# Logic for Computer Science. Lecture Notes

Andrzej Szałas

College of Economics and Computer Science, Olsztyn, Poland
and
Department of Computer Science, University of Linköping, Sweden

http://www.ida.liu.se/∼andsz

February 2002

# Contents

# Part I

# Introduction to Logics

# Chapter 1

# Introduction

## 1.1 Introduction to the Course

This set of lecture notes has been prepared as a material for a logic course given in the Swedish National Graduate School in Computer Science (CUGS).

The course is focused on various aspects of classical and non-classical logics, including:

- the classical propositional and predicate calculus
- modal logics, including logics of programs and temporal logics
- fixpoint calculus.

The main emphasis is put on automated deduction and computer science applications of considered logics. Automated deduction techniques are presented mainly in the context of the classical logics. The following techniques are considered:

- resolution method
- sequent calculus
- analytic tableaux.

Application areas include:

- formal specification and verification of software
- formal specification of data structures
- logic programming and deductive databases

- knowledge representation and commonsense reasoning.

This set of lecture notes is mainly based on the books [BDRS95, BDRS98, DŁSS02, Sza92, Sza95] and papers [DŁS97, NOS99]. An excellent textbook on mathematical logic is [EFT94]. These sources contain a comprehensive list of the relevant literature.

## 1.2 Introduction to Logics

Logical formalisms are applied in many areas of computer science. The extensive use of those formalisms resulted in defining hundreds of logics that fit nicely to particular application areas. Let us then first clarify what do we mean by a logic.

Recall first the rôle of logic in the clarification of human reasoning. In order to make the reasoning fruitful, first of all we have to decide what is the subject of reasoning or, in other words, what are we going to talk about and what language are we going to use. The next step is to associate a precise meaning to basic notions of the language, in order to avoid ambiguities and misunderstandings. Finally we have to state clearly what kind of opinions (sentences) can be formulated in the language we deal with and, moreover, which of those opinions are true (valid), and which are false (invalid). Now we can investigate the subject of reasoning via the validity of expressed opinions. Such an abstraction defines a specific logic.

Traditionally, there are two methodologies[1] to introduce a logic:

- syntactically, via a notion of a proof and proof system
- semantically, via a notion of a model, satisfiability and truth.

Both methodologies first require to chose a language that suits best a particular application. Of course we use different vocabulary talking about politics and about computer science. Moreover, we even use different linguistic tools. Logical language is defined by means of basic concepts, formulas and logical connectives or operators. Thus a syntax of the language is to be defined, so that one obtains a precise definition what are well formed sentences or formulas.

Having a language syntax defined, one knows how to "speak" correctly and does not know what do the spoken sentences mean and whether these are true or false. Thus a meaning (also called an interpretation) is to be attached to any well formed formula of the logic. The meaning is given either by the notion of provability, or satisfiability. Namely, a sentence (formula) is considered valid provided that it can be proved (syntactical approach) or it is valid in models accepted as semantical structures (semantical approach).

[1] And it is often desirable and even necessary to follow both methodologies, assuming that they lead to compatible results.

I find the semantical approach more fundamental and consider the syntactic approach as (still extremely important) tool. Thus logic will always be presented here primarily via the underlying semantics, showing also how various types of proof systems for the logics can be constructed. We then accept the following definition of logic.

**Definition 1.2.1** By a *logic* we shall mean triple $L = \langle F, C, \models \rangle$, where:

- $F$ is a set of *well-formed formulas*
- $C$ is a class of possible *interpretations* (*models*)
- $\models \subseteq C \times F$ is a *satisfiability relation*.

For $I \in C$ and $\alpha \in F$, if $I \models \alpha$ then we say that interpretation $I$ *satisfies* formula $\alpha$ or $I$ is a *model* od $\alpha$. For $C' \subseteq C$ and $F' \subseteq F$, by $C' \models F'$ we shall mean that for any interpretation $I \in C'$ and any formula $\alpha \in F'$ we have that $I \models \alpha$.

A formula is a *tautology* of $L$ iff for any interpretation $I \in C$, we have that $I \models \alpha$. A formula is *satisfiable* iff there is an interpretation $I \in C$, such that $I \models \alpha$. ■

Another fundamental notion in logics is that of consequence. Namely, Definition 1.2.1 provides us with a meaning of a formula, but at that abstract level we still do not know what it means that a formula is a consequence of other formulas. In the following definition we clarify the concept.

**Definition 1.2.2** Let $L = \langle F, C, \models \rangle$ be a logic. Then we say that a formula $\alpha \in F$ is a *semantic consequence* of a set of formulas $F' \subseteq F$ iff for any interpretation $I \in C$ we have that

$$I \models F' \text{ implies } I \models \alpha. \blacksquare$$

## 1.3 Introduction to Proof Systems

Observe that the definition of tautologies does not provide us with any tools for systematically proving that a given formula is indeed a tautology of a given logic or is a consequence of a set of formulas.

One of possible ways of providing such tools is to define suitable proof systems. Traditionally proof systems are often used to define new logic instead of defining them via semantics. Then one immediately obtains tools for proving the validity of formulas.

Hilbert-like axiomatic methods based on proof systems are accepted as a basis for formal and verifiable reasoning about both tautologies and properties of interpretations. They are very intuitive and easy to understand. However, there are currently much more open questions concerning the implementation of Hilbert-like proof systems than there are suitable solutions. The reasoning with Hilbert-like proof systems depends on accepting a set of basic axioms (i.e. "obvious" formulas admitted without proof), together with derivation rules, and then on deriving conclusions directly from axioms and/or theorems proved previously. Derivation rules are usually formulated according to the following scheme:

> if all formulas from a set of formulas (so-called *premises*) $A$ are proved then formula $\alpha$ (so-called *conclusion*) is proved, too.

Such a rule is denoted by $A \vdash \alpha$ or often by $\dfrac{A}{\alpha}$. The set of provable formulas is defined inductively as the least set of formulas satisfying the following conditions:

- every axiom is provable (note that some of the axioms may be so-called nonlogical axioms coming from the specific theory we are working in).
- if the premises of a rule are provable then its conclusion is provable, too.

One can then think of proof systems as nondeterministic procedures, for the process of proving theorems can be formulated as follows, where formula $\alpha$ is the one to be proved valid:

> if $\alpha$ is an axiom, or is already proved, then the proof is finished, otherwise select (nondeterministically) a set of axioms or previously proved theorems and then apply a nondeterministically chosen applicable derivation rule. Accept the thus obtained conclusion as the new theorem and repeat the described procedure.

As axioms are special kinds of derivation rules (namely those with the empty set of premises), nondeterminism can appear only when there are several derivation rules that can be applied during the proof.

Gentzen-like proof systems, also called *sequent calculus*, offer a more general form of derivation rules. The key rôle is played here by the notion of sequents taking the form $A \Rightarrow B$, where both $A$ and $B$ are finite sets of formulas. Intuitively, sequent $A \Rightarrow B$ means the following:

> the conjunction of formulas of set $A$ implies the disjunction of formulas of $B$, where, by convention, conjunction of the empty set of formulas is $\mathsf{True}$, while its disjunction is $\mathsf{False}$.

There is, however, an essential difference between the Hilbert and Gentzen methods of proofs. Namely, as Hilbert-like calculus is used to derive single formulas from sets of formulas, so sequent calculus allows us to derive sequents from other sequents. Moreover, Gentzen- and Hilbert-like proofs go in opposite directions. That is to say, in Hilbert-like systems the formula to be proved is obtained in the final stage of the proof, while in a Gentzen-like proof it is a starting point of the proof. The Gentzen-like (naîve) proof procedure can then be formulated as follows, where formula $\alpha$ is to be proved valid.

> Start the whole proof from sequent $\emptyset \Rightarrow \{\alpha\}$. If the sequent (or all other sequents obtained during derivation, if any) is (are) indecomposable (i.e. rules are no longer applicable) then check whether all of the final sequents are axioms. If the answer is yes, then $\alpha$ is proved valid, otherwise it is invalid. If some sequent is decomposable then first decompose it and then repeat the described procedure.

Axioms in Gentzen-like proof systems are usually very simple. For instance, any sequent $A \Rightarrow B$ such that $A \cap B \neq \emptyset$ is an axiom in many proof systems for tautologies. Derivation rules, however, take the more complicated form $S \vdash B$, where $S$ is a set of sequents and $B$ is a sequent. A rule $S \vdash B$ is often denoted by $\dfrac{B}{S}$.

Note that Hilbert-like proof systems are easy to use while reasoning about theories. One has only to add specific axioms of a theory to axioms of logic and then to apply derivation rules. Many theories do have nice Gentzen-like axiomatizations. However, obtaining them is often not a trivial task. Moreover, implementation of Gentzen-like axiomatizations of theories raises new problems and considerably complicates the process of finding proofs.

Let us summarize the above discussion with the following definitions.

**Definition 1.3.1** Let $L$ be a logic.

- By a *sequent* of logic $L$ we mean any expression of the form $A \Rightarrow B$, where $A$ and $B$ are finite sets of formulas of $L$.
- By a *Gentzen-like proof system* for logic $L$ we mean any pair $\langle GAx, G\rangle$ such that
  - $GAx$, called a set of *axioms*, is any set of sequents of $L$,
  - $G$ is any set of derivation rules of the form $S \vdash s$, where $S$ is a set of sequents of $L$, and $s$ is a sequent of $L$.
- We say that sequent $s$ is *indecomposable* in a given Gentzen-like proof system iff $s$ is an axiom or no rule of the system is applicable to $s$. A sequent is called *decomposable* iff it is not indecomposable.
- By a *Hilbert-like proof system* for logic $L$ we mean any pair $\langle HAx, H\rangle$ such that

- $HAx$, called a set of *axioms*, is any set of formulas of $L$,
- $H$ is any set of derivation rules of the form $A \vdash p$, where $A$ is a set of formulas of $L$, and $p$ is a formula of $L$. ■

Note that separating axioms from rules in the case of Hilbert-like proof systems is not necessary. Such a separation, however, allows us to treat both Gentzen- and Hilbert-like proof systems uniformly. Note also that axioms and derivation rules are usually given by schemes rather than by specific sequents, formulas or sets. For instance, writing $p, p \to q \vdash q$ we always think of $p$ and $q$ as variables ranging over set of formulas. Thus the above scheme of rules defines (usually infinitely) many rules that can be obtained by substituting $p$ and $q$ with specific formulas.

**Definition 1.3.2** Let $P = \langle Ax, C \rangle$ be a Gentzen (Hilbert)-like proof system for logic $L$. By a *proof* in $P$ we mean a rooted tree labelled by sequents (formulas) such that

- the height of the tree is finite,
- all leaves are labelled by elements of $Ax$ (sequents or formulas, respectively),
- any node $n$ in the tree is labelled either by an element of $Ax$, or by sequent (formula) $s$ for which there is a derivation rule $D \vdash s$ in $C$ with $D = \{t | t$ is a label of a son of $n$ in the tree$\}$.

We say that the sequent (formula) $s$ is provable in $P$ iff there is a proof in $P$ with a root labelled by $s$. ■

Since we are not going to discuss here applications of Gentzen-style proofs of theories, the next definition concerns Hilbert-like proof systems only.

**Definition 1.3.3** Let $P = \langle HAx, H \rangle$ be a Hilbert-like proof system for logic $L$. By a *syntactic consequence* (w.r.t. $P$) of a set of formulas $A$ we mean any formula provable in the proof system $\langle HAx \cup A, H \rangle$. The set of syntactic consequences (w.r.t. $P$) of set $A$ is denoted by $C_P(A)$. ■

We conclude this section with an informal discussion of soundness and completeness. As mentioned in the introduction, soundness is always the most fundamental property of any reasonable proof system. Soundness means that all proved conclusions are semantically true. In terms of procedures, one can define soundness as correctness of the procedure implementing the proof system. All the results of the procedure must then be correct. Completeness, however, means that all semantically true conclusions can be obtained as results of the procedure. In other words, soundness means that all answers given by a proof system are correct, while completeness means that all correct answers can be obtained using the proof system. As soundness is then always required, completeness serves as a measure of the quality of proof systems.

## 1.4 BNF Notation

We define the syntax of various logical languages using BNF notation with some commonly used additions. Elements (words) of the defined language are called *terminal symbols*. *Syntactic categories*, i.e. sets of well formed expressions are represented by so-called *non-terminal symbols* and denoted by $\langle Name\rangle$, where $Name$ is the name of a category. Syntactic categories are defined over non-terminal and terminal symbols using rules of the form:

$$\langle S\rangle ::= E_1 \mid\mid E_2 \mid\mid \ldots \mid\mid E_k$$

meaning that $\langle S\rangle$ is to be the least set of words containing only terminal symbols and formed according to expression $E_1$ or $E_2$ or ... or $E_k$. Notation $\{E\}$ is used to indicate that expression $E$ can be repeated 0 or more times and $[E]$ - to denote that expression $E$ is optional.

**Example 1.4.1** Assume we want do define arithmetic expressions containing the variable $x$ and using addition $+$ and brackets ( and ). Terminal symbols are then $x, +, (, )$. We need only one non-terminal symbol representing well-formed expressions and denoted by $\langle Expr\rangle$. The following rule defines the syntactic category $\langle Expr\rangle$, i.e. the set of all well-formed expressions:

$$\langle Expr\rangle ::= x \mid\mid \langle Expr\rangle + \langle Expr\rangle\{+\langle Expr\rangle\} \mid\mid (\langle Expr\rangle).$$

Now, for instance, $(x + x + x) + x$ is a well-formed expression, but $x + \langle Expr\rangle$ is not, since $\langle Expr\rangle$ is not a terminal symbol. ■

# Chapter 2

# Propositional Calculus

## 2.1 Introduction

According to the methodology described in section 1.2 we first have to decide upon the language we deal with.

Propositional logic is used to investigate properties of complex sentences built from elementary sentences by using propositional connectives like negation, conjunction, disjunction, implication, etc. Whether complex sentences are true or not, depends solely on logical values of elementary sentences involved in them. For example, elementary sentence $p$ implies elementary sentence $q$ if and when $p$ is false or $q$ is true. Thus in classical propositional logic we need a language containing constants denoting truth values, a set of propositional variables that serve to represent elementary sentences, and the set of propositional connectives used to build complex sentences. The sentences (or formulas) of the logic are formed from elementary sentences by applying propositional connectives. The meaning associated with complex sentences is given by valuations of elementary sentences together with a method of calculating values of complex sentences from their components.

## 2.2 Syntax of Propositional Calculus

Let $V_0$ be a set of *propositional variables* (or *boolean variables*), i.e. variables representing truth values $\mathsf{True}, \mathsf{False}$ standing for true and false, respectively. The set $\{\mathsf{True}, \mathsf{False}\}$ is denoted by $\mathsf{Bool}$. We further assume that truth values are ordered, $\mathsf{False} \leq \mathsf{True}$, and use $min(\ldots)$ and $max(\ldots)$ to denote minimum and maximum of a given set of truth values.

We build *propositional formulas* (*sentences*) from truth values and propositional variables by applying propositional connectives $\neg, \wedge, \vee, \rightarrow, \equiv$, standing for *nega-*

*tion*, *conjunction*, *disjunction*, *implication* and *equivalence*, respectively. The set of propositional formulas is denoted by $F_0$. More formally, the syntax of propositional formulas is defined by the following rules:

$$\begin{array}{ll}\langle F_0\rangle ::= & \langle \mathsf{Bool}\rangle \mid\mid \langle V_0\rangle \mid\mid \neg\langle F_0\rangle \mid\mid \langle F_0\rangle \wedge \langle F_0\rangle \mid\mid \langle F_0\rangle \vee \langle F_0\rangle \mid\mid \\ & \langle F_0\rangle \rightarrow \langle F_0\rangle \mid\mid \langle F_0\rangle \equiv \langle F_0\rangle \mid\mid (\langle F_0\rangle) \mid\mid [\langle F_0\rangle]\end{array}$$

## 2.3 Semantics of Propositional Calculus

The semantics of propositional formulas is given by assigning truth values to propositional variables and then calculating values of formulas. Let

$$v : V_0 \longrightarrow \mathsf{Bool}$$

be such an assignment (called a *valuation* of propositional variables). Then $v$ is extended to define the *truth value* of propositional formulas as follows:

$$\begin{array}{rcl}
v(\neg\alpha) & = & \begin{cases} \mathsf{True} & \text{if } v(\alpha) = \mathsf{False} \\ \mathsf{False} & \text{otherwise} \end{cases} \\
v(\alpha \wedge \beta) & = & min(v(\alpha), v(\beta)) \\
v(\alpha \vee \beta) & = & max(v(\alpha), v(\beta)) \\
v(\alpha \rightarrow \beta) & = & \mathsf{True} \text{ if and only if } v(\alpha) \leq v(\beta) \\
v(\alpha \equiv \beta) & = & \mathsf{True} \text{ if and only if } v(\alpha) = v(\beta).
\end{array}$$

A propositional formula $\alpha$ is *satisfiable* if there is a valuation $v$ such that $v(\alpha) = \mathsf{True}$. It is a *tautology* if for all valuations $v$ we have $v(\alpha) = \mathsf{True}$.

By a *propositional literal* we understand a propositional variable or its negation. A literal is positive if it is a variable and is negative if it is the negation of a variable. A *propositional clause* is any disjunction of propositional literals. A *propositional Horn clause* is a clause with at most one positive literal. We say that a formula is in *conjunctive normal form*, CNF, if it is a conjunction of clauses. It is in *disjunctive normal form*, DNF, if it is a disjunction of conjunctions of literals. It is in *negation normal form*, NNF, if any negation occurs only in literals.

Observe that Horn clauses are often equivalently presented in the form of implication:

$$(p_1 \wedge \ldots \wedge p_n) \rightarrow q$$

or

$$(p_1 \wedge \ldots \wedge p_n) \rightarrow \mathsf{False}.$$

Any formula can be equivalently transformed into CNF, DNF and NNF. The transformation, into CNF or DNF may exponentially increase the size of the formula, while the transformation into NNF may increase or decrease the size of the formula by a constant factor.

**Algorithm 2.3.1** Input: Formula $\alpha \in F_0$.

Output: Formula $\beta \in F_0$ such that $\beta \equiv \alpha$ and $\beta$ is in NNF.

Algorithm: Move negation inside of the formula $\alpha$ using the following rules, until no rule is applicable:

| sub-formula | substitute by |
|---|---|
| $\neg$True | False |
| $\neg$False | True |
| $\neg\neg\gamma$ | $\gamma$ |
| $\neg(\alpha \vee \beta)$ | $(\neg\alpha) \wedge (\neg\beta)$ |
| $\neg(\alpha \wedge \beta)$ | $(\neg\alpha) \vee (\neg\beta)$ |
| $\neg(\alpha \to \beta)$ | $\alpha \wedge (\neg\beta)$ |
| $\neg(\alpha \equiv \beta)$ | $[\alpha \wedge (\neg\beta)] \vee [(\neg\alpha) \wedge \beta]$ ■ |

## 2.4 The Complexity of Propositional Calculus

**Theorem 2.4.1** [Cook] The problem of checking satisfiability of propositional formulas is NPTime-complete. Checking whether a formula is a propositional tautology is a co-NPTime-complete problem. ■

There are some special cases where the reasoning is in PTime. One of the most interesting cases is provided by the following theorem.

**Theorem 2.4.2** The problem of checking satisfiability of propositional Horn clauses is in PTime. ■

## 2.5 Exercises

1. Design an algorithm to transform a given propositional formula into an equivalent propositional formula in the CNF.

2. Using the algorithm provided as a solution to exercise 1, transform formula $(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (p_3 \wedge q_3)$ into the CNF.

3. Find a formula size of which grows exponentially after the transformation defined by algorithm designed in exercise 1.

4. Design an algorithm to transform a given propositional formula into an equivalent propositional formula in the DNF.

5. Using the algorithm provided as a solution to exercise 4, transform formula $(p_1 \vee q_1 \vee r_1) \wedge (p_2 \vee q_2 \vee r_2)$ into the DNF.

6. Design PTime algorithms for:

   - checking the satisfiability of a formula in DNF
   - checking the validity of a formula in CNF.

# Chapter 3

# Predicate Calculus

## 3.1 Introduction

Predicate calculus, known also as the *classical first-order logic* serves as a means to express properties of individuals and relationships between individuals (objects of a domain). It is an extension of propositional calculus and provides syntax that allows one to talk about relations and functions. Quantifiers "for all" and "exists" are also available. The quantifiers range over individuals[1].

Predicate calculus is a very powerful formalism[2]. It has a great variety of applications ranging from mathematics to computer science and artificial intelligence. It is very well developed. In particular there are many well-developed automated theorem proving methods. Thus is is widely used as a "kernel" of many other formalisms.

## 3.2 Syntax of Predicate Calculus

Let $V_{\mathsf{I}}$ be a set of *individual variables* representing values of some domain. In order to define the language and semantics of *predicate calculus* (or, in other words, *first-order logic*) we assume that we are given a set of function symbols $\mathsf{Fun} = \{f_i : i \in I\}$ and relation symbols $\mathsf{Rel} = \{R_j : j \in J\}$, where $I, J$ are some finite sets. Functions and relations may have arguments. The number of arguments is called the *arity* of the function or relation, respectively. Functions and relations of arity 0 are called *individual constants*, or *constants*, for short, and *boolean constants*. The set of individual constants is denoted by $\mathsf{Const}$.

[1] In the second-order logic there are also quantifiers ranging over sets or relations, in the third-order logic over sets of sets etc.

[2] In fact, it is the strongest logic satisfying some natural conditions, with partially computable set of tautologies.

Symbols of arity 1 are usually called *unary* and of arity 2 are called *binary*.[3] The set of function symbols and relation symbols together with their arities is called the *signature* or *vocabulary*. We sometimes write $f_i^{a_i}, R_j^{a_j}$ to indicate that function symbol $f_i$ has $a_i$ arguments and relation symbol $R_j$ has $a_j$ arguments. By a *signature of a function or a relation symbol* we understand the number of arguments together with their types and the type of the result.

Functional expressions in predicate calculus are represented by terms. We define the set of *terms*, denoted by $\mathsf{Terms}$, by the following rule:

$$\langle \mathsf{Terms} \rangle ::= \mathsf{Const} \mid\mid V_{\mathsf{I}} \mid\mid \langle \mathsf{Fun} \rangle([\langle \mathsf{Terms} \rangle]\{, \langle \mathsf{Terms} \rangle\})$$

Terms without variables are called *ground terms*.

Formulas of predicate calculus, denoted by $F_{\mathsf{I}}$, are now defined by means of the following rules.

$$\begin{aligned} \langle F_{\mathsf{I}} \rangle ::= \quad & \langle \mathsf{Bool} \rangle \mid\mid \langle \mathsf{Rel} \rangle([\langle \mathsf{Terms} \rangle]\{, \langle \mathsf{Terms} \rangle\}) \mid\mid \\ & \neg \langle F_{\mathsf{I}} \rangle \mid\mid \langle F_{\mathsf{I}} \rangle \wedge \langle F_{\mathsf{I}} \rangle \mid\mid \langle F_{\mathsf{I}} \rangle \vee \langle F_{\mathsf{I}} \rangle \mid\mid \langle F_{\mathsf{I}} \rangle \to \langle F_{\mathsf{I}} \rangle \mid\mid \langle F_{\mathsf{I}} \rangle \equiv \langle F_{\mathsf{I}} \rangle \mid\mid \\ & \forall \langle V_{\mathsf{I}} \rangle . \langle F_{\mathsf{I}} \rangle \mid\mid \exists \langle V_{\mathsf{I}} \rangle . \langle F_{\mathsf{I}} \rangle \mid\mid (\langle F_{\mathsf{I}} \rangle) \mid\mid [\langle F_{\mathsf{I}} \rangle] \end{aligned}$$

Formulas of the form $\langle \mathsf{Rel} \rangle([\langle \mathsf{Terms} \rangle]\{, \langle \mathsf{Terms} \rangle\})$ are called *atomic formulas*. Atomic formulas without variables are called *ground formulas*.

A first-order formula is called *open* if it does not contain quantifiers. A variable occurrence is *free* in a formula if it is not bound by a quantifier, otherwise it is called *bound*. A formula is called *closed* (or a *sentence*) if it does not contain free occurrences of variables. Any set of sentences is called a *first-order theory*, or *theory*, for short. A theory is understood as a conjunction of its sentences.

A formula is in the *prenex normal form*, $\mathsf{PNF}$, if all its quantifiers are in its prefix, i.e. it is of the form $Q_1 x_1 \ldots Q_k x_k . \alpha$, where $Q_1, \ldots, Q_k \in \{\forall, \exists\}$ and $\alpha$ is an open formula. Any formula can be equivalently transformed into $\mathsf{PNF}$. The transformation into $\mathsf{PNF}$ may increase or decrease the size of the formula by a constant factor.

By a *universal formula* we mean a formula in the prenex normal form, without existential quantifiers. A set of universal formulas is called a *universal theory*. By a *first-order literal* (or *literal*, for short) we understand an atomic formula or its negation. A *first-order clause* is any disjunction of first-order literals, preceded by a (possibly empty) prefix of universal quantifiers. A literal is positive if it is an atomic formula and is negative if it is the negation of an atomic formula. A relation symbol $R$ occurs *positively* (resp. *negatively*) in a formula $\alpha$ if it appears under an even (resp. odd) number of negations. A relation symbol $R$ is *similar to a formula* $\alpha$ iff the arity of $R$ is equal to the number of free

[3] Observe that in the case of binary relations or functions we often use traditional infix notation. For instance we write $x \leq y$ rather than $\leq (x, y)$.

variables of $\alpha$. A *first-order Horn clause*, or *Horn clause*, for short, is a clause with at most one positive literal.

