization		$m_{10} \vee m_{11} \vee m_{14} \vee m_{15} = x_1 x_3$
3 from 4 triple factorization		$m_0 \vee m_1 \vee m_4 \vee m_5 \vee m_8 \vee m_9 \vee m_{12} \vee m_{13} = \bar{x}_3$
3 from 4 triple factorization		$m_1 \vee m_3 \vee m_5 \vee m_7 \vee m_9 \vee m_{11} \vee m_{13} \vee m_{15} = x_4$

A Computational Approach to Classical Logics and Circuits

The **central monoms** are the maximal monoms which cannot be covered by the disjunction of all the other maximal monoms. In the diagrams, the identification of the central monoms is very simple: if the group of the minterms covered by a maximal monom contains at least one cell (minterm) circled once, then the maximal monom is a central one and it belongs to all the simplified forms of the function.

We follow the general simplification algorithm and we have to identify the simplification case and obtain all the simplified forms.

The groups of the minterms corresponding to the central monoms are shaded in the diagram. The minterms from the function's expression which are unshaded, are not covered by the central monoms and they will be covered in all the possible ways using a minimum number of unused maximal monoms and with a minimum number of overlaps, resulting all the simplified forms of the initial function.

Example 7.18.

Simplify the following Boolean function given by its Boolean DCF expression.

$$f(x_1, x_2, x_3) = x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3$$

Using the standard notations for the minterms, the function can be written as:

$$f(x_1, x_2, x_3) = m_7 \vee m_6 \vee m_2 \vee m_5 \vee m_4 \vee m_1$$

The representation of the minterms in the diagrams:

Veitch diagram

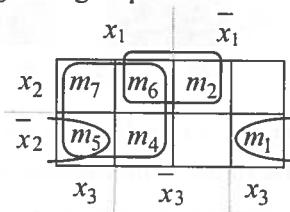
	x_1		\bar{x}_1
x_2	m_7	m_6	m_2
\bar{x}_2	m_5	m_4	m_1
x_3		\bar{x}_3	x_3

Karnaugh diagram

	x_1	x_2x_3	00	01	11	10
0				m_1		m_2
1			m_4	m_5	m_7	m_6

In the next steps we shall use the Veitch diagram.

The **factorization process** consists in factorizing all the possible groups of neighbor minterms, beginning with groups of 4 cells and then groups of 2 cells.



There is a group of 4 neighbor minterms and two groups of 2 neighbor minterms, circled in the diagram. The maximal monoms are obtained as follows:

Boolean Algebras and Boolean Functions

	x_1	\bar{x}_1		
x_2	m_7	m_6	m_2	
\bar{x}_2	m_5	m_4		m_1
	x_3	\bar{x}_3	\bar{x}_3	x_3

Maximal monom: $\max_1 = x_1 = m_7 \vee m_6 \vee m_5 \vee m_4$ (double factorization)

	x_1	x_1	\bar{x}_1	\bar{x}_1
x_2	m_7	m_6	m_2	
\bar{x}_2	m_5	m_4		m_1
	x_3	\bar{x}_3	\bar{x}_3	x_3

Maximal monom: $\max_2 = x_2 \bar{x}_3 = m_6 \vee m_2$ (simple factorization).

	x_1	\bar{x}_1	\bar{x}_1	\bar{x}_1
x_2	m_7	m_6	m_2	
\bar{x}_2	m_5	m_4		m_1
	x_3	\bar{x}_3	\bar{x}_3	x_3

Maximal monom: $\max_3 = \bar{x}_2 x_3 = m_5 \vee m_1$ (simple factorization).

The set of the maximal monoms is $M(f) = \{\max_1, \max_2, \max_3\} = \{x_1, x_2 \bar{x}_3, \bar{x}_2 x_3\}$.

The **central monoms** are identified. A maximal monom is a central monom if its corresponding group of minterms contains at least one minterm circled once. The minterms covered by the central monoms are shaded in the diagram.

	x_1	\bar{x}_1		
x_2	m_7	m_6	m_2	
\bar{x}_2	m_5	m_4		m_1
	x_3	\bar{x}_3	\bar{x}_3	x_3

The set of the central monoms is $C(f) = \{\max_1, \max_2, \max_3\} = \{x_1, x_2 \bar{x}_3, \bar{x}_2 x_3\}$
 $M(f) = C(f)$ --- the first case of the simplification algorithm.

There is a unique disjunctive simplified form of f , obtained as the disjunction of all the central monoms: $f^S(x_1, x_2, x_3) = x_1 \vee x_2 \bar{x}_3 \vee \bar{x}_2 x_3$

A Computational Approach to Classical Logics and Circuits

Example 7.19.

Simplify the following Boolean function:

$$f(x_1, x_2, x_3) = x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \vee \bar{x}_1 x_2 x_3.$$

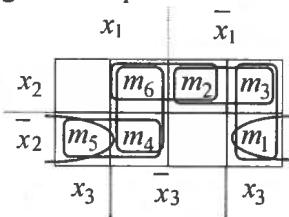
The function's expression is a disjunctive form but not a canonical one because the first and the third monoms are not minterms. In order to obtain the DCF we replace these monoms by the minterms covered by them:

- the monom $x_2 \bar{x}_3$ covers the minterms $x_1 x_2 \bar{x}_3$ and $\bar{x}_1 x_2 \bar{x}_3$ obtained from the initial monom and adding the missing variable, simple: x_1 and negated: \bar{x}_1 .
- the monom $x_1 \bar{x}_2$ covers the minterms $x_1 \bar{x}_2 x_3$ and $x_1 \bar{x}_2 \bar{x}_3$ obtained from the initial monom and adding the missing variable, simple: x_3 and negated: \bar{x}_3 .

The disjunctive canonical form of f is provided below using the Boolean expression and the standard notations of the minterms.

$$\begin{aligned} DCF(f) &= x_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_2 \bar{x}_3 = \\ &= m_6 \vee m_2 \vee m_3 \vee m_5 \vee m_4 \vee m_1 \end{aligned}$$

The corresponding Veitch diagram is depicted below.



The set of the **maximal monoms** is obtained by applying six simple factorizations on groups of two neighbor cells. All these six groups are circled in the diagram.

$$max_1 = x_1 \bar{x}_2 = m_5 \vee m_4,$$

$$max_3 = x_2 \bar{x}_3 = m_6 \vee m_2,$$

$$max_5 = x_1 x_2 = m_2 \vee m_3,$$

$$max_2 = x_1 \bar{x}_3 = m_4 \vee m_6$$

$$max_4 = \bar{x}_2 x_3 = m_5 \vee m_1$$

$$max_6 = \bar{x}_1 x_3 = m_3 \vee m_1$$

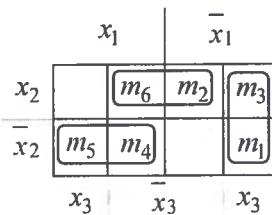
$$M(f) = \{max_1, max_2, max_3, max_4, max_5, max_6\} = \{x_1 \bar{x}_2, x_1 \bar{x}_3, x_2 \bar{x}_3, x_2 x_3, \bar{x}_1 x_2, \bar{x}_1 x_3\}$$

The set of the **central monoms**: $C(f) = \emptyset$ because all the minterms are circled twice, meaning that each minterm can be covered by two maximal monoms.

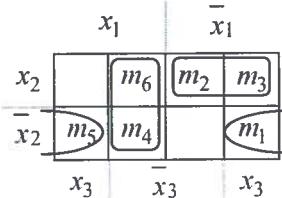
$M(f) \neq C(f)$ and $C(f) = \emptyset$, thus the third case of the simplification algorithm is applied.

All the minterms must be covered by a minimum number of maximal monoms, with a minimum number of overlaps. There are two such possibilities: all the minterms are covered by three maximal monoms and without overlaps, obtaining two equivalent disjunctive simplified forms of f .

Boolean Algebras and Boolean Functions



$$f_1^S(x_1, x_2, x_3) = x_1 \bar{x}_2 \vee x_2 \bar{x}_3 \vee \bar{x}_1 x_3 = max_1 \vee max_3 \vee max_6$$



$$f_2^S(x_1, x_2, x_3) = x_1 \bar{x}_3 \vee \bar{x}_1 x_2 \vee \bar{x}_2 x_3 = max_2 \vee max_5 \vee max_4$$

Example 7.20.

Using Karnaugh diagram simplify the Boolean function f given by its values 0: $f(0,0,0)=f(0,1,1)=f(1,1,1)=0$.

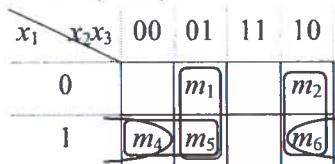
The table of its values is provided below.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

The disjunctive canonical form of f is obtained from the table:

$$f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3 = m_1 \vee m_2 \vee m_4 \vee m_5 \vee m_6$$

We fill the Karnaugh diagram as follows:



$M(f) = \{max_1, max_2, max_3, max_4\} = \{x_1 \bar{x}_2, x_1 \bar{x}_3, x_2 \bar{x}_3, \bar{x}_2 x_3\}$ is obtained by applying four simple factorizations:

A Computational Approach to Classical Logics and Circuits

$$\max_1 = \overline{x_1} \overline{x_2} = m_5 \vee m_4$$

$$\max_2 = \overline{x_1} x_3 = m_4 \vee m_6$$

$$\max_3 = x_2 \overline{x_3} = m_6 \vee m_2$$

$$\max_4 = x_2 x_3 = m_5 \vee m_1$$

The set of the central monoms is $C(f) = \{\max_3, \max_4\} = \{x_2 \overline{x_3}, \overline{x_2} x_3\}$

- \max_1 is not a central monom because it is covered by the disjunction of \max_2 and \max_4 :

$$\max_1 \leq \max_2 \vee \max_4;$$

- \max_2 is not a central monom because it is covered by the disjunction of \max_1 and \max_3 :

$$\max_2 \leq \max_1 \vee \max_3;$$

- \max_3 is a central monom because it is not covered by the disjunction of all the other maximal monoms:

$$\max_3 \not\leq \max_1 \vee \max_2 \vee \max_4, \text{ the minterm } m_2 \text{ is circled once};$$

- \max_4 is a central monom because it is not covered by the disjunction of all the other maximal monoms:

$$\max_4 \not\leq \max_1 \vee \max_2 \vee \max_3, \text{ the minterm } m_1 \text{ is circled once.}$$

$M(f) \neq C(f)$ and $C(f) \neq \emptyset$, this is the second case of the simplification algorithm.

We denote by $g(x_1, x_2, x_3) = \overline{x_2} \overline{x_3} \vee \overline{x_2} x_3 = \max_3 \vee \max_4$

(the disjunction of the central monoms)

x_1	x_2	x_3	00	01	11	10
0				m_1		m_2
1			m_4	m_5		m_6

The groups of cells corresponding to the central monoms are shaded in the diagram. The only unshaded minterm m_4 must be covered in all the possible ways by a minimum number of unused maximal monoms and with a minimum number of overlaps.

We can use two such functions h (see the simplification algorithm):

$$h_1(x_1, x_2, x_3) = \overline{x_1} \overline{x_2} = \max_1 \quad \text{or} \quad h_2(x_1, x_2, x_3) = \overline{x_1} x_3 = \max_2$$

Correspondingly, there are two disjunctive simplified forms of the initial function:

$$f_1^S(x_1, x_2, x_3) = g \vee h_1 = \max_3 \vee \max_4 \vee \max_1 = \overline{x_2} \overline{x_3} \vee \overline{x_2} x_3 \vee \overline{x_1} \overline{x_2}$$

$$f_2^S(x_1, x_2, x_3) = g \vee h_2 = \max_3 \vee \max_4 \vee \max_2 = \overline{x_2} \overline{x_3} \vee \overline{x_2} x_3 \vee x_1 \overline{x_3}$$

Boolean Algebras and Boolean Functions

Based on the simplification algorithm, having as input a *DCF*, a dual simplification algorithm that works on *CCF* (conjunction of maxterms) can be obtained.

While in the simplification of *DCF* the values 1 of the function (expressed by the minterms) are covered, in the simplification of *CCF* the values 0 of the function (expressed by the maxterms) are covered.

Similar steps are applied and conjunctive simplified forms are obtained. We build minimal and central disjunctions of a function, the duals of maximal and central monoms (conjunctions). In dual factorizations we work with groups of 2^k neighbor maxterms and the minimal disjunctions of a function are calculated. The maxterms (canonical disjunctions) from the function's expression are covered by a minimum number of minimal disjunctions and with a minimum number of overlaps. Thus, the conjunctive simplified forms are built.

We apply the dual simplification algorithm for the conjunctive canonical form of f . $CCF(f) = M_0 \wedge M_3 \wedge M_7 = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ is obtained from its table of values.

The Karnaugh diagram for $CCF(f)$ is depicted below.

x_1	x_2	x_3	00	01	11	10
0			M_0		M_3	
1					M_7	

In the diagram, the headers of the lines-columns are used to express the indices of the maxterms and represent the duals of the powers of the variables from the maxterms' expressions.

$$M_3 = M_{011_{(2)}} = x_1^0 \vee x_2^1 \vee x_3^1 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$$

There is an isolated maxterm (M_0) and a group of two neighbor maxterms (M_3, M_7). By applying a simple dual factorization we obtain a minimal disjunction of f :

- $min_1 = M_3 \wedge M_7 = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) = (x_1 \wedge \bar{x}_1) \vee \bar{x}_2 \vee \bar{x}_3 = 0 \vee \bar{x}_2 \vee \bar{x}_3 = \bar{x}_2 \vee \bar{x}_3$ (the common part of the maxterms).
- min_1 is smaller than both M_3 and M_7 and it covers the values 0 of both maxterms.

The minimal disjunctions of f are min_1 and $min_2 = M_0 = x_1 \vee x_2 \vee x_3$.

These are also central disjunctions of f , because each group of cells has at least one maxterm circled once. This is the first case of the dual simplification algorithm.

The function f has a unique conjunctive simplified form obtained as the conjunction of min_1 and min_2 .

$$f^{CS}(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

A Computational Approach to Classical Logics and Circuits

Let us compare the disjunctive simplified forms with the conjunctive form:

- f_1^S and f_2^S have 6 occurrences of variables, 5 binary operations and 3 unary operations.
- f^{CS} has 5 occurrences of variables, 4 binary operations and 2 unary operations.

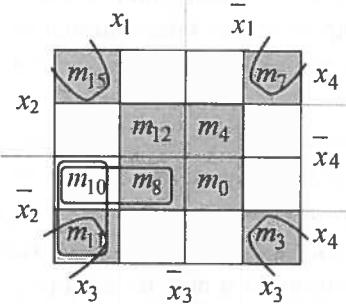
We conclude that $f^{CS}(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$ is the simplest equivalent form of f .

Example 7.21.

Using Veitch diagram simplify the Boolean function:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1 \bar{x}_2 x_3 x_4 \vee x_1 x_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 x_3 \bar{x}_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 x_3 x_4 \vee x_1 \bar{x}_2 x_3 x_4 \vee \\ &\quad \vee x_1 x_2 x_3 x_4 \vee \bar{x}_1 x_2 x_3 x_4 \vee x_1 x_2 \bar{x}_3 x_4 = \\ &= m_{15} \vee m_{12} \vee m_4 \vee m_{10} \vee m_7 \vee m_8 \vee m_{11} \vee m_3 \vee m_0 \end{aligned}$$

The function's expression is the disjunctive canonical form and we fill with the minterms the Veitch diagram of four variables:



$$m_{15} \vee m_7 \vee m_{11} \vee m_3 = x_3 x_4 = max_1 \text{ (double factorization)}$$

$$m_{12} \vee m_4 \vee m_0 \vee m_8 = \bar{x}_3 x_4 = max_2 \text{ (double factorization)}$$

$$m_{10} \vee m_{11} = x_1 \bar{x}_2 x_3 = max_3 \text{ (simple factorization)}$$

$$m_{10} \vee m_8 = x_1 x_2 \bar{x}_4 = max_4 \text{ (simple factorization)}$$

$$M(f) = \{x_3 x_4, \bar{x}_3 x_4, x_1 \bar{x}_2 x_3, x_1 \bar{x}_2 \bar{x}_4\} - \text{the set of maximal monoms}$$

$$C(f) = \{max_1, max_2\} - \text{the set of central monoms}$$

$$M(f) \neq C(f) \text{ and } C(f) \neq \emptyset \text{ --- the second case of the simplification algorithm.}$$

We denote by $g = max_1 \vee max_2 = x_3 x_4 \vee \bar{x}_3 x_4$ the disjunction of all the central monoms, belonging to all the simplified forms of f .

The minterms covered by the central monoms are shaded in the diagram.

Only the minterm m_{10} is not covered by the central monoms and we can cover it in two ways: using max_3 or using max_4 .

Thus, there are two disjunctive simplified forms of the initial function:

$$f_1^S(x_1, x_2, x_3, x_4) = g \vee max_3 = x_3 x_4 \vee \bar{x}_3 x_4 \vee x_1 \bar{x}_2 x_3$$

$$f_2^S(x_1, x_2, x_3, x_4) = g \vee max_4 = x_3 x_4 \vee \bar{x}_3 x_4 \vee x_1 x_2 \bar{x}_4$$

Boolean Algebras and Boolean Functions

7.6. Quine-Mc'Clusky's method

This method, used to simplify Boolean functions, is an analytical one which can be easily implemented and its advantage over the previous graphical method is that it can be applied for functions of any number of variables.

The Boolean function, f , is given in DCF.

The first step is to order the support set of f ,

$S_f = \{(x_1, x_2, \dots, x_n) \mid f(x_1, x_2, \dots, x_n) = 1\}$, in ascending/descending order, with respect to the number of values "1" in each n -uple.

The minterms from the function's expression are represented using the powers of the variables, in a tableau, each minterm on a row, in ascending/descending order, with respect to the number of the values "1" in the n -uples of the support of the function. The header of the tableau contains the variables names. We will group the minterms (delimited by horizontal lines), such that all the minterms belonging to the same group have the same number of values 1 as powers of the variables. A double horizontal line marks the end of the representation of the initial function.

We continue with simple factorizations obtaining new groups of monoms which will participate in double factorizations and so on.

Only two neighbor groups may contain two adjacent (neighbor) monoms (according to Definition 7.6), which factorize. The result of the **factorization** of two neighbor monoms (m and m') is a new monom represented in a new row at the end of the tableau. The row contains the same values (0,1, or $-$) in the columns corresponding to the common variables (of m and m') and the symbol " $-$ " for the variable which is eliminated. The rows corresponding to m and m' are marked (on the left side), with the meaning that they are not maximal monoms.

All the monoms obtained as results of the factorization of two neighbor groups will form a new group (delimited by a horizontal line), used further in higher order factorizations (double, triple,...).

The symbol " $-$ " cannot be combined with anything else. Thus, beginning with double factorization only rows from two neighbor groups having " $-$ " on the same position (column) will be combined.