*Semi-Horn formulas* are defined by the following syntax rules:

$$\begin{aligned}&\langle \textsf{semi-Horn Formula}\rangle ::= \\ &\quad \langle \textsf{Atomic Formula}\rangle \rightarrow \langle F_{\textsf{I}}\rangle \mid\mid \langle F_{\textsf{I}}\rangle \rightarrow \langle \textsf{Atomic Formula}\rangle\end{aligned}$$

where the formula $\langle F_{\textsf{I}}\rangle$ is an arbitrary classical first-order formula positive w.r.t. $\langle \textsf{Atomic Formula}\rangle$, and the only terms allowed in the atomic formula are variables. The atomic formula is called the *head* of the formula and the first-order formula is called the *body* of the formula. Semi-Horn formulas are assumed to be implicitly universally quantified, i.e. any free variable is bound by an implicit universal quantifier. *Semi-Horn rules* (or *rules*, for short), are semi-Horn formulas in which the only terms are constant and variable symbols.

If the head of a rule contains a relation symbol $R$, we call the rule semi-Horn w.r.t. $R$. If the body of a rule does not contain the relation symbol appearing in its head, the rule is called *nonrecursive.* A conjunction of rules (nonrecursive) w.r.t. a relation symbol $R$ is called a (*nonrecursive*) *semi-Horn theory w.r.t. R*.

## 3.3 Semantics of Predicate Calculus

The semantics of first-order formulas is given by a valuation of individual variables together with an interpretation of function symbols and relation symbols as functions and relations. The interpretation of function symbols and relation symbols is defined by *relational structures* of the form

$$\langle \textsf{Dom}, \{f_i^{\textsf{Dom}} : i \in I\}, \{R_j^{\textsf{Dom}} : j \in J\}\rangle,$$

where:

- $\textsf{Dom}$ is a non-empty set, called the *domain* or *universe* of the relational structure
- for $i \in I$, $f_i^{\textsf{Dom}}$ denote functions corresponding to function symbols $f_i$
- for $j \in J$, $R_j^{\textsf{Dom}}$ denote relations corresponding to relation symbols $R_j$.

For the sake of simplicity, in the rest of the book we often abbreviate $f_i^{\textsf{Dom}}$ and $R_j^{\textsf{Dom}}$ by $f_i$ and $R_j$, respectively.

For a given signature $Sig$, by $\textsf{STRUC}[Sig]$ we denote the class of all relational structures built over the signature $Sig$.

Let $v : V_{\textsf{I}} \longrightarrow \textsf{Dom}$ be a valuation of individual variables. By $v_a^x$ we shall denote the valuation obtained from $v$ by assigning value $a$ to variable $x$ and leaving all other variables unchanged, i.e.:

$$v_a^x(z) = \begin{cases} a & \text{if } z = x \\ v(z) & \text{otherwise} \end{cases}$$

The valuation $v$ is extended to provide values of terms as follows, where $f \in \mathsf{Fun}$ is a $k$-argument function symbol and $t_1, \ldots, t_k \in \mathsf{Terms}$:

$$v(f(t_1, \ldots, t_k)) = f^{\mathsf{Dom}}(v(t_1), \ldots, v(t_k)).$$

Then $v$ is extended to define the *truth value* of first-order formulas as follows, where $R \in \mathsf{Rel}$ is a $k$-argument relation:

$$\begin{array}{rcl}
v(R(t_1, \ldots, t_k)) & = & R^{\mathsf{Dom}}(v(t_1), \ldots, v(t_k)) \\
v(\neg\alpha) & = & \begin{cases} \mathsf{True} & \text{if } v(\alpha) = \mathsf{False} \\ \mathsf{False} & \text{otherwise} \end{cases} \\
v(\alpha \wedge \beta) & = & min(v(\alpha), v(\beta)) \\
v(\alpha \vee \beta) & = & max(v(\alpha), v(\beta)) \\
v(\alpha \rightarrow \beta) & = & \mathsf{True} \text{ if and only if } v(\alpha) \leq v(\beta) \\
v(\alpha \equiv \beta) & = & \mathsf{True} \text{ if and only if } v(\alpha) = v(\beta) \\
v(\forall x.\alpha(x)) & = & min(\{v_a^x(\alpha(x)) : a \in \mathsf{Dom}\}) \\
v(\exists x.\alpha(x)) & = & max(\{v_a^x(\alpha(x)) : a \in \mathsf{Dom}\}).
\end{array}$$

A first-order formula $\alpha$ is *satisfiable* if there is a relational structure

$$M = \langle \mathsf{Dom}, \{f_i^{\mathsf{Dom}} : i \in I\}, \{R_j^{\mathsf{Dom}} : j \in J\} \rangle$$

and a valuation $v : V_{\mathsf{I}} \longrightarrow \mathsf{Dom}$ such that its extension to $F_{\mathsf{I}}$ satisfies $v(\alpha) = \mathsf{True}$. Formula $\alpha$ is *valid* in a relational structure $M$ if for all valuations $v : V_{\mathsf{I}} \longrightarrow \mathsf{Dom}$, $v(\alpha) = \mathsf{True}$. In such a case we also say that $M$ is a *model* for $\alpha$. Formula $\alpha$ is a *tautology* if for all relational structures of suitable signature and all valuations $v$ we have $v(\alpha) = \mathsf{True}$. For a set of formulas $F \subseteq F_{\mathsf{I}}$ and formula $\alpha \in F_{\mathsf{I}}$, by *entailment* (or a *consequence relation*)[4], denoted by $F \models \alpha$, we mean that $\alpha$ is satisfied in any relational structure which is a model of all formulas of $F$. By $Cn(F)$ we mean the set of all semantical consequences of $F$, i.e.

$$Cn(F) = \{\alpha \in F_{\mathsf{I}} : F \models \alpha\}.$$

It is important to note that in many applications one deals with so-called *many-sorted* interpretations. In such a case one deals with elements of different types, called *sorts*, i.e. the underlying domain $\mathsf{Dom}$ is a union of disjoint

[4] We always assume that $G \models \beta$ means that formula $\beta$ is valid in all models of the set of formulas $G$, where formulas, validity and the notion of models depend on a given logic.

sets, $\mathsf{Dom} = \mathsf{Dom}_1 \cup \ldots \cup \mathsf{Dom}_n$ and arguments and results of function and relation symbols are typed. For instance, one can consider two-sorted relational structure $Stacks = \langle E \cup S, push, pop, top\rangle$, where $E$ is a set of elements and $S$ is a set of stacks. Now, for example *top* takes as argument a stack (from $S$) and returns as its result an element of $E$. The definitions provided previously can easily be adapted to deal with this case.

## 3.4 The Complexity of Predicate Calculus

Using predicate calculus as a practical reasoning tool is somewhat questionable, because of the complexity of the logic. Existing first-order theorem provers solve the reasoning problem partially and exploit the fact that checking whether a first-order formula is a tautology is partially computable.

The following theorem quotes the most important facts on the complexity of general first-order reasoning.

**Theorem 3.4.1**

1. The problem of checking whether a given first-order formula is a tautology is uncomputable [Church] but is partially computable [Gödel].
2. The problem of checking whether a given first-order formula is satisfiable, is not partially computable [Church]. ■

Fortunately, when fixing a finite domain relational structure, one ends up in a tractable situation, as stated in the following theorem.

**Theorem 3.4.2** Checking the satisfiability and validity of a given fixed first-order formula in a given finite domain relational structure is in PTime and LogSpace w.r.t. the size of the domain. ■

Observe that the above theorem holds only when a finite structure is fixed. If one would like to investigate properties of first-order formulas valid in all finite domain structures, one would end up in quite complex situation, as we have the following theorem.

**Theorem 3.4.3** [Trakhtenbrot]

1. The problem of checking whether a first-order formula is valid in all finite domain structures is not partially computable.
2. The problem of checking whether a first-order formula is satisfied in a finite domain structure is not computable but is partially computable. ■

## 3.5 Unification of terms

One of the substantial tools for many methods of automated deduction is that of the unification of terms. The unification means that given two terms, we are looking for a substitution of variables such that the terms become identical.

**Definition 3.5.1** By a *substitution* we understand any mapping from variables to terms. We say that a substitution $\sigma$ is a *unifier* for the set of terms $T = \{t_1, t_2, \ldots, t_k\}$ iff $\sigma(t_1) = \sigma(t_2) = \ldots = \sigma(t_k)$. We say that $\sigma$ is a *most general unifier* of $T$ iff it is a unifier and for any other unifier $\sigma'$ of $T$ there is a substitution $\sigma''$ such that $\sigma' = \sigma \circ \sigma''$. ■

**Example 3.5.2** Substitution $\sigma = [x := f(y)]$ unifies terms $g(f(y), f(x))$ and $g(x, f(f(y)))$. In fact, it is the most general unifier for the two terms.

Substitution $\sigma' = [x := f(h(z)); y := h(z)]$ is also a unifier of the terms, but it is not the most general unifier. In fact, $\sigma' = \sigma \circ [y := h(z)]$.

On the other hand, terms $f(x)$ and $g(x)$ are not unifiable as no substitution of variables can make them identical. ■

**Proposition 3.5.3** If $\sigma$ and $\sigma'$ are most general unifiers for terms $t_1, \ldots, t_n$ then there there is a renaming of variables $\rho$ such that $\sigma = \sigma' \circ \rho$. ■

Let us now formulate the algorithm that unifies two terms or answers that such a unification is not possible.

**Definition 3.5.4** Let $t_1, t_2$ be two terms. By a *disagreement* of $t_1$ and $t_2$ we understand a pair $t_1', t_2'$ of sub-terms of $t_1$ and $t_2$, respectively, such that:

- first symbols of $t_1'$ and $t_2'$ are different
- terms $t_1$ and $t_2$ are identical up to the occurrence of $t_1$ and $t_2$, respectively (counting from left to right). ■

**Example 3.5.5** The disagreement for terms $f(x, g(y, h(z))$ and $f(x, g(h(u), y))$ is the pair $y$, and $h(u)$. ■

**Algorithm 3.5.6** [A unification algorithm]

Input: terms $t_1, t_2$.

Output: the most general unifier $\sigma$ of $t_1, t_2$ if it exists, otherwise inform, that it does not exist.

```
Set $\sigma$ to be the identity;
while $\sigma(t_1) \neq \sigma(t_2)$ do steps 1-5:
```

1. let $t_1'$ and $t_2'$ be a disagreement for terms $\sigma(t_1)$ and $\sigma(t_2)$
2. if none of terms $t_1'$ and $t_2'$ is a variable then answer that the most general unifier does not exist and stop
3. if one of the terms $t_1'$ and $t_2'$ is a variable, say $x$, then denote the other term by $t$ (if both terms are variables then the choice is arbitrary)
4. if $x$ occurs in $t$ then answer that the most general unifier does not exist and stop
5. set $\sigma := \sigma \circ [x := t]$. ■

## 3.6 Skolemization

*Skolemization* is a technique for eliminating existential first-order quantifiers.

Assume we are given a formula $\alpha$ in the PNF form. In order to obtain a *Skolem form* of $\alpha$ we eliminate all existential quantifiers from left to right as follows:

- if $\alpha$ is of the form $\exists x.\beta(x, \bar{y})$ then remove the quantifier $\exists x$ and replace all occurrences of $x$ in $\alpha$ by a new constant symbol (the new symbol is called a *Skolem constant*)
- if $\alpha$ is of the form $\forall \bar{z}.\exists x.\bar{Q}.\beta(\bar{z}, x, \bar{y})$ then remove the quantifier $\exists x$ and replace all occurrences of $x$ in $\alpha$ by term $f(\bar{z})$, where $f$ is a new function symbol ($f$ is called a *Skolem function*).

**Example 3.6.1** Consider formula $\exists x.\forall y.\exists z.\forall t.\exists u.R(x, y, z, t, u)$. The Skolem form of the formula is $\forall y.\forall t.R(a, y, f(y), t, g(y, t))$. ■

**Proposition 3.6.2** Skolemization preserves the satisfiability of formulas, i.e., a formula is satisfiable iff its Skolem form is satisfiable. ■

## 3.7 Exercises

1. Show semantically that the following formulas are tautologies of predicate calculus, where $Q$ stands for a quantifier $\forall$ or $\exists$, and formula $\gamma$ does not contain variable $x$:

   (a) $\exists x.(\alpha(x) \vee \beta(x)) \ \equiv\ \exists x.\alpha(x) \vee \exists x.\beta(x)$

   (b) $\forall x.(\alpha(x) \wedge \beta(x)) \ \equiv\ \forall x.\alpha(x) \wedge \forall x.\beta(x)$

   (c) $Qx.(\alpha(x)) \wedge \gamma \ \equiv\ Qx.(\alpha(x) \wedge \gamma)$

   (d) $\gamma \wedge Qx.(\alpha(x)) \ \equiv\ Qx.(\gamma \wedge \alpha(x))$

(e) $Qx.(\alpha(x)) \vee \gamma \;\equiv\; Qx.(\alpha(x) \vee \gamma)$

(f) $\gamma \vee Qx.(\alpha(x)) \;\equiv\; Qx.(\gamma \vee \alpha(x))$

(g) $\alpha(\bar{t}) \;\equiv\; \forall\bar{x}(\alpha(\bar{t} \leftarrow \bar{x}) \vee \bar{x} \neq \bar{t})$

(h) $\alpha(\bar{t}_1) \vee \cdots \vee \alpha(\bar{t}_n) \;\equiv\; \exists\bar{x}.(\bar{x} = \bar{t}_1 \vee \cdots \vee \bar{x} = \bar{t}_n) \wedge \alpha(\bar{t}_1 := \bar{x}))$.

2. Design an algorithm to transform a given predicate formula into an equivalent predicate formula in the PNF.

3. Using the algorithm provided as a solution to exercise 2, transform formula $\forall x.[\exists y.(R(x) \vee P(y) \vee \forall x.S(x, y))] \wedge \exists x.R(x)$ into the PNF.

4. Design algorithms that prove Theorem 3.4.2.

5. Generalize Algorithm 3.5.1 to an arbitrary number of terms.

6. Transform the formula $\exists x.\forall y.\exists z.(P(x, y) \vee Q(z))$ into the Skolem form.

7. Show that Skolemization does not preserve the validity of formulas.

# Chapter 4

# Applications of Predicate Calculus

## 4.1 Specifying Data Structures

Data structures play an important rôle in programming. Formal specification and analysis of abstract data types occupies an important place in current research on the description of semantics of programming languages and on the process of program design and verification. In this section we shall show how to specify data structures by means of predicate calculus.

Data structure consists of objects (elements) and operations performed on the objects. Let us start with a simple example.

**Example 4.1.1** Consider a data structure representing stacks. In this case we deal with two types of objects, $E$ and $S$, representing elements and stacks, respectively. The signatures of typical operations on stacks are defined as follows[1]:

$$
\begin{aligned}
&\mathit{top} : S \longrightarrow E\\
&\mathit{pop} : S \longrightarrow S\\
&\mathit{push} : E \times S \longrightarrow S\\
&\mathit{empty} : \emptyset \longrightarrow S\\
&\mathit{err} : \emptyset \longrightarrow E
\end{aligned}
$$

Now we are ready to specify the properties of operations.

$$\forall e.\forall s.[\mathit{push}(e, s) \neq \mathit{empty}] \tag{4.1}$$

[1] Observe that *empty* and *err* have no arguments and thus are constant symbols. Their intended meaning is to represent the empty stack and an error, respectively.

$$top(empty) = err \tag{4.2}$$
$$\forall e.\forall s.[e = top(push(e, s))] \tag{4.3}$$
$$pop(empty) = empty \tag{4.4}$$
$$\forall e.\forall s.[s = pop(push(e, s))] \tag{4.5}$$
$$\forall s.[s \neq empty \rightarrow s = push(top(s), pop(s))] \tag{4.6}$$
$$\forall s.\forall s'.[s = s' \equiv (top(s) = top(s') \wedge pop(s) = pop(s'))].\blacksquare \tag{4.7}$$

Of course, one would like to know what is the quality of specification provided in Example 4.1. Are there any missing important details? The specification might have many models. What then should be accepted as its semantics?

Let us first start with the question of semantics. There are many approaches, but I like best the one originating from the algorithmic logic (see section 11.3). Here, we assume that stacks are built only by applying the available operations. In the context of programming, such an assumption is well justified. We just do not allow unusual stacks appearing in the model "from nowhere". Thus we assume that any stack is obtained from the empty stack by iterating *push*, i.e. any stack $s$ is assumed to be definable by a term of the following form:

$$s = push(e_1, push(e_2, push(\ldots, push(e_n, empty)))), \tag{4.8}$$

where $e_1, \ldots, e_n$ are elements placed on the stack, and in the case when $n = 0$, $s$ is defined to be *empty*. Unfortunately, the property (4.8) is not expressible by means of the predicate calculus[2].

Let us now fix the set $E$ of elements. We shall show that all models in which any stack is definable as in formula (4.8) are isomorphic, i.e. that these models have logically the same properties and the implementation may only differ in such details as representation of elements and stacks. Logical properties of stacks as well as properties of programs using stacks remain the same, provided that the implementation satisfies properties (4.1)-(4.7) together with the meta-property (4.8).

**Definition 4.1.2** Relational structures $S_1$ and $S_2$ are *similar* if they consist of the same number of sorts, functions and relations and the signatures of corresponding functions and relations are the same. By a *homomorphism* between $S_1$ and $S_2$ we shall mean a mapping $h : S_1 \longrightarrow S_2$ such that:

1. for any function $f_i^1$ of $S_1$ and corresponding function $f_i^2$ of $S_2$

$$h(f_i^1(a_1, \ldots, a_n)) = f_i^2(h(a_1), \ldots, h(a_n))$$

2. for any relation $r_i^1$ of $S_1$ and corresponding function $r_i^2$ of $S_2$

$$r_i^1(a_1, \ldots, a_n)) \text{ iff } r_i^2(h(a_1), \ldots, h(a_n)).$$

[2] It is expressible in stronger logics, like algorithmic or dynamic logic.

By an *isomorphism* we understand a homomorphism which is one-to-one and onto. ■

**Example 4.1.3** [Example 4.1 continued] Assume that the set of elements $E$ is fixed and (4.1)-(4.7) together with the meta-property (4.8) hold. We shall show that all models of the specification are isomorphic. Let $S$ and $T$ be two models of (4.1)-(4.7). We define a mapping $i : S \longrightarrow T$ as follows:

- $i(e) \stackrel{\text{def}}{=} e$ for $e \in E$
- $i(push(e_1, push^S(e_2, push^S(\ldots, push^S(e_n, empty^S))))) \stackrel{\text{def}}{=}$ $push^T(e_1, push^T(e_2, push^T(\ldots, push(e_n, empty^T))))$, where the superscripts indicate the appropriate structure.

By (4.8), $i$ is defined on all elements and stacks and $i$ is onto. Using property (4.7) one can show that $i$ is a one-to-one mapping. Let us now show that all operations on stacks are preserved under $i$.

1. By definition of $i$, $i(empty^S) = empty^T$.
2. Consider $i(top^S(s))$. By (4.8), $s = empty^S$ or $s = push^S(e, t)$ for some $e$ and $t$. In the first case, by definition of $i$ and property (4.2), $i(top^S(s)) = i(empty^S) = err^S = err^T = top^T(empty^T) = top^T(i(empty^S))$. In the second case, by (4.3), (4.7) and definition of $i$,
$i(top^S(s)) = i(top^S(push^S(e, t))) = e = top^T(push^T(e, t)) = top^T(i(s))$.
3. The case of $i(pop(s))$ is similar to that of $i(top(s))$. Here properties (4.4), (4.5) and (4.7) are to be applied.
4. Consider $i(push^S(e, s))$. Here the proof is carried out by structural induction on terms representing stacks. Consider first the case when $s = empty$. By definition of $i$, $i(push^S(e, empty^S)) = push^T(e, empty^T) = push^T(i(e), i(empty^S))$. In order to prove the induction step consider a non-empty stack $s$. By (4.6) we have that $s = push^S(top^S(s), pop^S(s))$. Now $push^S(e, t) = s$, for some $s$. Thus
$i(push^S(e, t)) = i(s) = i(push^S(top^S(s), pop^S(s))$.
By the definition of $i$ and the fact that $i$ preserves operations *top* and *pop*,
$$i(push^S(top^S(s), pop^S(s)) = push^T(i(top^S(s)), i(pop^S(s))) = \\ push^T(top^T(s), pop^T(s)).$$ ■

## 4.2 Predicate Calculus as a Programming Language

Predicate logic is a successful application as a tool to express programs. In fact, declarative programming, in particular the *logic programming paradigm* uses predicate calculus to express programs. PROLOG serves as the most popular programming language based on the paradigm.

The man idea is to define computation by means of Horn clauses. In logic programming, one deals with clauses written in the following form:

$$R(\bar{x}) \leftarrow R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k)$$

with intended meaning that the conjunction of $R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k)$ implies $R(\bar{x})$. A logic program is simply a set of clauses, interpreted as the conjunction of clauses. The semantics is given by the minimal model of all relations occurring in a given program.

The underlying computation mechanism heavily depends on resolution and unification, which are discussed more deeply in further parts of the notes.

**Example 4.2.1** Consider the well-known problem of Hanoi towers. One has three towers and $n$ rings, each of different size. Rings can be put only on rings of a greater size. Initially all rings are on the first tower. Move all the rings onto the second tower, assuming that the third tower might be used, too. Consider the following logic program, where the intended meaning of $H(n, A, B, C)$ is "move $n$ rings from $A$ to $B$, using $C$, and the meaning of relation $Move(X, Y)$ is that the upper ring of tower $X$ is moved into the tower $Y$:

$$\begin{array}{l} H(n, A, B, C) \leftarrow n = 0 \\ H(n, A, B, C) \leftarrow H(n-1, A, C, B), Move(A, B), H(n-1, C, B, A). \end{array}$$

The above two simple clauses solve the problem. ■

## 4.3 Predicate Calculus as a Query Language

Predicate calculus appears as powerful as SQL when querying databases. In fact one can use function-free fragment of predicate calculus as a query language. Any formula of predicate calculus defines a new relation of arity equal to the number of free variables occurring in the formula. Since in the case of databases, one deals with finite structures, the complexity results quoted as Theorem 3.4.2 apply here, too.

Let us explain the idea using the following example.

**Example 4.3.1** Consider a database containing the following relations:

- $M(x), W(x)$ stating that person $x$ is a man or a woman, respectively
- $MC(x, y)$ stating that $x$ and $y$ are a married couple.

$M(x) \wedge \exists y.[MC(x,y) \vee MC(y,x)]$ expresses that $x$ is a husband. Observe that $x$ is the only free variable in the query. Thus one can consider the query as a definition of a new relation, say $H(x)$, defined as

$$H(x) \equiv [M(x) \wedge \exists y.[MC(x,y) \vee MC(y,x)]].$$

The query $MC(x,y) \vee MC(y,x)$ defines a binary relation of all married couples. One can introduce a new relation symbol to denote this relation, e.g. $Married(x,y) \equiv MC(x,y) \vee MC(y,x)$ and use it in other queries. This does not increase the expressiveness of the language, but makes querying more convenient.

The query $\exists x.H(x)$ has no free variables. It defines a "yes or no" query, returning a (zero-argument) boolean constant True or False and checking whether in the database there is a person who is a husband.

Another example of a "yes or no" query could be

$$\forall x.[W(x) \rightarrow \exists y.Married(x,y)].$$

The query checks whether all women in the database are married.

The query $M(x) \wedge \forall y.[\neg Married(x,y)]$ selects all unmarried men. ■

## 4.4 Exercises

1. Provide a specification of lists in the predicate calculus.
2. Provide a specification of trees in the predicate calculus.
3. Provide a specification of graphs in the predicate calculus.
4. Assume that:
   - $List(x,h,t)$ means that $h$ is the head and $t$ is the tail of list $x$
   - $Empty(x)$ means that list $x$ is empty.

   Specify the following relations within the logic programming paradigm:

   (a) $C(e,x)$ meaning that list $x$ contains element $e$

   (b) $Conc(x,y,z)$ meaning that list $z$ results from concatenating lists $x$ and $y$

   (c) $Shr(x,y)$ meaning that list $x$ is shorter than list $y$.
5. Consider a library database containing the following relations:
   - $T(x,y)$ meaning that book number $x$ has title $y$
   - $W(x,y)$ meaning that book number $x$ has been written by writer $y$

- $R(x, y)$ meaning that book number $x$ is borrowed by reader $y$.

Construct the following queries in the language of predicate calculus:

(a) select titles and writers of all books borrowed by reader $r$

(b) select titles of all books written by writer $w$

(c) select all books borrowed by readers of the library.

# Part II

# Automated Deduction in Classical Logic

# Chapter 5

# Automated Deduction in Propositional Calculus

## 5.1 Introduction

There are many techniques for proving satisfiability/validity of propositional formulas. Here we shall concentrate on purely logical calculus.

## 5.2 Resolution Method

The resolution method was introduced by Robinson in 1965 in the context of predicate calculus (see section 6.2 for presentation of the method in this more general context). It is one of the most frequently used technique for proving the validity of formulas.

The resolution method applies to clauses. Here the *reductio ad absurdum*, i.e., a *reduction to absurdity* argument is applied. This means that in order to prove that a formula is valid one first negates it, and shows that the negated formula is inconsistent. In the context of resolution, one has to transform the negated formula into a conjunction of clauses (CNF) and proves that the obtained set of clauses is inconsistent. Inconsistency is represented by the *empty clause*, denoted here by $\nabla$. Observe that the empty clause is the empty disjunction which, by definition, is False.

The resolution method is based on applying the following two rules:

$$p_1 \vee \ldots \vee p_k \vee r, \quad \neg r \vee q_1 \vee \ldots \vee q_m \vdash p_1 \vee \ldots \vee p_k \vee q_1 \vee \ldots \vee q_m \tag{5.1}$$

$$p_1 \vee \ldots \vee p_k \vee p_k \vdash p_1 \vee \ldots \vee p_k, \tag{5.2}$$

where $p_1, \ldots, p_k, q_1, \ldots, q_m$ are literals and $r$ is a propositional variable.

Rule (5.1) is called the *resolution rule* and rule (5.2) is called the *factoring rule.*

Note that the resolution rule can be presented equivalently in the following form:

$$\begin{aligned}&\neg(p_1 \vee \ldots \vee p_k) \rightarrow r, \quad r \rightarrow (q_1 \vee \ldots \vee q_m) \vdash \\ &\quad \neg(p_1 \vee \ldots \vee p_k) \rightarrow (q_1 \vee \ldots \vee q_m).\end{aligned}$$

Thus, in the case of propositional calculus, it reflects the transitivity of implication.

It is important to note that the resolution rule does not preserve validity of formulas. However it preserves their satisfiability, which suffices for the method, as it shows inconsistency, i.e., unsatisfiability of sets of clauses.

The following theorems state that the resolution method is sound and complete.

**Theorem 5.2.1** For any set of propositional clauses $S$,

1. if $S \vdash \nabla$ then $S$ is inconsistent (soundness)
2. if $S$ is inconsistent then $S \vdash \nabla$ (completeness),

where $\vdash$ denotes the proof system consisting of rules (5.1) and (5.2). ■

The following examples show some applications of the method.

**Example 5.2.2** Consider clauses $p \vee p$ and $\neg p \vee \neg p$. Obviously, the clauses are inconsistent. The following proof shows that $\nabla$ can be obtained by applying the resolution method[1].

$$\dfrac{\dfrac{p \vee p}{p}(5.2) \quad \dfrac{\neg p \vee \neg p}{\neg p}(5.2)}{\nabla}(5.1) \quad \blacksquare$$

**Example 5.2.3** Let us prove that formula $[p \rightarrow (q \vee r)] \rightarrow [(p \wedge \neg q) \rightarrow r]$ is a propositional tautology. We first negate the formula and obtain $[p \rightarrow (q \vee r)] \wedge (p \wedge \neg q) \wedge \neg r]$. The negated formula is represented by four clauses: $(\neg p \vee q \vee r)$, $p$, $\neg q$, $\neg r$. Let us now prove that $\nabla$ follows from the set of clauses:

$$\dfrac{\dfrac{\dfrac{\neg p \vee q \vee r, \quad p}{q \vee r}(5.1) \quad \neg q}{r}(5.1) \quad \neg r}{\nabla}(5.1) \quad \blacksquare$$

[1] Observe that the factoring rule is substantial here. The resolution rule itself does not lead to the result.

**Example 5.2.4** The set of clauses $p \vee q$, $\neg q \vee s$ is consistent. Consequently, one cannot derive $\nabla$ from the set, as the only possible application of the resolution rule results in clause $p \vee s$ and then no new clauses can further be obtained. ■

## 5.3 Sequent Calculus

Sequent calculus were introduced by Gentzen in 1934 and are also known as Gentzen-like calculus and sometimes as a *natural deduction.* In fact, a similar method was independently formulated by Jaśkowski, also in 1934. In sequent calculus one deals with sets of formulas rather that with single formulas. Proofs are driven by syntax of formulas which simplifies the process of proving validity.

**Definition 5.3.1** By a *sequent* we shall understand any expression of the form

$$\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m,$$

where $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m$ are propositional formulas. ■

A sequent $\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m$ represents a single propositional formula

$$(\alpha_1 \wedge \ldots \wedge \alpha_n) \rightarrow (\beta_1 \vee \ldots \vee \beta_m).$$

We assume that the order of formulas in sequents is inessential.

Let us now define sequent calculus for propositional logic.

**Definition 5.3.2** By an *axiom of sequent calculus* we shall understand any sequent of the form $\alpha_1, \ldots, \alpha_n, \gamma \Rightarrow \beta_1, \ldots, \beta_m, \gamma$. The *rules of sequent calculus for propositional connectives* are of the following form:

$$(\neg l)\frac{\neg\alpha, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m}{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha} \qquad (\neg r)\frac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \neg\alpha}{\alpha, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m}$$

$$(\wedge l)\frac{\alpha \wedge \beta, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m}{\alpha, \beta, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m} \qquad (\vee r)\frac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha \vee \beta}{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha, \beta}$$

$$(\wedge r)\frac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha \wedge \beta}{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha; \quad \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \beta}$$

$$(\vee l)\frac{\alpha \vee \beta, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m}{\alpha, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m; \quad \beta, \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m} \blacksquare$$

The rules for other boolean connectives can easily be derived.

It is important to observe that the rules preserve validity of formulas, assuming that semicolons in consequents of rules $(\wedge r)$ and $(\vee l)$ are interpreted as a conjunction.

We have the following theorem stating soundness and completeness of sequent calculus.

**Theorem 5.3.3** Let $\alpha$ be a propositional formula. Then:

- if the sequent $\emptyset \Rightarrow \alpha$ is provable then $\alpha$ is a tautology of propositional calculus (soundness)
- if $\alpha$ is a tautology of propositional calculus then the sequent $\emptyset \Rightarrow \alpha$ is provable (completeness),

where provability refers to sequent calculus defined in Definition 5.3.2. ■

Consider now some examples of applications of the sequent calculus.

**Example 5.3.4** Let us prove that $\neg[(p \wedge q) \vee s] \vee (p \vee s)$ is a tautology. We start the proof with the sequent $\emptyset \Rightarrow \neg[(p \wedge q) \vee s] \vee (p \vee s)$:

$$\dfrac{\emptyset \Rightarrow \neg[(p \wedge q) \vee s] \vee (p \vee s)}{\dfrac{\emptyset \Rightarrow \neg[(p \wedge q) \vee s], (p \vee s)}{\dfrac{(p \wedge q) \vee s \Rightarrow (p \vee s)}{\dfrac{(p \wedge q) \vee s \Rightarrow p, s}{\dfrac{p \wedge q \Rightarrow p, s}{p, q \Rightarrow p, s}(\wedge l); \quad \dfrac{s \Rightarrow p, s}{}}(\vee l)}(\vee r)}(\neg r)}(\vee r)$$

The bottom sequents are axioms, thus the initial formula is a propositional tautology. ■

The following example shows how to derive rules for other propositional connectives.

**Example 5.3.5** Let us derive rules for implication. Assume we are given a sequent of the form $\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha \rightarrow \beta$. We use the fact that $(\alpha \rightarrow \beta) \equiv (\neg\alpha \vee \beta)$. Consider the following derivation:

$$\dfrac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha \rightarrow \beta}{\dfrac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \neg\alpha \vee \beta}{\dfrac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \neg\alpha, \beta}{\alpha_1, \ldots, \alpha_n, \alpha \Rightarrow \beta_1, \ldots, \beta_m, \beta}(\neg r)}(\vee r)}$$

Thus the rule for implication on the righthand side of sequents can look as follows:

$$(\rightarrow r)\dfrac{\alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m, \alpha \rightarrow \beta}{\alpha_1, \ldots, \alpha_n, \alpha \Rightarrow \beta_1, \ldots, \beta_m, \beta}$$

Assume we are now given a sequent of the form $\alpha_1,\ldots,\alpha_n,\alpha\to\beta\Rightarrow\beta_1,\ldots,\beta_m$. Consider the following derivation:

$$\dfrac{\dfrac{\alpha_1,\ldots,\alpha_n,\alpha\to\beta\Rightarrow\beta_1,\ldots,\beta_m}{\alpha_1,\ldots,\alpha_n,\neg\alpha\vee\beta\Rightarrow\beta_1,\ldots,\beta_m}}{\dfrac{\alpha_1,\ldots,\alpha_n,\neg\alpha\Rightarrow\beta_1,\ldots,\beta_m}{\alpha_1,\ldots,\alpha_n,\Rightarrow\beta_1,\ldots,\beta_m,\alpha}(\neg l);\quad\dfrac{\alpha_1,\ldots,\alpha_n,\beta\Rightarrow\beta_1,\ldots,\beta_m}{}}(\vee l)$$

Thus the rule for implication on the lefthand side of sequents can look as follows:

$$(\to l)\dfrac{\alpha_1,\ldots,\alpha_n,\alpha\to\beta\Rightarrow\beta_1,\ldots,\beta_m}{\alpha_1,\ldots,\alpha_n\Rightarrow\beta_1,\ldots,\beta_m,\alpha;\quad\alpha_1,\ldots,\alpha_n,\beta\Rightarrow\beta_1,\ldots,\beta_m}\quad\blacksquare$$

Let us now show a derivation tree for a formula which is not a tautology and show how can one construct a valuation of propositional variables falsifying the formula.

**Example 5.3.6** Consider formula $(p\vee q)\to p$. In order to prove it, we start with the sequent $\emptyset\Rightarrow(p\vee q)\to p$. The derivation tree looks as follows[2]:

$$\dfrac{\emptyset\Rightarrow(p\vee q)\to p}{\dfrac{p\vee q\Rightarrow p}{p\Rightarrow p;\quad q\Rightarrow p}(\vee l)}(\to r)$$

The first bottom sequent is an axiom, but the second one is not. One can falsify it by assuming that $q$ is $\mathsf{True}$ and $p$ is $\mathsf{False}$. Observe that this valuation falsifies the input formula.

In general, such a construction gives a partial valuation which can further be extended to a total valuation falsifying the formula. One uses here fact that validity (thus invalidity, too) is preserved by the rules of sequent calculus. ■

## 5.4 Analytic Tableaux

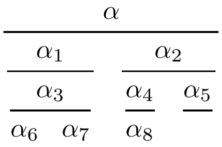
Analytic tableaux were invented by Beth in 1955 (under the name of *semantic tableaux*) and then simplified by Smullyan in 1968. Analytic tableaux, like resolution, apply the reduction to absurdity paradigm.

The name *tableaux* comes from the representation of proof trees in the form of tables. In these notes, however, we shall use tree-like rather than the tabular presentation.

The following example provides a hint how do the tables look like.

[2] We use here the rule $(\to r)$ derived in Example 5.3.5.

**Example 5.4.1** Consider the following proof tree:



It can be represented by the following table:

<table>
<tr><td colspan="4">$\alpha$</td></tr>
<tr><td colspan="2">$\alpha_1$</td><td colspan="2">$\alpha_2$</td></tr>
<tr><td colspan="2">$\alpha_3$</td><td>$\alpha_4$</td><td rowspan="2">$\alpha_5$</td></tr>
<tr><td>$\alpha_6$</td><td>$\alpha_7$</td><td>$\alpha_8$</td></tr>
</table>

■

Any tableau represents a formula as follows:

- each branch of tableau represents the conjunction of formulas appearing on this branch
- the tableau represents the disjunction of all its branches.

The tableau of Example 5.4.1 represents the following formula:

$$(\alpha \wedge \alpha_1 \wedge \alpha_3 \wedge \alpha_6) \vee (\alpha \wedge \alpha_1 \wedge \alpha_3 \wedge \alpha_7) \vee \\ (\alpha \wedge \alpha_2 \wedge \alpha_4 \wedge \alpha_8) \vee (\alpha \wedge \alpha_2 \wedge \alpha_5).$$

The rules in propositional analytic tableaux are grouped into two types, the so-called *rules of type* $\alpha$, corresponding to conjunction, and *rules of type* $\beta$, corresponding to disjunction. The following table summarizes the types.

| $\alpha$ | $\alpha_1, \ldots, \alpha_n$ | $\beta$ | $\beta_1, \ldots, \beta_n$ |
|---|---|---|---|
| $\phi_1 \wedge \ldots \wedge \phi_n$ | $\phi_1, \ldots, \phi_n$ | $\phi_1 \vee \ldots \vee \phi_n$ | $\phi_1, \ldots, \phi_n$ |
| $\neg(\phi_1 \vee \ldots \vee \phi_n)$ | $\neg\phi_1, \ldots, \neg\phi_n$ | $\neg(\phi_1 \wedge \ldots \wedge \phi_n)$ | $\neg\phi_1, \ldots, \neg\phi_n$ |
| $\neg\neg\phi$ | $\phi$ | | |
| $\neg$True | False | | |
| $\neg$False | True | | |

Observe that the only propositional connectives considered are $\vee$, $\wedge$ and $\neg$. The rules for analytic tableaux are defined for those connectives. The rules for other connectives can easily be derived.

**Definition 5.4.2** By an *analytic tableau* for a formula $\phi$ we understand a labelled tree constructed as follows:

- the root of the tree is labelled by formula $\phi$
- the branches are obtained by applying the following rules:

$$(\alpha)\dfrac{\alpha}{\dfrac{\alpha_1}{\dfrac{\dots}{\alpha_n}}}$$

$$(\beta)\dfrac{\beta}{\beta_1;\quad \dots\quad \beta_n}$$

A branch of a tableau is called *closed* iff it contains a formula and its negation. A tableau is *closed* provided that all its branches are closed. ■

Observe that closed tableaux represent formulas equivalent to False. Thus the proof method based on analytic tableaux depends on searching for a closed tableau for the negation of the input formula.

The following examples explain the method.

**Example 5.4.3** Let us prove that formula $q \to ((p \wedge q) \vee q)$ is a propositional tautology. We first negate the formula and obtain $q \wedge ((\neg p \vee \neg q) \wedge \neg q)$. The following tableau is constructed for the negated formula:

$$\dfrac{q \wedge ((\neg p \vee \neg q) \wedge \neg q)}{\dfrac{q}{\dfrac{(\neg p \vee \neg q) \wedge \neg q}{\dfrac{\neg p \vee \neg q}{\neg q}}(\alpha)}}(\alpha)$$

Observe that the above tableau contains one branch, which is closed (contains both $q$ and $\neg q$). In consequence the tableau is closed, thus the input formula is a propositional tautology. ■

**Example 5.4.4** Let us check whether formula $\neg(p \wedge q) \wedge q$ is a propositional tautology. We first negate it and obtain $(p \wedge q) \vee \neg q$. The tableau for the formula is constructed as follows:

$$\dfrac{(p \wedge q) \vee \neg q}{\dfrac{p \wedge q}{\dfrac{p}{q}}(\alpha) \qquad \neg q}(\beta)$$

The first branch of the tableau is closed, but the second one is not. The input formula is then not a propositional tautology. ■

We have the following theorem stating soundness and completeness of analytic tableaux.

**Theorem 5.4.5** Let $\alpha$ be a propositional formula. Then:

- if there is a closed tableau for $\neg\alpha$ then $\alpha$ is a propositional tautology (soundness)
- if $\alpha$ is a tautology of propositional calculus then there is a closed tableau for $\neg\alpha$ (completeness). ■

## 5.5 Exercises

1. Show that the resolution rule preserves satisfiability and does not preserve validity of propositional formulas.
2. Show that the rules of sequent calculus preserve validity of propositional formulas.
3. Derive sequent calculus rules for the propositional connective $\oplus$ (exclusive or) defined as $(p \oplus q) \equiv \mathsf{True}$ iff $p$ is $\mathsf{True}$ or $q$ is $\mathsf{True}$ but not both together.
4. Define analytic tableaux rules for implication.
5. Using resolution method, sequent calculus and analytic tableaux check whether the following formulas are propositional tautologies:

   (a) $(p \to q) \to [(r \to s) \to ((p \wedge r) \to (q \wedge s))]$

   (b) $[(p \vee q) \to r] \to [(p \to r) \wedge (q \to r)]$

   (c) $[(p \wedge q) \to r] \to [(p \to r) \wedge (q \to r)]$

   (d) $(p \to q) \to [(p \to \neg q) \to \neg p]$.

# Chapter 6

# Automated Deduction in Predicate Calculus

## 6.1 Introduction

In this chapter we shall extend methods defined in Chapter 5 for the case of propositional calculus.

## 6.2 Resolution Method

The resolution method in predicate calculus is based on the resolution rule and the factoring rule. However, because of the language, which includes terms, one has to use the unification.

Let us start with an example illustrating the need for unification of terms.

**Example 6.2.1** Consider two formulas $\forall x, y.[M(f(y), h(x)) \rightarrow T(y, x)]$ and $\forall x.M(f(g(x)), h(a))$. In order to prove that $T(g(a), a)$ one has to unify terms $f(y)$ with $f(g(x))$ and $h(x)$ with $h(a)$. The unification is done by applying substitution $[x := a; y := g(x)]$. Now the result is obtained by the transitivity of implication (which corresponds to an application of the resolution rule). ■

Consequently, the resolution and factoring rules are formulated as follows.

$$\begin{aligned}
&p_1(\bar{t}_1) \vee \ldots \vee p_k(\bar{t}_k) \vee r(\bar{t}),\ \ \neg r(\bar{t}') \vee q_1(\bar{s}_1) \vee \ldots \vee q_m(\bar{s}_m) \vdash && (6.1)\\
&\qquad p_1(\sigma(\bar{t}_1)) \vee \ldots \vee p_k(\sigma(\bar{t}_k)) \vee q_1(\sigma(\bar{s}_1)) \vee \ldots \vee q_m(\sigma(\bar{s}_m)) &&\\
&p_1(\bar{t}_1) \vee \ldots \vee p_k(\bar{t}) \vee p_k(\bar{t}') \vdash p_1(\sigma(\bar{t}_1)) \vee \ldots \vee p_k(\sigma(\bar{t}_k)), && (6.2)
\end{aligned}$$

where:

- $p_1, \ldots, p_k, q_1, \ldots, q_m$ are literals and $r$ is a relation symbol
- $\bar{t}_1, \ldots \bar{t}_k, \bar{t}, \bar{t}', \bar{s}_1, \ldots, \bar{s}_m$ are vectors of terms
- $\sigma$ is the most general unifier for $\bar{t}$ and $\bar{t}'$
- all repeating occurrences of literals in the consequent of rule (6.2) are deleted.

Rule (6.1) is called the *resolution rule* and rule (6.2) is called the *factoring rule.*

It is again important to note that the resolution rule does not preserve validity of formulas. However it preserves their satisfiability.

The following theorems state that the resolution method is sound and complete.

**Theorem 6.2.2** For any set of first-order clauses $S$,

1. if $S \vdash \nabla$ then $S$ is inconsistent (soundness)
2. if $S$ is inconsistent then $S \vdash \nabla$ (completeness),

where $\vdash$ denotes the proof system consisting of rules (6.1) and (6.2). ■

**Example 6.2.3** Let us prove that formula $\exists x.\forall y.R(y, x) \to \forall z.\exists u.R(z, u)$ is a tautology. Negating the formula results in $\exists x.\forall y.R(y, x) \wedge \exists z.\forall u.\neg R(z, u)$. The formula is not in the form of a set of clauses. We apply Skolemization and obtain two clauses[1] $R(y, a)$ and $\neg R(b, u)$, where $a, b$ are Skolem constants. A single resolution step with the most general unifier $[y := b; u := a]$ gives as the result the empty clause:

$$\frac{R(y, a) \qquad R(b, u)}{\nabla}(6.1;\ \sigma = [y := b; u := a]),$$

which proves the input formula. ■

**Example 6.2.4** Consider formula $\neg\exists y.\forall z.[P(z, y) \wedge \forall x.(\neg P(z, x) \vee \neg P(x, z))]$. We first negate the formula and obtain $\exists y.\forall z.[P(z, y) \wedge \forall x.(\neg P(z, x) \vee \neg P(x, z))]$, which is equivalent to the formula in PNF form $\exists y.\forall z.\forall x.[P(z, y) \wedge (\neg P(z, x) \vee \neg P(x, z))]$. After Skolemization and renaming of variable $z$ in the first clause we obtain two clauses $P(z', a)$ and $(\neg P(z, x) \vee \neg P(x, z))$, where $a$ is a Skolem constant. The proof can now be obtained as follows:

$$\cfrac{P(z', a) \qquad \cfrac{\neg P(z, x) \vee \neg P(x, z)}{\neg P(a, a)}(6.2;\ \sigma = [x := a; z := a])}{\nabla}(6.1;\ \sigma = [z' := a]).$$

Since the empty clause is obtained, the negated formula leads to contradiction, which shows that the input formula is a first-order tautology. ■

[1] The universal quantifiers are, as usual, implicit here.

## 6.3 Sequent Calculus

We define sequent calculus for the predicate calculus by accepting axioms and rules as given for propositional in section 5.3 (see Definition 5.3.2) and by adding new rules for first-order quantifiers.

**Definition 6.3.1** The *rules of sequent calculus for quantifiers* are of the following form:

$$
\begin{array}{ll}
(\forall l) & \dfrac{\forall x.\alpha(x), \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m}{\alpha(x := t), \alpha_1, \ldots, \alpha_n, \forall x.\alpha(x) \Rightarrow \beta_1, \ldots, \beta_m} \\
(\forall r) & \dfrac{\alpha_1, \ldots, \alpha_n \Rightarrow \forall x.\beta(x), \beta_1, \ldots, \beta_m}{\alpha_1, \ldots, \alpha_n \Rightarrow \beta(x), \beta_1, \ldots, \beta_m} \\
(\exists l) & \dfrac{\exists x.\alpha(x), \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m}{\alpha(x), \alpha_1, \ldots, \alpha_n \Rightarrow \beta_1, \ldots, \beta_m} \\
(\exists r) & \dfrac{\alpha_1, \ldots, \alpha_n \Rightarrow \exists x.\beta(x), \beta_1, \ldots, \beta_m}{\alpha_1, \ldots, \alpha_n \Rightarrow \beta(x := t), \beta_1, \ldots, \beta_m, \exists x.\beta(x)},
\end{array}
$$

where $t$ is a term, and in the substitution $x := t$ in rules $(\forall l)$ and $(\exists r)$ all variables that would become bound by a quantifier are renamed by using new variables. ■

It is important to observe that the rules preserve validity of formulas.

We have the following theorem stating soundness and completeness of sequent calculus.

**Theorem 6.3.2** Let $\alpha$ be a first-order formula. Then:

- if the sequent $\emptyset \Rightarrow \alpha$ is provable then $\alpha$ is a tautology of predicate calculus (soundness)
- if $\alpha$ is a tautology of predicate calculus then the sequent $\emptyset \Rightarrow \alpha$ is provable (completeness). ■

The following example illustrates the method.

**Example 6.3.3** Consider formula $\forall x.[P(x) \lor Q(x)] \to [\exists x.P(x) \lor \forall x.Q(x)]$. In order to prove that the formula is a tautology we start with the sequent $\emptyset \Rightarrow \forall x.[P(x) \lor Q(x)] \to [\exists x.P(x) \lor \forall x.Q(x)]$, then we apply rules $(\to r)$, $(\lor r)$, $(\forall l)$, $(\exists r, \forall r)$, $(\lor l)$:

$$
\begin{array}{c}
\emptyset \Rightarrow \forall x.[P(x) \lor Q(x)] \to [\exists x.P(x) \lor \forall x.Q(x)] \\
\hline
\forall x.[P(x) \lor Q(x)] \Rightarrow \exists x.P(x) \lor \forall x.Q(x) \\
\hline
\forall x.[P(x) \lor Q(x)] \Rightarrow \exists x.P(x), \forall x.Q(x) \\
\hline
P(x) \lor Q(x), \forall x.[P(x) \lor Q(x)] \Rightarrow \exists x.P(x), \forall x.Q(x) \\
\hline
P(x) \lor Q(x), \forall x.[P(x) \lor Q(x)] \Rightarrow P(x), Q(x), \exists x.P(x) \\
\hline
P(x), \forall x.[P(x) \lor Q(x)] \Rightarrow P(x), Q(x), \exists x.P(x); \quad Q(x), \forall x.[P(x) \lor Q(x)] \Rightarrow P(x), Q(x), \exists x.P(x)
\end{array}
$$

Since both leaves of the proof tree are axioms, the proof is completed. ■

## 6.4 Analytic Tableaux

We extend tableaux given for propositional connectives in section 5.4. We need rules for quantifiers. The rules for quantifiers are grouped into two types, the so-called *rules of type* $\gamma$, corresponding to the universal quantification, and *rules of type* $\delta$, corresponding to the existential quantification. The following table summarizes the types.

| $\gamma$ | $\gamma_1$ | $\delta$ | $\delta_1$ |
|---|---|---|---|
| $\forall x.\phi(x)$ | $\phi(t)$ | $\neg\forall x.\phi(x)$ | $\neg\phi(c)$ |
| $\neg\exists x.\phi(x)$ | $\neg\phi(t)$ | $\exists x.\phi(x)$ | $\phi(c)$ |

where $t$ is a term not containing free variables that could become bound after the substitution, and $c$ is a constant symbol.