A double horizontal line symbolizes the end of a simple, double, triple,... factorization. The end of the factorization process is represented by a triple horizontal line and the tableau is closed.

The set of the **maximal monoms** contains the monoms corresponding to all the unmarked rows from the tableau.

For obtaining the **central monoms** a new tableau representing the correspondence between the maximal monoms (on columns) and the minterms from the function's expression (on rows) is used. A cell in the tableau is marked

A Computational Approach to Classical Logics and Circuits

with an asterisk (*) if the minterm corresponding to the row was used in factorization to obtain the maximal monom from the column. A maximal monom is a central monom if there is a * (on its column), which is unique on its row. The disjunction of all central monoms belongs to all the simplified forms of the initial function.

According to the previous tableau, the minterms from the function's expression which are uncovered by the central monoms will be covered in all the possible ways using a minimum number of unused maximal monoms, with a minimum number of overlaps, resulting all the simplified forms of the initial function.

Example 7.22.

Let f be a Boolean function of 3 variables given by the following expression:
 $f(x_1, x_2, x_3) = \overline{x_1}x_2x_3 \vee \overline{x_1}x_2\overline{x_3} \vee \overline{x_1}x_2x_3 \vee x_1\overline{x_2}\overline{x_3}$.

Using the standard notations for the minterms: $f(x_1, x_2, x_3) = m_1 \vee m_2 \vee m_3 \vee m_4$.

The ascending order, with respect to the number of "1" values in each triplet, of the support of f is $S_f = \{(0,0,1), (0,1,0), (1,0,0), (0,1,1)\}$

		Group			
		x_1	x_2	x_3	
Simple factorizations	Representation	✓	0	0	1 m_1
		I	✓	0	1 m_2
			1	0	0 $m_4 = x_1\overline{x_2}\overline{x_3} = max_1$
III=I+II	Group	II	✓	0	1 m_3
			0	1	$m_1 \vee m_3 = \overline{x_1}x_3 = max_2$
			0	—	0 $m_2 \vee m_3 = \overline{x_1}x_2 = max_3$

- Only simple factorizations can be applied: $m_1 \vee m_3$ and $m_2 \vee m_3$. The minterm m_4 does not have neighbors, it is an isolated minterm which is a maximal monom. The unmarked rows correspond to the maximal monoms:

$$M(f) = \{x_1\overline{x_2}\overline{x_3}, \overline{x_1}x_3, \overline{x_1}x_2\} = \{max_1, max_2, max_3\}.$$

- We build the following correspondence tableau (minterms - maximal monoms) and we circled the '*' which are unique on their rows. We can read the tableau on columns or on rows as follows:
 - third row: the minterm m_3 is covered by max_2 and by max_3
 - second column: the maximal monom max_2 was obtained as a result of a simple factorization between the minterms m_1 and m_3

All the columns have a \oplus , thus all the maximal monoms are central monoms:

$$M(f) = C(f).$$

Boolean Algebras and Boolean Functions

minterms \ max.monom			
	max ₁	max ₂	max ₃
m_1		*	
m_2			*
m_3		*	*
m_4	*		

- 3. This is the first case of the simplification algorithm.
- 4. f has a unique disjunctive simplified form obtained as the disjunction of all the maximal monoms:

$$f^S(x_1, x_2, x_3) = \text{max}_1 \vee \text{max}_2 \vee \text{max}_3 = x_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_3 \vee \bar{x}_1 x_2$$

Example 7.23.

Let f be a Boolean function of 3 variables given by the following table.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

The disjunctive canonical form is:

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3 \\ &= m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_6 \vee m_7 \end{aligned}$$

$$S_f = \{(0,0,0), (0,0,1), (0,1,1), (1,0,0), (1,1,0), (1,1,1)\}$$

The ascending order, with respect to the number of "1" values in each triplet, of the support of f is $S_f = \{(0,0,0), (1,0,0), (0,0,1), (1,1,0), (0,1,1), (1,1,1)\}$

Representation	Group	x_1	x_2	x_3	
		✓	✓	✓	
I	✓	0	0	0	m_0
	✓	1	0	0	m_4
II	✓	0	0	1	m_1
	✓	1	1	0	m_6
III	✓	0	1	1	m_3
	✓	1	1	1	m_7
IV	✓				

A Computational Approach to Classical Logics and Circuits

Simple factorizations V=I+II VI=II+III VII=III+IV	<table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="text-align: center;">-</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td>$m_0 \vee m_4 = \overline{x_2} \overline{x_3} = max_1$</td></tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">-</td><td>$m_0 \vee m_1 = \overline{x_1} \overline{x_2} = max_2$</td></tr> <tr> <td colspan="3" style="border-top: 1px solid black;"></td><td>$m_4 \vee m_6 = \overline{x_1} x_3 = max_3$</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">-</td><td style="text-align: center;">0</td><td>$m_1 \vee m_3 = \overline{x_1} x_3 = max_4$</td></tr> <tr> <td colspan="3" style="border-top: 1px solid black;"></td><td>$m_6 \vee m_7 = x_1 x_2 = max_5$</td></tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">-</td><td style="text-align: center;">1</td><td>$m_3 \vee m_7 = x_2 x_3 = max_6$</td></tr> <tr> <td colspan="3" style="border-top: 1px solid black;"></td><td></td></tr> </tbody> </table>	-	0	0	$m_0 \vee m_4 = \overline{x_2} \overline{x_3} = max_1$	0	0	-	$m_0 \vee m_1 = \overline{x_1} \overline{x_2} = max_2$				$m_4 \vee m_6 = \overline{x_1} x_3 = max_3$	1	-	0	$m_1 \vee m_3 = \overline{x_1} x_3 = max_4$				$m_6 \vee m_7 = x_1 x_2 = max_5$	0	-	1	$m_3 \vee m_7 = x_2 x_3 = max_6$					
-	0	0	$m_0 \vee m_4 = \overline{x_2} \overline{x_3} = max_1$																											
0	0	-	$m_0 \vee m_1 = \overline{x_1} \overline{x_2} = max_2$																											
			$m_4 \vee m_6 = \overline{x_1} x_3 = max_3$																											
1	-	0	$m_1 \vee m_3 = \overline{x_1} x_3 = max_4$																											
			$m_6 \vee m_7 = x_1 x_2 = max_5$																											
0	-	1	$m_3 \vee m_7 = x_2 x_3 = max_6$																											

- Only simple factorizations can be applied. The neighbor groups (V, VI) and (VI, VII) do not have neighbor monoms to be used in a double factorization. The unmarked rows correspond to the maximal monoms:
 $M(f) = \{\overline{x_2} \overline{x_3}, \overline{x_1} \overline{x_2}, \overline{x_1} \overline{x_3}, \overline{x_1} x_3, x_1 \overline{x_2}, x_2 \overline{x_3}\} = \{max_1, max_2, max_3, max_4, max_5, max_6\}$

We build the following correspondence tableau:

max.monom	max_1	max_2	max_3	max_4	max_5	max_6
m_0	*	*				
m_4	*		*			
m_1		*		*		
m_6			*		*	
m_3				*		*
m_7					*	*

- There is no '*' which is unique on a certain row, meaning that all the minterms are covered by two maximal monoms, so there are no central monoms:
 $C(f) = \emptyset$.
- This is the third case of the simplification algorithm.
- There are two simplified forms of f , obtained by covering all the minterms with a minimum number of maximal monoms, without overlaps:

$$f_1^S(x_1, x_2, x_3) = max_1 \vee max_4 \vee max_5 = \overline{x_2} \overline{x_3} \vee \overline{x_1} x_3 \vee x_1 x_2 \quad (*)$$

$$f_2^S(x_1, x_2, x_3) = max_2 \vee max_3 \vee max_6 = \overline{x_1} x_2 \vee \overline{x_1} x_3 \vee x_2 x_3 \quad (**)$$

Example 7.24.

Simplify the following Boolean function of 4 variables:

$$f(x_1, x_2, x_3, x_4) = x_1 \overline{x_2} \overline{x_3} x_4 \vee \overline{x_1} x_2 \overline{x_3} \vee \overline{x_1} \overline{x_2} \overline{x_3} x_4 \vee x_1 \overline{x_2} \overline{x_3} \overline{x_4} \vee \overline{x_1} x_3 x_4 \vee \overline{x_1} x_3 \vee x_2 \overline{x_3} x_4$$

The support set of f is obtained as the reunion of the supports of the monoms from the function's expression, which is a disjunctive form but not a canonical one:

Boolean Algebras and Boolean Functions

$$\begin{aligned}
 S_f = & \{(1,0,1,0)\} \cup \{(0,1,1,0), (0,1,1,1)\} \cup \{(0,1,0,1)\} \cup \{(1,1,0,0)\} \cup \\
 & \cup \{(0,0,0,1), (0,1,0,1)\} \cup \{(0,0,1,0), (0,0,1,1), (0,1,1,0), (0,1,1,1)\} \cup \\
 & \cup \{(0,0,0,1), (1,0,0,1)\} = \\
 & = \{(0,1,1,1), (1,0,1,0), (0,1,1,0), (1,1,0,0), (0,1,0,1), (1,0,0,1), (0,0,1,1), \\
 & \quad (0,0,0,1), (0,0,1,0)\}
 \end{aligned}$$

From S_f the disjunctive canonical form of f is written as follows:

$$f(x_1, x_2, x_3, x_4) = m_7 \vee m_{10} \vee m_6 \vee m_{12} \vee m_5 \vee m_9 \vee m_3 \vee m_1 \vee m_2$$

1. The support of f is represented in descending order, with respect to the number of "1" values in each 4-uple and then factorizations are applied as can be seen in the following tableau.

		Group				
		x_1	x_2	x_3	x_4	
Representation	I	✓	0	1	1	1
		✓	1	0	1	0
		✓	0	1	1	0
	II	✓	1	1	0	0
		✓	0	1	0	1
		✓	1	0	0	1
Simple factorization	III	✓	0	0	1	1
		✓	0	0	0	1
		✓	0	0	1	0
	IV=I+II	✓	0	1	1	-
		✓	0	1	-	1
		✓	0	-	1	1
V=II+III		✓	-	0	1	0
		✓	0	-	1	0
		✓	0	-	0	1
		✓	-	0	0	1
		✓	0	0	-	1
		✓	0	0	1	-
Double factorization	VI=IV+V		0	-	1	-
			0	-	-	1

The monoms from the groups IV and V are obtained by applying simple factorizations and further the neighbor monoms from these groups are used in double factorizations obtaining group VI.

There is an isolated minterm (without neighbors): m_{12} , which is a maximal monom.

A Computational Approach to Classical Logics and Circuits

$$M(f) = \{max_1, max_2, max_3, max_4, max_5\} = \{x_1x_2\bar{x}_3\bar{x}_4, \bar{x}_2x_3\bar{x}_4, \bar{x}_2\bar{x}_3x_4, \bar{x}_1x_3, \bar{x}_1x_4\}$$

Note that a result of a double factorization is obtained twice as follows:

$$max_4 = (m_7 \vee m_6) \vee (m_3 \vee m_2) = (m_7 \vee m_3) \vee (m_6 \vee m_2) \text{ and}$$

$$max_5 = (m_7 \vee m_5) \vee (m_3 \vee m_1) = (m_7 \vee m_3) \vee (m_5 \vee m_1)$$

- We identify the central monoms from the correspondence tableau built below.

max.monom minterms	max ₁	max ₂	max ₃	max ₄	max ₅
m_7				*	*
m_{10}		*			
m_6				*	
m_{12}	*				
m_5					*
m_9			*		
m_3				*	*
m_1			*		*
m_2		*		*	

We circled the '*' which are unique on their rows. All the columns have a *, thus all the maximal monoms are central monoms: $M(f) = C(f)$.

- According to the first case of the simplification algorithm there is a unique disjunctive simplified form of the initial function, obtained as the disjunction of all the central monoms:
- $f^S(x_1, x_2, x_3, x_4) = max_1 \vee max_2 \vee max_3 \vee max_4 \vee max_5 =$
 $= x_1x_2\bar{x}_3\bar{x}_4 \vee \bar{x}_2x_3\bar{x}_4 \vee \bar{x}_2\bar{x}_3x_4 \vee \bar{x}_1x_3 \vee \bar{x}_1x_4$

Example 7.25.

Simplify the following Boolean function of 4 variables:

$$f(x_1, x_2, x_3, x_4) = m_4 \vee m_6 \vee m_7 \vee m_8 \vee m_9 \vee m_{10} \vee m_{11} \vee m_{12}.$$

The support of f is represented in a descending order, with respect to the number of "1" values in each 4-uple.

$$S_f = \{(0,1,1,1), (1,0,1,1), (0,1,1,0), (1,0,0,1), (1,0,1,0), (1,1,0,0), (0,1,0,0), (1,0,0,0)\}$$

- Factorization process:** there are 4 simple factorizations and one double factorization according to the following table.

The set of the maximal monoms is:

$$M(f) = \{max_1, max_2, max_3, max_4, max_5\} = \{\bar{x}_1x_2x_3, \bar{x}_1x_2\bar{x}_4, x_2\bar{x}_3\bar{x}_4, x_1\bar{x}_3\bar{x}_4, x_1\bar{x}_2\}$$

Boolean Algebras and Boolean Functions

		Group	x_1	x_2	x_3	x_4	
Simple factorization	I	\checkmark	0	1	1	1	m_7
		\checkmark	1	0	1	1	m_{11}
		\checkmark	0	1	1	0	m_6
Simple factorization	II	\checkmark	1	0	0	1	m_9
		\checkmark	1	0	1	0	m_{10}
		\checkmark	1	1	0	0	m_{12}
Simple factorization	III	\checkmark	0	1	0	0	m_4
		\checkmark	1	0	0	0	m_8
			0	1	1	-	$m_7 \vee m_6 = \bar{x}_1x_2x_3 = max_1$
Simple factorization	IV=I+II	\checkmark	1	0	-	1	$m_{11} \vee m_9$
		\checkmark	1	0	1	-	$m_{11} \vee m_{10}$
			0	1	-	0	$m_6 \vee m_4 = \bar{x}_1x_2\bar{x}_4 = max_2$
Simple factorization	V=II+III	\checkmark	1	0	0	-	$m_9 \vee m_8$
		\checkmark	1	0	-	0	$m_{10} \vee m_8$
			-	1	0	0	$m_{12} \vee m_4 = \bar{x}_2\bar{x}_3\bar{x}_4 = max_3$
Double factorization	VI=IV+V		1	-	0	0	$m_{12} \vee m_8 = \bar{x}_1\bar{x}_3\bar{x}_4 = max_4$
			1	0	-	-	$m_{11} \vee m_9 \vee m_{10} \vee m_8 = \bar{x}_1\bar{x}_2 = max_5$

2. Identification of the central monoms:

max.monom minterms	max_1	max_2	max_3	max_4	max_5
m_7	*				
m_{11}					*
m_6	*	*			
m_9					*
m_{10}					*
m_{12}			*	*	
m_4		*	*		
m_8				*	*

The shaded columns (max_1 and max_5) correspond to the central monoms because they have at least one '**', which is unique on its row.

max_1 has on the first row a '**', which is unique on its row, meaning that m_7 is covered only by max_1 , so max_1 must belong to all simplified forms of f .

A Computational Approach to Classical Logics and Circuits

The minterms m_{11} , m_9 , m_{10} are covered only by max_5 (see the second, fourth and fifth rows), therefore max_5 is a central monom.

$$C(f) = \{\text{max}_1, \text{max}_5\}$$

3. $M(f) \neq C(f)$ and $C(f) \neq \emptyset$, the second case of the simplification algorithm.

$$g(x_1, x_2, x_3, x_4) = \text{not. } \text{max}_1 \vee \text{max}_5 = \overline{x_1}x_2x_3 \vee x_1\overline{x_2}.$$

4. We also shade the rows (minterms) covered by the central monoms. To identify the simplified forms of f we have to choose functions $h(x_1, x_2, x_3, x_4)$, as simple as possible, such that they cover the minterms m_{12} and m_4 uncovered by $g(x_1, x_2, x_3, x_4)$.

We choose the maximal monom corresponding to the column with the maximum number of unshaded '*' : $\text{max}_3 = x_2\overline{x_3}\overline{x_4}$.

Note that $h(x_1, x_2, x_3, x_4) = \text{max}_3$, covers $S_f \setminus S_g$.

There is a unique simplified form of f :

$$f^*(x_1, x_2, x_3, x_4) = g(x_1, x_2, x_3, x_4) \vee h(x_1, x_2, x_3, x_4) = \overline{x_1}x_2x_3 \vee x_1\overline{x_2} \vee x_2\overline{x_3}\overline{x_4}$$

7.7. Moisil's simplification method

Grigore Constantin Moisil was a Romanian mathematician, computer pioneer. His research was mainly in the fields of mathematical logic, algebraic logic and differential equations. Moisil is considered to be the father of Romanian informatics with his invention of three-stable circuits.

Moisil's method uses propositional logic to obtain the simplified forms of a Boolean function from the set of maximal monoms. The set of the maximal monoms are calculated using one of the previous methods (Veitch-Karnaugh diagrams, Quine's method). In the next example we follow the approach proposed in paper [62].

Example 7.26.

We apply Moisil's method to the function from Example 7.22:

$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{x_1}\overline{x_2}\overline{x_3} \vee \overline{x_1}\overline{x_2}x_3 \vee \overline{x_1}x_2\overline{x_3} \vee x_1\overline{x_2}\overline{x_3} \vee x_1x_2\overline{x_3} \vee x_1x_2x_3 = \\ &= m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_6 \vee m_7 - \text{DCF (disjunctive canonical form)} \end{aligned}$$

$$S_f = \{(0,0,0), (0,0,1), (0,1,1), (1,0,0), (1,1,0), (1,1,1)\}$$

The maximal monoms were obtained using Quine's method by applying six double factorizations:

$$\text{max}_1 = m_0 \vee m_4 = \overline{x_2}\overline{x_3}$$

$$\text{max}_2 = m_0 \vee m_1 = \overline{x_1}\overline{x_2}$$

$$\text{max}_3 = m_4 \vee m_6 = x_1\overline{x_3}$$

$$\text{max}_4 = m_1 \vee m_3 = \overline{x_1}x_3$$

$$\text{max}_5 = m_6 \vee m_7 = x_1x_2$$

$$\text{max}_6 = m_3 \vee m_7 = x_2x_3$$

$$M(f) = \{x_2x_3, x_1x_2, x_1x_3, x_1x_3, x_1x_2, x_2x_3\} = \{\text{max}_1, \text{max}_2, \text{max}_3, \text{max}_4, \text{max}_5, \text{max}_6\}.$$

Boolean Algebras and Boolean Functions

Because the simplified forms of a function contain only maximal monoms we consider the following propositional sentences:

p_i : "max_i belongs to a simplified form of f ", $i=1,2,\dots,6$.