**Definition 6.4.1** By an *analytic tableau* for a formula $\phi$ we understand a labelled tree constructed as follows:

- the root of the tree is labelled by formula $\phi$
- the branches are obtained by applying rules given in Definition 5.4.2 and the following rules:

$$(\gamma)\ \frac{\gamma}{\gamma_1} \qquad (\delta)\ \frac{\delta}{\delta_1}$$

A branch of a tableau is called *closed* iff it contains a formula and its negation. A tableau is *closed* provided that all its branches are closed. ■

**Example 6.4.2** Let us prove the formula

$$\forall x.[P(x) \to Q(x)] \to [\forall x.P(x) \to \forall x.Q(x)].$$

We first negate it, eliminate implication, and obtain:

$$\forall x.(\neg P(x) \lor Q(x)) \land \forall x.P(x) \land \neg\forall x.Q(x).$$

The following tableau can be constructed for the formula:

$$\dfrac{\forall x.(\neg P(x) \lor Q(x)) \land \forall x.P(x) \land \neg\forall x.Q(x)}{\begin{array}{c}\forall x.(\neg P(x) \lor Q(x))\\ \forall x.P(x)\\ \neg\forall x.Q(x)\end{array}}(\alpha)$$

$$\dfrac{\begin{array}{c}\forall x.(\neg P(x) \lor Q(x))\\ \forall x.P(x)\\ \neg\forall x.Q(x)\end{array}}{\begin{array}{c}\neg P(c) \lor Q(c)\\ P(c)\\ \neg\forall x.Q(x)\end{array}}(\gamma \text{ with } t = c)$$

$$\dfrac{\neg\forall x.Q(x)}{\begin{array}{c}\neg P(c) \lor Q(c)\\ \neg Q(c)\end{array}}(\delta)$$

$$\dfrac{\neg Q(c)}{\neg P(c) \qquad Q(c)}(\beta)$$

The tableau is closed, which proves the formula. ■

We have the following theorem stating soundness and completeness of analytic tableaux.

**Theorem 6.4.3** Let $\alpha$ be a first-order formula. Then:

- if there is a closed tableau for $\neg\alpha$ then $\alpha$ is a tautology of the predicate calculus (soundness)
- if $\alpha$ is a tautology of predicate calculus then there is a closed tableau for $\neg\alpha$ (completeness). ■

## 6.5 Exercises

1. Check which of the following formulas are valid, using the resolution method, sequent calculus and analytic tableaux:
   (a) $\exists x.(P(x) \lor Q(x)) \equiv (\exists x.P(x) \lor \exists x.Q(x))$
   (b) $\forall x.(P(x) \land Q(x)) \equiv (\forall x.P(x) \land \forall x.Q(x))$
   (c) $\forall x.\exists y.R(x, y) \to \exists y.\forall x.R(x, y)$
   (d) $\forall x.\forall y.[R(x, y) \lor R(y, x)] \to \exists x.\exists y.R(x, y)$
   (e) $\forall x.\exists y.[R(x, y) \lor \exists z.R(x, z)] \to \exists x.R(x, x)$.
2. For a chosen method define rules for equality and prove the following formulas:
   (a) $[P(x) \lor P(y)] \equiv \exists z.[(z = x \lor z = y) \land P(z)]$
   (b) $P(t) \equiv \forall x.(x \neq t \lor P(x))$.

# Part III

# Second-Order Logic and its Applications

# Chapter 7

# Second-Order Logic

## 7.1 Introduction

In many computer science applications, including AI, databases and formal verification of software, it is often necessary to formulate properties using phrases of the form "there is a relation", "for any relation", i.e. to use quantifiers over relations. Such quantifiers are called *second-order quantifiers* and are allowed in the *second-order logic*.

## 7.2 Syntax of Second-Order Logic

Second-order logic is an extension of predicate calculus obtained by admitting second-order quantifiers. In order to define this logic we have to add variables representing relations. The set of *relational variables*[1] is denoted by $V_{\mathsf{II}}$.

*Formulas of second-order logic*, denoted by $F_{\mathsf{II}}$, are defined by means of the following rules.

$$\begin{aligned} \langle F_{\mathsf{II}}\rangle ::= \quad & \langle \mathsf{Bool}\rangle \;||\; \langle V_{\mathsf{II}}\rangle \;||\; \langle F_{\mathsf{I}}\rangle \;||\; \neg\langle F_{\mathsf{II}}\rangle \;||\; \langle F_{\mathsf{II}}\rangle \wedge \langle F_{\mathsf{II}}\rangle \;||\; \langle F_{\mathsf{II}}\rangle \vee \langle F_{\mathsf{II}}\rangle \;|| \\ & \langle F_{\mathsf{II}}\rangle \to \langle F_{\mathsf{II}}\rangle \;||\; \langle F_{\mathsf{II}}\rangle \equiv \langle F_{\mathsf{II}}\rangle \;||\; \forall\langle V_{\mathsf{I}}\rangle.\langle F_{\mathsf{II}}\rangle \;||\; \exists\langle V_{\mathsf{I}}\rangle.\langle F_{\mathsf{II}}\rangle \;|| \\ & \forall\langle \mathsf{Rel}\rangle.\langle F_{\mathsf{II}}\rangle \;||\; \exists\langle \mathsf{Rel}\rangle.\langle F_{\mathsf{II}}\rangle \;||\; (\langle F_{\mathsf{II}}\rangle) \;||\; [\langle F_{\mathsf{II}}\rangle] \end{aligned}$$

By an *existential fragment of second-order logic* we shall mean the set of second-order formulas, whose second-order quantifiers can only be existential and appear in front of the formula.

[1] Relational variables are sometimes called *second-order variables*.

## 7.3 Semantics of Second-Order Logic

The semantics of the second-order logic is an extension of the semantics of the predicate calculus. We then only have to provide the semantics for relational variables and second-order quantifiers.

Let $R \in \mathsf{Rel}$ be a $k$-argument relation symbol. Assume we are given a relational structure $M = \langle \mathsf{Dom}, \{f_i^{\mathsf{Dom}} : i \in I\}, \{R_j^{\mathsf{Dom}} : j \in J\}\rangle$,. Denote by $\mathsf{Rel}(M)$ the set of all relations over $\mathsf{Dom}$. Let $v' : V_{\mathsf{I}} \longrightarrow \mathsf{Dom}$ be a valuation of individual variables and $v'' : V_{\mathsf{II}} \longrightarrow \mathsf{Rel}(M)$ be a valuation assigning relations to relational variables. Valuations $v', v''$ can be extended to the valuation assigning truth values to second-order formulas, $v : F_{\mathsf{II}} \longrightarrow \mathsf{Bool}$, as follows, assuming that first-order connectives and quantifiers are defined as in section 3.3:

$$\begin{array}{rcl}
v(X) & = & v''(X) \text{ for } X \in V_{\mathsf{II}} \\
v(\forall X.A(X)) & = & min(\{v_S^X(A(X))) : S \subseteq \mathsf{Dom}^k\}) \\
v(\exists X.A(X)) & = & max(\{v_S^X(A(X)) : S \subseteq \mathsf{Dom}^k\}),
\end{array}$$

where $X$ is a $k$-argument relational variable and by $v_S^X$ we denote the valuation obtained from $v$ by assigning value $S$ to variable $X$ and leaving all other variables unchanged.

Observe that in many cases second-order quantification, say $\forall R$, is intended to range over a subset of all relations. For instance in the *weak second-order logic* only finite relations are considered. Another example is the so-called *Henkin-like semantics*, where one restricts possible relations, e.g. to first-order definable relations. This is motivated by the high complexity of the second-order logic (see section 7.4) as well as the observation that quite often one has in mind a restricted subset of relations. For example, consider the sentence "John has all features of a good student. The second-order quantification "all features ranges over relations that can be defined by a first-order formula.

## 7.4 The Complexity of Second-Order Logic

**Theorem 7.4.1** Both checking whether a second-order formula is satisfiable or whether it is a tautology are not partially computable problems[2]. ■

**Theorem 7.4.2** Let $\alpha$ be a second-order formula $\alpha$. Then:

- checking satisfiability and validity of $\alpha$ over a given finite domain relational structure is PSPACE-complete
- [Fagin] if $\alpha$ belongs to the existential fragment of second-order logic, then checking its satisfiability is NPTIME-complete. ■

[2] In fact, the problem is even much more complex as these are not even arithmetical in the sense of Kleene and Mostowski hierarchy.

## 7.5 Second-Order Logic in Commonsense Reasoning

One of the most prominent applications of the second-order logic is the formulation of circumscription[3] Circumscription is a powerful non-monotonic formalism centered around the following idea: the objects (tuples of objects) that can be shown to satisfy a certain relation are all the objects (tuples of objects) satisfying it. For instance, to circumscribe the relation of being red is to assume that any object that cannot be proved red is not red.

Circumscription can be viewed as a form of minimization of relations. Observe also that circumscription can be used for maximizing relations, since maximizing a relation corresponds to minimizing its negation. Given a theory $T$, a list $P_1, \ldots, P_n$ of relation symbols to be minimized and a list $Q_1, \ldots, Q_m$ of relation symbols that are allowed to vary, circumscription of $P_1, \ldots, P_n$ in $T$ with variable $Q_1, \ldots, Q_m$ amounts to implicitly adding to $T$ a special second-order sentence, called *circumscription axiom*, capturing the desired minimization.

**Definition 7.5.1** [Circumscription] Let $\overline{P} = (P_1 \ldots, P_n)$ be a tuple of distinct relation symbols, $\overline{S} = (S_1, \ldots, S_m)$ be a tuple of distinct relation symbols disjoint with $\overline{P}$, and let $T(\overline{P}, \overline{S})$ be a theory. The *circumscription* of $\overline{P}$ *in* $T(\overline{P}, \overline{S})$ *with variable* $\overline{S}$, written $CIRC(T; \overline{P}; \overline{S})$, is the sentence

$$T(\overline{P}, \overline{S}) \wedge \forall \overline{X}. \forall \overline{Y}. \neg[T(\overline{X}, \overline{Y}) \wedge \overline{X} < \overline{P}] \tag{7.1}$$

where $\overline{X} = (X_1 \ldots, X_n)$ and $\overline{Y} = (Y_1, \ldots, Y_m)$ are tuples of relation variables similar to $\overline{P}$ and $\overline{S}$, respectively.[4]

Observe that (7.1) can be rewritten as

$$T(\overline{P}, \overline{S}) \wedge \forall \overline{X}. \forall \overline{Y}. \ \big[[T(\overline{X}, \overline{Y}) \wedge \overline{X} \leq \overline{P}] \rightarrow \overline{P} \leq \overline{X}\big]$$

which, in turn, is an abbreviation for

$$T(\overline{P}, \overline{S}) \wedge \forall \overline{X}. \forall \overline{Y}. \ \Bigg[[T(\overline{X}, \overline{Y}) \wedge \bigwedge_{i=1}^{n} \forall \overline{x}.(X_i(\overline{x}) \rightarrow P_i(\overline{x}))] \\ \rightarrow \bigwedge_{i=1}^{n} \forall \overline{x}.(P_i(\overline{x}) \rightarrow X_i(\overline{x}))\Bigg]. \blacksquare$$

[3] Circumscription was introduced by McCarthy. Its second-order formulation is due to Lifschitz.

[4] $T(\overline{X}, \overline{Y})$ is the sentence obtained from $T(\overline{P}, \overline{S})$ by replacing all occurrences of $P_1 \ldots, P_n$ by $X_1 \ldots, X_n$, respectively, and all occurrences of $S_1 \ldots, S_m$ by $Y_1 \ldots, Y_m$, respectively.

**Example 7.5.2** Let $T$ consists of:

$$\begin{array}{l} Bird(Tweety) \\ \forall x.[Bird(x) \wedge \neg Ab(x) \to Flies(x)]. \end{array}$$

We take $\overline{P} = (Ab)$ and $\overline{S} = (Flies)$.

$$\begin{array}{l} CIRC(T; \overline{P}; \overline{S}) = T(\overline{P}, \overline{S}) \wedge \\ \qquad \forall X.\forall Y.\ [[Bird(Tweety) \wedge \forall x.[(Bird(x) \wedge \neg X(x)) \to Y(x)] \wedge \\ \qquad \forall x.(X(x) \to Ab(x))] \to \forall x.(Ab(x) \to X(x))]\,. \end{array}$$

Substituting $\mathsf{False}$ for $X$ and $Bird(x)$ for $Y$, we conclude that

$$CIRC(T; \overline{P}; \overline{S}) \models T \wedge A \tag{7.2}$$

where $A$ is

$$\begin{array}{l} T \wedge [\forall x.(Bird(x) \wedge \neg \mathsf{False}) \to Bird(x)) \wedge \forall x.(\mathsf{False} \to Ab(x))] \to \\ [\forall x.Ab(x) \to \mathsf{False}]). \end{array}$$

Since $A$ simplifies to $\forall x.Ab(x) \to \mathsf{False}$, we conclude, by (7.2), that

$$CIRC(T; \overline{P}; \overline{S}) \models Flies(Tweety). \blacksquare$$

**Example 7.5.3** Let $T$ consists of:

$$\begin{array}{l} R(n) \wedge Q(n) \\ \forall x.R(x) \wedge \neg Ab_1(x) \to \neg P(x) \\ \forall x.Q(x) \wedge \neg Ab_2(x) \to P(x), \end{array}$$

with $R$, $P$, $Q$, $n$ standing for *Republican*, *Pacifist*, *Quaker* and *Nixon*, respectively.[5] Let $M$ and $N$ be models of $T$ such that $|M| = |N| = \{nixon\}$, $M|n| = N\,|n| = nixon$, $M\,|R| = N\,|R| = \{nixon\}$, $M\,|Q| = N\,|Q| = \{nixon\}$ and

$$\begin{array}{ll} M\,|P| = \{nixon\} & N\,|P| = \emptyset \\ M\,|Ab_1| = \{nixon\} & N\,|AB_1| = \emptyset \\ M\,|Ab_2| = \emptyset & N\,|Ab_2| = \{nixon\}. \end{array}$$

It is easily seen that for any $\overline{S}$, $M$ and $N$ are both $(\overline{AB}, \overline{S})$-minimal models of $T$, where $\overline{AB} = (Ab_1, Ab_2)$. Furthermore, $M \models P(n)$ and $N \models \neg P(n)$. It follows,

[5] Observe that we use two abnormality relations here, namely $Ab_1$ and $Ab_2$. This is because being abnormal with respect to pacifism as a quaker is a different notion than being abnormal with respect to pacifism as a republican.

therefore, that when we circumscribe $\overline{AB}$ in $T$, we shall not be able to infer whether Nixon is a pacifist or not. The best we can obtain is the disjunction $\neg Ab_1(n) \vee \neg Ab_2(n)$, stating that Nixon is either normal as a republican or as a quaker. ■

## 7.6 Exercises

1. Prove that the following formula[6] is a second-order tautology:

$$\forall \bar{x} \exists y . \alpha(\bar{x} \ldots) \;\equiv\; \exists f \forall \bar{x} \alpha(\bar{x}, y := f(\bar{x}), \ldots).$$

2. Prove theorem of Fagin (the second part of Theorem 7.4.2).

3. Characterize natural numbers up to isomorphism using the second-order logic.

4. Express in the second-order logic that $S(x, y)$ is the transitive closure of relation $R(x, y)$.

[6] Called second-order Skolemization.

# Chapter 8

# Second-Order Quantifier Elimination

## 8.1 Introduction

Second-order logic is a natural tool to express many interesting properties. It is usually avoided because of its high complexity. On the other hand, there are methods that allow one to eliminate second-order quantifiers. The most fruitful methods are described below.

## 8.2 SCAN Algorithm

The Scan algorithm[1] was proposed by Gabbay and Ohlbach. It can be considered as a refinement of Ackermann's resolution-like method, but it was discovered independently. Scan takes as input second-order formulae of the form

$$\alpha = \exists P_1 \dots \exists P_k . \Phi$$

with existentially quantified predicate variables $P_i$ and a first-order formula $\Phi$. Scan eliminates all predicate variables at once.

The following three steps are performed by Scan:

1. $\Phi$ is transformed into clause form.
2. All C-resolvents and C-factors with the predicate variables $P_1, \dots, P_k$ are generated. *C-resolution rule* ('C' is short for constraint) is defined as

[1] Scan means "Synthesizing Correspondence Axioms for Normal Logics". The name was chosen before the general nature of the procedure was recognized.

follows:

$$\begin{array}{ll} P(s_1,\ldots s_n)\vee C & P(\ldots)\text{ and }\neg P(\ldots)\\ \neg P(t_1,\ldots,t_n)\vee D & \text{are the }\textit{resolution literals}\\ \hline C\vee D\vee s_1\neq t_1\vee\ldots\vee s_n\neq t_n & \end{array}$$

and the *C-factoring rule* is defined analogously:

$$\frac{P(s_1,\ldots,s_n)\vee P(t_1,\ldots,t_n)\vee C}{P(s_1,\ldots,s_n)\vee C\vee s_1\neq t_1\vee\ldots\vee s_n\neq t_n}.$$

When *all* resolvents and factors between a particular literal and the rest of the clause set have been generated (the literal is then said to be "resolved away"), the clause containing this literal is deleted (this is called "purity deletion"). If all clauses have been deleted this way, we know $\alpha$ is a tautology. If an empty clause is generated, we know $\alpha$ is inconsistent.

3. If step 2 terminates and the set of clauses is non-empty then the quantifiers for the Skolem functions are reconstructed.

The next example illustrates the various steps of the SCAN algorithm in detail.

**Example 8.2.1** Let the input for SCAN be:

$$\exists P.\forall x.\forall y.\exists z.[(\neg P(a)\vee Q(x))\wedge(P(y)\vee Q(a))\wedge P(z)].$$

In the first step the clause form is computed:

$$\begin{array}{ll} C_1: & \neg P(a)\vee Q(x)\\ C_2: & \phantom{\neg}P(y)\vee Q(a)\\ C_3: & \phantom{\neg}P(f(x,y)) \end{array}$$

where $f$ is a Skolem function.

In the second step of SCAN we begin by choosing $\neg P(a)$ to be resolved away. The resolvent between $C_1$ and $C_2$ is $C_4 = Q(x), Q(a)$ which is equivalent to $Q(a)$ (this is one of the equivalence preserving simplifications). The C-resolvent between $C_1$ and $C_3$ is $C_5 = (a\neq f(x,y), Q(x))$. There are no more resolvents with $\neg P(a)$, so $C_1$ is deleted. We are left with the clauses

$$\begin{array}{ll} C_2: & P(y)\vee Q(a)\\ C_3: & P(f(x,y))\\ C_4: & Q(a)\\ C_5: & a\neq f(x,y)\vee Q(x). \end{array}$$

Selecting the next two $P$-literals to be resolved away yields no new resolvents, so $C_2$ and $C_3$ can be deleted as well. All $P$-literals have now been eliminated. Restoring the quantifiers, we then get

$$\forall x.\exists z.[Q(a)\wedge(a\neq z\vee Q(x))]$$

as the final result ($y$ is no longer needed). ■

There are two critical steps in the algorithm. First of all the C-resolution loop need not always terminate. This may but need not indicate that there is no first–order equivalent for the input formula. If the resolution eventually terminates the next critical step is the unskolemization. Since this is essentially a quantifier elimination problem for existentially quantified function variables, there is also no complete solution. The algorithm usually used is heuristics based.

Preventing C-resolution from looping is a difficult control issue. Some equivalence preserving transformations on clause sets turned out to be quite useful. In the algorithm we have implemented each new resolvent can be tested whether it is implied by the non-parent clauses. In the affirmative case it is deleted even if more resolutions are possible.

**Example 8.2.2** Consider three colorability of a graph. The following second-order formula expresses this property:

$$\exists C. \left( \begin{array}{l} \forall x.[C(x,R) \vee C(x,Y) \vee C(x,G)] \wedge \\ \forall x.\forall y.[(C(x,y) \wedge C(x,z)) \rightarrow y = z] \wedge \\ \forall x.\forall y.[E(x,y) \rightarrow \neg \exists z.(C(x,z) \wedge C(y,z))] \end{array} \right) \tag{8.1}$$

where $R, Y, G$ are constant symbols denoting colors, $C(x,y)$ is interpreted as "node $x$ in a graph has color $y$" and $E(x,y)$ as "$y$ is adjacent to $x$ in a graph". The first formula states that each node is colored with one of the three colors. The second axiom says that each node has at most one color and finally the last axiom requires adjacent nodes to have different colors.

The clause normal form of (8.1) is

$$\begin{array}{l} C(x,R) \vee C(x,Y) \vee C(x,G) \vee C(x,B) \\ \neg C(x,y) \vee \neg C(x,z) \vee x = z \\ \neg C(x,z) \vee \neg C(y,z) \vee \neg N(x,y) \end{array}$$

Given this to SCAN and asking it to eliminate $C$ the resolution loops. If we however replace the first clause by the equivalent formula

$$\forall x \; \exists c \; (c = R \vee c = Y \vee c = G \vee c = B) \wedge C(x,c)$$

whose clause normal form is

$$\begin{array}{l} c(x) = R \vee c(x) = Y \vee c(x) = G \vee c(x) = B \\ C(x, c(x)) \end{array}$$

there is no problem anymore. The two successive resolutions with the second clause yields a tautology. The two successive resolutions with the third clause yields $c(x) \neq c(y) \vee \neg N(x,y)$. The result is now

$$\begin{array}{l} \exists c.\forall x.(c(x) = R \vee c(x) = Y \vee c(x) = G \vee c(x) = B) \\ \wedge \forall x.\forall y.(N(x,y) \Rightarrow c(x) \neq c(y)) \end{array}$$

which is again second–order. It is just a reformulation of the original formula in terms of the coloring function $c$. However, it would be quite surprising to get a better result with such a simple method. ■

## 8.3 DLS Algorithm

We say that a formula $\Phi$ is *positive* w.r.t. a predicate $P$ iff there is no occurrence of $\neg P$ in the negation normal form of $\Phi$. Dually, we say that $\Phi$ is *negative* w.r.t. $P$ iff every of its occurrences in the negation normal form of $\Phi$ is of the form $\neg P$.

The following lemma was proved by Ackermann in 1934.

**Lemma 8.3.1** [Ackermann] Let $P$ be a predicate variable and let $\Phi$ and $\Psi(P)$ be first–order formulae.

- If $\Psi(P)$ is positive w.r.t. $P$ and $\Phi$ contains no occurrences of $P$ at all, then

$$\exists P.\forall \overline{x}.(P(\overline{x}) \to \Phi) \wedge \Psi(P) \quad \equiv \quad \Psi\left[P(\overline{\alpha}) := [\Phi]^{\overline{x}}_{\overline{\alpha}}\right]$$

- If $\Psi(P)$ is negative w.r.t. $P$ and $\Phi$ contains no occurrences of $P$ at all, then

$$\exists P.\forall \overline{x}.(\Phi \to P(\overline{x})) \wedge \Psi(P) \quad \equiv \quad \Psi\left[P(\overline{\alpha}) := [\Phi]^{\overline{x}}_{\overline{\alpha}}\right]$$

The right hand formulas are to be read as: "every occurrence of $P$ in $\Psi$ is to be replaced by $\Phi$ where the actual argument of $P$, say $\overline{\alpha}$, replaces the variables of $\overline{x}$ in $\Phi$ (and the bound variables are renamed if necessary)". ■

Hence, if the second–order formula under consideration has the syntactic form of one of the forms given as the left-hand-side of equivalences given in Lemma 8.3.1 then this lemma can immediately be applied for the elimination of $P$.

The DLS algorithm was defined by Doherty, Łukaszewicz and Szałas as a refinement of an earlier algorithm of Szałas. The algorithm tries to transform the input formula into the form suitable for application of Lemma 8.3.1.

More precisely, the algorithm takes a formula of the form $\exists P.\Phi$, where $\Phi$ is a first-order formula, as an input and returns its first-order equivalent or reports failure[2]. Of course, the algorithm can also be used for formulas of the form $\forall P.\Phi$, since the latter formula is equivalent to $\neg\exists P.\neg\Phi$. Thus, by repeating the

[2] The failure of the algorithm does not mean that the second-order formula at hand cannot be reduced to its first-order equivalent. The problem we are dealing with is not even partially decidable, for first-order definability of the formulas we consider is totally uncomputable.

algorithm one can deal with formulas containing many arbitrary second-order quantifiers.

The following purity deletion rule is used in the algorithm.

> if there is a predicate $Q$ among the list of predicates to be eliminated such that $Q$ occurs with mixed sign in some clauses and either only with positive or only with negative sign in the other clauses then all clauses containing $Q$ are deleted. For example in the two clauses $\neg Q(x) \vee Q(f(x))$ and $Q(a)$ there is no clause containing $Q$ only with negative sign. If these are the only clauses with $Q$, they can be deleted. (Since $Q$ is existentially quantified, a model making $Q$ true everywhere satisfies the clauses.)

The elimination algorithm consists of three phases: (1) preprocessing; (2) preparation for Lemma 8.3.1; (3) application of Lemma 8.3.1. These phases are described below. It is always assumed that (1) whenever the goal specific for a current phase is reached, then the remaining steps of the phase are skipped, (2) every time the extended purity deletion rule is applicable, it should be applied.