Each minterm from the function's expression must be covered by a maximal monom in a simplified form, therefore according to the result of the factorization process we have the following true sentences:

„ m_0 is covered by max₁ or by max₂”, translated as: $p_1 \vee p_2 \equiv T$

„ m_4 is covered by max₁ or by max₃”, translated as: $p_1 \vee p_3 \equiv T$

„ m_1 is covered by max₂ or by max₄”, translated as: $p_2 \vee p_4 \equiv T$

„ m_6 is covered by max₃ or by max₅”, translated as: $p_3 \vee p_5 \equiv T$

„ m_3 is covered by max₄ or by max₆”, translated as: $p_4 \vee p_6 \equiv T$

„ m_7 is covered by max₅ or by max₆”, translated as: $p_5 \vee p_6 \equiv T$

In order to obtain a simplified form of f , all the minterms from the function's expression must be covered by a minimum number of maximal monoms, with a minimum number of overlaps. This statement is modeled by the following propositional formula obtained as a conjunction of all the previous true sentences:

$$(p_1 \vee p_2) \wedge (p_1 \vee p_3) \wedge (p_2 \vee p_4) \wedge (p_3 \vee p_5) \wedge (p_4 \vee p_6) \wedge (p_5 \vee p_6) \equiv T$$

The formula is in conjunctive normal form (CNF – a conjunction of clauses) and we have to transform it into its equivalent disjunctive normal form (DNF).

We transform syntactically the previous formula:

$$T \equiv (p_1 \vee (p_2 \wedge p_3)) \wedge (p_4 \vee (p_2 \wedge p_6)) \wedge (p_5 \vee (p_3 \wedge p_6))$$

Apply distributive laws:

$$\begin{aligned} T &\equiv (p_1 \wedge p_4 \wedge p_5) \vee (p_1 \wedge p_4 \wedge p_3 \wedge p_6) \vee (p_1 \wedge p_2 \wedge p_6 \wedge p_5) \vee \\ &\quad \vee (p_1 \wedge p_2 \wedge p_6 \wedge p_3 \wedge p_6) \vee (p_2 \wedge p_3 \wedge p_4 \wedge p_5) \vee \\ &\quad \vee (p_2 \wedge p_3 \wedge p_4 \wedge p_3 \wedge p_6) \vee (p_2 \wedge p_3 \wedge p_2 \wedge p_6 \wedge p_5) \vee \\ &\quad \vee (p_2 \wedge p_3 \wedge p_2 \wedge p_6 \wedge p_3 \wedge p_6) \end{aligned}$$

Apply idempotency:

$$\begin{aligned} T &\equiv (p_1 \wedge p_4 \wedge p_5) \vee (p_1 \wedge p_4 \wedge p_3 \wedge p_6) \vee (p_1 \wedge p_2 \wedge p_6 \wedge p_5) \vee \\ &\quad \vee (p_1 \wedge p_2 \wedge p_3 \wedge p_6) \vee (p_2 \wedge p_3 \wedge p_4 \wedge p_5) \vee (p_2 \wedge p_3 \wedge p_4 \wedge p_3 \wedge p_6) \vee \\ &\quad \vee (p_2 \wedge p_3 \wedge p_6 \wedge p_5) \vee (p_2 \wedge p_3 \wedge p_6) \end{aligned}$$

Apply absorption:

$$\begin{aligned} T &\equiv (\mathbf{p}_1 \wedge \mathbf{p}_4 \wedge \mathbf{p}_5) \vee (\mathbf{p}_1 \wedge \mathbf{p}_4 \wedge \mathbf{p}_3 \wedge \mathbf{p}_6) \vee (\mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_6 \wedge \mathbf{p}_5) \vee \\ &\quad \vee (\mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{p}_4 \wedge \mathbf{p}_5) \vee (\mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{p}_6) - DNF \text{ with 5 cubes} \end{aligned}$$

A DNF (disjunction of cubes) is true if at least one of its cubes is true.

From DNF we consider the cubes with the minimum number of propositional variables (marked in bold) and correspondingly we obtain the simplified forms of f .

A Computational Approach to Classical Logics and Circuits

For the cube $p_1 \wedge p_4 \wedge p_5 \equiv T$ the corresponding simplified form is:

$$f_1^S(x_1, x_2, x_3) = max_1 \vee max_4 \vee max_5 = \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_3 \vee x_1 x_2.$$

For the cube $p_2 \wedge p_3 \wedge p_6 \equiv T$, the corresponding simplified form is:

$$f_2^S(x_1, x_2, x_3) = max_2 \vee max_3 \vee max_6 = \bar{x}_1 \bar{x}_2 \vee x_1 \bar{x}_3 \vee x_2 x_3.$$

If we take the other cubes from DNF, there will be overlaps: a minterm will be covered by two maximal monoms and these forms do not correspond to the simplest forms of f .

7.8. Application

A direct application of Boolean functions simplification is automatic simplification of conditional expressions. In the beginning, the simple conditional expressions must be identified. These simple conditional expressions are connected by logical operations. They are denoted by variables and thus a logical expression that can be seen as a Boolean function is obtained. The simplification process is easily implemented according to the steps described in the following.

- **Step1:** The conditional expression is transformed from current form into disjunctive normal form, using the appropriate algorithm from the first chapter.
- **Step2:** To obtain the disjunctive canonical form of the Boolean function, attach to each cube all missing variables, both simple and negated.
- **Step3:** The Quine's method is applied to perform factorizations and to obtain the maximal monoms.
- **Step4:** The simplified form is obtained using Moisil's method.

Example 7.27.

We consider the following Pascal conditional expression:

$$((x < y) \text{ and } (y < z) \text{ and } (z < 5)) \text{ or } ((x < y) \text{ and } (y \geq z) \text{ and } (z < 5))$$

Note that the expression $y \geq z$ can be written as: $\neg(y < z)$.

So, the initial expression becomes:

$$((x < y) \text{ and } (y < z) \text{ and } (z < 5)) \text{ or } ((x < y) \text{ and } (\neg(y < z)) \text{ and } (z < 5))$$

In order to track more easily the simplification process, we denote the simple conditional expressions with variables, as follows:

- $x < y$ by x_1 , $y < z$ by x_2 , $z < 5$ by x_3 .

The following Boolean function is obtained:

$$f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3)$$

The simplified form of f function is: $f^S(x_1, x_2, x_3) = x_1 \wedge x_3$

Thus, the initial expression was reduced to $(x < y) \text{ and } (z < 5)$.

An application which implements the above algorithm was presented at the ICMI2006 conference, see [39].

7.9. DCF versus CCF in simplification

All three simplification methods presented in the previous sections use the disjunctive canonical form (*DCF*) of the initial function. The simplification purpose is to obtain the simplest form of the function. This form can be a conjunctive or a disjunctive form. But by negation, the *CCF* of the function is transformed into *DCF* of the negated function, which can be simplified using the previous presented methods and then the negated simplified form can be changed back into a conjunctive simplified form of the initial function.

Example 7.28.

Let us consider the following function given in its conjunctive canonical form:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= M_0 \wedge M_4, \\ \bar{f}(x_1, x_2, x_3, x_4) &= m_0 \vee m_4 = \overline{x_1 x_2 x_3 x_4} \vee \overline{x_1 x_2 x_3 x_4} = \overline{x_1 x_3 x_4} (\overline{x_2} \vee x_2) = \overline{x_1 x_3 x_4} \end{aligned} \quad - \text{the simplified form}$$

By negating the simplified form obtained previously, the simplified form of the initial function f is obtained: $f^S(x_1, x_2, x_3, x_4) = x_1 \vee x_3 \vee x_4$

Sometimes, the simplified form of a Boolean function is a conjunctive one, not a disjunctive form, as can be seen from the following example. Thus, it is recommended to simplify both: the function and its negation, and in the end to choose the simplest form.

Example 7.29.

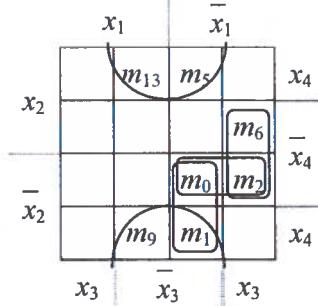
Determine the simplest equivalent form of the following Boolean function:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1 \overline{x_2} \overline{x_3} x_4 \vee \overline{x_1} \overline{x_2} \overline{x_3} x_4 \vee \overline{x_1} x_2 \overline{x_3} \overline{x_4} \vee \overline{x_1} x_2 x_3 \overline{x_4} \vee \overline{x_1} x_2 \overline{x_3} \overline{x_4} \vee \\ &\vee x_1 \overline{x_2} x_3 x_4 \vee x_1 x_2 \overline{x_3} x_4 = \\ &= m_{13} \vee m_5 \vee m_6 \vee m_0 \vee m_2 \vee m_9 \vee m_1 \quad (\text{DCF}) \\ &= M_{15} \wedge M_7 \wedge M_{14} \wedge M_{12} \wedge M_4 \wedge M_{10} \wedge M_8 \wedge M_{11} \wedge M_3 \quad (\text{CCF}) \end{aligned}$$

The negation function of f is:

$$\bar{f}(x_1, x_2, x_3, x_4) = m_{15} \vee m_7 \vee m_{14} \vee m_{12} \vee m_4 \vee m_{10} \vee m_8 \vee m_{11} \vee m_3$$

We use the Veitch simplification method in order to obtain the simplified form for both f and \bar{f} . First we simplify $f(x_1, x_2, x_3, x_4)$:



A Computational Approach to Classical Logics and Circuits

1. The set of the maximal monoms is:

$$M(f) = \{\bar{x}_3x_4, \bar{x}_1\bar{x}_2x_3, \bar{x}_1x_2\bar{x}_4, \bar{x}_1x_3\bar{x}_4\}$$

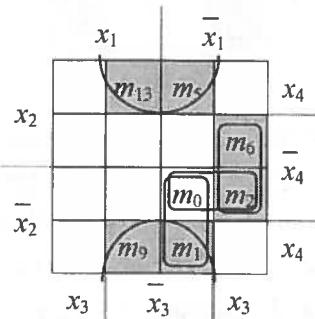
$$\max_1 = \underline{x}_3x_4 = m_{13} \vee m_5 \vee m_9 \vee m_1,$$

$$\max_3 = \underline{x}_1x_2\bar{x}_4 = m_0 \vee m_2$$

$$\max_2 = \bar{x}_1\bar{x}_2x_3 = m_1 \vee m_0$$

$$\max_4 = \bar{x}_1x_3\bar{x}_4 = m_6 \vee m_2$$

2. $C(f) = \{ \max_1 = \bar{x}_3x_4, \max_4 = \bar{x}_1x_3\bar{x}_4 \}$



3. This is the second case of the simplification algorithm.

4. There is only one uncovered minterm, m_0 , which can be covered in two ways, with the same complexity, using \max_2 or \max_3 .

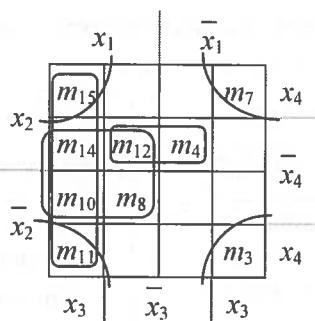
$$f_1^S(x_1, x_2, x_3, x_4) = \bar{x}_3x_4 \vee \bar{x}_1\bar{x}_3x_4 \vee \bar{x}_1x_2x_3$$

$$f_2^S(x_1, x_2, x_3, x_4) = \bar{x}_3x_4 \vee \bar{x}_1x_3\bar{x}_4 \vee x_1x_2x_4$$

Both *disjunctive simplified forms* of the function contain 8 occurrences of the variables, 7 binary operations and 6 unary operations.

In the following we simplify the function:

$$\bar{f}(x_1, x_2, x_3, x_4) = m_{15} \vee m_7 \vee m_{14} \vee m_{12} \vee m_4 \vee m_{10} \vee m_8 \vee m_{11} \vee m_3$$



1. The set of maximal monoms is:

$$M(\bar{f}) = \{x_1x_3, x_1\bar{x}_4, x_2\bar{x}_3\bar{x}_4, x_3x_4\}$$

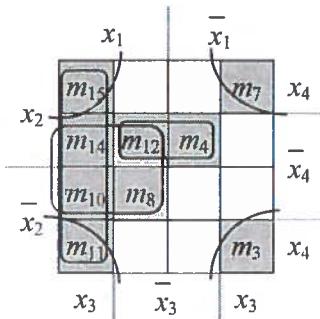
$$\max'_1 = x_1x_3 = m_{15} \vee m_{14} \vee m_{10} \vee m_{11}$$

$$\max'_3 = x_2\bar{x}_3\bar{x}_4 = m_{12} \vee m_4$$

$$\max'_2 = \bar{x}_1\bar{x}_4 = m_{14} \vee m_{12} \vee m_{10} \vee m_8$$

$$\max'_4 = x_3x_4 = m_{15} \vee m_{11} \vee m_7 \vee m_3$$

Boolean Algebras and Boolean Functions



2. $C(\bar{f}) = \{ \max'_2 = x_1\bar{x}_4, \max'_3 = x_2\bar{x}_3\bar{x}_4, \max'_4 = x_3x_4 \}$
3. This is the **second case** of the simplification algorithm, but as can be seen in the above diagram, the central monoms cover all the minterms of \bar{f} .
4. Thus, there is a unique simplified form of the function \bar{f} , obtained as a disjunction of all the central monoms: $\bar{f}^S(x_1, x_2, x_3, x_4) = x_1\bar{x}_4 \vee x_2\bar{x}_3\bar{x}_4 \vee x_3x_4$

Negating again we obtain the *conjunctive simplified form* of f :

$$f^{CS}(x_1, x_2, x_3, x_4) = (\bar{x}_1 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_4).$$

Note that this new simplified form of the initial function contains only 7 variable occurrences, 6 binary operations and 4 unary operations, so is simpler than the disjunctive simplified forms obtained earlier.

We conclude that the simplest form of f is the conjunctive form, obtained by negating the simplified form of \bar{f} :

$$f^{CS}(x_1, x_2, x_3, x_4) = (\bar{x}_1 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_4).$$

Conclusion:

In order to obtain the simplest equivalent form (conjunctive or disjunctive) of a Boolean function f we have two possibilities:

see Example 7.10.	see Example 7.19.
- simplify $DCF(f) \Rightarrow$ disjunctive simplified form: f^S	
- simplify $CCF(f)$ using a dual simplification algorithm \Rightarrow conjunctive simplified form: f^{CS}	<ul style="list-style-type: none"> - $CCF(f)$ is negated obtaining a disjunctive canonical form - simplify $\overline{CCF(f)} \Rightarrow$ disjunctive simplified form: \bar{f}^S - \bar{f}^S is negated and we obtain the conjunctive simplified form f^{CS}
- from f^S and f^{CS} choose the form with the fewest occurrences of variables and connectives	

A Computational Approach to Classical Logics and Circuits

7.10. Exercises

Exercise 7.1.

For the following Boolean functions of 3 variables, given by their tables of values, write the corresponding *disjunctive canonical form (DCF)* and *conjunctive canonical form (CCF)*. Using Veitch diagrams simplify the functions.

x	y	z	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
0	0	0	0	1	1	1	0	1	0	1
0	0	1	1	1	1	0	1	0	0	0
0	1	0	0	0	1	0	1	1	1	1
0	1	1	1	0	0	1	0	1	0	0
1	0	0	1	0	1	0	0	0	1	0
1	0	1	0	1	0	0	1	0	1	1
1	1	0	0	0	0	1	1	1	0	1
1	1	1	1	1	0	1	0	0	1	0

Exercise 7.2.

Simplify the following Boolean functions of 4 variables using Veitch diagrams.

1. $f_1(x_1, x_2, x_3, x_4) = \overline{x_1}x_2\overline{x_3}x_4 \vee x_1\overline{x_2}\overline{x_3}x_4 \vee x_1x_2\overline{x_3}\overline{x_4} \vee \overline{x_1}\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}\overline{x_2}x_3x_4 \vee x_1\overline{x_2}x_3\overline{x_4}$
 $\vee x_1x_2\overline{x_3}x_4 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_3x_4;$
2. $f_2(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4 \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}\overline{x_2}x_3\overline{x_4} \vee x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2x_3x_4$
 $\vee \overline{x_1}x_2\overline{x_3}x_4 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_3x_4;$
3. $f_3(x_1, x_2, x_3, x_4) = \overline{x_1}x_2x_3x_4 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}\overline{x_2}x_3\overline{x_4} \vee x_1\overline{x_2}x_3\overline{x_4} \vee x_1x_2\overline{x_3}x_4 \vee x_1x_2\overline{x_3}\overline{x_4}$
 $\vee x_1\overline{x_2}x_3x_4 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_3x_4;$
4. $f_4(x_1, x_2, x_3, x_4) = \overline{x_1}x_2\overline{x_3}x_4 \vee x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4} \vee x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4}$
 $\vee x_1x_2\overline{x_3}x_4 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_3x_4;$
5. $f_5(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4 \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}\overline{x_4}$
 $\vee \overline{x_1}x_2x_3\overline{x_4} \vee x_1\overline{x_2}x_3\overline{x_4} \vee x_1x_2\overline{x_3}x_4;$
6. $f_6(x_1, x_2, x_3, x_4) = \overline{x_1}x_2x_3x_4 \vee \overline{x_1}\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_3\overline{x_4} \vee x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}x_4$
 $\vee x_1\overline{x_2}x_3x_4 \vee x_1\overline{x_2}x_3x_4 \vee x_1x_2\overline{x_3}x_4;$
7. $f_7(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4 \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3x_4 \vee x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}x_4$
 $\vee \overline{x_1}x_2x_3\overline{x_4} \vee x_1\overline{x_2}x_3x_4 \vee x_1x_2\overline{x_3}x_4;$
8. $f_8(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4 \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3\overline{x_4} \vee x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}x_4$
 $\vee x_1\overline{x_2}x_3\overline{x_4} \vee x_1\overline{x_2}x_3x_4 \vee x_1x_2\overline{x_3}x_4.$

Boolean Algebras and Boolean Functions

Exercise 7.3.