1. *Preprocessing.* The purpose of this phase is to transform the formula $\exists P.\Phi$ into a form that separates positive and negative occurrences of the quantified predicate variable $P$. The form we want to obtain is

   $$\exists \bar{x}.\exists P.[(\Phi_1(P) \wedge \Psi_1(P)) \vee \cdots \vee (\Phi_n(P) \wedge \Psi_n(P))],$$

   where, for each $1 \leq i \leq n$, $\Phi_i(P)$ is positive w.r.t. $P$ and $\Psi_i(P)$ is negative w.r.t. $P$. The steps of this phase are the following. (i) Eliminate the connectives $\Rightarrow$ and $\equiv$ using the usual definitions. Remove redundant quantifiers. Rename individual variables until all quantified variables are different and no variable is both bound and free. Using the usual equivalences, move the negation connective to the right until all its occurrences immediately precede atomic formulas. (ii) Move universal quantifiers to the right and existential quantifiers to the left, applying as long as possible the usual quantifier rules. (iii) In the matrix of the formula obtained so far, distribute all top-level conjunctions over the disjunctions that occur among their conjuncts. (iv) If the resulting formula is not in the required form, then report the failure of the algorithm. Otherwise replace the input formula by its equivalent given by

   $$\exists \bar{x}.(\exists P.(\Phi_1(P) \wedge \Psi_1(P)) \vee \cdots \vee \exists P.(\Phi_n(P) \wedge \Psi_n(P))).$$

   Try to find first-order equivalent of the above formula by applying the next phases in the algorithm to each its disjunct separately. If the first-order equivalents of each disjunct are successfully obtained then return their disjunction, preceded by the prefix $\exists \bar{x}$, as the output of the algorithm.

2. *Preparation for the Ackermann lemma.* The goal of this phase is to transform a formula of the form $\exists P(\Phi(P) \wedge \Psi(P))$, where $\Phi(P)$ (respectively. $\Psi(P)$) is positive (respectively. negative) w.r.t. $P$, into one of the forms required in Lemma 8.3.1. Both forms can always be obtained by using equivalences given in section 3.7 and both transformations should be performed because none, one or both forms may require Skolemization. Unskolemization, which occurs in the next phase, could fail in one form, but not the other. In addition, one form may be substantially smaller than the other.

3. *Application of the Ackermann Lemma.* The goal of this phase is to eliminate the second-order quantification over $P$, by applying Lemma 8.3.1, and then to unskolemize the function variables possibly introduced. This latter step employs the second-order Skolemization equivalence.

4. *Simplification.* Generally, application of Lemma 8.3.1 in step (3) often involves the use of equivalences mentioned in section 3.7 in the left to right direction. If so, the same equivalences may often be used after application in the right to left direction, substantially shortening the resulting formula.

## 8.4 Reducing Circumscription

**Example 8.4.1** Consider a variant of the Vancouver example of Reiter. Rather than using the function *city* as Reiter does, we will use a relation $C(x, y)$ with suitable axioms. The intention is that that $C(x, y)$ holds iff $y$ is the home town of $x$.

Let $\Phi(Ab, C)$ be the theory

$$[\forall x.\forall y.\forall z.(\neg Ab(x) \wedge C(x,y) \wedge C(wife(x), z)) \rightarrow y = z] \wedge \\ [\forall x.\forall y.\forall z.\ (C(x,y) \wedge C(x,z)) \rightarrow y = z].$$

The circumscription of $\Phi(Ab, C)$ with $Ab$ minimized and $C$ varied is

$$circ(\Phi(Ab, C); Ab; C) \equiv \Phi(Ab, C) \wedge \\ \forall P^*.\forall Z^*.[\Phi(P^*, Z^*) \wedge [P^* \leq Ab] \rightarrow [Ab \leq P^*],$$

where

$$\Phi(P^*, Z^*) \equiv \quad [\forall x.\forall y.\forall z.\neg P^*(x) \wedge Z^*(x,y) \wedge Z^*(wife(x), z) \rightarrow y = z] \wedge \\ [\forall x.\forall y.\forall z.Z^*(x,y) \wedge Z^*(x,z) \rightarrow y = z].$$

The DLS algorithm reduces the second-order part of circumscription:

$$\forall P^*.\forall Z^*.[\Phi(P^*, Z^*) \wedge [P^* \leq Ab] \rightarrow [Ab \leq P^*]].$$

After two iterations (the first for reducing $P^*$ and the second for reducing $Z^*$) one obtains the result $\forall t.\neg Ab(t)$. Consequently,

$$circ(\Phi(Ab, C); Ab; C) \equiv \Phi(Ab, C) \wedge \forall t.\neg Ab(t). \blacksquare$$

## 8.5 Exercises

1. Give an example where SCAN fails.

2. Give an example where DLS fails.

3. Let $\Gamma(Ab, On)$ be the theory

$$\begin{aligned}&[b1 \neq b2 \wedge B(b1) \wedge B(b2) \wedge \neg On(b1)] \wedge\\ &\forall x.(B(x) \wedge \neg Ab(x) \rightarrow On(x)),\end{aligned}$$

   where $B$ and $On$ are abbreviations for *Block* and *Ontable*, respectively. Consider the circumscription of $\Gamma(Ab, On)$ with $Ab$ minimized and $On$ varied, and eliminate second-order quantifiers using SCAN and DLS.

4. Let $\Gamma(Ab, G)$ be the theory

$$\begin{aligned}&[\exists x.\exists y.(B(y) \wedge F(x, y) \wedge \neg G(x, y))]\\ &\quad \wedge[\forall x.\forall y.(B(y) \wedge F(x, y) \wedge \neg Ab(x, y) \rightarrow G(x, y))],\end{aligned}$$

   where $B$, $F$ and $G$ are abbreviations for *Birthday*, *Friend* and *GivesGift*, respectively. Here $Ab(x, y)$ has the following intuitive interpretation: "$x$ behaves abnormally w.r.t. $y$ in the situation when $y$ has a birthday and $x$ is a friend of $y$". Consider the circumscription of $\Gamma(Ab, G)$ with $Ab$ minimized and $G$ varied, and eliminate second-order quantifiers using SCAN and DLS.

5. Let $\Gamma$ be the theory

$$\begin{aligned}&[\forall x.\exists y.(Ab(x, y) \rightarrow H(x, y))] \wedge\\ &\forall x.\exists y.(\neg Ab(x, y) \rightarrow H(x, y)).\end{aligned}$$

   Here $H(x, y)$ and $Ab(x, y)$ are to be intuitively interpreted as "$x$ is in a hospital in a situation $y$" and "$x$ behaves abnormally in a situation $y$", respectively. Consider the circumscription of $\Gamma$, with $Ab$ minimized and $H$ varied, and eliminate second-order quantifiers using SCAN and DLS.

# Part IV

# Other Important Logics

# Chapter 9

# Modal Logics

## 9.1 Introduction

Classical logic is *extensional* in the sense that one can always substitute a term by an equal term and a subformula by equivalent subformula. On the other hand, in many applications one deals with *intensional notions*, i.e., notions that are not extensional in its nature.For instance, many natural language expressions are not extensional.

Consider the operator $Knows(x, y)$ meaning that person $x$ knows who is person $y$. Assume that $\neg Knows(Jack, X)$ and that $Knows(Jack, Mary)$. Here we assume that $X$ is a masked person. It might, however, happen that $Mary = X$. If expression $Knows$ was extensional, one could then replace $X$ by $Mary$ and obtain $\neg Knows(Jack, Mary)$ leading to an obvious contradiction.

Intensional notions are called *modal operators* or *modalities* and logics allowing such notions are called modal logics and sometimes *intensional logics*. In the simplest case one deals with a single unary modality $\Box$ and its dual $\Diamond$. Modalities $\Box$ and $\Diamond$ have many possible readings, dependent on a particular application. The following table summarizes the most frequent readings of modalities.

| a possible reading of $\Box\alpha$ | a possible reading of $\Diamond\alpha$ |
|---|---|
| $\alpha$ is necessary | $\alpha$ is possible |
| $\alpha$ is obligatory | $\alpha$ is allowed |
| always $\alpha$ | sometimes $\alpha$ |
| in the next moment of time $\alpha$ | in the next moment of time $\alpha$ |
| $\alpha$ is known | $\alpha$ is believed |
| every program state satisfies $\alpha$ | there is a program state satisfying $\alpha$ |
| $\alpha$ is provable | $\alpha$ is not provable |

Observe that $\Box$ usually corresponds to a kind of universal quantification and $\Diamond$

usually corresponds to a kind of existential quantification.

The types of modal logics reflect the possible readings of modalities. The following table summarizes types of logics corresponding to the possible readings of modalities.

| the possible reading of modalities | the type of modal logic |
|---|---|
| necessary, possible | aletic logics |
| obligatory, allowed | deontic logics |
| always, sometimes, in the next time | temporal logics |
| known, believed | epistemic logics |
| every (some) state of a program satisfies | logics of programs |
| (un)provable | provability logics |

Modal logics have a long history. However we will focus on their modern developments.

## 9.2 Syntax of Propositional Modal Logics

The simplest version of modal logics allows for a single unary modal operator, usually denoted by $\Box$ and its dual form, denoted by $\Diamond$. In more complex applications, e.g., involving multi-agent systems or software verification, one deals with many modalities included in so-called multi-modal logics.

Let us start with the case of one modality.

**Definition 9.2.1** *Formulas of propositional modal logic*, denoted by $F_M$ are defined by augmenting the syntax rules for propositional formulas by the following rule:

$$\langle F_M \rangle ::= \langle F_0 \rangle \mid\mid \Box \langle F_M \rangle.$$

Dual modal operator, $\Diamond$, is defined by $\Diamond\alpha \overset{\text{def}}{\equiv} \neg\Box\neg\alpha$. ■

## 9.3 How to Develop a Modal Logic for an Application

There are two approaches to defining modal logics suitable for a given application. The first one depends on providing a set of axioms describing the desired properties of modal operators. The second one starts with suitable models (e.g., generated by a program) and then develops syntactic (proof-theoretical) characterization of modalities.

Let us start with the first approach (the second one will be represented in Chapters 10 and 11 on temporal logics and logics of programs). Suppose we are interested in formalizing knowledge operator. One can postulate, or instance, the following properties:

1. $\Box\alpha \to \Diamond\alpha$ — if $\alpha$ is known then $\alpha$ is believed
2. $\Box\alpha \to \alpha$ — if $\alpha$ is known then it actually holds
3. $\alpha \to \Diamond\alpha$ — if $\alpha$ holds then it is believed.

Of course there are many questions at this point, including the following:

- are the properties consistent?
- do the properties express all the desired phenomena?
- is a property a consequence of another property?
- how to formalize the reasoning involving modalities?

Usually answers to such questions are given by semantic investigations.

One should also note that many modal logics are well understood and can be used as building blocks for more advanced applications.

Let us now define a well-known classification of modal logics, based on properties of modalities. The starting point for the classification is the notion of normal modal logics as defined below.

**Definition 9.3.1** Let $\mathcal{F}$ be a modal language, as defined in Definition 9.2.1. Then:

- By a *propositional modal logic* we understand any logic with the set of formulas $\mathcal{F}$.
- By a *normal modal logic* we shall understand any modal logic $\langle \mathcal{F}, \mathcal{C}, \models \rangle$, satisfying the following conditions:
  - $\models \Box(\alpha \to \beta) \to (\Box\alpha \to \Box\beta)$,
  - $\models \alpha$ implies $\models \Box\alpha$.

The least normal modal logic (in the sense of the set of tautologies) is denoted by $\mathbf{K}$[1]. ■

The above conditions concerning normal modal logics are reflected by a Hilbert-like proof system defined below.

[1] $\mathbf{K}$ comes from the name of Kripke.

**Definition 9.3.2** Proof system for **K** is defined by extending the propositional calculus by adding the axiom:

$$\Box(\alpha \to \beta) \to (\Box\alpha \to \Box\beta)$$

together with the following *modal generalization rule*:

$$\alpha \;\vdash\; \Box\alpha. \;\blacksquare$$

Other normal modal logics are defined by additional axioms expressing the desired properties of modalities. Bellow we follow the classification introduced by Lemmon.

| | | |
|---|---|---|
| **D.** | = | $\Box\alpha \to \Diamond\alpha$ |
| **T.** | = | $\Box\alpha \to \alpha$ |
| **4.** | = | $\Box\alpha \to \Box\Box\alpha$ |
| **E.** | = | $\Diamond\alpha \to \Box\Diamond\alpha$ |
| **B.** | = | $\alpha \to \Box\Diamond\alpha$ |
| **Tr.** | = | $\Box\alpha \equiv \alpha$ |
| **M.** | = | $\Box\Diamond\alpha \to \Diamond\Box\alpha$ |
| **G.** | = | $\Diamond\Box\alpha \to \Box\Diamond\alpha$ |
| **H.** | = | $(\Diamond\alpha \land \Diamond\beta) \to (\Diamond(\alpha \land \beta) \lor \Diamond(\alpha \land \Diamond\beta) \lor \Diamond(\beta \land \Diamond\alpha))$ |
| **Grz.** | = | $\Box(\Box(\alpha \to \Box\alpha) \to \alpha) \to \alpha$ |
| **Dum.** | = | $\Box(\Box(\alpha \to \Box\alpha) \to \alpha) \to (\Diamond\Box\alpha \to \alpha)$ |
| **W.** | = | $\Box(\Box\alpha \to \alpha) \to \Box\alpha.$ |

In the formulas given above **D** comes from *deontic*, **T** is a traditional name of the axiom (after Feys), **4** is characteristic for logic **S4** of Lewis, **E** comes from *Euclidean*[2], **B** comes from *Brouwer*[3], **Tr** abbreviates *trivial*, **M** comes from *McKinsey*, **G** from *Geach*, **H** from *Hintikka*, **Grz** from *Grzegorczyk*, **Dum** from *Dummett*, **W** from *reverse well founded*[4].

The above formulas are used to define many well-known modal logics. The notational convention $\mathbf{KX_0...X_m}$ means the least normal logic in which formulas $X_0, ..., X_m$ are accepted as axioms. The following logics are frequently used and applied:

[2] This axiom is often denoted by **5**.

[3] Because of its similarity with **KTB** and the intuitionistic logic.

[4] It is also known as the Löb axiom.

| | | |
|---|---|---|
| **KT** | = | **T** = logic of Gödel/Feys/Von Wright |
| **KT4** | = | **S4** |
| **KT4B** | = | **KT4E** = **S5** |
| **K4E** | = | **K45** |
| **KD** | = | deontic **T** |
| **KD4** | = | deontic **S4** |
| **KD4B** | = | deontic **S5** |
| **KTB** | = | Brouwer logic |
| **KT4M** | = | **S4.1** |
| **KT4G** | = | **S4.2** |
| **KT4H** | = | **S4.3** |
| **KT4Dum** | = | **D** = Prior logic |
| **KT4Grz** | = | **KGrz** = Grzegorczyk logic |
| **K4W** | = | **KW** = Löb logic |
| **KTr** | = | **KT4BM** = trivial logic. |

Observe that Hilbert-like proof method formalizes the process of reasoning in the above logics.

## 9.4 Semantics of Propositional Modal Logics

There are many semantics for modal logics. Here we shall follow the Kripke-like style of defining semantics for modal logics[5].

Kripke published his famous paper in 1959. The basis for the Kripke semantics is the universe of *possible worlds*. Formulas evaluated in the so-called *actual world*. The other worlds, alternatives to the actual worlds correspond to possible situations. Modalities $\Box$ and $\Diamond$ have the following intuitive meaning:

- formula $\Box\alpha$ holds in a given world $w$, if $\alpha$ holds in all worlds alternative for $w$
- formula $\Diamond\alpha$ holds in a given world $w$, if $\alpha$ holds in some world alternative for $w$.

What remains to do is to define the notion of alternative worlds. Kripke assumes that this is done by a binary relation on worlds, as defined in the following definition.

**Definition 9.4.1** By a *Kripke frame* we mean nay relational system of the form $\langle \mathcal{W}, R \rangle$, where $\mathcal{W}$ is any set and $R$ id a binary relation defined on $\mathcal{W}$. Elements of $\mathcal{W}$ are called *worlds*. ■

We are now ready to define the notion of a Kripke structure.

[5] In fact, similar semantics was a couple of years earlier defined by Kanger. Then, in the same year, the similar semantics was given by Hintikka and also Guillaume.

**Definition 9.4.2** By a *Kripke structure* we mean triple

$$\langle \mathcal{K}, w, v \rangle,$$

where $\mathcal{K} = \langle \mathcal{W}, R \rangle$ is a Kripke frame, $w \in \mathcal{W}$ is the *actual world* and $v$ is a mapping, $v : \mathcal{F} \times \mathcal{W} \longrightarrow \{\mathsf{True}, \mathsf{False}\}$, assigning truth values to atoms in worlds. ■

Observe that a very general definition of modal logics was given in Definition 9.2.1. However, from now on we shall restrict ourselves to those modal logics, whose semantics is defined via Kripke structures. The following definition reflects this approach.

**Definition 9.4.3** By a *propositional modal logic* we shall mean any logic $\mathcal{M} = \langle \mathcal{F}, \mathcal{C}, \models_{\mathcal{M}} \rangle$, where:

- the set of formulas $\mathcal{F}$ is defined as in Definition 9.2.1,
- $\mathcal{C}$ is a subclass of the class of all Kripke structures,
- $\models_{\mathcal{M}}$ is defined as follows:
  - $\mathcal{K}, w, v \models_{\mathcal{M}} \alpha$ iff $v(\alpha, w) = \mathsf{True}$, where $\alpha$ is an atom,
  - the meaning of propositional connectives is the same as in the case of propositional calculus,
  - $\mathcal{K}, w, v \models_{\mathcal{M}} \Box\alpha$ iff for any $w'$ such that $R(w, w')$ holds, we have that $\mathcal{K}, w', v \models_{\mathcal{M}} \alpha$, where $R$ is the relation of the Kripke frame $\mathcal{K}$,
  - $\mathcal{K}, w, v \models_{\mathcal{M}} \Diamond\alpha$ iff there is $w'$ such that $R(w, w')$ and $\mathcal{K}, w', v \models_{\mathcal{M}} \alpha$, where $R$ is the relation of the Kripke frame $\mathcal{K}$.

By $\mathcal{K} \models_{\mathcal{M}} \alpha$ we shall mean that formula $\alpha$ holds in any Kripke structure with the frame $\mathcal{K}$. ■

Observe that we now face the question, what are the properties of relation $R$ that guarantee that particular axioms are indeed valid in all Kripke structures. If the properties can be expressed by means of predicate calculus, we will be able to automatize the reasoning using the techniques defined for the predicate calculus.

It appears that the properties of $R$ can often be calculated using the second-order quantifier elimination techniques. We shall return to the subject later. Here we only summarize some correspondences between modal logics and predicate calculus. We use notation $\exists! y$, meaning that there is exactly one $y$. Such a quantifier can be defined as follows:

$$\exists! y.\alpha(y) \;\equiv\; \exists y.(\alpha(y) \land \forall z.(\alpha(z) \to z = y)).$$

| Formula | Property of $R$ |
|---|---|
| **D** | $\forall x.\exists y.R(x,y)$ ($R$ is serial) |
| **T** | $\forall x.R(x,x)$ ($R$ is reflexive) |
| **4** | $\forall x,y,z.(R(x,y)\wedge R(y,z))\rightarrow R(x,z)$ ($R$ is transitive) |
| **E** | $\forall x,y,z.(R(x,y)\wedge R(x,z))\rightarrow R(y,z)$ ($R$ is Euclidean) |
| **B** | $\forall x,y.(R(x,y)\rightarrow R(y,x))$ ($R$ is symmetric) |
| **Tr** | $\forall x,y.(R(x,y)\equiv x=y)$ ($R$ is trivial) |
| **G** | $\forall x,y,z.((R(x,y)\wedge R(x,z))\rightarrow \exists w(R(y,w)\wedge R(z,w))$<br>($R$ is directed) |
| $\Diamond\alpha\rightarrow\Box\alpha$ | $\forall x,y,z.(R(x,y)\wedge R(x,z))\rightarrow y=z$ ($R$ is a partial function) |
| $\Diamond\alpha\equiv\Box\alpha$ | $\forall x.\exists! y.R(x,y)$ ($R$ is a function) |
| $\Box\Box\alpha\rightarrow\Box\alpha$ | $\forall x,y.R(x,y)\rightarrow\exists z(R(x,z)\wedge R(z,y))$ ($R$ is dense) |

Unfortunately, not all modal axioms correspond to first-order expressible properties of $R$. Axioms **Grz**, **W M** are examples of the lack of the desired correspondence.

Similarly, there are some natural properties of relation $R$, which cannot be captured by any modal axiom. For instance,

| | |
|---|---|
| $\forall x.\neg R(x,x)$ | - antireflexivity |
| $\forall x,y.(R(x,y)\rightarrow\neg R(y,x))$ | - antisymmetry |
| $\forall x,y.(R(x,y)\wedge R(y,x))\rightarrow x=y$ | - weak antisymmetry. |

## 9.5 Multi-Modal Logics

Consider now the case where one deals with many modalities[6].

**Definition 9.5.1** *Formulas of a propositional multi-modal logic* with modalities $\Box_m$, where $(m=1,\ldots,k)$, are defined by replacing in Definition 9.2.1 the clause introducing $\Box$ by the following clause:

$$\langle F_M\rangle ::= \langle F_0\rangle \mid\mid \Box_1\langle F_M\rangle \mid\mid \ldots \mid\mid \Box_k\langle F_M\rangle.$$

Modalities $\Diamond_m$, dual $\Box_m$ are defined by $\Diamond_m\alpha \overset{\text{def}}{\equiv} \neg\Box_m(\neg\alpha)$. ■

The notions of the Kripke frames and structures is easily obtained by introducing relations $R_m$, for $m=1,\ldots,k$ modelling modalities $\Box_m$.

**Definition 9.5.2**

- By a *multi-modal Kripke frame* we understand any relational system $\langle\mathcal{W},\{R_m\}_{m\in M}\rangle$, where $\mathcal{W}$ is any set, and $R_m$ are binary relations on $\mathcal{W}$.

[6] We consider here only unary modalities. One can easily extend the definitions form many-argument modalities.

- By a *multi-modal Kripke structure* we understand any triple $\langle \mathcal{K}, w, v \rangle$, where $\mathcal{K} = \langle \mathcal{W}, \{R_m\}_{m\in M} \rangle$ is a multi-modal Kripke frame, $w \in \mathcal{W}$ is a world and $v$ is a function,

$$v : \mathcal{F} \times \mathcal{W} \longrightarrow \{\mathsf{True}, \mathsf{False}\}. \blacksquare$$

**Definition 9.5.3** By a *propositional multi-modal logic* with modalities $\Box_m$ ($m = 1, \dots, k$) we call any logic $\mathcal{M} = \langle \mathcal{F}, \mathcal{C}, \models_{\mathcal{M}} \rangle$, where:

- $\mathcal{F}$ the set of formulas as defined in Definition 9.5.1,
- $\mathcal{C}$ is a subclass the class of Kripke structures,
- $\models_{\mathcal{M}}$ is defined by extending Definition 9.4.3, where $m = 1, \dots, k$):
  - $\mathcal{K}, w, v \models_{\mathcal{M}} \Box_m \alpha$ iff for any $w'$ such that $R_m(w, w')$ we have $\mathcal{K}, w', v \models_{\mathcal{M}} \alpha$, where $R_m$ is a relation of $\mathcal{K}$,
  - $\mathcal{K}, w, v \models_{\mathcal{M}} \Diamond_m \alpha$ iff there is $w'$ such that $R_m(w, w')$ and $\mathcal{K}, w', v \models_{\mathcal{M}} \alpha$, where $R_m$ is a relation of $\mathcal{K}$. ■

## 9.6 Computing Correspondences between Modal Logics and Predicate Calculus

**Example 9.6.1** Consider first the modal axioms $\Box p \to p$, $\Box p \to \Box\Box p$ and $p \to \Box\Diamond p$. As we treat implication as abbreviation, those should be equivalently rewritten as $\neg\Box p \lor p$, $\neg\Box p \lor \Box\Box p$ and $\neg p \lor \Box\Diamond p$

Let us now apply the algorithm to those axioms. Let us start with the first one.

— translated axiom:
$\forall P.\forall x.\neg(\forall y.\neg R(x, y) \lor P(y)) \lor P(x)$
— negated: $\exists x.\exists P.(\forall y.\neg R(x, y) \lor P(y)) \land \neg P(x)$ — note that Lemma 8.3.1 can be applied
— $P$ eliminated: $\exists x.\neg R(x, x)$
— unnegated: $\forall x.R(x, x)$ — i.e. the reflexivity of $R$

Let us now consider the second axiom.

— translated axiom:
$\forall P.\forall x.\neg(\forall y.\neg R(x, y) \lor P(y)) \lor \forall y.(\neg R(x, y) \lor \forall z.(\neg R(y, z) \lor P(z)))$
— negated:
$\exists x.\exists P.(\forall y.\neg R(x, y) \lor P(y)) \land \exists y.(R(x, y) \land \exists z.(R(y, z) \land \neg P(z)))$
— transformed:
$\exists x.\exists P.\forall y.[P(y) \lor \neg R(x, y)] \land [\exists y.R(x, y) \land \exists z.(R(y, z) \land \neg P(z))]$
— now Lemma 8.3.1 can be applied

— $P$ eliminated: $\exists x.\exists y.R(x,y) \wedge \exists z.(R(y,z) \wedge \neg R(x,z))$
— unnegated: $\forall x.\forall y.\neg R(x,y) \vee \forall z.(\neg R(y,z) \vee R(x,z))$ — the algorithm stops here, however the formula can still be simplified as follows
— simplified: $\forall x,y,z.R(x,y) \to (R(y,z) \to R(x,z))$ — i.e., the transitivity of $R$

The elimination of $P$ from the third axiom proceeds as follows.