Using Karnaugh diagrams, simplify the following Boolean functions:

1. $f_1(x_1, x_2, x_3) = \overline{x_1}(x_2 \downarrow x_3) \vee \overline{x_1}\overline{x_2}x_3 \vee \overline{(x_1 \vee (x_2 \uparrow x_3))} \vee x_1x_2\overline{x_3}$
2. $f_2(x_1, x_2, x_3) = \overline{x_1}\overline{x_2}\overline{x_3} \vee x_1(\overline{x_2} \downarrow x_3) \vee (x_1 \vee (x_2 \uparrow \overline{x_3})) \vee x_1\overline{x_2}x_3$
3. $f_3(x_1, x_2, x_3) = \overline{x_1}(x_2 \downarrow x_3) \vee x_1\overline{x_2}x_3 \vee \overline{(x_1 \vee (x_2 \uparrow x_3))} \vee x_1x_2x_3$
4. $f_4(x_1, x_2, x_3) = \overline{x_1}\overline{x_2}\overline{x_3} \vee x_1(\overline{x_2} \downarrow x_3) \vee (x_1 \vee (\overline{x_2} \uparrow x_3)) \vee x_1\overline{x_2}x_3$
5. $f_5(x_1, x_2, x_3) = x_1(x_2 \downarrow x_3) \vee \overline{x_1}\overline{x_2}x_3 \vee \overline{(x_1 \vee (\overline{x_2} \uparrow x_3))} \vee \overline{x_1}x_2\overline{x_3}$
6. $f_6(x_1, x_2, x_3) = x_1\overline{x_2}x_3 \vee \overline{x_1}(\overline{x_2} \downarrow x_3) \vee \overline{(x_1 \vee (x_2 \uparrow x_3))} \vee x_1\overline{x_2}\overline{x_3}$
7. $f_7(x_1, x_2, x_3) = \overline{x_1}(x_2 \downarrow x_3) \vee x_1\overline{x_2}x_3 \vee \overline{(x_1 \vee (x_2 \uparrow x_3))} \vee \overline{x_1}x_2\overline{x_3}$
8. $f_8(x_1, x_2, x_3) = \overline{x_1}\overline{x_2}x_3 \vee x_1(x_2 \downarrow x_3) \vee \overline{(x_1 \vee (x_2 \uparrow x_3))} \vee \overline{x_1}x_2\overline{x_3}$

Exercise 7.4.

Simplify the following Boolean functions of 4 variables using Karnaugh diagrams.

1. $f_1(x_1, x_2, x_3, x_4) = x_1x_4 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_4 \vee \overline{x_1}x_3 \vee x_3\overline{x_4};$
2. $f_2(x_1, x_2, x_3, x_4) = x_1x_2 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_4 \vee \overline{x_1}x_3 \vee x_2x_3;$
3. $f_3(x_1, x_2, x_3, x_4) = x_1x_4 \vee x_1x_2x_3x_4 \vee \overline{x_1}x_2x_4 \vee \overline{x_1}x_3 \vee x_3x_4;$
4. $f_4(x_1, x_2, x_3, x_4) = \overline{x_1}x_2 \vee x_1\overline{x_2}x_3x_4 \vee \overline{x_1}x_2x_4 \vee \overline{x_1}x_3 \vee \overline{x_2}x_3;$
5. $f_5(x_1, x_2, x_3, x_4) = x_3x_4 \vee x_1x_2x_3x_4 \vee x_3\overline{x_2}x_4 \vee \overline{x_1}x_3 \vee \overline{x_1}x_4;$
6. $f_6(x_1, x_2, x_3, x_4) = \overline{x_1}x_4 \vee \overline{x_1}x_2x_3x_4 \vee x_1x_2x_4 \vee \overline{x_1}x_3 \vee x_3\overline{x_4};$
7. $f_7(x_1, x_2, x_3, x_4) = x_3x_4 \vee \overline{x_1}x_2\overline{x_3}x_4 \vee x_2x_3x_4 \vee \overline{x_1}x_3 \vee \overline{x_1}x_4;$
8. $f_8(x_1, x_2, x_3, x_4) = x_3x_4 \vee x_1\overline{x_2}x_3x_4 \vee x_2x_3\overline{x_4} \vee \overline{x_1}x_3 \vee x_1x_4.$

Exercise 7.5.

Using Quine's method, simplify the following Boolean functions given by their values 0.

1. $f_1(0,1,0) = f_1(0,1,1) = f_1(1,0,1) = 0; \quad 2. \quad f_2(0,0,0) = f_2(0,0,1) = f_2(1,1,1) = 0;$
3. $f_3(0,0,1) = f_3(0,1,0) = f_3(1,1,0) = 0; \quad 4. \quad f_4(0,0,0) = f_4(0,1,1) = f_4(1,0,0) = 0;$
5. $f_5(0,0,0) = f_5(1,1,0) = f_5(1,1,1) = 0; \quad 6. \quad f_6(0,1,0) = f_6(1,0,0) = f_6(1,0,1) = 0;$
7. $f_7(0,1,1) = f_7(1,0,0) = f_7(1,1,1) = 0; \quad 8. \quad f_8(0,0,1) = f_8(1,0,1) = f_8(1,1,0) = 0.$

Exercise 7.6.

Using Quine's method, simplify the following Boolean functions given in DCF (disjunction of minterms):

1. $f_1(x_1, x_2, x_3) = m_0 \vee m_3 \vee m_4 \vee m_5 \vee m_6 \vee m_7;$
2. $f_2(x_1, x_2, x_3) = m_1 \vee m_2 \vee m_4 \vee m_5 \vee m_6 \vee m_7;$
3. $f_3(x_1, x_2, x_3) = m_1 \vee m_2 \vee m_3 \vee m_4 \vee m_5 \vee m_7;$
4. $f_4(x_1, x_2, x_3) = m_0 \vee m_1 \vee m_2 \vee m_3 \vee m_5 \vee m_6;$

A Computational Approach to Classical Logics and Circuits

5. $f_5(x_1, x_2, x_3) = m_0 \vee m_1 \vee m_2 \vee m_4 \vee m_6 \vee m_7;$
6. $f_6(x_1, x_2, x_3) = m_0 \vee m_1 \vee m_3 \vee m_5 \vee m_6 \vee m_7;$
7. $f_7(x_1, x_2, x_3) = m_0 \vee m_1 \vee m_2 \vee m_3 \vee m_4 \vee m_7;$
8. $f_8(x_1, x_2, x_3) = m_0 \vee m_2 \vee m_3 \vee m_4 \vee m_5 \vee m_6.$

Exercise 7.7.

Simplify the following Boolean functions of 4 variables given by their values 1, using Quine's method.

1. $f_1(1,1,1,1) = f_1(1,1,0,1) = f_1(0,1,1,1) = f_1(1,1,0,0) = f_1(0,1,0,0) = f_1(0,0,0,0) = f_1(0,0,0,1) = f_1(0,0,1,1) = 1;$
2. $f_2(1,1,0,1) = f_2(0,1,0,1) = f_2(0,1,0,0) = f_2(0,0,0,0) = f_2(0,0,1,0) = f_2(1,0,1,1) = f_2(1,0,0,1) = f_2(0,0,1,1) = 1;$
3. $f_3(0,1,0,1) = f_3(0,1,0,0) = f_3(0,1,1,0) = f_3(1,0,1,0) = f_3(1,0,0,0) = f_3(0,0,1,0) = f_3(1,0,0,1) = f_3(0,0,0,1) = 1;$
4. $f_4(0,1,0,1) = f_4(0,1,1,1) = f_4(1,1,1,0) = f_4(1,1,0,0) = f_4(0,1,1,0) = f_4(1,0,0,0) = f_4(0,0,0,0) = f_4(0,0,0,1) = 1;$
5. $f_5(1,1,1,1) = f_5(0,1,0,1) = f_5(0,1,1,1) = f_5(1,1,1,0) = f_5(1,1,0,0) = f_5(1,0,0,0) = f_5(1,0,0,1) = f_5(0,0,0,1) = 1;$
6. $f_6(1,1,0,1) = f_6(0,1,0,1) = f_6(0,1,1,1) = f_6(1,1,1,0) = f_6(0,1,1,0) = f_6(1,0,1,0) = f_6(1,0,1,1) = f_6(1,0,0,1) = 1;$
7. $f_7(1,1,1,1) = f_7(1,1,0,1) = f_7(0,1,0,1) = f_7(0,1,0,0) = f_7(0,1,1,0) = f_7(0,0,1,0) = f_7(1,0,1,1) = f_7(0,0,1,1) = 1;$
8. $f_8(1,1,1,1) = f_8(1,1,1,0) = f_8(1,1,0,0) = f_8(1,0,0,0) = f_8(0,0,0,0) = f_8(0,0,1,0) = f_8(1,0,1,1) = f_8(0,0,1,1) = 1.$

Exercise 7.8.

Using Moisil's method simplify the following Boolean functions of 3 variables:

1. $f_1(x_1, x_2, x_3) = m_0 \vee m_1 \vee m_2 \vee m_5 \vee m_6;$
2. $f_2(x_1, x_2, x_3) = m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_7;$
3. $f_3(x_1, x_2, x_3) = m_1 \vee m_2 \vee m_3 \vee m_5 \vee m_6;$
4. $f_4(x_1, x_2, x_3) = m_0 \vee m_3 \vee m_4 \vee m_5 \vee m_7;$
5. $f_5(x_1, x_2, x_3) = m_1 \vee m_2 \vee m_5 \vee m_6 \vee m_7;$
6. $f_6(x_1, x_2, x_3) = m_0 \vee m_2 \vee m_3 \vee m_4 \vee m_7;$
7. $f_7(x_1, x_2, x_3) = m_1 \vee m_2 \vee m_4 \vee m_5 \vee m_6;$
8. $f_8(x_1, x_2, x_3) = m_0 \vee m_3 \vee m_4 \vee m_6 \vee m_7.$

8. LOGIC CIRCUITS

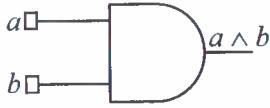
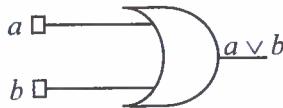
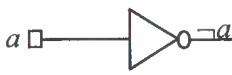
Computers and other electronic devices are composed of simple electronic circuits which are based on the two states of electricity. Digital logic circuits handle data encoded in binary form, i.e. signal that have only two values, 0 and 1. The Boolean functions and the logic circuits describe algebraically and graphically the functionality of electronic circuits. The papers [4, 14, 25, 28, 37, 41, 62] were used as bibliographic references in this chapter.

8.1. Basic concepts

Boolean algebra provides a concise way to express the operation of a logic circuit formed by a combination of logic gates so that the output can be determined for various combinations of input values. A Boolean function models the functionality of a logic circuit. The variables used in the function's expression are represented by the inputs of the circuit. Each wire in the circuit represents some part of the expression. A gate takes the values from its input wires and combines them with the appropriate Boolean operation, AND, OR, NOT, to produce the label on its output wire. The final output of the whole circuit represents the function's expression.

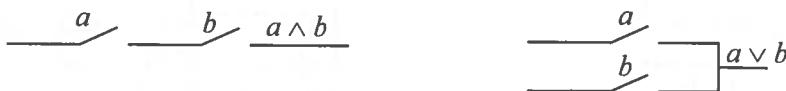
The basic elements used to build electronic circuits are called **basic gates** and they implement one of the Boolean operations: \wedge , \vee , \neg .

According to the IEEE standards, the basic gates are represented as follows:

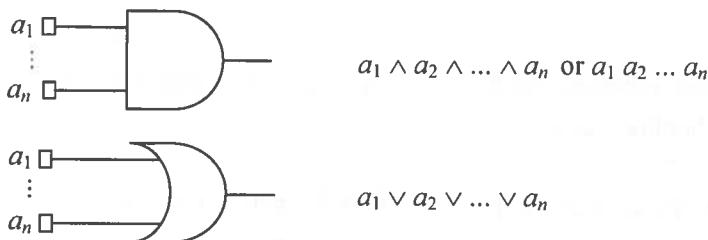
gate	symbol for the gate	Boolean operation, symbols used
AND		conjunction $a \wedge b$, $a \cap b$, $a * b$, ab
OR		disjunction $a \vee b$, $a \cup b$, $a + b$
NOT		negation $\neg a$, \bar{a}

A Computational Approach to Classical Logics and Circuits

At the physical level electronic circuits are composed of a power supply connection and contacts. A contact is described by a Boolean variable having one of the values: 0 for an **open contact** or 1 for a **closed contact**. The **AND**, **OR** gates correspond to a **series connection** and to a **parallel connection** respectively.



The **AND** and **OR** gates can be generalized to have more input variables:



Derived gates XOR, NAND, NOR:

gate	symbol	Boolean function	alternative symbol
XOR		$a \oplus b = \bar{a}b \vee a\bar{b}$	
NAND		$a \uparrow b = \bar{a}\bar{b} = a \vee \bar{b}$	
NOR		$a \downarrow b = \overline{a \vee b} = \bar{a}\bar{b}$	

The **AND**, **OR**, **NAND**, **NOR** gates may have any number of inputs, but practical commercial gates are mostly limited to 2, 3 and 4 inputs to fit the standard IC (integrated circuits) packages. A logic circuit can be implemented directly from the function's expression.

Example 8.1.

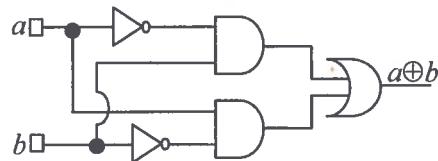
We want to build the logic circuit corresponding to "**exclusive OR**" (XOR) operation using only basic gates. The Boolean function which models its functionality is $f : \{0, 1\}^2 \rightarrow \{0, 1\}$, $f(a, b) = a \oplus b$.

Logic Circuits

Its truth table is provided below:

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

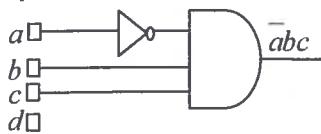
The disjunctive canonical form of $f(a,b) = \bar{a}b \vee a\bar{b}$. Using only basic gates we draw the circuit diagram.



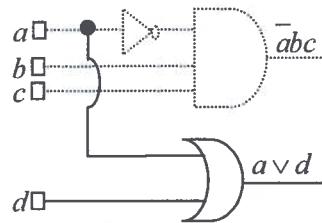
Example 8.2.

Given the Boolean function $f(a,b,c,d) = \bar{abc} \vee (\bar{a} \vee d)$ of four variables, we draw the logic circuit having f as output.

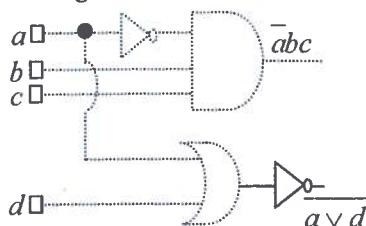
1. a NOT gate for \bar{a} and the AND gate for \bar{abc} are placed:



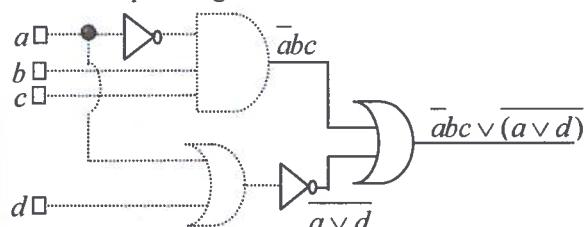
2. we draw the gate for $\bar{a} \vee d$



3. the branch $\bar{a} \vee d$ is negated:



4. finally, an OR gate has as inputs the wires corresponding to \bar{abc} and $\bar{a} \vee d$:

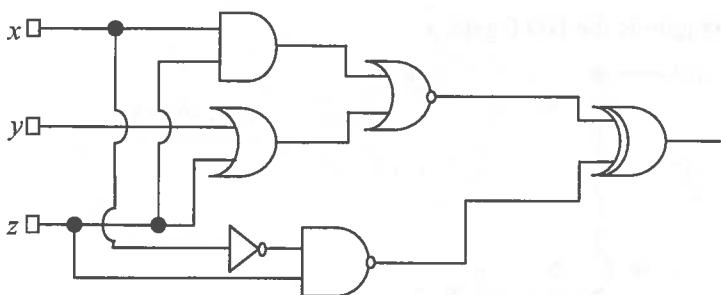


The *logic circuit analysis* is the task of determining the functionality of a given logic circuit from its diagram. To derive the Boolean expression for a given logic circuit, begin in the diagram at the left-most inputs and work toward the final output, writing the expression for each gate.

Example 8.3.

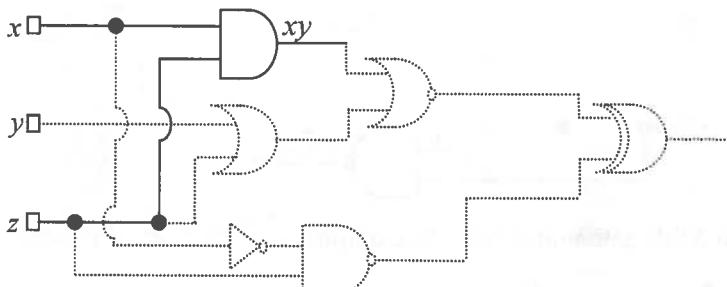
Let us consider the logic circuit depicted below, containing all types of gates, basic and derived ones.

A Computational Approach to Classical Logics and Circuits

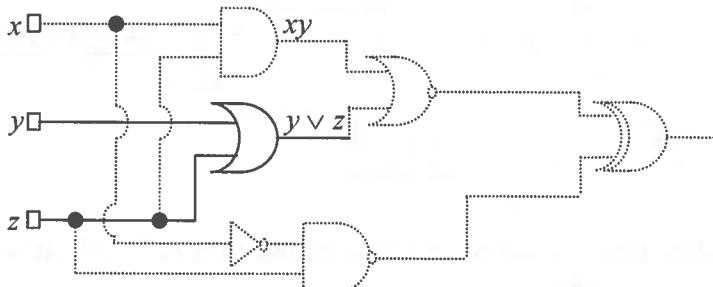


The Boolean expression which models its functionality is deduced step by step in the following:

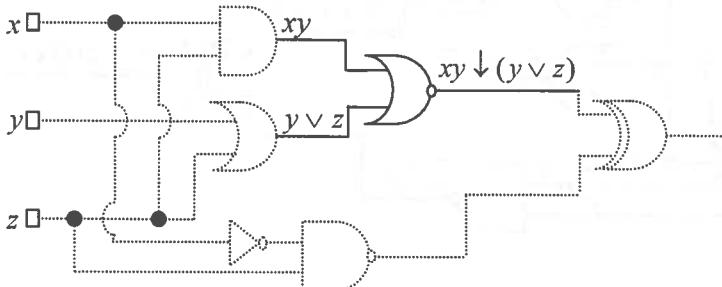
- the output of each gate is identified one at a time, starting from left to right, up to bottom.
 - the first gate to be identified is the AND gate for xy :



- the next gate is the OR gate for the expression $y \vee z$:

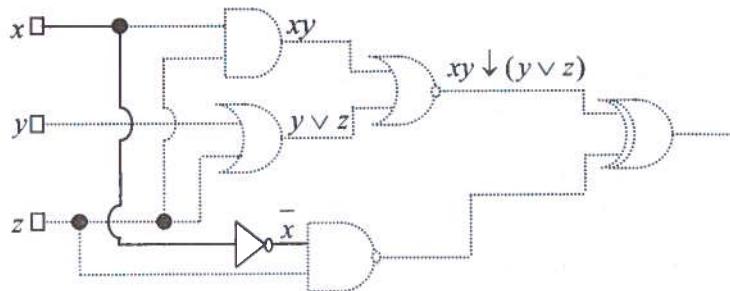


- the outputs of the previous two gates are the inputs of a NOR gate:

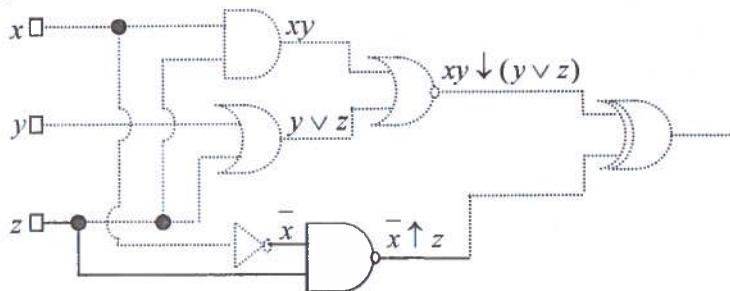


Logic Circuits

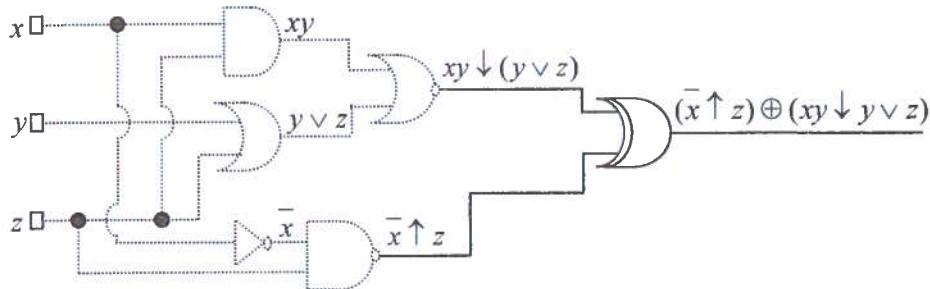
- the next gate is the NOT gate, \bar{x} :



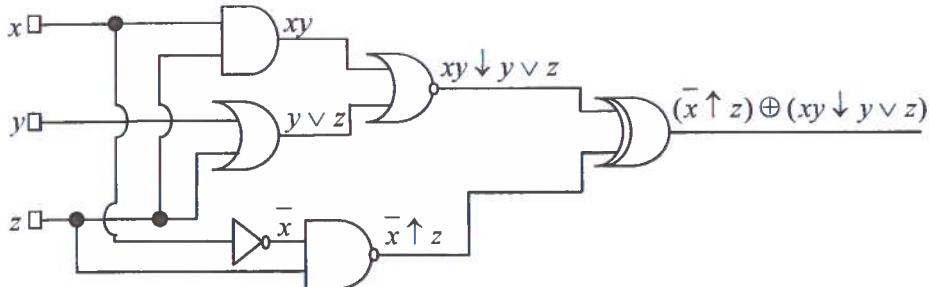
- the output of \bar{x} enters into an NAND gate, $\bar{x} \uparrow z$:



- the last XOR gate unites those two outputs $(\bar{x} \uparrow z) \oplus (xy \downarrow y \vee z)$:



So, the Boolean function corresponding to the circuit is $(\bar{x} \uparrow z) \oplus (xy \downarrow y \vee z)$:



A Computational Approach to Classical Logics and Circuits

Example 8.4.

The NAND gate is a universal gate because it can be used to produce the NOT, AND, OR, and NOR functions. We express the Boolean operations: negation, conjunction, disjunction using only the NAND operation, symbolized by ' \uparrow ' and having the definition: $a \uparrow b = \overline{a \wedge b}$.

$$\bar{a} = \overline{a \wedge a} = a \uparrow a$$

$$a \wedge b = \overline{\overline{a \wedge b}} = \overline{a \uparrow b} = (a \uparrow b) \uparrow (a \uparrow b)$$

$$a \vee b = \overline{\overline{a \wedge \bar{b}}} = \overline{\overline{a \uparrow \bar{b}}} = (a \uparrow a) \uparrow (b \uparrow b)$$

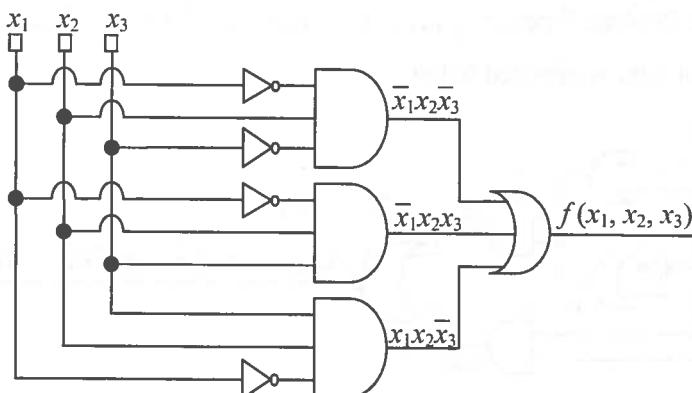
$$a \downarrow b = \overline{a \vee b} = \overline{\overline{(a \uparrow b)}} = ((a \uparrow a) \uparrow (b \uparrow b)) \uparrow ((a \uparrow a) \uparrow (b \uparrow b))$$

The corresponding logic circuits are as follows:

$\bar{a} = \overline{a \wedge a} = a \uparrow a$	
$a \wedge b = (a \uparrow b) \uparrow (a \uparrow b)$	
$a \vee b = \overline{\overline{a \wedge \bar{b}}} = (a \uparrow a) \uparrow (b \uparrow b)$	
$a \downarrow b = \overline{a \vee b} = \overline{\overline{(a \uparrow b)}} = ((a \uparrow a) \uparrow (b \uparrow b)) \uparrow ((a \uparrow a) \uparrow (b \uparrow b))$	

Example 8.5.

Given the initial circuit drawn below, we want to obtain a simplified circuit (with less gates) equivalent to the initial one.



Logic Circuits

In order to obtain the simplified circuit we have to write the Boolean function corresponding to the initial circuit, then simplify the function and finally draw the circuit modeled by the simplified function.

The Boolean function which models the functionality of the logic circuit is

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 x_2 \bar{x}_3 = m_2 \vee m_3 \vee m_6$$

Using Veitch diagram we simplify f :

	x_1	\bar{x}_1	
x_2	m_6	m_2	m_3
\bar{x}_2			
	x_3	\bar{x}_3	x_3

$$\max_1 = m_6 \vee m_2 = x_2 \bar{x}_3$$

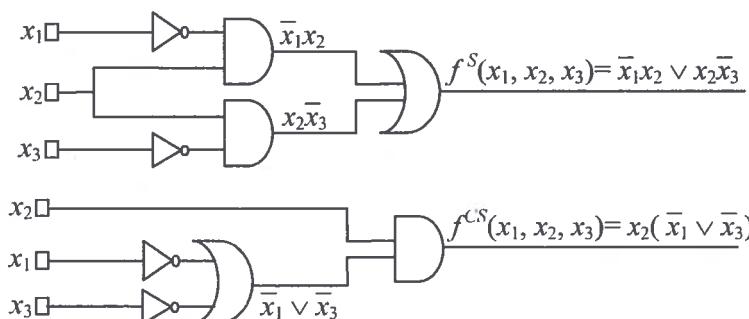
$$\max_2 = m_2 \vee m_3 = \bar{x}_1 x_2$$

$$M(f) = \{ \max_1, \max_2 \} = C(f)$$

It is the first case of the simplification algorithm, thus there is a unique *disjunctive simplified form*: $f^S(x_1, x_2, x_3) = \bar{x}_1 x_2 \vee x_2 \bar{x}_3$.

By applying distributive and commutative laws, a *conjunctive simplified form* is obtained: $f^{CS}(x_1, x_2, x_3) = x_2(\bar{x}_1 \vee \bar{x}_3)$.

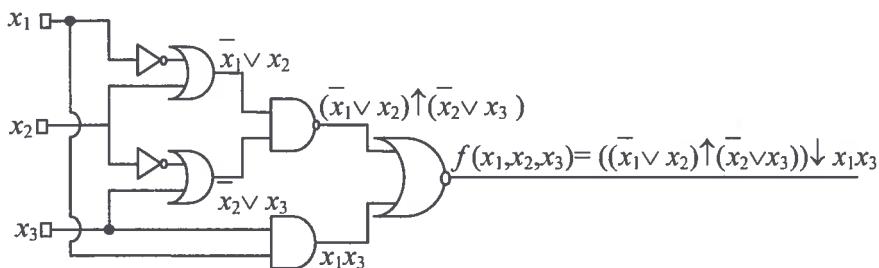
The simplified circuits equivalent to the initial one are as follows:



Example 8.6.

Draw the corresponding circuit (with basic and derived gates) and a simplified circuit for the Boolean function: $f(x_1, x_2, x_3) = ((\bar{x}_1 \vee x_2) \uparrow (\bar{x}_2 \vee x_3)) \downarrow x_1 x_3$.

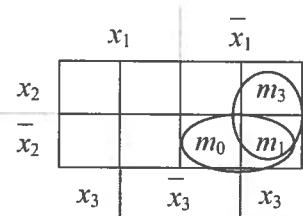
The circuit diagram is depicted below.



A Computational Approach to Classical Logics and Circuits

We transform the function into its disjunctive canonical form. We replace the NAND and NOR operations with equivalent expressions using only: \wedge , \vee , \neg operations and we apply DeMorgan, idempotency, commutativity, simplification and distributive laws.

$$\begin{aligned}
 f(x_1, x_2, x_3) &= ((\bar{x}_1 \vee x_2) \uparrow (\bar{x}_2 \vee x_3)) \downarrow x_1 x_3 = \\
 &= \overline{(\bar{x}_1 \vee x_2)(\bar{x}_2 \vee x_3)} \vee x_1 x_3 = \\
 &= (\bar{x}_1 \vee x_2)(\bar{x}_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_3) = (\bar{x}_1 \bar{x}_2 \vee \bar{x}_1 x_3 \vee \bar{x}_2 \bar{x}_3 \vee x_2 x_3)(\bar{x}_1 \vee \bar{x}_3) = \\
 &= (\bar{x}_1 \bar{x}_2 \vee \bar{x}_1 x_3 \vee 0 \vee x_2 x_3)(\bar{x}_1 \vee \bar{x}_3) = (\bar{x}_1 \bar{x}_2 \vee \bar{x}_1 x_3 \vee x_2 x_3)(\bar{x}_1 \vee \bar{x}_3) = \\
 &= \bar{x}_1 \bar{x}_2 \bar{x}_1 \vee \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_3 \bar{x}_1 \vee \bar{x}_1 x_3 x_3 \vee x_2 \bar{x}_3 \bar{x}_1 \vee x_2 x_3 x_3 = \\
 &= \bar{x}_1 \bar{x}_2 \vee \bar{x}_1 x_2 x_3 \vee \bar{x}_1 x_3 \vee 0 \vee \bar{x}_1 x_2 x_3 \vee 0 = \\
 &= \bar{x}_1 x_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3 = m_1 \vee m_0 \vee m_3
 \end{aligned}$$



By applying factorizations on Veitch diagram we obtain the maximal and central monoms:

$$\begin{aligned}
 max_1 &= m_0 \vee m_1 = \bar{x}_1 \bar{x}_2; \\
 max_2 &= m_1 \vee m_3 = \bar{x}_1 x_3
 \end{aligned}$$

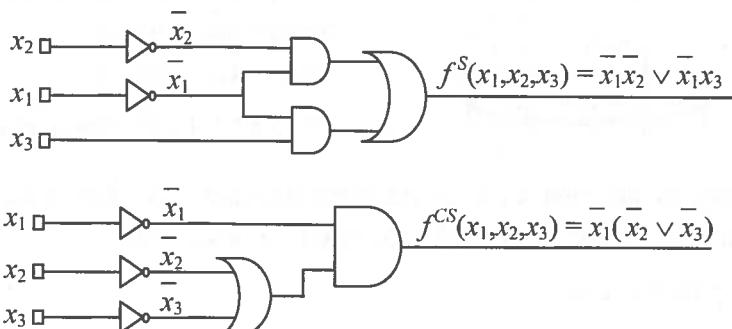
$$M(f) = C(f) = \{max_1, max_2\}$$

$$\text{There is a unique disjunctive simplified form: } f^S(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 \vee \bar{x}_1 x_3$$

A simpler conjunctive form, with less operations and less occurrences of variables, is obtained by applying distributivity:

$$f^{CS}(x_1, x_2, x_3) = \bar{x}_1(\bar{x}_2 \vee \bar{x}_3)$$

The simplified circuits for f are depicted below:



Example 8.7.

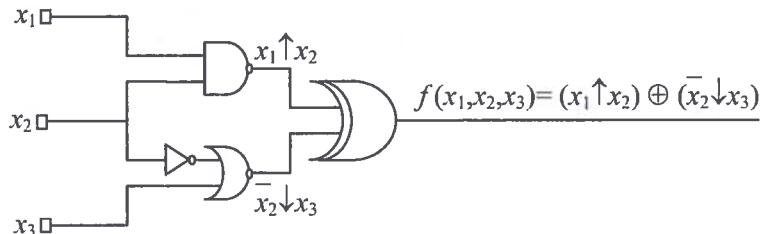
The following Boolean function contains the operations: \oplus , \uparrow , \downarrow :

$$f(x_1, x_2, x_3) = (x_1 \uparrow x_2) \oplus (\bar{x}_2 \downarrow x_3).$$

Draw a simplified circuit containing only basic gates.

Logic Circuits

The circuit diagram with derived gates:



Using the definitions of the Boolean operations: \oplus , \uparrow , \downarrow and by applying DeMorgan, absorption, idempotency and distributive laws we obtain a disjunctive form (not canonical):

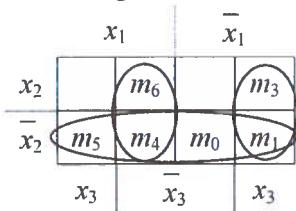
$$\begin{aligned}
 f(x_1, x_2, x_3) &= (x_1 \uparrow x_2) \oplus (\bar{x}_2 \downarrow x_3) = \\
 &= \overline{(x_1 \uparrow x_2)(\bar{x}_2 \downarrow x_3)} \vee (x_1 \uparrow x_2)\overline{(\bar{x}_2 \downarrow x_3)} = \\
 &= \overline{(x_1 x_2)} \overline{(\bar{x}_2 \vee x_3)} \vee \overline{(x_1 x_2)} \overline{\overline{(\bar{x}_2 \vee x_3)}} = \\
 &= x_1 x_2 \bar{x}_3 \vee (\bar{x}_1 \vee \bar{x}_2)(\bar{x}_2 \vee x_3) = x_1 x_2 \bar{x}_3 \vee \bar{x}_2 \vee \bar{x}_1 x_3
 \end{aligned}$$

This form is simple but we try to simplify it more.

We build the Veitch diagram from this disjunctive form as follows:

- for the minterm $x_1 x_2 \bar{x}_3 = m_6$ we mark the corresponding cell;
- for the monom \bar{x}_2 we mark in the diagram the 4 cells (m_1, m_0, m_5, m_4) which have in common the variable: \bar{x}_2
- for the monom $\bar{x}_1 x_3$ which covers the minterms: $\bar{x}_1 \bar{x}_2 x_3 = m_1$ and $\bar{x}_1 x_2 x_3 = m_3$ we mark the corresponding 2 cells.

The Veitch diagram is:



Factorizations:

$$max_1 = \bar{x}_2 = m_1 \vee m_0 \vee m_5 \vee m_4$$

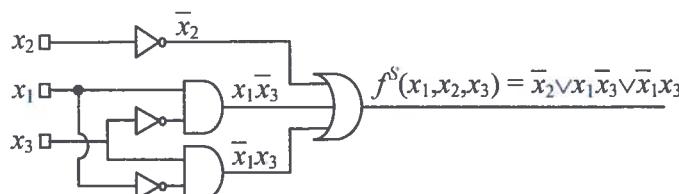
$$max_2 = x_1 \bar{x}_3 = m_6 \vee m_4$$

$$max_3 = \bar{x}_1 x_3 = m_1 \vee m_3$$

$$M(f) = C(f) = \{max_1, max_2, max_3\}$$

According to the first case of the simplification algorithm there is a unique disjunctive simplified form of f : $f^S(x_1, x_2, x_3) = \bar{x}_2 \vee x_1 \bar{x}_3 \vee \bar{x}_1 x_3$.

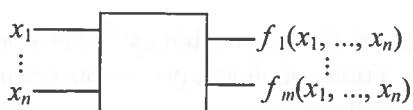
The simplified circuit:



A Computational Approach to Classical Logics and Circuits

A logic circuit which does not use any memory and whose outputs respond immediately to the inputs is called a *combinational circuit*.

It can have m number of outputs. Each output corresponds to a Boolean function.



An elementary logic circuit is a particular case of a combinational circuit ($m = 1$).

In the *combinational logic circuit synthesis task* we have as input the problem specification describing the functionality of the desired circuit and as output the implementation of the logic circuit. To solve the task, first we build the truth tables defining the functions, then we minimize them and finally we build the circuit using the logic gates.

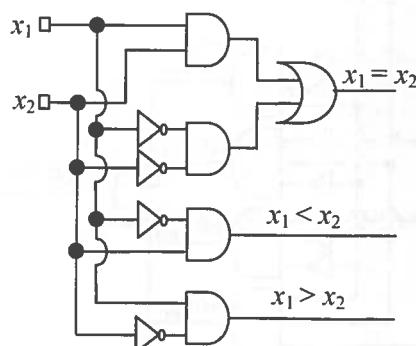
8.2. Examples of useful combinational circuits

Combinational logic circuits can be used to perform arithmetic and logical operations (adders, subtractors, comparators), for data transmission (encoders, decoders, multiplexors, de-multiplexors). Some of these combinational circuits used in the hardware of computers will be described in this section.

Example 8.8.

The **comparator** circuit checks the relation between two binary digits. It is a combinational circuit with two inputs and three outputs. The truth tables of the output functions are as follows:

x_1	x_2	<i>equal</i> ($x_1 = x_2$)	<i>smaller</i> ($x_1 < x_2$)	<i>bigger</i> ($x_1 > x_2$)
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0



We obtain the Boolean expressions for the output functions and we draw the circuit diagram.

$$\text{equal}(x_1, x_2) = \bar{x}_1 \bar{x}_2 \vee x_1 x_2 = \bar{x}_1 \oplus x_2$$

$$\text{smaller}(x_1, x_2) = \bar{x}_1 x_2$$

$$\text{bigger}(x_1, x_2) = x_1 \bar{x}_2$$

Logic Circuits

Boolean functions corresponding to the following binary relations can be written:

- smaller or equal: $x_1 \leq x_2$, $\text{seq}(x_1, x_2) = \bar{x}_1 \vee x_2$
- bigger or equal: $x_1 \geq x_2$, $\text{beq}(x_1, x_2) = x_1 \vee \bar{x}_2$

Example 8.9.