— translated axiom:
$\forall P.\forall x.\neg P(x) \vee (\forall y.\neg R(x,y) \vee \exists z.(R(y,z) \wedge P(z))$
— negated: $\exists x.\exists P.P(x) \wedge \exists y.(R(x,y) \wedge \forall z.(\neg R(y,z) \vee \neg P(z))$
— transformed:
$\exists x.\exists P.\forall z.[P(z) \vee x \neq z] \wedge [\exists y.R(x,y) \wedge \forall z.(\neg R(y,z) \vee \neg P(z))]$ — now Lemma 8.3.1 can be applied
— $P$ eliminated: $\exists x.\exists y.R(x,y) \wedge \forall z.(\neg R(y,z) \vee x \neq z)$
— unnegated: $\forall x.\forall y.\neg R(x,y) \vee \exists z.(R(y,z) \wedge x = z)$ — the algorithm stops here, but we can simplify the formula
— simplified: $\forall x,y.R(x,y) \to R(y,x)$ — i.e., the symmetry of $R$

Let us now illustrate the algorithm in case of elimination of two second-order quantifications. For this purpose consider the axiom $\Box(p \vee q) \to (\Box p \vee \Box q)$, i.e., $\neg\Box(p \vee q) \vee (\Box p \vee \Box q)$.

— translated axiom: $\forall P.\forall Q.\forall x.\neg(\forall y.(R(x,y) \to (P(y) \vee Q(y))) \vee (\forall z.(R(x,z) \to P(z)) \vee \forall v.(R(x,v) \to Q(v)))$
— negated: $\exists x.\exists P.\exists Q.(\forall y.(\neg R(x,y) \vee P(y) \vee Q(y)) \wedge (\exists z.(R(x,z) \wedge \neg P(z)) \wedge \exists v.(R(x,v) \wedge \neg Q(v))$
— separated (w.r.t. $Q$): $\exists x,z,v.\exists P.\exists Q.\forall y.[Q(y) \vee (\neg R(x,y) \vee P(y))] \wedge [R(x,z) \wedge \neg P(z) \wedge R(x,v) \wedge \neg Q(v)]$
— $Q$ eliminated:
$\exists x,z,v.\exists P.(R(x,z) \wedge \neg P(z) \wedge R(x,v) \wedge (\neg R(x,v) \vee P(v))$
— separated (w.r.t. $P$):
$\exists x,z,v.\exists P.[P(v) \vee \neg R(x,v)] \wedge R(x,z) \wedge \neg P(z) \wedge R(x,v) \wedge \neg P(v)$
— transformed:
$\exists x,z,v.\exists P.\forall u.[P(u) \vee (u \neq v \vee \neg R(x,u))] \wedge [R(x,z) \wedge \neg P(z) \wedge R(x,v)]$
— $P$ eliminated: $\exists x,z,v.[R(x,z) \wedge (z \neq v \vee \neg R(x,z)) \wedge R(x,v)]$
— negated: $\forall x,z,v.[\neg R(x,z) \vee (z = v \wedge R(x,z)) \vee \neg R(x,v)]$ — the algorithm stops here, but we can still make some simplifications:
— simplified: $\forall x,z,v.((R(x,z) \wedge R(x,v)) \to (z = v \wedge R(x,z))$, i.e., $\forall x,z,v.((R(x,z) \wedge R(x,v)) \to z = v$ ■

## 9.7 Exercises

1. Provide a precise definition of translation of modal formulas into formulas of the predicate calculus.

2. Compute correspondences of Example 9.6.1 using SCAN.

3. Compute correspondences for axioms **D** and $\Box\Box\alpha \rightarrow \Box\alpha$, using a chosen second-order quantifier elimination technique.

4. Check whether the following formulas are tautologies of **K**:

   - $\alpha \rightarrow \Box\alpha$
   - $\alpha \rightarrow \Box\mathsf{True}$
   - $\Diamond\mathsf{True}$
   - $\Diamond\mathsf{True} \rightarrow (\Box\alpha \rightarrow \Diamond\alpha)$.

# Chapter 10

# Temporal Logic

## 10.1 Introduction

As temporal logics serve as a tool for expressing and verifying properties dynamically changing in time, one first of all has to decide what is the structure of time, and what are time-dependent properties. Both time structure and time-dependent properties are strongly dependent on the specific application. As we are now mainly interested in the temporal logics of programs, we shall not attempt to give any general definition of temporal logics. Instead, we shall present a definition that follows intuitions related to programs and their computations. Consider then typical programs. Usual programs are algorithms that compute over fixed, time-independent data structures. Program variables are thus those symbols that change while passing from one program state to another. For our purposes it is then sufficient to admit that the only time-dependent symbols are program variables, and that all other symbols are independent of the flow of time. Time-dependent symbols are usually called *flexible* or *local*, while time-independent ones are *rigid* or *global.* The situation with the notion of time is not that simple, for temporal logics of programs are mainly applied to specification and verification of concurrent computations. Thus there can be as many reasonable structures of time as there are possible models of computations. Accepting the so-called interleaving model, where actions of processes are shuffled, one deals with a linear time. For semantics based on partial orders, partially ordered time is the most natural and apparent. It is not difficult to give examples of applications where multilinear time suits best. Even when one deals with linear time, there is a wide choice. The following are typical examples:

- time is reals, ordered by the usual relation $\leq$, i.e. the corresponding time structure is $\langle R, \leq \rangle$,
- time is integers ordered by the usual relation $\leq$, i.e. the time structure is $\langle Z, \leq \rangle$,

- time is natural numbers with a successor function (corresponding to next-time modality) ordered by the relation $\leq$, i.e. the time structure is $\langle N, succ, \leq \rangle$
- time is natural numbers with distinguished constants 0, 1, addition, multiplication and ordered by the relation $\leq$, i.e. the time structure is $\langle N, 0, 1, +, *, \leq \rangle$.

In the general definition of the temporal logic of programs we have to accept all possible notions of time. We shall assume that time is given by means of classical first-order interpretation. Since data structures are usually defined by first-order interpretations, too, we have the following definition.

**Definition 10.1.1** By *temporal logic* we mean any logic $TL = \langle F, Mod, . \rangle$ such that

- $F$ contains a distinguished set of constants $C$ (elements of $C$ are called *flexible constants*); no symbols except those of $C$ can have a time-dependent meaning,
- $Mod$ is a class of classical two-sorted interpretations of the form $\langle T$ , $D$, $(f_c)_{c \in C} \rangle$, where
  - $T$ is a *time structure*,
  - $D$ is a *data structure* that gives meaning to function and relation symbols (as in classical first-order logic),
  - for any $c \in C$, $f_c : T \longrightarrow D$ is a function from $T$ to $D$ which serves to interpret constant $c$ (the value of $c$ at time $t$ is $f_c(t)$). ■

Note that in the case of propositional versions of temporal logics, data structure $D$ of the above definition consists of two-element boolean algebra (with universe $\{\mathsf{True}, \mathsf{False}\}$).

Observe also that flexible constants are usually called *variables* in the literature as they correspond to program variables. In what follows we shall often use similar terminology.

## 10.2 Propositional Temporal Logic of Programs

In the following definition we shall consider only two temporal operators, namely *atNext* (introduced by Kröger) and *atPrev*. These operators are similar. Their main difference is that *atNext* concerns the future, while *atPrev* deals with the past. Intuitively, $p \, atNext \, q$ means that $p$ will be satisfied at the nearest of the future time points with $q$ satisfied and $p \, atPrev \, q$ means that $p$ has been satisfied at the nearest of the past time points with $q$ satisfied. Note that we deal with the strong versions of the temporal operators, in that we require that $p\,atNext\,q$ implies satisfiability of $q$ at some future time point and $p\,atPrev\,q$ implies satisfiability of $q$ somewhere in the past.

**Definition 10.2.1** By a *propositional temporal logic of programs* we mean triple $PTL = \langle F_{PTL}, Mod_{PTL}, \models_{PTL}\rangle$, where

- $F_{PTL}$ is the set of *propositional temporal formulas* containing a distinguished set $P$ of *flexible propositional variables* and defined by the following syntax rules

$$\begin{aligned}\langle F_{PTL}\rangle ::= {} & \langle P\rangle \mid\mid \neg\langle F_{PTL}\rangle \mid\mid \langle F_{PTL}\rangle \wedge \langle F_{PTL}\rangle \mid\mid \langle F_{PTL}\rangle \vee \langle F_{PTL}\rangle \mid\mid \\ & \langle F_{PTL}\rangle \rightarrow \langle F_{PTL}\rangle \mid\mid \langle F_{PTL}\rangle \equiv \langle F_{PTL}\rangle \mid\mid \\ & \langle F_{PTL}\rangle\, atNext\, \langle F_{PTL}\rangle \mid\mid \langle F_{PTL}\rangle\, atPrev\, \langle F_{PTL}\rangle \mid\mid \\ & \mid\mid (\langle F_{PTL}\rangle) \mid\mid [\langle F_{PTL}\rangle]\end{aligned}$$

- $Mod$ is the class of two-sorted first-order interpretations $\langle \mathbf{Z}, D, (f_p)_{p\in P}\rangle$, where
  1. $\mathbf{Z}$ is the structure of integers ordered by the usual relation $\leq$, i.e. $\mathbf{Z} = \langle Z, \leq\rangle$,
  2. $D$ is the two-element boolean algebra with universe $\{\mathsf{True}, \mathsf{False}\}$,
  3. for any $p \in P$, $f_p : Z \longrightarrow \{\mathsf{True}, \mathsf{False}\}$ is a function assigning boolean values to propositional variables (intuitively $f_p(t)$ is the value of $p$ at time point $t$)
- for $\mathcal{M} \in Mod_{PTL}$, and $\gamma \in F_{PTL}$, $\mathcal{M} \models_{PTL} \gamma$ iff for all $t \in Z$, $\mathcal{M}, t \models \gamma$, where for $\alpha, \beta \in F_{PTL}$:
  1. $\mathcal{M}, t \models \alpha$, for $\alpha \in P$, iff $f_\alpha(t) = \mathsf{True}$
  2. $\mathcal{M}, t \models \neg\alpha$ iff not $\mathcal{M}, t \models \alpha$
  3. $\mathcal{M}, t \models \alpha \wedge \beta$ iff $\mathcal{M}, t \models \alpha$ and $\mathcal{M}, t \models \beta$
  4. $\mathcal{M}, t \models \alpha\, atNext\, \beta$ iff there is $t_1 \in Z$ such that $t_1 > t$, $\mathcal{M}, t_1 \models \alpha \wedge \beta$ and for all $t < t_2 < t_1$, $\mathcal{M}, t_2 \models \neg\beta$
  5. $\mathcal{M}, t \models \alpha\, atPrev\, \beta$ iff there is $t_1 \in Z$ such that $t_1 < t$, $\mathcal{M}, t_1 \models \alpha \wedge \beta$ and for all $t_1 < t_2 < t$, $\mathcal{M}, t_2 \models \neg\beta$. ■

Observe that other known temporal operators can easily be defined, e.g.:

- the operator $\overleftarrow{\bigcirc}$ with truth table $\exists t' < t.[\forall t'' < t.(t'' \leq t') \;\wedge\; Q(t')]$ (i.e., $Q$ is true at the immediate predecessor of $t$) is defined by

$$(\overleftarrow{\bigcirc} q) \;\overset{\mathrm{def}}{\equiv}\; q\, atPrev\, \mathsf{True},$$

- the operator $\bigcirc$ with truth table $\exists t' > t.[\forall t'' > t.(t'' \geq t') \ \wedge \ Q(t')]$ (i.e., $Q$ is true at the immediate successor of $t$) is defined by

$$(\bigcirc q) \overset{\text{def}}{\equiv} q \, \mathit{atNext} \, \mathsf{True}.$$

- the operator $\Diamond$ with truth table $\exists t' > t.[Q(t')]$ is defined by

$$\Diamond \alpha \overset{\text{def}}{\equiv} \mathsf{True} \, \mathit{atNext} \, \alpha$$

There are many other temporal operators discussed in the literature, for instance:

- the operator *Since* with truth table $\exists t' < t.[P(t') \wedge \forall s.(t' < s < t \rightarrow Q(s))]$,
- the operator *Until* with truth table $\exists t' > t.[P(t') \wedge \forall s.(t < s < t' \rightarrow Q(s))]$,
- the operator $\overleftarrow{\Diamond}$ with truth table $\exists t' < t.[Q(t')]$,
- the operator $\overleftarrow{\Box}$ with truth table $\forall t' < t.[Q(t')]$,
- the operator $\Box$ with truth table $\forall t' > t.[Q(t')]$,
- the operator $\widehat{\Box}$ with truth table $\forall t.[Q(t)]$,
- the operator $\widehat{\Diamond}$ with truth table $\exists t.[Q(t)]$.

The definitions those operators using *atNext* and *atPrev* are left as exercises.

Let us now consider a few examples of properties of programs expressible by means of temporal logics of programs (thus by $PTL$, too). In the examples we assume that formulas $p$ and $q$ do not contain past time operators.

- **Invariance (safety) properties**
  - $p \rightarrow \Box q$ (all states reached by a program after the state satisfying $p$ will satisfy $q$)
  - $(atFirst \rightarrow p) \rightarrow \Box(atEnd \rightarrow q)$ (partial correctness w.r.t conditions $p$ and $q$, where propositional variable $atFirst$ is true only at the beginning of the specified program, while $atEnd$ is true only when the program reaches its terminal state)
  - $\Box((\neg q) \vee (\neg p))$ (the program cannot enter critical regions $p$ and $q$ simultaneously (mutual exclusion)).

- **Eventuality properties**
  - $p \to \Diamond q$ (there is a program state satisfying $q$ reached by a program after the state satisfying $p$)
  - $(atFirst \to p) \to q\ atNext\ atEnd$ (total correctness w.r.t. conditions $p$ and $q$)
  - $\Box\Diamond p \to \Diamond q$ (repeating a request $p$ will force a response $q$)
  - $\Box p \to \Diamond q$ (permanent holding a request $p$ will force a response $q$).

Note that precedence properties are easier and more natural to express by past time operators than by future time ones. For instance, a typical precedence property stating that any occurrence of a program state (event) satisfying $p$ (if any) must be preceded by a state (event) satisfying $q$, can be formulated by past time operator $\overleftarrow{\Diamond}$ as $p \to \overleftarrow{\Diamond} q$ while a suitable formula involving future time operators only could to look like $\Diamond p \to (\neg p\ \ Until\ q)$.

The propositional temporal logic we deal with, $PTL$, is decidable. This means that there is an algorithm to decide whether a given formula is satisfiable. More precisely, given a formula $p$ of $PTL$, we can automatically check whether there is an interpretation $\mathcal{M} = \langle \mathbf{Z}, D, (f_p)_{p\in P}\rangle \in Mod_0$ satisfying $p$ (i.e. such that $\mathcal{M} \models p$). Such an algorithm can, for instance, use the following important theorem.

**Theorem 10.2.2** If a formula of $PTL$ is satisfiable then it is satisfiable by a finitely representable interpretation. Moreover, the size of the interpretation can be calculated from size of the formula. ■

## 10.3 Hilbert and Gentzen-like Proof Systems for PTL

Let us now define the Hilbert-like proof system for the propositional temporal logic $PTL$.

**Definition 10.3.1** By the Hilbert-like proof system $HP$ for logic $PTL$ we mean the system $\langle HAx, H\rangle$, where

- set $HAx$ consists of the following (schemes of) axioms
  1. $\vdash\ \neg\bigcirc\alpha \equiv \bigcirc(\neg\alpha)$
  2. $\vdash\ (\bigcirc\alpha \wedge \bigcirc\beta) \to \bigcirc(\alpha \wedge \beta)$
  3. $\vdash\ \alpha\ atNext\beta\ \ \equiv\ \ \bigcirc((\alpha \wedge \beta) \vee (\neg\beta \wedge \alpha\ atNext\beta))$
  4. $\vdash\ \neg\overleftarrow{\bigcirc}\alpha \equiv \overleftarrow{\bigcirc}(\neg\alpha)$

5. $\vdash\ (\overleftarrow{\bigcirc}\alpha \wedge \overleftarrow{\bigcirc}\beta) \rightarrow \overleftarrow{\bigcirc}(\alpha \wedge \beta)$
6. $\vdash\ \alpha\ \mathit{atPrev}\,\beta\ \equiv\ \overleftarrow{\bigcirc}((\alpha \wedge \beta) \vee (\neg\beta \wedge \alpha\ \mathit{atPrev}\,\beta))$
7. $\vdash\ \alpha \equiv \bigcirc(\overleftarrow{\bigcirc}\alpha)$
8. $\vdash\ \alpha \equiv \overleftarrow{\bigcirc}(\bigcirc\alpha)$
9. $\vdash\ \Box(\alpha \rightarrow \bigcirc\alpha)\ \rightarrow\ (\bigcirc\alpha \rightarrow \Box\alpha)$
10. $\vdash\ \overleftarrow{\Box}(\alpha \rightarrow \overleftarrow{\bigcirc}\alpha)\ \rightarrow\ (\overleftarrow{\bigcirc}\alpha \rightarrow \overleftarrow{\Box}\beta)$

- set $H$ consists of the following (schemes of) derivation rules

  1. for each substitution instance of a classical propositional tautology $\alpha$, $\vdash \alpha$
  2. $\alpha,\ \alpha \rightarrow \beta\ \vdash\ \beta$
  3. $\alpha \rightarrow \beta\ \vdash\ \alpha\ \mathit{atNext}\,\gamma \rightarrow \beta\ \mathit{atNext}\,\gamma$
  4. $\bigcirc(\alpha \wedge \beta)\ \rightarrow \gamma,\ \bigcirc(\neg\beta \wedge \gamma) \rightarrow \gamma,\ \gamma \rightarrow \delta\ \vdash\ \alpha\ \mathit{atNext}\,\beta \rightarrow \delta$
  5. $\alpha \rightarrow \beta\ \vdash\ \alpha\ \mathit{atPrev}\,\gamma \rightarrow \beta\ \mathit{atPrev}\,\gamma$
  6. $\overleftarrow{\bigcirc}(\alpha \wedge \beta)\ \rightarrow \gamma,\ \overleftarrow{\bigcirc}(\neg\beta \wedge \gamma) \rightarrow \gamma,\ \gamma \rightarrow \delta\ \vdash\ \alpha\ \mathit{atPrev}\,\beta \rightarrow \delta.$ ■

A few examples of applications of $HP$ now follow.

**Example 10.3.2** First we shall show simple formal proofs that the following usual axioms of many propositional temporal proof systems

$$\alpha\ \mathit{atNext}\,\beta\ \rightarrow\ \Diamond\beta \tag{10.1}$$

$$\bigcirc(\alpha \wedge \beta) \rightarrow (\bigcirc\alpha\ \wedge\ \bigcirc\beta) \tag{10.2}$$

$$\overleftarrow{\bigcirc}(\alpha \wedge \beta) \rightarrow (\overleftarrow{\bigcirc}\alpha\ \wedge\ \overleftarrow{\bigcirc}\beta) \tag{10.3}$$

are derivable in system $HP$.

First note that $\Diamond\beta$ is, by definition, equivalent to $\mathsf{True}\ \mathit{atNext}\,\beta$. Formula $\alpha \rightarrow \mathsf{True}$ is derivable in system $HP$ by application of rule 1, as it is simply a classical propositional tautology. Thus, applying rule 3 with $\beta$ in the rule substituted by $\mathsf{True}$ we obtain $\alpha\ \mathit{atNext}\,\beta \rightarrow \mathsf{True}\ \mathit{atNext}\,\beta$, i.e., by definition, $\alpha\ \mathit{atNext}\,\beta\ \rightarrow\ \Diamond\beta$. This proves (10.1).

As proofs of (10.2) and (10.3) are similar, we shall present the second of them. According to the definition, $\overleftarrow{\bigcirc}p \equiv p\ \mathit{atPrev}\,\mathsf{True}$, thus, by rule 1, it suffices to prove that

$$(\alpha \wedge \beta)\ \mathit{atPrev}\,\mathsf{True} \rightarrow \alpha\ \mathit{atPrev}\,\mathsf{True} \tag{10.4}$$

and

$$(\alpha \wedge \beta)\ \mathit{atPrev}\, \mathsf{True} \rightarrow \beta\ \mathit{atPrev}\, \mathsf{True}. \tag{10.5}$$

Since formulas $(\alpha \wedge \beta) \rightarrow \alpha$ and $(\alpha \wedge \beta) \rightarrow \beta$ are classical propositional tautologies, we prove (10.4) and (10.5) by application of rule 1 and then 5 with $\gamma$ substituted by $\mathsf{True}$.

Note that the following tautologies can be proved applying axioms 1, 4, tautologies (10.2), (10.3) and rule 1

$$\vdash\ \bigcirc(\alpha \rightarrow \beta) \equiv (\bigcirc\alpha \rightarrow \bigcirc\beta), \tag{10.6}$$

$$\vdash\ \overleftarrow{\bigcirc}(\alpha \rightarrow \beta) \equiv (\overleftarrow{\bigcirc}\alpha \rightarrow \overleftarrow{\bigcirc}\beta). \tag{10.7}$$

■

The following example shows a formal proof of a temporal property that combines past and future. It also displays the usefulness of proof rule 4.

**Example 10.3.3** Let us present a formal proof of the following temporal tautology:

$$\alpha\ \rightarrow\ \neg(\mathsf{True}\ \mathit{atNext}\,(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))).$$

By classical propositional reasoning, we have to prove the formula

$$\mathsf{True}\ \mathit{atNext}\,(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))\ \rightarrow\ \neg\alpha.$$

Its form suggests the use of rule 4 of proof system $HP$. It is then sufficient to find an invariant $\gamma$ such that premises of the rule can be proved valid. After applying suitable substitutions, those premises take the following form

$$\bigcirc(\mathsf{True} \wedge (\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha)))\ \rightarrow\ \gamma, \tag{10.8}$$

$$\bigcirc(\mathsf{True}\ \mathit{atPrev}\,\alpha\ \wedge\ \gamma)\ \rightarrow\ \gamma, \tag{10.9}$$

$$\gamma\ \rightarrow\ (\neg\alpha). \tag{10.10}$$

At first glance it seems that substituting $\gamma$ by $\neg\alpha$ should work. Unfortunately, it is not the case. It is, however, quite a usual situation in the logics of programs (e.g. in that of Hoare for partial correctness of **while**-programs) that the invariant has to be stronger than the desired conclusion suggests. No wonder then, that a similar phenomenon is inherited by the temporal logic of programs. In our proof we shall accept formula $\bigcirc(\overleftarrow{\Box}(\neg\alpha))$ as a suitable invariant. By the definition of $\overleftarrow{\Box}$, the invariant takes the form $\bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))$.

The premise (10.8) reduces now to $\bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha)) \ \rightarrow\ \bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))$, which as an obvious classical propositional tautology is provable (by rule (1) of $HP$).

After replacing $\gamma$ by our invariant and applying classical propositional reasoning, the second premise (10.9) takes the form

$$\bigcirc(\mathsf{True}\ \mathit{atPrev}\,\alpha\ \wedge\ \bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha)))\ \rightarrow\ \bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha)).$$

Note that provability of formula

$$\bigcirc\bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))\ \rightarrow\ \bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))$$

implies, by axiom 1, tautology (10.6) of the previous example and rule (1) of $HP$, provability of the premise (10.9). By tautology (10.6) it suffices to show provability of the formula $\bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))\ \rightarrow\ (\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))$, i.e. by axiom 1 and rule 1, provability of formula

$$\mathsf{True}\ \mathit{atPrev}\,\alpha\ \rightarrow\ \bigcirc(\mathsf{True}\ \mathit{atPrev}\,\alpha). \tag{10.11}$$

Consider the right-hand side of the implication (10.11). After application of axiom 6 we can rewrite it as $\bigcirc(\overleftarrow{\bigcirc}(\mathsf{True} \wedge \alpha) \vee \overleftarrow{\bigcirc}(\neg\alpha \wedge \mathsf{True}\ \mathit{atPrev}\,\alpha))$. By applying axioms 1, 7 and tautology (10.6), we obtain the formula $\alpha \vee (\neg\alpha \wedge \mathsf{True}\ \mathit{atPrev}\,\alpha)$, i.e. by rule 1, formula $(\alpha \vee \mathsf{True}\ \mathit{atPrev}\,\alpha)$, which, again by rule 1, is implied by the left-hand side of implication (10.11).

The premise (10.8), $\gamma\ \rightarrow\ (\neg\alpha)$, takes the form $\bigcirc(\neg(\mathsf{True}\ \mathit{atPrev}\,\alpha))\ \rightarrow\ (\neg\alpha)$. To prove this formula it suffices to apply axiom 1 together with rule 1 in order to obtain the formula $\alpha \rightarrow \bigcirc(\mathsf{True}\ \mathit{atPrev}\,\alpha)$. This formula can be proved by applying axiom 6 and then 7 together with axiom 1, tautology (10.6) and rule 1.

To complete the proof it now suffices to apply rule 4 of $HP$. ■

We have the following important theorem.

**Theorem 10.3.4** Proof system $HP$ for propositional temporal logic $PTL$ is sound and complete. ■

Let us now present a Gentzen-like proof system for $PTL$. According to the notational conventions used in the literature we denote finite sets of formulas by $\Delta$, $\Gamma$, $\Pi$ and $\Sigma$. Similarly, by $\Delta, \alpha, \Gamma$ we mean set $\Delta \cup \{\alpha\} \cup \Gamma$. Thus a colon corresponds to the set-theoretical union. A semicolon will be used to separate different sequents.

**Definition 10.3.5** By Gentzen-like proof system $GP$ for logic $PTL$ we mean system $\langle GAx, G\rangle$ such that

- $GAx$ consists of a single (scheme of) axioms of the form $\vdash\ \Gamma \Rightarrow \Delta$ when $\Gamma \cap \Delta \neq \emptyset$
- $G$ consists of the rules for the classical propositional calculus (see section 5.3) together with the following (schemes of) derivation rules

$$(\bigcirc\neg l)\ \frac{\Gamma, \bigcirc(\neg\alpha) \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \bigcirc\alpha} \qquad (\bigcirc\neg r)\ \frac{\Gamma \Rightarrow \Delta, \bigcirc(\neg\alpha)}{\Gamma, \bigcirc\alpha \Rightarrow \Delta}$$

$$(\overleftarrow{\bigcirc}\neg l)\ \frac{\Gamma, \overleftarrow{\bigcirc}(\neg\alpha) \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \overleftarrow{\bigcirc}\alpha} \qquad (\overleftarrow{\bigcirc}\neg r)\ \frac{\Gamma \Rightarrow \Delta, \overleftarrow{\bigcirc}(\neg\alpha)}{\Gamma, \overleftarrow{\bigcirc}\alpha \Rightarrow \Delta}$$

$$(atNm)\ \frac{\Gamma\, \mathit{atNext}\, \alpha \Rightarrow \Delta\, \mathit{atNext}\, \alpha}{\Gamma \Rightarrow \Delta} \qquad (atPm)\ \frac{\Gamma\, \mathit{atPrev}\, \alpha \Rightarrow \Delta\, \mathit{atPrev}\, \alpha}{\Gamma \Rightarrow \Delta},$$

where $\Sigma\, \mathit{atNext}\, \gamma$ denotes set $\{\beta\, \mathit{atNext}\, \gamma|\ \beta \in \Sigma\}$ and $\Sigma\, \mathit{atPrev}\, \gamma$ denotes set $\{\beta\, \mathit{atPrev}\, \gamma|\ \beta \in \Sigma\}$

$$(\overleftarrow{\bigcirc}\bigcirc l)\frac{\overleftarrow{\bigcirc}\bigcirc\alpha, \Gamma \Rightarrow \Delta}{\alpha, \Gamma \Rightarrow \Delta} \qquad (\overleftarrow{\bigcirc}\bigcirc r)\frac{\Gamma \Rightarrow \Delta, \overleftarrow{\bigcirc}\bigcirc\alpha}{\Gamma \Rightarrow \Delta, \alpha}$$

$$(\bigcirc\overleftarrow{\bigcirc} l)\frac{\bigcirc\overleftarrow{\bigcirc}\alpha, \Gamma \Rightarrow \Delta}{\alpha, \Gamma \Rightarrow \Delta} \qquad (\bigcirc\overleftarrow{\bigcirc} r)\frac{\Gamma \Rightarrow \Delta, \bigcirc\overleftarrow{\bigcirc}\alpha}{\Gamma \Rightarrow \Delta, \alpha}$$

$$(atNl)\frac{\Gamma, \alpha\, \mathit{atNext}\, \beta \Rightarrow \Delta}{\Gamma, \bigcirc((\alpha \wedge \beta) \vee (\neg\beta \wedge \alpha\, \mathit{atNext}\, \beta)) \Rightarrow \Delta}$$

$$(atNr)\frac{\Gamma \Rightarrow \alpha\, \mathit{atNext}\, \beta, \Delta}{\Gamma \Rightarrow \Delta, \bigcirc((\alpha \wedge \beta) \vee (\neg\beta \wedge \alpha\, \mathit{atNext}\, \beta))}$$

$$(atNi)\frac{\alpha\, \mathit{atNext}\, \beta, \Gamma \Rightarrow \Delta}{\bigcirc(\alpha \wedge \beta) \Rightarrow \gamma;\ \bigcirc(\neg\beta \wedge \gamma) \Rightarrow \gamma;\ \Gamma, \gamma \Rightarrow \Delta}$$

$$(atPl)\frac{\Gamma, \alpha\, \mathit{atPrev}\, \beta \Rightarrow \Delta}{\Gamma, \overleftarrow{\bigcirc}((\alpha \wedge \beta) \vee (\neg\beta \wedge \alpha\, \mathit{atPrev}\, \beta)) \Rightarrow \Delta}$$

$$(atPr)\frac{\Gamma \Rightarrow \alpha\, \mathit{atPrev}\, \beta, \Delta}{\Gamma \Rightarrow \Delta, \overleftarrow{\bigcirc}((\alpha \wedge \beta) \vee (\neg\beta \wedge \alpha\, \mathit{atPrev}\, \beta))}$$

$$(atPi)\frac{\alpha\, \mathit{atPrev}\, \beta, \Gamma \Rightarrow \Delta}{\overleftarrow{\bigcirc}(\alpha \wedge \beta) \Rightarrow \gamma;\ \overleftarrow{\bigcirc}(\neg\beta \wedge \gamma) \Rightarrow \gamma;\ \Gamma, \gamma \Rightarrow \Delta}$$

$$(iF)\ \frac{\Gamma \Rightarrow \Delta, \bigcirc\alpha \to \Box\alpha}{\Gamma \Rightarrow \Delta, \Box(\alpha \to \bigcirc\alpha)} \qquad (iP)\ \frac{\Gamma \Rightarrow \Delta, \bigcirc\alpha \to \Box\alpha}{\Gamma \Rightarrow \Delta, \Box(\alpha \to \bigcirc\alpha)}\ \blacksquare$$

The following *cut rule* is useful, as it simplifies some derivations. Usually one avoids cut rule since it complicates the implementation of the proof system (because the formula $\gamma$ of the rule is to be guessed).

$$(cut)\ \frac{\Gamma\Rightarrow\Sigma}{\Gamma\Rightarrow\gamma;\quad\gamma\Rightarrow\Sigma}$$

**Example 10.3.6** Let us first show that the following rules can be derived in *GP*:

$$\frac{\Gamma\Rightarrow\Delta,\bigcirc(\alpha\wedge\beta)}{\Gamma\Rightarrow\bigcirc\alpha,\Delta;\ \Gamma\Rightarrow\bigcirc\beta,\Delta}\tag{10.12}$$

$$\frac{\Gamma,\bigcirc(\alpha\wedge\beta)\Rightarrow\Delta}{\Gamma,\bigcirc\alpha,\bigcirc\beta\Rightarrow\Delta}\tag{10.13}$$

Below $(pc)$ refers to the rules of the classical propositional calculus. Derivation of (10.12) is the following:

$$\frac{\Gamma\Rightarrow\Delta,\bigcirc(\alpha\wedge\beta)}{\dfrac{\Gamma\Rightarrow\bigcirc\alpha\wedge\bigcirc\beta,\Delta}{\Gamma\Rightarrow\bigcirc\alpha,\Delta;\ \Gamma\Rightarrow\bigcirc\beta,\Delta}(pc)\quad\dfrac{\bigcirc\alpha\wedge\bigcirc\beta\Rightarrow\bigcirc(\alpha\wedge\beta)}{\dfrac{\bigcirc\alpha,\bigcirc\beta\Rightarrow\bigcirc(\alpha\wedge\beta)}{\dfrac{\alpha,\beta\Rightarrow\alpha\wedge\beta}{\alpha,\beta\Rightarrow\alpha;\quad\alpha,\beta\Rightarrow\beta}(pc)}(atN)}(pc)}(cut)$$

Note that in application of rule $(atN)$ we use the definition of $\bigcirc$. The right-hand branch of this derivation reduces to axioms. Observe then that the conclusion of the derivation depends only on premises $\Gamma\Rightarrow\bigcirc p,\Delta$; $\Gamma\Rightarrow\bigcirc q,\Delta$. This justifies rule (10.12).

The derivation of rule (10.13) can look as follows (applications of rules of the classical propositional calculus are not marked here):

$$\frac{\Gamma,\bigcirc(\alpha\wedge\beta)\Rightarrow\Delta}{\dfrac{\Gamma\Rightarrow\Delta,\neg(\bigcirc\alpha\wedge\bigcirc\beta)}{\dfrac{\bigcirc\alpha\wedge\bigcirc\beta,\Gamma\Rightarrow\Delta}{\bigcirc\alpha,\bigcirc\beta,\Gamma\Rightarrow\Delta}}\quad\dfrac{\neg(\bigcirc\alpha\wedge\bigcirc\beta),\bigcirc(\alpha\wedge\beta)\Rightarrow}{\dfrac{\bigcirc(\alpha\wedge\beta)\Rightarrow\bigcirc\alpha\wedge\bigcirc\beta}{\dfrac{\bigcirc(\alpha\wedge\beta)\Rightarrow\bigcirc\alpha}{\dfrac{\alpha\wedge\beta\Rightarrow\alpha}{\alpha,\beta\Rightarrow\alpha}}(atN)\dfrac{\bigcirc(\alpha\wedge\beta)\Rightarrow\bigcirc\beta}{\dfrac{\alpha\wedge\beta\Rightarrow\beta}{\alpha,\beta\Rightarrow\beta}}(atN)}}}(cut)$$

As before, in order to complete the justification of rule (10.13) it now suffices to forget those branches of the proof that reduce to axioms. ■

**Example 10.3.7** Let us now show examples of two simple proofs of temporal tautologies $\neg\bigcirc\mathsf{False}$ and $\alpha\ \mathit{atNext}\,\beta \rightarrow \Diamond\beta$.

The proof of the first tautology can be the following:

$$\begin{array}{cl}
\Rightarrow \neg\bigcirc\mathsf{False} & \\ \hline
\Rightarrow \neg\bigcirc(\alpha \wedge \neg\alpha) & (def.\,of\ \mathsf{False}) \\ \hline
\bigcirc(\alpha \wedge \neg\alpha) \Rightarrow & (pc) \\ \hline
\bigcirc\alpha, \bigcirc(\neg\alpha) \Rightarrow & (10.13) \\ \hline
\bigcirc\alpha, \neg\bigcirc\alpha \Rightarrow & (\bigcirc\neg l) \\ \hline
\bigcirc\alpha \Rightarrow \bigcirc\alpha & (pc)
\end{array}$$

Note that the second tautology has already been proved in Example 10.3.2. Observe, however, how the former proof can be automated in Gentzen-like formalism.

$$\begin{array}{cl}
\Rightarrow \alpha\ \mathit{atNext}\,\beta \rightarrow \Diamond\beta & \\ \hline
\Rightarrow \alpha\ \mathit{atNext}\,\beta \rightarrow \mathsf{True}\ \mathit{atNext}\,\beta & (def.\,of\ \Diamond) \\ \hline
\alpha\ \mathit{atNext}\,\beta \Rightarrow \mathsf{True}\ \mathit{atNext}\,\beta & (pc) \\ \hline
\alpha \Rightarrow \mathsf{True} & (atNm)
\end{array}$$

The rest of the proof can be carried out by simple applications of rules for propositional calculus and the definition $\mathsf{True} \overset{\text{def}}{\equiv} (p \vee \neg p)$. ■

We have the following theorem.

**Theorem 10.3.8** Proof system $GP$ for propositional temporal logic $PTL$ is sound and complete. ■

## 10.4 Exercises

1. Define the following temporal operators using *atPrev* and *atNext*.

   - the operator *Since* with truth table $\exists t' < t.[P(t') \wedge \forall s.(t' < s < t \rightarrow Q(s))]$,
   - the operator *Until* with truth table $\exists t' > t.[P(t') \wedge \forall s.(t < s < t' \rightarrow Q(s))]$,
   - the operator $\overleftarrow{\Diamond}$ with truth table $\exists t' < t.[Q(t')]$,
   - the operator $\overleftarrow{\Box}$ with truth table $\forall t' < t.[Q(t')]$,
   - the operator $\Box$ with truth table $\forall t' > t.[Q(t')]$,
   - the operator $\widehat{\Box}$ with truth table $\forall t.[Q(t)]$,

- the operator $\widehat{\Diamond}$ with truth table $\exists t.[Q(t)]$.

2. Using the system HP and GP, prove that $\alpha \;\rightarrow\; \Box(\overleftarrow{\Diamond}\,\alpha)$ is a tautology of PTL.

3. Show that the rule $\alpha \vdash \overleftarrow{\Box}\alpha$ is derivable in $HP$.

4. Check the validity of the following formulas:

   - $\Box\alpha \rightarrow \Box(p \rightarrow \bigcirc\alpha)$
   - $\Box\alpha \rightarrow \Box\Diamond\alpha$
   - $\Box\Diamond\alpha \rightarrow \Diamond\Box\alpha$
   - $\Diamond\Box\alpha \rightarrow \Box\Diamond\alpha$.

# Chapter 11

# Logics of Programs

## 11.1 Introduction

Logics of programs play a similar rôle in computer science to that of the classical logic in "pure" mathematics. Classical formulae, however, mirror the static nature of mathematic notions. On the other hand, dynamic behavior of programs requires an another approach. Namely, the dynamic character of phenomena appearing in most areas of computer science have their counterparts in non-classical logics.

The history of development of logics of programs was initiated in the late sixties and may seem a rather short one. Nevertheless the research on logics of programs was very intensive. Many logics have been defined and investigated. They were strongly influenced by development of new programming tools for expanding applications of computers, and by essential progress in programming methodology. The explosion of various applications of computers resulted in development of many new programming concepts. Over fifteen hundred different programming languages have been defined. They were usually accompanied by less or more suitable axiom systems for proving correctness of programs. Thus one of the major trends in the area of logics of programs concerns proof systems that enable formal reasoning about program properties.

First-order logics of programs that are intended to express at least the most basic properties of programs as e.g. halting property, cannot be characterized completely (in classical sense) by finitistic proof systems. On the other hand, in order to stay within a finitary framework, one can try to weaken classical notion of completeness. Various non-classical notions of completeness were defined and new completeness proving techniques were developed. The most widely accepted non-classical notion of completeness is that of relative completeness defined by Cook. He separated the reasoning about programs from reasoning about first-order properties of underlying interpretation, and proved that the famous Hoare's system for proving partial correctness of programs is complete relative to the class of expressive interpretations.

**Definition 11.1.1** [Cook] We say that a proof system $P$ for a logic is *sound* (*complete*) *relative to the class $C$ of interpretations*, provided that for any $I \in C$ and any formula $\alpha$,

$Th_I \vdash \alpha$ implies (is implied by) $I \models \alpha$,

where $Th_I$ denotes the set of all classical first-order formulae valid in $I$. ∎

In fact, in the definition of Cook, $C$ is assumed to be the class of expressive interpretations.

Arithmetical completeness was derived from relative completeness by Harel in his works on dynamic logic. Harel restricts the class of admissible interpretations to those containing arithmetic of natural numbers. One can also consider a Henkin-like semantics, where non-standard models of computations are allowed.

Below we shall show proof systems that use a form of induction and are then close to Harel's ideas. The proof systems given below can be automatically obtained from a definition of their semantics[1]

## 11.2 Strictly arithmetical interpretations

Let us now discuss the class of admissible interpretations we consider here. First, we assume that s-arithmetical interpretation contains sort $\omega$ of natural numbers together with constants 0, 1 and functions $+$, $*$. Next note that programmers deal with potentially infinite data types whose elements, however, are represented by finitistic means. Queues, stacks, arrays, trees, symbols, etc. are always finite. We shall formulate this condition as assumption that for each sort there is a relation "encoding" its elements as natural numbers.

Let us note that we do not consider finite interpretations, for $\omega$ is infinite.

The following definition summarizes the discussion.

**Definition 11.2.1** First-order interpretation $I$ is called *strictly arithmetical* provided that:

1. $I$ contains sort $\omega$ of natural numbers together with constants 0, 1 and functions $+$, $*$ interpreted as usual
2. for each sort $s$ of $I$ there is an effective binary relation $e_s$ such that for each $x$ of sort $s$ there is exactly one $i \in \omega$ with $e_s(x, i)$ true in $I$. ∎

[1] The general method of obtaining such proof systems was worked out by the author of these notes and published in a series of papers.

Let us remark here that because of effectiveness of relation $e_s$ all sorts of $I$ are effective (i.e. some natural kind of Turing computability is defined on sorts of $I$). Note also that the class of strictly arithmetical interpretations are a proper subclass of arithmetical interpretations of Harel.

In order to simplify our considerations, in what follows we shall consider one-sorted strictly arithmetical interpretations with sort $\omega$, operations 0, 1, $+$, $*$ and additional functions having signature $\omega \to \omega$. In presence of encoding relations this can be done without loss of generality. Namely, functions and relations on sorts other than $\omega$ can be represented by functions with signature $\omega \to \omega$ or $\omega \to \{0, 1\}$, respectively.

**Definition 11.2.2** We say that a proof system $P$ for a logic is *strictly arithmetically sound* (*complete*) provided that it is sound (complete) relative to the class of strictly arithmetical interpretations. ■

The above definition differs from relative completeness of Cook and arithmetical completeness of Harel in class of admissible interpretations.

## 11.3 Algorithmic Logic

Algorithmic logic was introduced by Salwicki. Algorithmic logic is a multimodal logic, where modal operators are (labelled by) programs. Namely, if $P$ is a program and $\alpha$ is a formula, then $[P]\alpha$ is a formula, too. Its intended meaning is that program $P$ stops and its results satisfy formula $\alpha$.

**Definition 11.3.1** By *algorithmic logic* (AL) we shall mean the logic satisfying the following conditions:

1. non-classical connectives are of the form $[P]$, where $P$ is a program, i.e. an expression defined as follows:

$$\begin{array}{ll} \langle Program \rangle ::= & z := t \mid\mid \langle Program \rangle ; \langle Program \rangle \mid\mid \\ & if\ \gamma\ then\ \langle Program \rangle\ else\ \langle Program \rangle\ fi \mid\mid \\ & while\ \gamma\ do\ \langle Program \rangle\ od \end{array}$$

   where $z$ is a variable, $t$ is a term and $\gamma$ is an open formula,

2. the class of admissible interpretations is the class of classical first-order interpretations

3. the satisfiability relation of AL, $\models$, is defined as follows:

   - for classical connectives and first-order quantifiers $\models$ agrees with satisfiability relation of the predicate calculus

- $I, v \models [z := t]\alpha$ iff $I, v \models \alpha(z := t)$,
- $I, v \models [P; Q]\alpha$ iff $I, v \models [P]([Q]\alpha)$
- $I, v \models [\mathit{if}\ \gamma\ \mathit{then}\ P\ \mathit{else}\ Q\ \mathit{fi}]\alpha$ iff $I, v \models (\gamma \rightarrow [P]\alpha) \wedge (\neg\gamma \rightarrow [Q]\alpha)$
- $I, v \models [\mathit{while}\ \gamma\ \mathit{do}\ P\ \mathit{od}]\alpha$ iff there is $i \in \omega$ such that $I, v \models [P^i](\neg\gamma)$, and for the first such $i$ we have that $I, v \models [P^i]\alpha)$. ■

**Example 11.3.2** Observe that:

- $[P]\mathsf{True}$ expresses that program $P$ stops
- $\alpha \rightarrow [P]\beta$ expresses the *total correctness of program* $P$ w.r.t. $\alpha$ and $\beta$
- $(\alpha \wedge [P]\mathsf{True}) \rightarrow [P]\beta$ expresses the *partial correctness of program* $P$ w.r.t. $\alpha$ and $\beta$
- $\forall x.[y := 0;\ \mathit{while}\ y \neq x\ \mathit{do}\ y := y + 1\ \mathit{od}]\mathsf{True}$ over the domain of natural numbers expresses the fact that every natural number is "finite", i.e., can be obtained from 0 by a finite number of applications of the successor function.

Many other interesting properties of programs and data structures can also be expressed. ■

**Definition 11.3.3** Given a fixed strictly arithmetical interpretation, we define the following proof system for AL:

1. all instances of classical propositional tautologies
2. $\vdash_{AL} [z := t]\alpha \equiv \alpha(z := t)$ where $\alpha$ is an open formula
3. $\vdash_{AL} [P; Q]\alpha \equiv [P]([Q]\alpha)$
4. $\vdash_{AL} [\mathit{if}\ \gamma\ \mathit{then}\ P\ \mathit{else}\ Q\ \mathit{fi}]\alpha \equiv (\gamma \rightarrow [P]\alpha) \wedge (\neg\gamma \rightarrow [Q]\alpha)$
5. $(\alpha \wedge \neg\gamma) \rightarrow \delta,\ (\gamma \wedge [P]\delta) \rightarrow \delta,\ \delta \rightarrow \beta \vdash_{AL} [\mathit{while}\ \gamma\ \mathit{do}\ P\ \mathit{od}]\alpha \rightarrow \beta$
6. $\beta \rightarrow \exists n\, \delta(n),\ \delta(n+1) \rightarrow ((\alpha \wedge \neg\gamma) \vee (\gamma \wedge [P]\delta(n)),\ \neg\delta(0) \vdash_{AL}$
   $\beta \rightarrow [\mathit{while}\ \gamma\ \mathit{do}\ P\ \mathit{od}]\alpha,$
   where $n$ does not appear in $\gamma, \alpha$ and $P$
7. $\alpha, \alpha \rightarrow \beta \vdash_{AL} \beta$
   $\alpha \rightarrow \beta \vdash_{AL} \forall x.\alpha \rightarrow \forall x.\beta$
   $\alpha \rightarrow \beta \vdash_{AL} [P]\alpha \rightarrow [P]\beta.$ ■

The following theorem is a consequence of more general Theorem 12.7.6.

**Theorem 11.3.4** The proof system defined in Definition 11.3.3 is strictly arithmetically sound and complete. ■

## 11.4 Dynamic Logic

**Definition 11.4.1** By *dynamic logic* (DL) we shall mean the logic satisfying the following conditions:

(a) non-classical connectives are of the form $\langle P\rangle$, where $P$ is a program, i.e. an expression defined as follows:

$$\begin{array}{ll}\langle Program\rangle ::= & z := t \mid\mid \gamma? \mid\mid \langle Program\rangle;\langle Program\rangle \mid\mid \\ & \langle Program\rangle \cup \langle Program\rangle \mid\mid \langle Program\rangle^{*}\end{array}$$

where $z$ is a variable, $t$ is a term and $\gamma$ is an open formula,

(b) the class of admissible interpretations is the class of classical first-order interpretations

(c) the satisfiability relation of DL, $\models$, is defined as follows:

- for classical connectives and first-order quantifiers $\models$ agrees with satisfiability relation of classical first-order logic
- $I, v \models \langle z := t\rangle\alpha$ iff $I, v \models \alpha(z := t)$,
- $I, v \models \langle\gamma?\rangle\alpha$ iff $I, v \models \gamma \wedge \alpha$
- $I, v \models \langle P; Q\rangle\alpha$ iff $I, v \models \langle P\rangle(\langle Q\rangle\alpha)$
- $I, v \models \langle P \cup Q\rangle\alpha$ iff $I, v \models \langle P\rangle\alpha$ or $I, v \models \langle Q\rangle\alpha$
- $I, v \models \langle P^{*}\rangle\alpha$ iff there is $i \in \omega$ such that $I, v \models \langle P^{i}\rangle\alpha$. ■

Note that the main difference between AL and DL is that AL concerns deterministic programs, whilst DL also non-deterministic ones (due to program connectives $\cup$ and $^{*}$). Consequently, $\langle\rangle$ means modal possibility.

**Definition 11.4.2** By *concurrent dynamic logic* (CDL) we shall mean DL augmented with additional program connective $\cap$, and the following rule concerning its semantics:

$$I, v \models \langle P \cap Q\rangle\alpha \;\text{ iff }\; I, v \models \langle P\rangle\alpha \;\text{ and }\; I, v \models \langle Q\rangle\alpha.\blacksquare$$

**Definition 11.4.3** Given a fixed strictly arithmetical interpretation, we define the following proof system for (C)DL:

1. all instances of classical propositional tautologies
2. $\vdash_{(C)DL}< z := t > \alpha \equiv \alpha(z := t)$ where $\alpha$ is an open formula
3. $\vdash_{(C)DL}< \gamma? > \alpha \equiv \gamma \wedge \alpha$

4. $\vdash_{(C)DL} \langle P;Q\rangle\alpha \equiv \langle P\rangle(\langle Q\rangle\alpha)$
5. $\vdash_{(C)DL} < P \cup Q > \alpha \equiv < P > \alpha \vee < Q > \alpha$
6. $\vdash_{CDL} < P \cap Q > \alpha \equiv < P > \alpha \wedge < Q > \alpha$
7. $(\alpha \vee < P > \delta) \to \delta,\ \delta \to \beta \vdash_{(C)DL} < P^* > \alpha \to \beta$
8. $\beta \to \exists n\, \delta(n),\ \delta(n+1) \to (\alpha \vee < P > \delta(n)),\ \neg\delta(0) \vdash_{(C)DL} \beta \to < P^* > \alpha$
   where $n$ does not appear in $< P^* > \alpha$
9. $\alpha, \alpha \to \beta \vdash_{(C)DL} \beta$
   $\alpha \to \beta \vdash_{(C)DL} \forall x.\alpha \to \forall x.\beta$
   $\alpha \to \beta \vdash_{(C)DL} < P > \alpha \to < P > \beta$ ■

The following theorem is a consequence of more general Theorem 12.7.6.

**Theorem 11.4.4** The proof system defined in Definition 11.4.3 is strictly arithmetically sound and complete. ■

Gentzen-like rules for DL could include the following ones (see also section 12.7):

- $\dfrac{\Gamma, \langle P^*\rangle\alpha, \Sigma \Rightarrow \Delta}{\alpha \vee \langle P\rangle\delta \Rightarrow \delta;\ \Gamma, \delta, \Sigma \Rightarrow \Delta}$
- $\dfrac{\Gamma \Rightarrow \Sigma, \langle P^*\rangle\alpha, \Delta}{\delta(n \leftarrow n+1) \Rightarrow \alpha \vee \langle P\rangle\delta(n);\ \delta(n \leftarrow 0) \Rightarrow;\ \Gamma \Rightarrow \Sigma, \exists n(\delta(n)), \Delta}$,
  where n does not appear in $\langle P^*\rangle\alpha$.