The **full adder** computes the sum of two binary digits: a and b of the same rank in two binary numbers. It is a three input and two output combinational circuit.

input variables: a, b - *two digits*

c_{in} - the *input carry* digit from the previous rank (less significant)

output functions: s - the *one-digit sum* of a and b and c_{in} ,

c_{out} - the generated carry digit, called *output carry*

The table of truth values is presented below.

inputs			outputs	
a	b	c_{in}	$s =$ $a + b + c_{in}$	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

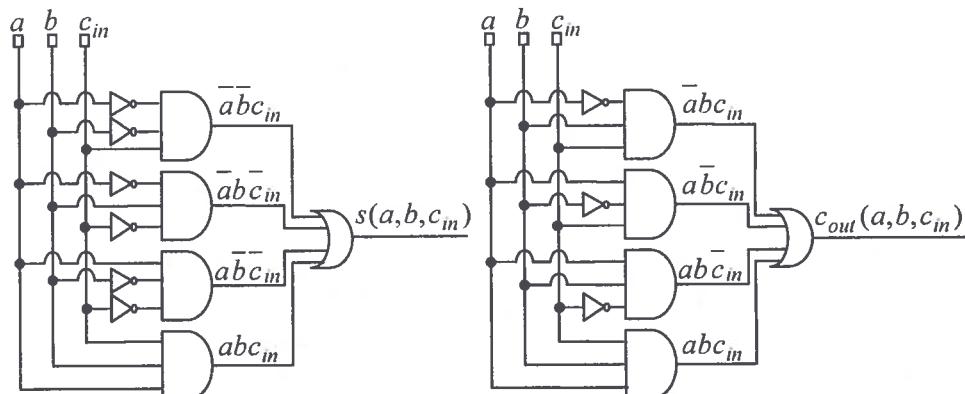


The output functions have the following disjunctive canonical forms:

$$s(a, b, c_{in}) = \bar{a}\bar{b}c_{in} \vee \bar{a}b\bar{c}_{in} \vee a\bar{b}c_{in} \vee abc_{in}$$

$$c_{out}(a, b, c_{in}) = \bar{a}bc_{in} \vee a\bar{b}c_{in} \vee ab\bar{c}_{in} \vee abc_{in}$$

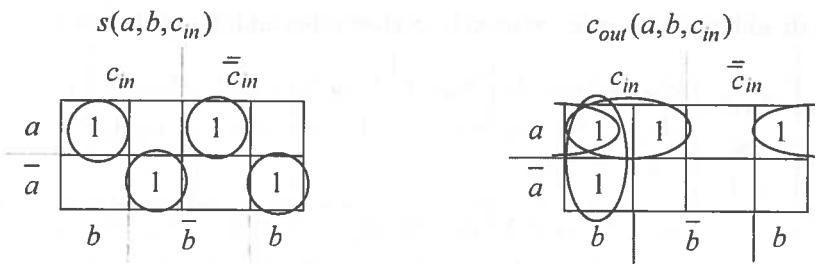
The DCF circuits for the functions are as follows:



Before drawing the corresponding combinational logic circuit we have to simplify the Boolean functions s and c_{out} .

Veitch diagrams are used in simplification.

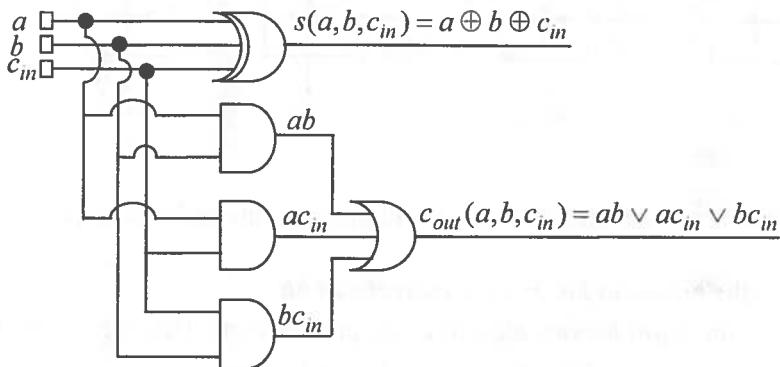
A Computational Approach to Classical Logics and Circuits



Note that the function s is in a simplified form and the simplified form of c_{out} using only basic gates is $c_{out}(a, b, c_{in}) = ab \vee ac_{in} \vee bc_{in}$.

For the function s there is a simplified form with fewest gates (derived): $s(a, b, c_{in}) = a \oplus b \oplus c_{in}$, as can be seen in the truth table.

The combinational circuit diagram is depicted below.

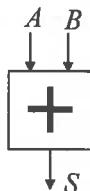


Example 8.10.

To add two n -bit binary numbers we need to use the **n - bit parallel adder**. It is composed of n full adders in cascade. The output carry of the previous full adder is connected to the input carry of the next full adder.

Let $A = (a_{n-1}a_{n-2}\dots a_0)_{(2)}$ and $B = (b_{n-1}b_{n-2}\dots b_0)_{(2)}$ be two binary numbers and $S = A + B$, $S = (s_n s_{n-1} \dots s_0)_{(2)}$. For $A = 10011011_{(2)}$ and $B = 11001111_{(2)}$, $S = A + B = 101101010_{(2)}$.

The logic circuit is symbolized as:

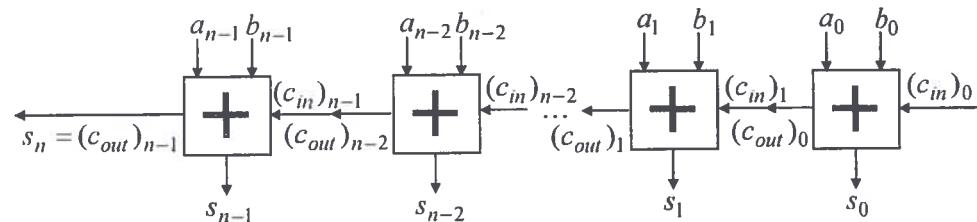


Logic Circuits

Eight full adders in cascade are used to perform this addition as follows:

carries	$(c_{out})_7$ $= s_8$ $= 1$	$(c_{out})_6$ $= (c_{in})_7$ $= 0$	$(c_{out})_5$ $= (c_{in})_6$ $= 0$	$(c_{out})_4$ $= (c_{in})_5$ $= 1$	$(c_{out})_3$ $= (c_{in})_4$ $= 1$	$(c_{out})_2$ $= (c_{in})_3$ $= 1$	$(c_{out})_1$ $= (c_{in})_2$ $= 1$	$(c_{out})_0$ $= (c_{in})_1$ $= 1$	$(c_{in})_0$ $= 0$
<i>A</i>		$a_7 = 1$	$a_6 = 0$	$a_5 = 0$	$a_4 = 1$	$a_3 = 1$	$a_2 = 0$	$a_1 = 1$	$a_0 = 1$
<i>B</i>		$b_7 = 1$	$b_6 = 1$	$b_5 = 0$	$b_4 = 0$	$b_3 = 1$	$b_2 = 1$	$b_1 = 1$	$b_0 = 1$
$S = A + B$	$s_8 = 1$	$s_7 = 0$	$s_6 = 1$	$s_5 = 1$	$s_4 = 0$	$s_3 = 1$	$s_2 = 0$	$s_1 = 1$	$s_0 = 0$

The combinational circuit is composed of n full adders:



Example 8.11.

The **full subtractor** is a combinational circuit with three inputs and two outputs.
input variables:

a - the *minuend bit*, b - the *subtrahend bit*

c_{in} - the *input borrow* digit from the previous rank (less significant)

output functions: d - the one-digit *difference* of a and b ,

c_{out} - the *output borrow*

The table of truth values is presented below.

inputs			outputs	
a	b	c_{in}	d	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



If $a - b - c_{in} \geq 0$ then $d = a - b - c_{in}$ and $c_{out} = 0$
else $d = 2 + a - b - c_{in}$ and $c_{out} = 1$

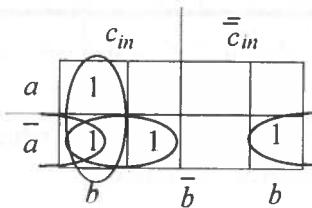
The output functions have the following disjunctive canonical forms:

A Computational Approach to Classical Logics and Circuits

$$d(a, b, c_{in}) = \bar{a}\bar{b}c_{in} \vee \bar{a}b\bar{c}_{in} \vee a\bar{b}c_{in} \vee abc_{in}$$

$$c_{out}(a, b, c_{in}) = \bar{a}\bar{b}c_{in} \vee \bar{a}b\bar{c}_{in} \vee \bar{a}bc_{in} \vee abc_{in}$$

Note that the function d is the same as the output function s from the full adder circuit. The function c_{out} is simplified using Veitch diagram:

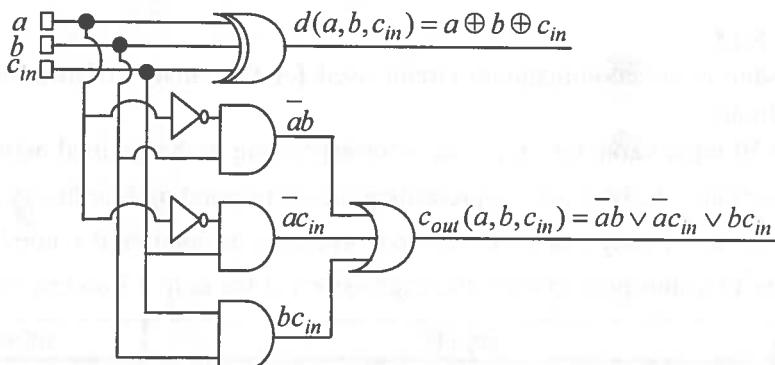


The simplified forms are:

$$d(a, b, c_{in}) = a \oplus b \oplus c_{in}$$

$$c_{out}(a, b, c_{in}) = \bar{a}b \vee \bar{a}c_{in} \vee bc_{in}$$

The combinational circuit diagram is depicted below.



Example 8.12.

The n -bit parallel subtractor is used to subtract two n -bit binary numbers.

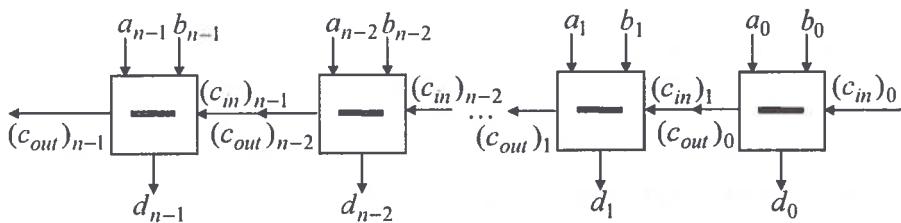
Let $A = (a_{n-1}a_{n-2}...a_0)_{(2)}$ be the minuend and $B = (b_{n-1}b_{n-2}...b_0)_{(2)}$ the subtrahend. The difference $D = A - B$, $D = (d_{n-1}d_{n-2}...d_0)_{(2)}$ is calculated using n full subtractors in cascade. The output borrow of the previous full subtractor is connected to the input borrow of the next full subtractor.

The subtraction in binary on 8 bits for $A = 11101010_{(2)}$ and $B = 10011111_{(2)}$, $D = A - B = 10010111_{(2)}$ is modeled as follows:

Logic Circuits

carries	$(c_{out})_6$	$(c_{out})_5$	$(c_{out})_4$	$(c_{out})_3$	$(c_{out})_2$	$(c_{out})_1$	$(c_{out})_0$	$(c_{in})_0$
$(c_{out})_7 = 0$	$= (c_{in})_7$	$= (c_{in})_6$	$= (c_{in})_5$	$= (c_{in})_4$	$= (c_{in})_3$	$= (c_{in})_2$	$= (c_{in})_1$	$= 0$
A	$a_7 = 1$	$a_6 = 1$	$a_5 = 1$	$a_4 = 0$	$a_3 = 1$	$a_2 = 0$	$a_1 = 1$	$a_0 = 0$
B	$b_7 = 1$	$b_6 = 0$	$b_5 = 0$	$b_4 = 1$	$b_3 = 1$	$b_2 = 1$	$b_1 = 1$	$b_0 = 1$
$D = A - B$	$d_7 = 0$	$d_6 = 1$	$d_5 = 0$	$d_4 = 0$	$d_3 = 1$	$d_2 = 0$	$d_1 = 1$	$d_0 = 1$

The circuit diagram of the *n-bit subtractor*, which is composed of *n* full subtractors in cascade is represented as follows:



Example 8.13.

The **encoder** is the combinational circuit used for the binary codification of the decimal digits.

There are 10 input variables: x_0, \dots, x_9 - corresponding to the decimal digits, and 4 output functions: f_1, f_2, f_3, f_4 - representing the corresponding four binary digits.

The variable $x_i = 1$ only when we want to convert the decimal digit i into binary.

First we build a tableau to identify the expressions of the output Boolean functions:

decimal digit	inputs										outputs			
	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	f_1	f_2	f_3	f_4
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	1	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	1	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	1	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	1	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	0	0	1

A Computational Approach to Classical Logics and Circuits

The disjunctive forms of the Boolean functions are:

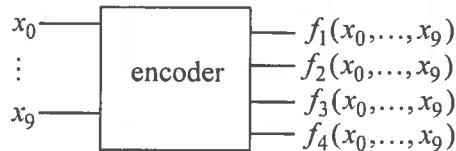
$$f_1(x_0, \dots, x_9) = x_8 \vee x_9$$

$$f_2(x_0, \dots, x_9) = x_4 \vee x_5 \vee x_6 \vee x_7$$

$$f_3(x_0, \dots, x_9) = x_2 \vee x_3 \vee x_6 \vee x_7$$

$$f_4(x_0, \dots, x_9) = x_1 \vee x_3 \vee x_5 \vee x_7 \vee x_9$$

The corresponding diagram of the combinational circuit is:



Example 8.14.

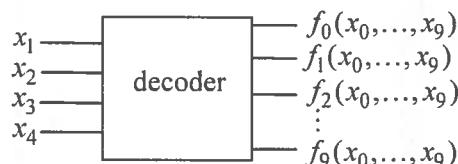
The **decoder** converts a group of 4 binary digits into a decimal digit.

$$f_i(x_1, x_2, x_3, x_4) = 1 \text{ only for } x_1x_2x_3x_4_{(2)} = i_{(10)}, i = 0, 9.$$

- $f_0(0,0,0,0) = 1, 0000_{(2)} = 0_{(10)}$, for all the other arguments the function takes the value 0;
- $f_7(0,1,1,1) = 1, 0111_{(2)} = 7_{(10)}$, for all the other arguments the function takes the value 0.

inputs				outputs										disjunctive canonical form
x_1	x_2	x_3	x_4	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	$f_0(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4$
0	0	0	1	0	1	0	0	0	0	0	0	0	0	$f_1(x_1, x_2, x_3, x_4) = \overline{x_1}x_2\overline{x}_3x_4$
0	0	1	0	0	0	1	0	0	0	0	0	0	0	$f_2(x_1, x_2, x_3, x_4) = x_1\overline{x}_2x_3\overline{x}_4$
0	0	1	1	0	0	0	1	0	0	0	0	0	0	$f_3(x_1, x_2, x_3, x_4) = \overline{x_1}x_2x_3x_4$
0	1	0	0	0	0	0	0	1	0	0	0	0	0	$f_4(x_1, x_2, x_3, x_4) = \overline{x_1}\overline{x}_2x_3\overline{x}_4$
0	1	0	1	0	0	0	0	0	1	0	0	0	0	$f_5(x_1, x_2, x_3, x_4) = \overline{x_1}x_2\overline{x}_3x_4$
0	1	1	0	0	0	0	0	0	0	1	0	0	0	$f_6(x_1, x_2, x_3, x_4) = \overline{x_1}x_2x_3\overline{x}_4$
0	1	1	1	0	0	0	0	0	0	0	1	0	0	$f_7(x_1, x_2, x_3, x_4) = x_1\overline{x}_2x_3\overline{x}_4$
1	0	0	0	0	0	0	0	0	0	0	0	1	0	$f_8(x_1, x_2, x_3, x_4) = \overline{x_1}x_2\overline{x}_3x_4$
1	0	0	1	0	0	0	0	0	0	0	0	0	1	$f_9(x_1, x_2, x_3, x_4) = x_1\overline{x}_2x_3\overline{x}_4$

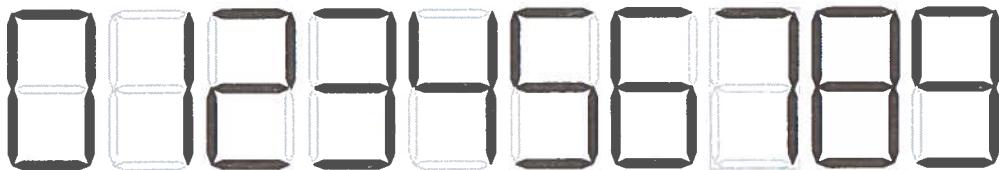
The corresponding combinational circuit is:



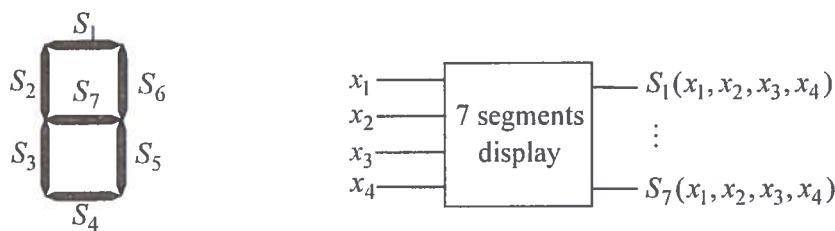
Logic Circuits

Example 8.15.

Electronic display of the decimal digits using 7 segments (LEDs):



The combinational circuit having the functionality described above has as inputs 4 variables and as outputs 7 functions corresponding to the segments.



The combinations of 4 binary digits which do not correspond to a decimal digit will generate an emission of light only for the segment S_7 .

Decimal digit	inputs				outputs						
	x_1	x_2	x_3	x_4	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	0	0	0	1	1	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	0	0	1	1	1	1
4	0	1	0	0	0	1	0	0	1	1	1
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	1	1	1	0	1
7	0	1	1	1	1	0	0	0	1	1	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	1	1	1	1

A Computational Approach to Classical Logics and Circuits

Σ	1	0	1	0	0	0	0	0	0	0	1
Σ	1	0	1	1	0	0	0	0	0	0	1
Σ	1	1	0	0	0	0	0	0	0	0	1
Σ	1	1	0	1	0	0	0	0	0	0	1
Σ	1	1	1	0	0	0	0	0	0	0	1
Σ	1	1	1	1	0	0	0	0	0	0	1

The canonical forms of the output functions are presented.

$$S_1 = m_0 \vee m_2 \vee m_3 \vee m_5 \vee m_6 \vee m_8 \vee m_9 \quad (\text{DCF})$$

$$= M_1 \wedge M_4 \wedge M_{10} \wedge M_{11} \wedge M_{12} \wedge M_{13} \wedge M_{14} \wedge M_{15} \quad (\text{CCF})$$

$$S_2 = m_0 \vee m_4 \vee m_5 \vee m_6 \vee m_8 \vee m_9 \quad (\text{DCF})$$

$$= M_1 \wedge M_2 \wedge M_3 \wedge M_7 \wedge M_{10} \wedge M_{11} \wedge M_{12} \wedge M_{13} \wedge M_{14} \wedge M_{15} \quad (\text{CCF})$$

$$S_3 = m_0 \vee m_2 \vee m_6 \vee m_8 \quad (\text{DCF})$$

$$= M_1 \wedge M_3 \wedge M_4 \wedge M_5 \wedge M_7 \wedge M_9 \wedge M_{10} \wedge M_{11} \wedge M_{12} \wedge M_{13} \wedge M_{14} \wedge M_{15} \quad (\text{CCF})$$

$$S_4 = m_0 \vee m_2 \vee m_3 \vee m_5 \vee m_6 \vee m_8 \vee m_9 \quad (\text{DCF})$$

$$= M_1 \wedge M_4 \wedge M_7 \wedge M_{10} \wedge M_{11} \wedge M_{12} \wedge M_{13} \wedge M_{14} \wedge M_{15} \quad (\text{CCF})$$

$$S_5 = m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_5 \vee m_6 \vee m_7 \vee m_8 \vee m_9 \quad (\text{DCF})$$

$$= M_2 \wedge M_{10} \wedge M_{11} \wedge M_{12} \wedge M_{13} \wedge M_{14} \wedge M_{15} \quad (\text{CCF})$$

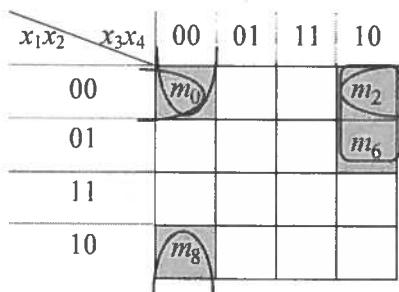
$$S_6 = m_0 \vee m_1 \vee m_2 \vee m_3 \vee m_7 \vee m_8 \vee m_9 \quad (\text{DCF})$$

$$= M_5 \wedge M_6 \wedge M_{10} \wedge M_{11} \wedge M_{12} \wedge M_{13} \wedge M_{14} \wedge M_{15} \quad (\text{CCF})$$

$$S_7 = m_2 \vee m_3 \vee m_4 \vee m_5 \vee m_8 \vee m_9 \vee m_{10} \vee m_{11} \vee m_{12} \vee m_{13} \vee m_{14} \vee m_{15} \quad (\text{DCF})$$

$$= M_0 \wedge M_1 \wedge M_7 \quad (\text{CCF})$$

In the following we simplify S_3 , first its DCF, using Karnaugh diagram:



Logic Circuits

The set of the maximal monoms is $M(S_3) = \{max_1, max_2, max_3\}$. These maximal monoms are obtained by applying three simple factorizations:

$$max_1 = \overline{\underline{x}_1} \overline{\underline{x}_2} \overline{\underline{x}_4} = m_0 \vee m_2$$

$$max_2 = \overline{\underline{x}_1} \underline{x}_3 \overline{\underline{x}_4} = m_2 \vee m_6$$

$$max_3 = \underline{x}_2 \underline{x}_3 \overline{\underline{x}_4} = m_0 \vee m_8$$

The set of the central monoms, $C(S_3) = \{max_2, max_3\} \neq M(S_3)$.

There is a unique disjunctive simplified form of S_3 :

$S_3^S = max_2 \vee max_3 = \overline{\underline{x}_1} \overline{\underline{x}_3} \overline{\underline{x}_4} \vee \overline{\underline{x}_2} \overline{\underline{x}_3} \overline{\underline{x}_4}$. This expression has 6 occurrences of variables, 5 binary operations and 5 unary operations.

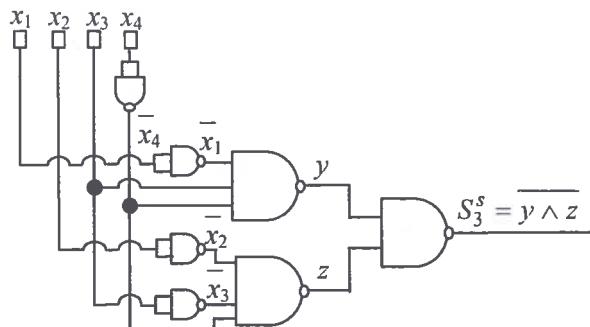
We double negate the expression of S_3^S and we transform it using the results from Example 8.5 in order to implement the corresponding circuit using only NAND gates.

$$\begin{aligned} S_3^S &= (\overline{\underline{x}_1} \wedge \overline{\underline{x}_3} \wedge \overline{\underline{x}_4}) \vee (\overline{\underline{x}_2} \wedge \overline{\underline{x}_3} \wedge \overline{\underline{x}_4}) = \overline{\overline{(\underline{x}_1} \wedge \underline{x}_3 \wedge \underline{x}_4)} \vee \overline{\overline{(\underline{x}_2} \wedge \underline{x}_3 \wedge \underline{x}_4)} = \\ &= \overline{\underline{x}_1} \wedge \overline{\underline{x}_3} \wedge \overline{\underline{x}_4} \wedge \overline{\underline{x}_2} \wedge \overline{\underline{x}_3} \wedge \overline{\underline{x}_4} = \\ &= \overline{\overline{(\underline{x}_1} \wedge \underline{x}_1) \wedge \underline{x}_3 \wedge (\overline{\underline{x}_4} \wedge \underline{x}_4)} \wedge \overline{\overline{(\underline{x}_2} \wedge \underline{x}_2) \wedge (\underline{x}_3 \wedge \underline{x}_3) \wedge (\overline{\underline{x}_4} \wedge \underline{x}_4)} \end{aligned}$$

The logic circuit has:

- four NAND gates which implement the negations of the variables (ex: $\overline{\underline{x}_1} = \underline{x}_1 \wedge \underline{x}_1$);
- two NAND gates, with 3 inputs each, for the expressions $y = \overline{\overline{(\underline{x}_1} \wedge \underline{x}_1) \wedge \underline{x}_3 \wedge (\overline{\underline{x}_4} \wedge \underline{x}_4)}$ and $z = \overline{\overline{(\underline{x}_2} \wedge \underline{x}_2) \wedge (\underline{x}_3 \wedge \underline{x}_3) \wedge (\overline{\underline{x}_4} \wedge \underline{x}_4)}$
- a two-input NAND gate corresponding to $S_3^S = \overline{y \wedge z}$

The circuit diagram is depicted below:



A Computational Approach to Classical Logics and Circuits

The simplification of $CCF(S_3)$ follows a dual simplification algorithm presented in Example 7.10. In the Karnaugh diagram for CCF , the headers of the lines-columns are used to express the indices of the maxterms and represent the duals of the powers of the variables from the maxterms' expressions.

$x_1x_2 \backslash x_3x_4$	00	01	11	10
00		M_1	M_3	
01	M_4	M_5	M_7	
11	M_{12}	M_{13}	M_{15}	M_{14}
10		M_9	M_{11}	M_{10}

By applying a triple dual factorization and two double dual factorizations we obtain the minimal disjunctions of S_3 :

$$min_1 = M_1 \wedge M_3 \wedge M_5 \wedge M_7 \wedge M_{13} \wedge M_{15} \wedge M_9 \wedge M_{11} = \bar{x}_4$$

$$min_2 = M_4 \wedge M_5 \wedge M_{12} \wedge M_{13} = \bar{x}_2 \vee x_3$$

$$min_3 = M_{15} \wedge M_{14} \wedge M_{11} \wedge M_{10} = \bar{x}_1 \vee \bar{x}_3$$

These minimal disjunctions are also central disjunctions of S_3 , because in the diagram each group of cells has at least one maxterm circled once.

According to the first case of the dual simplification algorithm there is a unique conjunctive simplified form:

$$S_3^{CS} = \bar{x}_4 \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3)$$

This expression has 5 occurrences of variables, 4 binary operations and 4 unary operations and is simpler than the disjunctive simplified form S_3^S .

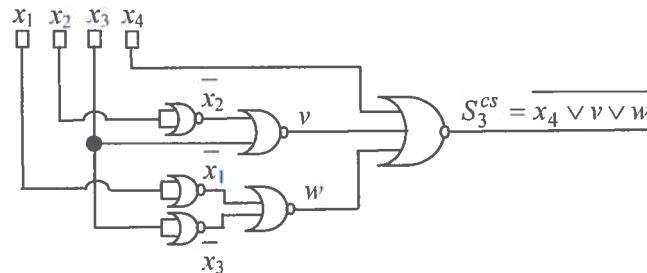
From the conjunctive simplified form we can implement very easily the corresponding logic circuit using only NOR gates. The transformations applied on S_3^{CS} are presented in the following:

$$\begin{aligned} S_3^{CS} &= \bar{x}_4 \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) = \overline{\overline{\bar{x}_4}} \wedge \overline{\overline{(\bar{x}_2 \vee x_3)}} \wedge \overline{\overline{(\bar{x}_1 \vee \bar{x}_3)}} = \\ &= \overline{\overline{x_4}} \vee \overline{\overline{(\bar{x}_2 \vee x_3)}} \vee \overline{\overline{(\bar{x}_1 \vee \bar{x}_3)}} = \\ &= \overline{\overline{x_4}} \vee \overline{\overline{(x_2 \vee x_3)}} \vee \overline{\overline{(\bar{x}_1 \vee x_3)}} = \overline{\overline{x_4}} \vee \overline{\overline{(x_2 \vee x_2)}} \vee \overline{\overline{x_3}} \vee \overline{\overline{(x_1 \vee x_1)}} \vee \overline{\overline{(x_3 \vee x_3)}} \end{aligned}$$

Logic Circuits

In the implementation of the logic circuit six NOR gates are used as follows:

- three gates for $\bar{x}_1, \bar{x}_2, \bar{x}_3$ (ex: $\bar{x}_1 = \bar{x}_1 \vee x_1$);
- two gates, with two inputs each, for the expressions:
 $v = (\bar{x}_2 \vee x_2) \vee x_3$ and $w = (\bar{x}_1 \vee x_1) \vee (x_3 \vee x_3)$;
- a final gate with three inputs corresponding to $S_3^{cs} = \bar{x}_4 \vee v \vee w$



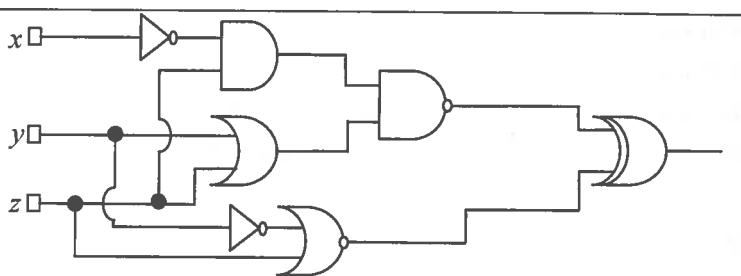
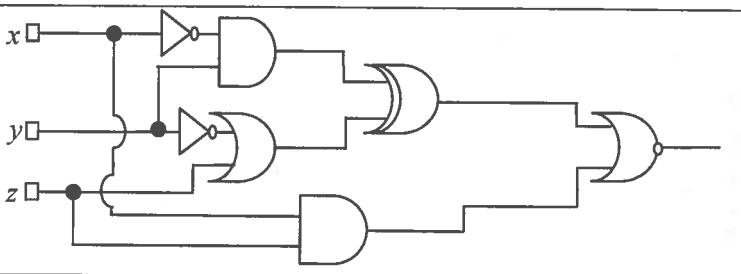
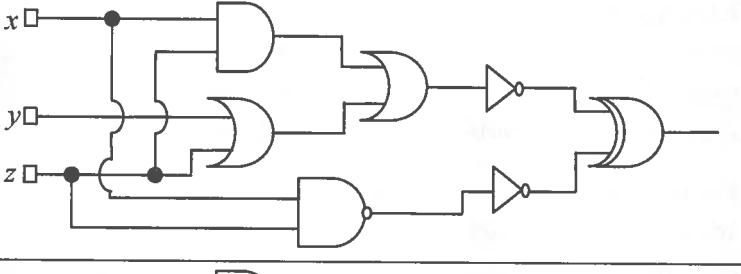
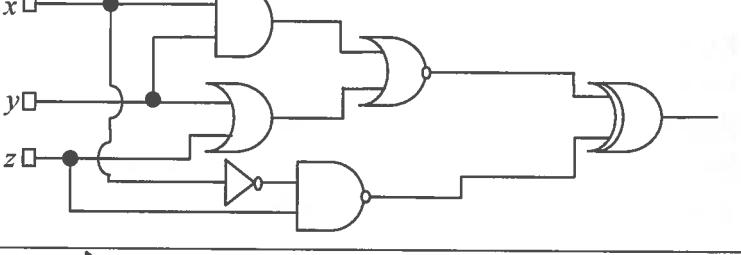
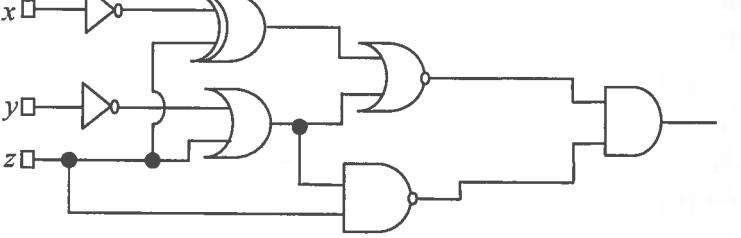
8.3. Exercises

Exercise 8.1.

Write the corresponding Boolean function associated to the following logic circuit:

1.	
2.	
3.	

A Computational Approach to Classical Logics and Circuits

4.	
5.	
6.	
7.	
8.	

Exercise 8.2.

Prove that NOR is a universal gate.

Logic Circuits

Exercise 8.3.

For each of the following Boolean functions draw the corresponding logic circuit using derived gates, simplify the function and draw the logic circuits associated to all simplified forms of the initial function using only basic gates.

1. $f_1(x, y, z) = x(y \oplus z) \vee y(x \oplus z) \vee x(\bar{y} \downarrow \bar{z}) \vee (x \downarrow y)\bar{z}$;
2. $f_2(x, y, z) = x(y \uparrow z) \vee \bar{x}(\bar{y} \oplus z) \vee y(\bar{x} \oplus \bar{z})$;
3. $f_3(x, y, z) = x(\bar{y} \oplus z) \vee y(\bar{x} \oplus z) \vee \bar{x}(\bar{y} \downarrow z) \vee (\bar{x} \downarrow y)z$;
4. $f_4(x, y, z) = \bar{x}(y \uparrow \bar{z}) \vee x(\bar{y} \oplus z) \vee \bar{y}(\bar{x} \oplus z)$;
5. $f_5(x, y, z) = \bar{x}(y \oplus \bar{z}) \vee \bar{y}(x \oplus z) \vee \bar{x}(y \downarrow z) \vee (\bar{x} \downarrow y)\bar{z}$;
6. $f_6(x, y, z) = x(\bar{y} \uparrow \bar{z}) \vee \bar{x}(y \oplus z) \vee \bar{y}(\bar{x} \oplus z)$;
7. $f_7(x, y, z) = x(y \oplus \bar{z}) \vee y(\bar{x} \oplus z) \vee x(y \downarrow z) \vee (x \downarrow y)\bar{z}$;
8. $f_8(x, y, z) = x(\bar{y} \uparrow z) \vee \bar{x}(\bar{y} \oplus z) \vee y(x \oplus \bar{z})$.

Exercise 8.4.

Draw a logic circuit having 3 input wires and containing all basic and derived gates. Write the corresponding Boolean function, simplify it and then draw a simplified circuit equivalent to the initial one.

Exercise 8.5.

Write a Boolean function of 4 variables given by its table of values, simplify it and draw the logic circuits corresponding to all its simplified forms.

Exercise 8.6.

Simplify the Boolean functions ($S_1, S_2, S_3, S_4, S_5, S_6, S_7$) corresponding to the outputs(segments) of the electronic display of the decimal digits using 7 segments (see Example 8.15). Draw the logic circuits for the disjunctive/conjunctive simplified forms using only NAND/NOR gates.

Exercise 8.7.

Binary codes are used to represent the ten decimal digits on 4 bits. There exist *weighted codes* and *unweighted codes*. In a weighted code each binary digit has a ‘weight’ and the decimal value is obtained as the weighted sum of all 4 bits.

BCD - Binary Coded Decimal is an example of a weighted code, with the weights for the digits: 8,4,2,1 (from the most significant digit to the least significant one).

The most used unweighted codes are *Excess 3* (obtained by adding ‘0011’ to each *BCD* word) and the *Gray code* (a cyclic code with the property that all the successive words differ by one single digit).

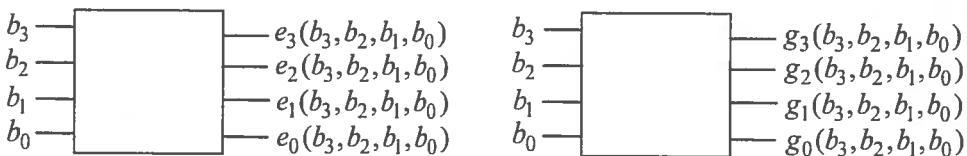
The table below provides the *BCD*, *Excess 3* and *Gray* code on 4 bits.

A Computational Approach to Classical Logics and Circuits

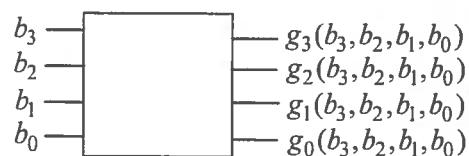
Decimal value	BCD				Excess 3				Gray code			
	b ₃	b ₂	b ₁	b ₀	e ₃	e ₂	e ₁	e ₀	g ₃	g ₂	g ₁	g ₀
0	0	0	0	0	0	0	1	1	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	1
2	0	0	1	0	0	1	0	1	0	0	1	1
3	0	0	1	1	0	1	1	0	0	0	1	0
4	0	1	0	0	0	1	1	1	0	1	1	0
5	0	1	0	1	1	0	0	0	0	1	1	1
6	0	1	1	0	1	0	0	1	0	1	0	1
7	0	1	1	1	1	0	1	0	0	1	0	0
8	1	0	0	0	1	0	1	1	1	1	0	0
9	1	0	0	1	1	1	0	0	1	1	0	1
10	1	0	1	0	-	-	-	-	1	1	1	1
11	1	0	1	1	-	-	-	-	1	1	1	0
12	1	1	0	0	-	-	-	-	1	0	1	0
13	1	1	0	1	-	-	-	-	1	0	1	1
14	1	1	1	0	-	-	-	-	1	0	0	1
15	1	1	1	1	-	-	-	-	1	0	0	0

Implement the combinational circuits for converting:

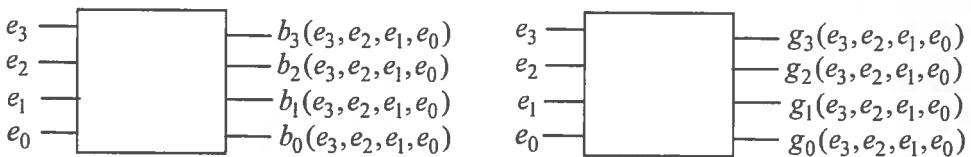
1. from BCD to Excess 3 code;



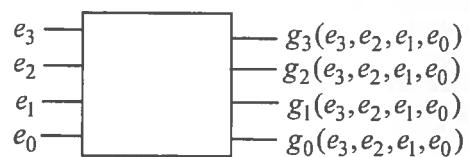
2. from BCD to Gray code;



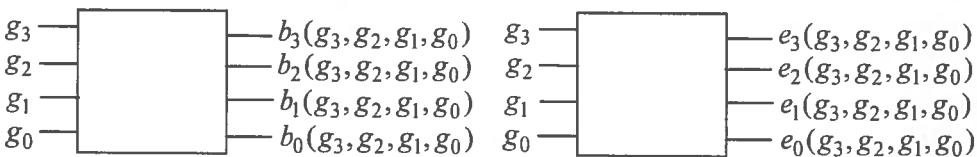
3. from Excess 3 code to BCD;



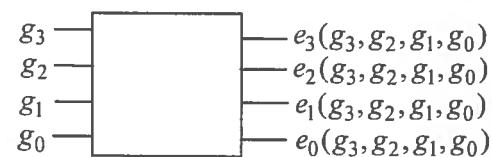
4. from Excess 3 code to Gray code;



5. from Gray code to BCD;



6. from Gray code to Excess 3 code;



Index

INDEX

$\alpha, \beta, \gamma, \delta$ (type formula, rule)	67, 68	binary	185
$\wedge(l, r), \vee(l, r), \neg(l, r)$	98	Boolean function	184, 186
$\exists(l, r), \forall(l, r)$	98	canonical form	188
\Rightarrow (provable)	97	branch	69, 82
$\uparrow, \downarrow, \oplus, \neg, \wedge, \vee, \rightarrow, \leftrightarrow$	9	closed	69, 83
7 segments display	241	complete	69
absorption laws	5	open	69, 83
adder	234, 236	canonical disjunction	190
addition rule	18	CCF	188
adjacent (neighbor) monoms	194	circuit	
Algorithm		comparator	234
build tree from postfix	90	clause	10
construction semantic tableau	92	central	133
for computing the mgu	55	clashing	118, 138
Herbrand	59	empty	118
level-saturation-strategy	123	Horn	134
normalization	11	input	134
positive hyperresolution	152	mixed	149
postfix notation	88	negative	134
predicate_resolution	139	parent	118
propositional resolution	119	positive	134, 149
semantic tableaux method	94	side	133
simplification	196	top	133
and gate	224	unit	121
anti-model	3	CNF	11
first-order logic	45	coherence	24
anti-sequent		combinational circuit	233
basic	106	common instance	54
non-derivability	108	commutative laws	5
anti-sequent calculus	106	compactness	22, 23
antecedent		conclusion	7, 15
anti-sequent	106	sequent calculus	99
sequent	97	conjunction	1, 190
associative laws	6	rule	19
ATOMS	36	conjunction in conclusions	9
axiom	36	conjunctive canonical form	188
axiomatic system		conjunctive normal form	11
anti-sequent	107	connective	1, 35
first-order logic	35	consequent	
propositional logic	14	anti-sequent	106
resolution	118	sequent	97
sequent calculus	98	consistent	
axiom	15	formula	3, 44
backtracking	133	set	7, 45
basic gate	224	constant	43
Boolean algebra	184	Skolem	51
		correspondence tableau	209

A Computational Approach to Classical Logics and Circuits

cube	10	first-order logic	35
cut	9	propositional logic	14
Davis-Putman procedure	121	resolution	118
DCF	188	sequent	98
De Morgan's laws	6	formula	1
decidability	24, 100	closed	37
decision problems	24	ground	58
decoder	240	open	37
decomposition rules	67, 82	function	
deducible	15, 37	Skolem	51
deduction	38	function symbol	43
theorem	20	functionally complete	6
derivable	15, 37	ground atom	57
derived gate	225	ground instance	58
direct method	24	Herbrand	57
disjunction	1	Algorithm	59
disjunction in premises	9	base	58
disjunctive canonical form	188	system	58
disjunctive normal form	10	Theoreme	59
distributive laws	6	universe	58
DNF	10, 69	hyperresolution	149
domain		negative	153
of interpretation	43	positive	151
substitution	53	hypothesis	7, 15, 37
dual		idempotency laws	5
concepts	12	implication	1
connectives	6	inconsistent	3, 44
logical equivalences	6	set	7, 45
rules	99	induction	15
truth values	6	inferable	15, 37
duality principle	6	inference rules	
Boolean algebra	185	set of	15
empty clause	10	interpretation	2, 43
encoder	239	Karnaugh diagram	197
equivalence	1	law	
equivalences		absorption	5
logical	5	associative	6
equivalent		commutative	5
logically	4	De Morgan's	6, 47
exclusive or	2	distributive	6, 47
existential		expansion	47
quantifier	35	extraction of quantifiers	47
existential generalization	38	idempotency	5
existential instantiation	38	quantifiers interchanging	47
facts	7	semi-distributive	47
factor	138	simplification	5
factoring rule	138	literal	10, 36
factorization	194, 198	pure	121
factorization process	202	logic circuit	224
false	9	logical connectives	
under the interpretation	45	first-order logic	45
first-order logic	35	set of	15
formal system		logical consequence	4
anti-sequent	107	of a set	7

Index

logical equivalences	5	syntactic	24
first-order logic	46	properties	
logically equivalent	4	propositional logic	22
first-order logic	45	property	
logically implies		conjunction in conclusions	9
first-order logic	45	disjunction in premises	9
matrix	50	transitivity	9
maxterm	190	property	
minterm	190	cut	9
model	3	monotonicity	9
first-order (predicate) logic	44	property	
first-order formula	70	non-contradictory	23
propositional formula	70	property	
modus ponens	15, 19, 36, 37	coherent	24
modus tollens	19, 36	property	
Moisil	216	decidable	24
monom	190	propositional logic	1, 155
central	195, 201, 209	axiomatic system	14
maximal	195, 199, 209	formal system	14
monotonicity	9	normal forms	10
nand	2	properties	22
gate	225	semantics	1
negation	1	soundness and completeness	23
non-contradictory		propositional variables	
property	23	set of	1, 14
nor	2	quantifier	
gate	225	existential	35
normal form		universal	35
clausal	51	quantifiers	35
conjunctive prefix	50	Quine-Mc'Clusky	209
prefix	50	reductio ad absurdum	40
Skolem	51	refutation proof method	25, 40
normal forms		reasoning modeling	155
first-order logic	50	resolution	
propositional logic	10	axiomatic system	118
normalization algorithm	11	deletion strategy	123
not gate	224	factoring rule	138
or gate	224	first-order (predicate) logic	138
predicate logic	162	incompleteness	126, 129, 130, 134
predicate symbol	44	input	134
prefix	50	linear	132
premise		lock	127
sequent calculus	99	method	118
premises	7	refinements	125
program verification	178	rule	19, 118, 138
proof by contradiction	40	semantic	145
proof method		set-of support strategy	124
anti-sequent calculus	108	unit	134
direct	24	resolvent	
refutation	25	binary	138
resolution	118	predicate	138
semantic	24	reverse of the theorem of deduction	20
semantic tableaux	67	rule	
sequent calculus	99	addition	18

A Computational Approach to Classical Logics and Circuits

anti-sequent	calculus	-	term	36
inference/reduction		107	ground	57
conjunction		19	theorem	38, 49
inference		99	Theorem	
modus ponens		19	Church	49, 140
modus tollens		19	complementarity of sequent and anti-	
reduction		99	sequent calculi	108
resolution		19	completeness	23
sequent calculus		98	completeness - resolution	120
simplification		19	completeness lock resolution	127
sylllogism		19	deduction	20, 40
satisfiable		3, 44	Herbrand	59
semantic domain		1	refutation	40
semantic proof			soundness	23
method		24	soundness - resolution	120
semantic tableau		82	soundness and completeness	
closed		69, 83	of first order logic	49
complete		69	of linear resolution	133
construction		68	of propositional logic	23
open		69, 83	of resolution	120
simplified form		83	of sequent calculus	100
semantic tableaux method		67	of the semantic tableaux method	70
semantics			of TP	83
first-order (predicate) logic		43	soundness lock resolution	127
propositional logic		1	theorem prover	87
sequent calculus		97	TP function	82
semi-decidable		49, 71	transitivity	9
first-order logic		101	true	9
sequent		97	anti-sequent	107
basic		97	under the interpretation	44
sequent calculus		97	truth tables	2
axiomatic system		98	truth values	1
derivability		108	unsatisfiable	3
method		99	true	9
sequent calculus			under the interpretation	44
inference/reduction rules		98	undecidable	49
set of well-formed formulas		1	unification	53
simplification			unifier	54
Boolean function		193	most general	55
dual method CCF		219	universal quantifier	35
rule		19	universal generalization	36, 37, 38
simplification laws		5	universal instantiation	36, 38
simplification method			unsatisfiable	44
Moisil		216	valid	45
Quine-Mc'Clusky's		209	variable	
Veitch-Karnaugh diagrams		197	bound	37
substitution		53	free	37
subtractor		237, 238	Veitch-Karnaugh diagram	197
sylllogism rule		19	valid	3
syntactic proof			vocabulary	1, 14, 35
method		24	well-formed formula	15
syntax		1	xor	2
sequent calculus		97	gate	225
tautology		3, 45, 49		

BIBLIOGRAPHY

1. Ben-Ari, M., *Mathematical Logic for Computer Science*, Springer, 2001.
2. Benzaken, C., *Systèmes Formels. Introduction à la Logique*, Masson, 1991.
3. Bibel, W., *Automated Theorem Proving*, View Verlag, Braunschweig, second edition, 1987.
4. Boian, F. M., *De la aritmetică la calculatoare*, Presa Universitară Clujeană, 1996.
5. Bonatti, P., Olivetti, N., *Sequent Calculi for Propositional Nonmonotonic Logics*, ACM Trans. Comput. Log., 2002, pag. 226-278.
6. Boole, G., 1847, *The Mathematical Analysis of Logic. Being an Essay Towards a Calculus of Deductive Reasoning*, Cambridge: Macmillan, Barclay, 1847.
7. Boole, G., 1854, *An Investigation of the Laws of Thought, on which are Founded the Mathematical Theories of Logic and Probabilities*, London: Walton & Maberly, 1854.
8. Both, N., *Algebra logicii cu aplicații*, Editura Dacia, Cluj-Napoca, 1984.
9. Both, N., *Capitole Speciale de Logică Matematică*, Universitatea Babeș-Bolyai, Cluj-Napoca, 1994.
10. Boyer, R.S., *Locking: A Restriction of Resolution*, Ph.D. Thesis, University of Texas at Austin, Texas, 1971.
11. Chang, C.L., *The unit proof and the input proof in theorem proving*, Journal Assoc. Comput. Mach. 17, 1970, pp.698-707.
12. Chang, C.L., Lee, R.C., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
13. Church, A., *An unsolvable problem of number theory*, American Journal of Mathematics, 58, pag.345-363, 1936.
14. Cocan, M., Pop, B., *Bazele Matematice ale Sistemelor de Calcul*, Editura Albastră, Cluj-Napoca, 2001.
15. Davis, M., Putnam, H., *A computing procedure for quantification theory*, Journal Assoc. Comput. Mach., 7, 1960, pp. 201-215.
16. Delahaye, J.P., *Outils Logiques pour l'Intelligence Artificielle*, Eyrolles, 1986.
17. Dreben, B., Goldfarb, W., *The Decision Problem: Solvable Classes of Quantificational Formulas*, Addison-Wesley, 1979.
18. Duffy, D.A., *Principles of Automated Theorem Proving*, John Wiley & Sons, 1991.
19. Dumitrescu, D., *Principiile Inteligenței Artificiale*, Editura Albastră, Cluj-Napoca, 2002.
20. Fitting, M., *First-order logic and Automated Theorem Proving*, Springer Verlag, 1990.
21. Floarea, A., Boangiu, A., *Inteligenta Artificială*, Universitatea Tehnică București, 1994
22. Gentzen, G.K.E., *Untersuchungen über das logische Schließen. I*, Mathematische Zeitschrift 39 (2), 1934, pp. 176–210.

Bibliography

23. Genesereth, M.R., Nilsson, N.J., *Logical Foundations of Artificial Intelligence*, Morgan Kaufman, 1992.
24. Hilbert, D., Ackermann, W., *Principles of Mathematical Logic*, Chelsea, New York, 1950.
25. Hsu, J. Z., *Computer logic. Design Principles and Applications*, Springer-Verlag, New York, 2002.
26. Kleene, S.C., *Mathematical Logic*, Wiley, New York, 1967.
27. Lenzen, W., *Leibniz's Logic*, in *Handbook of the History of Logic*, D. M. Gabbay/J. Woods (eds.), volume 3: The Rise of Modern Logic: From Leibniz to Frege, Amsterdam et al.: Elsevier-North-Holland, pp. 1–83, 2004.
28. Livovschi, L., *Circuite cu contacte de relee*, Editura Academiei Republicii Socialiste România, 1968.
29. Littlewood, J.E., *Varietăți matematice*, București, 1969.
30. Loveland, D.W., *A linear format for resolution*, Proceedings IRIA Symp. Automatic Demonstration, Versailles, France, 1968, Springer-Verlag, New York, 1970, pp. 147–162.
31. Loveland, D.W., *Automated Theorem Proving: A Logical Basis*, North Holland, Amsterdam, 1978.
32. Luckham, D., *Refinements in resolution theory*, Proceedings IRIA Symp. Automatic Demonstration, Versailles, France, 1968, Springer-Verlag, New York, 1970, pp. 147–162.
33. Lupea, M., *Lock resolution - a refinement of resolution*, Seminar on Computer Science, Editura Universitatii Babes-Bolyai, Cluj-Napoca, 1994, pp. 57-64.
34. Lupea, M., *Semantic tableaux to compute extensions for different versions of default logic*, Research Seminar on Computer Science, Editura Universitatii Babes-Bolyai, Cluj-Napoca, 2000, pp. 31-48.
35. Lupea, M., *Raționament nemonoton prin logici implicate*, Ph.D Thesis, Babeș-Bolyai University, Cluj-Napoca, 2002.
36. Lupea, M., *Axiomatization of credulous reasoning in rational default logic*, Studia Universitas Babeș-Bolyai, Informatica, LII(1), pag:101-111, 2007.
37. Lupea, M., Mihiș, A., *Logici clasice și circuite logice. Teorie și exemple*, Editura Albastra, Cluj-Napoca, first edition (2008), second edition (2009), third edition (2011).
38. Malița, M., Mircea M., *Bazele Inteligenței Artificiale*, Vol.I, Logici propozitionale, Editura Tehnică, București, 1987.
39. Mihiș, A.D., Chisăliță-Crețu C., Mihăilă C., Șerban C., *BOOFS - a tool that supports simplifying conditional expressions using boolean functions simplification methods*, Proceedings of International Conference of Mathematics & Informatics, ICMI45, Bacău, Septembrie 18-20, 2006, Studii și cercetări științifice, nr.16 – 2006 Supplement, Universitatea din Bacău, Facultatea de Științe Matematice, ISN 1224-2519.
40. Moșcenski, V.A., *Lecții po matematicheskoi logike*, Minsk, 1973.
41. Nasin, P., *Circuite logice și automatizări sevențiale*, Editura Tehnică, București 1967.

A Computational Approach to Classical Logics and Circuits

42. Nilsson, N.J., *Principles of Artificial Intelligence*, Tioga, Palo Alto, Morgan Kaufmann, 1980.
43. Paulson, L.C., *Logic and Proof*, Cambridge University, 2000, www.cl.cam.ac.uk/teaching/2000/LogicProof/notes.pdf.
44. Popa, C., *Logica predicatelor*, Editura Hyperion, Bucureşti, 1992.
45. Possega, M., *Deduction Systems*, Institute of Informatics, 2002, on-line course.
46. Rădulescu, V., *Revanşa minții*, Editura Militară, Bucureşti, 1974.
47. Reeves S., Clarke, M., *Logic for Computer Science*, Eddison-Wesley, 1990.
48. Rich, E., *Artificial Intelligence*, Mac Graw-Hill, New York, 1983.
49. Risch, V., Schwind, C.B., *Tableau-Based Characterization and Theorem Proving for Default Logic*, Journal of Automated Reasoning, Vol 13, Nr. 2, 1994, pag. 223-242.
50. Robinson, J.A., *A machine oriented logic based on resolution principle*, Journal Assoc. Comput. Mach, 12, 1965, pp. 23-41.
51. Robinson, J.A., *Automatic deduction with hyper-resolution*, International Journal Computation Math, 1, 1965, pp. 227-234.
52. Robinson, J.A., *Logic, Form and Function. The Mechanization of Deductive Reasoning*, University Press, Edinburg, 1979.
53. Roul de Palma, *Algebra binară a lui Boole și aplicațiile ei în informatică*, Editura Tehnică, Bucureşti, 1976.
54. Russel, S., Norwig, P., *Artificial Intelligence*, Prentice Hall, 1995.
55. Rusu, E., *Cum gândim și rezolvăm 200 de probleme*, Editura Albastros, Bucureşti, 1972.
56. Schwind, C.B., *A tableaux-based theorem prover for a decidable subset of default logic*. 10-th International Conference on Automated Deduction. Lecture Notes in A.I. 449, 1990.
57. Slagle, J.R., *Automatic theorem proving with renamable and semantic resolution*, Journal Assoc. Comput. Mach, 12, 1965, pp. 536-541.
58. Smullyan, R.M., *First-Order Logic*, Springer Verlag, Berlin, 1968. Revised Edition, Dover Press, New York, 1996.
59. State, L., *Elemente de Logică Matematică și Demonstrarea Automată a Teoremelor*, Universitatea Bucureşti, 1989.
60. Tarski, A., *Introduction to Logic*, Oxford University Press, 1976.
61. Thayse, A., editor: *From Standard Logic to Logic Programming*, John Wiley & Sons, vol1 (1989), vol2 (1989), vol3 (1990).
62. Tătar, D., *Bazele Matematice ale Calculatoarelor*, Lito, Babeș-Bolyai University, Cluj-Napoca, 1999.
63. Tătar, D., Inteligența Artificială: Demonstrare Automată de Teoreme și NLP, Editura Microinformatica, Cluj-Napoca, 2001.
64. Tîrnoveanu, M., *Elemente de logică matematică, vol.1, Logica propozițiilor bivalente*, Editura didactică și pedagogică, Bucureşti, 1964.
65. Wos, L., Robinson J.A., Carson, D.F., *Efficiency and completeness of the set of support strategy in theorem proving*, Journal Assoc. Comput. Mach, 12, 1965, pp. 536-541.