## 11.5 Exercises

1. Prove rules 5, 6 of Definition 11.3.3.
2. Prove rules 7, 8 of Definition 11.4.3.
3. Check whether the following formulas are tautologies of AL:
   - $[P](\alpha \wedge \beta) \equiv ([P]\alpha \wedge [P]\beta)$
   - $[P](\alpha \vee \beta) \equiv ([P]\alpha \vee [P]\beta)$.
4. Check whether the following formulas are tautologies of (C)DL:
   - $\langle P\rangle(\alpha \vee \beta) \equiv (\langle P\rangle\alpha \wedge \langle P\rangle\beta)$
   - $\langle P\rangle(\alpha \vee \beta) \equiv (\langle P\rangle\alpha \vee \langle P\rangle\beta)$.
5. Express in AL and in (C)DL the following properties of stacks and queues:
   - every stack is finite
   - every queue is finite.

# Chapter 12

# Fixpoint Calculus

## 12.1 Introduction

Fixpoint calculus are important in computer science applications. First, the semantics of many computational processes have a nice fixpoint characterization. Second, fixpoint queries are computable in PTime. Moreover, as Immerman and Vardi proved, fixpoint calculus express all PTime computable queried over ordered databases.

This chapter is then devoted to fixpoint calculus and their applications.

## 12.2 Syntax of Fixpoint Calculus

Formulas of fixpoint calculus, denoted by $F_{\mathsf{X}}$, are defined by means of the following rules.

$$\begin{aligned}
\langle F_{\mathsf{X}}\rangle ::= \quad & \langle F_{\mathsf{I}}\rangle \mathrel{||} \mathsf{lfp}\ \langle \mathsf{Rel}\rangle.\langle F_{\mathsf{X}}\rangle \text{ where } \langle F_{\mathsf{X}}\rangle \text{ is positive w.r.t. } \langle \mathsf{Rel}\rangle \mathrel{||} \\
& \mathsf{gfp}\ \langle \mathsf{Rel}\rangle.\langle F_{\mathsf{X}}\rangle \text{ where } \langle F_{\mathsf{X}}\rangle \text{ is positive w.r.t. } \langle \mathsf{Rel}\rangle \mathrel{||} \\
& \neg\langle F_{\mathsf{X}}\rangle \mathrel{||} \langle F_{\mathsf{X}}\rangle \land \langle F_{\mathsf{X}}\rangle \mathrel{||} \langle F_{\mathsf{X}}\rangle \lor \langle F_{\mathsf{X}}\rangle \mathrel{||} \langle F_{\mathsf{X}}\rangle \to \langle F_{\mathsf{X}}\rangle \mathrel{||} \\
& \langle F_{\mathsf{X}}\rangle \equiv \langle F_{\mathsf{X}}\rangle \mathrel{||} \forall\langle V_{\mathsf{I}}\rangle.\langle F_{\mathsf{X}}\rangle \mathrel{||} \exists\langle V_{\mathsf{I}}\rangle.\langle F_{\mathsf{X}}\rangle \mathrel{||} (\langle F_{\mathsf{X}}\rangle) \mathrel{||} [\langle F_{\mathsf{X}}\rangle]
\end{aligned}$$

It is sometimes convenient to define more than one relation by means of fixpoint equations. This gives raise to so-called *simultaneous fixpoints* defined by allowing many relations as arguments of fixpoint operators. The syntax is then modified by assuming new syntax rules for fixpoint operators and leaving the other rules unchanged. The new rules are the following:

$$\mathsf{lfp}\ \langle \mathsf{Rel}\rangle\{,\langle \mathsf{Rel}\rangle\}.\langle F_{\mathsf{X}}\rangle \mid\mid \mathsf{gfp}\ \langle \mathsf{Rel}\rangle\{,\langle \mathsf{Rel}\rangle\}.\langle F_{\mathsf{X}}\rangle$$

where $\langle F_{\mathsf{X}}\rangle$ are positive w.r.t. all relations in $\langle \mathsf{Rel}\rangle\{,\langle \mathsf{Rel}\rangle\}$.

## 12.3 Semantics of Fixpoint Calculus

The semantics of $\mathsf{lfp}\ X.T(X)$ and $\mathsf{gfp}\ X.T(X)$ is the least and the greatest fixpoint of $T(X)$, i.e. the least and the greatest relation $X$ such that $X \equiv T(X)$. Since $T$ is assumed positive w.r.t. $X$, such fixpoints exist. More precisely, given a relational structure $\langle \mathsf{Dom}, \{f_i^{\mathsf{Dom}} : i \in I\}, \{R_j^{\mathsf{Dom}} : j \in J\}\rangle$, and any valuation $v : V_{\mathsf{I}} \longrightarrow \mathsf{Dom}$ can be extended to valuation $v : F_{\mathsf{X}} \longrightarrow \mathsf{Bool}$ as follows, assuming that first-order connectives and quantifiers are defined as in the case of predicate calculus:

$$\begin{array}{rcl}
v(\mathsf{lfp}\ X(\bar{x}).A(X)) & = & \text{the least (w.r.t. } \subseteq)) \text{ relation } S \text{ such that} \\
 & & S(x) \equiv v_S^X(A(X)) \\
v(\mathsf{gfp}\ X(\bar{x}).A(X)) & = & \text{the greatest (w.r.t. } \subseteq)) \text{ relation } S \text{ such that} \\
 & & S(x) \equiv v_S^X(A(X)).
\end{array}$$

The semantics of the least fixpoint operator $\mathsf{lfp}\ X_1, \ldots, X_n.T(X_1, \ldots, X_n)$ and the greatest fixpoint operator $\mathsf{gfp}\ X_1, \ldots, X_n.T(X_1, \ldots, X_n)$ is the tuple of the least and the greatest fixpoints of $T(X_1, \ldots, X_n)$, i.e. the least and the greatest relations $X_1, \ldots, X_n$ such that for all $i = 1, \ldots, n$, $X_i \equiv T(X_1, \ldots, X_n)$. Since $T$ is assumed positive w.r.t. $X_1, \ldots, X_n$, such fixpoints exist.

## 12.4 The Characterization of Fixpoints

Consider the following simultaneous fixpoint.

$$\mathsf{lfp}\ R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k).A(R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k))$$

Each $R_i$ is called the $i$-th coordinate of the fixpoint and also denoted by $A/i$.

Fixpoint formulas have a very nice computational characterization, which allows one to compute simultaneous fixpoints over databases in time polynomial in the size of the database[1]. Namely, given an extensional database $B$, we have the following definition of the least fixpoint,

$$\langle R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k)\rangle = \bigvee_{i \in \omega} A^i(\mathsf{False}, \ldots, \mathsf{False})$$

[1] In the case of infinite domains, $\omega$ is to be replaced by a suitable ordinal number.

where brackets $\langle , \rangle$ are used to denote a vector of relations, $\omega$ stands for the set of natural numbers,

$$A^0(\mathsf{False},\dots,\mathsf{False}) = \langle\underbrace{\mathsf{False},\dots,\mathsf{False}}_{k-\text{times}}\rangle$$
$$A^{i+1}(\mathsf{False},\dots,\mathsf{False}) = \langle A^i/1(\mathsf{False},\dots,\mathsf{False}),\dots,A^i/k(\mathsf{False},\dots,\mathsf{False})\rangle.$$

Similarly, we have the following definition of the greatest fixpoint,

$$\langle R_1(\bar{x}_1),\dots,R_k(\bar{x}_k)\rangle = \bigwedge_{i\in\omega} A^i(\mathsf{True},\dots,\mathsf{True})$$

where

$$A^0(\mathsf{True},\dots,\mathsf{True}) = \langle\underbrace{\mathsf{True},\dots,\mathsf{True}}_{k-\text{times}}\rangle$$
$$A^{i+1}(\mathsf{True},\dots,\mathsf{True}) = \langle A^i/1(\mathsf{True},\dots,\mathsf{True}),\dots,A^i/k(\mathsf{True},\dots,\mathsf{True})\rangle.$$

## 12.5 The Complexity of Fixpoint Calculus

**Theorem 12.5.1** Both checking whether a fixpoint formula is satisfiable or whether it is a tautology are not partially computable problems[2]. ■

**Theorem 12.5.2** Given a fixpoint formula, checking its satisfiability or validity over a given finite domain relational structure is in PTIME. ■

## 12.6 Fixpoint Calculus as a Query Language

**Example 12.6.1** The transitive closure of a binary relation $R$ can be defined by the following fixpoint formula:

$$\mathsf{TC}(R)(x,y) \equiv \mathsf{lfp}\ X(x,y).[R(x,y) \lor \exists z.(R(x,z) \land X(z,y))]. \blacksquare$$

**Example 12.6.2** Consider now the following example, where we are given a unary relation *Wise* and a binary relation *Colleague* defined on the set *Persons* and suppose we want to calculate the relation *Wisest* as the greatest relation satisfying the following constraint, meaning that *Wisest* are those who are wise and have only wisest colleagues:

$$\forall x.Wisest(x) \to (Wise(x) \land \forall y.(Colleague(x,y) \to Wisest(y))).$$

[2] In fact, as in the case of the second-order logic, the problem is even much more complex than partially computable problems or their complements.

The $Wisest$ relation is defined by the following fixpoint formula:

$$\mathsf{gfp}\ X(x).[\forall x.X(x) \to (Wise(x) \wedge \forall y.(Colleague(x,y) \to X(y)))].$$

since we are interested in calculating the greatest such relation. ■

**Example 12.6.3** The example concerns a game with states $a, b, \ldots$. The game is between two players. The possible moves of the games are held in a binary relation *moves*. A tuple $\langle a, b\rangle$ in *moves* indicates that when in a state $a$, one can chose to move to state $b$. A player loses if he or she is in a state from which there are no moves. The goal is to compute the set of winning states (i.e., the set of states such that there exists a winning strategy for a player in this state). These are obtained as the extension of a unary predicate *win*.

Let $M(x,y)$ and $W(x)$ denote the predicates *moves* and *win*, respectively.

Observe that $W$ satisfies the following conditions:

1. $\forall x[(\exists y M(x,y) \wedge \forall z \neg M(y,z)) \to W(x)]$, i.e. from $x$ there is a move to $y$ from which the opposing player has no move;
2. $\forall x[(\exists y M(x,y) \wedge \forall z(M(y,z) \to W(z))) \to W(x)]$, i.e. from $x$ there is a move to $y$ from which all choices of the opposing player lead to a state where the other player (the player that moved from $x$ to $y$) has a winning strategy.

Note that (1) is subsumed by (2), thus the definition of $W$ is given by,

$$W(x) \equiv \mu W(x).\{\exists y.[M(x,y) \wedge \forall z.(M(y,z) \to W(z))]\}. \ ■$$

## 12.7 Designing Proof Systems

By $M$ we shall denote an enumerable set of non-classical connectives. We assume that the connectives are unary. However, the approach can easily be generalized to many-argument non-classical connectives. In the sequel we shall always assume that a first-order signature is fixed. By $L$ we shall then denote the set of many-sorted classical first-order formulas.

**Definition 12.7.1** Let $M$ be an enumerable set of non-classical connectives. We form an *M-extension of classical first-order logic* ($M$-logic, in short) as triple $\langle L(M), C, \models\rangle$, where:

1. $L(M)$ is the set of formulas obtained from $L$ augmented with the following syntax rule:

$$\langle L(M)\rangle ::= \langle L\rangle || \langle M\rangle(\langle L(M)\rangle)$$

2. $C$ is a class of admissible interpretations (we assume $C$ is a subclass of classical first-order interpretations in relational structures)
3. $\models$ is a satisfiability relation that agrees with the classical one for classical first-order formulas. ■

The semantics of many logics can be characterized by means of fixpoint equations. The fixpoint calculus are an obvious example. Also many other logics can be characterized similarly.

**Example 12.7.2** For any first-order interpretation $I$ and valuation $v$ of free variables the following conditions hold:

- $I, v \models \langle P^*\rangle\alpha \equiv (\alpha \vee \langle P\rangle\langle P^*\rangle\alpha)$
- $I, v \models \alpha\, \mathit{atNext}\, \beta \equiv \bigcirc(\alpha \wedge B) \vee \bigcirc(\neg\beta \wedge \alpha\, \mathit{atNext}\, \beta)$ ■

The above example points out the most essential characterization of non-classical connectives in considered logics. Namely, equivalences given there have the following common form:

$$x \equiv \Gamma(x),$$

e.g, $x \equiv (\alpha \vee \langle P\rangle x)$.

**Definition 12.7.3** We say that set $\Gamma(M) = \{\Gamma_{m(A)}(x) \mid m \in M, A \in L(M)\}$ *defines set* $M$ of non-classical connectives of $M$-logic provided that the following conditions hold:

- for any first-order interpretation $I$ and valuation $v$ of free variables,
  - $I, v \models m(A) \equiv \Gamma_{m(A)}(m(A))$
  - $I, v \models m(A)$ iff there is $i \in \omega$ such that $I, v \models \Gamma^i_{m(A)}(\mathsf{False})$.
- there is a well-founded[3] relation $<_M$ on $M$ such that righthand sides of equalities defining functionals $\Gamma_{m(A)} \in \Gamma(M)$, contain (syntactically) only connectives less (w.r.t. $<_M$) than $m$. ■

**Definition 12.7.4**

- Given an $M$-logic, we shall say that set $M$ of non-classical connectives is *monotone* iff for any interpretation $I$, $m \in M$, and formulas $\alpha$, $\beta$:

$$I \models \alpha \to \beta \text{ implies } I \models m(\alpha) \to m(\beta)$$

[3] A binary relation is *well-founded* if there are no infinite decreasing chains w.r.t. the relation.

- Given an $M$-logic, we shall say that set $\Gamma$ of functionals is *monotone* iff for any interpretation $I$, functional $G \in \Gamma$, and formulas $\alpha$, $\beta$:
$$I \models \alpha \to \beta \text{ implies } I \models G(\alpha) \to G(\beta) \qquad \blacksquare$$

**Definition 12.7.5** By proof system $P_{(M,\Gamma)}$ for $(M, \Gamma)$-logic we shall mean the proof system containing the following axioms and inference rules:

(1) all instances of classical propositional tautologies

(2) for all $m \in M$ and formula $\alpha$ such that $\Gamma_{m(\alpha)}(x)$ is a constant functional (syntactically, i.e. functional containing no occurrences of $x$) we assume the following axiom:

(LR) $\vdash_{(M,\Gamma)} m(\alpha) \equiv \Gamma_{m(\alpha)}(\mathsf{False})$

(3) for all $m \in M$ other then those in (2) we assume the following inference rules:

(L) $\Gamma_{m(\alpha)}(\gamma) \to \gamma,\ \gamma \to \beta \vdash_{(M,\Gamma)} m(\alpha) \to \beta$

(R) $\beta \to \exists n\, \gamma(n),\ \gamma(n+1) \to \Gamma_{m(\alpha)}(\gamma(n)), \neg\gamma(0) \vdash_{(M,\Gamma)} \beta \to m(\alpha)$ where $n$ is a variable not appearing in $m(\alpha)$

(4) rules:

(MP) $\alpha, \alpha \to \beta \vdash_{(M,\Gamma)} \beta$

(G) $\alpha \to \beta \vdash_{(M,\Gamma)} \forall x.\alpha \to \forall x.\beta$

(M) $\alpha \to \beta \vdash_{(M,\Gamma)} m(\alpha) \to m(\beta)$ ■

Note that distinction between cases (2) and (3) is not essential. In fact, we could consider case (3) only, particular case of which is case (2). However, we make the distinction in order to obtain more elegant proof systems.

**Theorem 12.7.6** [Szałas] For any $(M, \Gamma)$-logic proof system $P_{(M,\Gamma)}$ is strictly arithmetically sound and complete. ■

**Definition 12.7.7** Let $\mathcal{L}$ be an $(M, \mathbf{G})$-logic. By $GP_{\mathcal{L}}$ we shall mean the following proof system

I. axioms:
$\vdash \Gamma \Rightarrow \Delta$, when $\Gamma \cap \Delta \neq \emptyset$

II. rules:

1. rules for the predicate calculus
2. for all $m \in M$ and formula $\alpha$ such that $G_{m(\alpha)}(x)$ is a constant functional (syntactically, i.e. functional containing no occurrences of $x$) we assume the following rules:

(a) $\dfrac{\Sigma, m(\alpha), \Gamma \Rightarrow \Delta}{\Sigma, G_{m(\alpha)}(\mathsf{False}), \Gamma \Rightarrow \Delta}$

(b) $\dfrac{\Gamma \Rightarrow \Sigma, m(\alpha), \Delta}{\Gamma \Rightarrow \Sigma, G_{m(\alpha)}(\mathsf{False}), \Delta}$

3. for all $m \in M$ other than those above we assume the following rules:

(a) $\dfrac{\Gamma, m(\alpha), \Sigma \Rightarrow \Delta}{G_{m(\alpha)}(\gamma) \Rightarrow \gamma;\ \Gamma, \gamma, \Sigma \Rightarrow \Delta}$

(b) $\dfrac{\Gamma \Rightarrow \Sigma, m(\alpha), \Delta}{\gamma(n+1) \Rightarrow G_{m(A)}(\gamma(n));\ \gamma(0) \Rightarrow;\ \Gamma \Rightarrow \Sigma, \exists n.(\gamma(n)), \Delta}$

where $n$ does not appear in $m(A)$. ■

Note that, as in the case of the Hilbert-like proof system, the rules $2(a)$ and $2(b)$ are special cases of rules $3(a)$ and $3(b)$.

**Theorem 12.7.8** [Szałas] For any $(M, \Gamma)$-logic proof system $GP_{(M,\Gamma)}$ is strictly arithmetically sound and complete. ■

## 12.8 A Fixpoint Approach to Quantifier Elimination

The idea that lead to Lemma 8.3.1 can be generalized by the use of fixpoint operators.

As an example consider the second–order formula

$$\exists P.[P(a) \wedge \neg P(b) \wedge \forall x.\forall y.(\neg P(x) \vee P(y) \vee N(x,y))].$$

The problem we have if we try to apply Lemma 8.3.1 is that we are not able to separate the positive from the negative occurrences of $P$ such that the requirements for the lemma are fulfilled. This is certainly not too surprising for otherwise we would be able to find an equivalent first–order formula which is impossible as Ackermann's result shows. The idea is therefore to describe these many conjunctive elements in a finite language and that with the help of fixpoint operators as follows:

$$\Big[\nu P(x).\ x \neq a \wedge \forall y\ P(y) \vee N(y,x)\Big]_b^x$$

where $[\Phi]_b^x$ is meant to express that every occurrence of $x$ in $\Phi$ is to be replaced by $b$.

**Theorem 12.8.1** [Nonnegart and Szałas] If $\Phi$ and $\Psi$ are positive w.r.t. $P$ then the closure ordinal for $\Phi(P)$ is less than or equal to $\omega$ and

$$\exists P\ \forall \overline{y}\ (P(\overline{y}) \rightarrow \Phi(P)) \wedge \Psi(P)\ \equiv\ \Psi\left[P(\overline{\alpha}) := \Big[\nu P(\overline{y}).\Phi(P)\Big]_{\overline{\alpha}}^{\overline{y}}\right]$$

and similarly for the case where the sign of $P$ is switched and $\Phi$ and $\Psi$ are negative w.r.t. $P$. ■

Note the strong similarities between Lemma 8.3.1 and Lemma 12.8.1. In fact, it can quite easily be observed that this fixpoint result subsumes the former result as described in Lemma 8.3.1 for in case that $\Phi$ does not contain any $P$ at all we have that $\nu P(\overline{y}).\Phi$ is equivalent to $\Phi$, hence Theorem 12.8.1 is a proper generalization of Lemma 8.3.1.

Again it is usually necessary to apply some equivalence preserving transformations in order to obtain a formula in the form required for applying Lemma 12.8.1. This can be done by the initial phases of the DLS algorithm (see section 8.3). Recall that the syntactic form required in Lemma 8.3.1 cannot always be obtained. This is not the case anymore for Lemma 12.8.1 for any formula can be transformed into the form required provided second–order Skolemization is allowed. This Skolemization evidently cannot be avoided in general for otherwise every second–order formula could be transformed into a (possibly infinite) first–order formula. Nevertheless, the lemma can always be applied and returns some result which is usually a fixpoint formula and sometimes another second–order formula. Such fixpoints can be tried to be simplified then and in particular in case where the fixpoint is bounded a final first–order result can be found.

The following example illustrates the use of Theorem 12.8.1 in case of the second-order induction axiom.

**Example 12.8.2** Consider the second-order induction, where $S(x, y)$ means that $y$ is a successor of $x$:

$$\forall P.[P(0) \land \forall x.\forall y.((P(x) \land S(x, y)) \to P(y))] \to \forall z.P(z) \tag{12.1}$$

Let us now transform the formula into the form required in Theorem 12.8.1 and then apply the theorem[4]:

Negated —
$\exists P.[P(0) \land \forall x.\forall y.((P(x) \land S(x, y)) \to P(y))] \land \exists z.\neg P(z)$
Transformed into the conjunctive normal form —
$\exists z.\exists P.\forall y.[(y \neq 0 \lor P(y)) \land \forall x.(\neg P(x) \lor \neg S(x, y) \lor P(y))] \land \neg P(z)$
Transformed into the form required in Theorem 12.8.1 —
$\exists z.\exists P.\forall y.[P(y) \lor (y \neq 0 \land \forall x.(\neg S(x, y) \lor \neg P(x)))] \land \neg P(z)$
After application of Theorem 12.8.1 —
$\exists z.\nu\neg P(z).(z \neq 0 \land \forall x.[\neg S(x, z) \lor \neg P(x))]$
Unnegated —
$\forall z.\mu P(z).[z = 0 \lor \exists x.(S(x, z) \land P(x))]$.

[4] We use the well-known equivalence $(p \to q) \equiv (\neg p \lor q)$

By Theorem 12.8.1 we have that the above formula is equivalent to the following infinite disjunction:

$$\forall z. \bigvee_{i\in\omega} [z = 0 \vee \exists x.(S(x,z) \wedge P(x))]^i(\mathsf{False}).$$

Denote by $\Phi(\Psi(z))$ the formula $z = 0 \vee \exists x.(S(x,z) \wedge \Psi(x))$. Now

$\Phi^0(\mathsf{False}) \;\equiv\; \mathsf{False}$
$\Phi^1(\mathsf{False}) \;\equiv\; [z = 0 \vee \exists x.(S(x,z) \wedge \mathsf{False})] \;\equiv\; [z = 0 \vee \mathsf{False}] \;\equiv$
$\quad [z = 0]$
$\Phi^2(\mathsf{False}) \;\equiv\; \Phi(\Phi^1(\mathsf{False})) \;\equiv\; [z = 0 \vee \exists x.(S(x,z) \wedge x = 0)] \;\equiv$
$\quad [z = 0 \vee S(0,z)]$
$\Phi^3(\mathsf{False}) \;\equiv\; \Phi(\Phi^2(\mathsf{False})) \;\equiv$
$\quad [z = 0 \vee \exists x.(S(x,z) \wedge (x = 0 \vee S(0,x)))] \;\equiv$
$\quad [z = 0 \vee S(0,z) \vee S^2(0,z)]$
...
$\Phi^i(\mathsf{False}) \;\equiv\; [z = 0 \vee S(0,z) \vee ... \vee S^{i-1}(0,z)].$

Thus, by a simple calculation we obtain the well-known, but not trivial fact that the second-order induction (12.1) is equivalent to $\forall z. \bigvee_{i\in\omega}[S^i(0,z)]$, i.e. "every natural number is obtained from 0 by a finite number of applications of the successor relation". In fact, the other Peano axioms say that the successor relation is a function, etc. ■

Let us now show some correspondence-theoretical applications of Theorem 12.8.1.

**Example 12.8.3** Consider the Löb Axiom

$$\Box(\Box P \to P) \;\to\; \Box P \tag{12.2}$$

Translated:
$\forall P.\forall x.[\forall y.(R(x,y) \to \forall z.(R(y,z) \to P(z)) \to P(y))] \to$
$\qquad \forall u.(R(x,u) \to P(u))$
After negation and elimination of →:
$\exists P.\exists x.[\forall y.(\neg R(x,y) \vee \exists z.(R(y,z) \wedge \neg P(z) \vee P(y)))] \wedge$
$\qquad \exists u.(R(x,u) \wedge \neg P(u))$
Transformed into the form required in Theorem 12.8.1:
$\exists x.\exists P.[\forall y.(P(y) \vee \neg R(x,y) \vee \exists z.(R(y,z) \wedge \neg P(z)))] \wedge$
$\qquad \exists u.(R(x,u) \wedge \neg P(u))$
After application of Theorem 12.8.1:
$\exists x.\exists u.R(x,u) \wedge \nu\neg P(u).[\neg R(x,u) \vee \exists z.(R(u,z) \wedge \neg P(z))]$
Unnegated:
$\forall x.\forall u.R(x,u) \to \mu P(u).[R(x,u) \wedge \forall z.(R(u,z) \to P(z))]$

Denote by $\Phi(\Psi(u))$ the formula $R(x,u) \wedge \forall z.(R(u,z) \rightarrow \Psi(z))$. Now

$\Phi^0(\mathsf{False}) \;\equiv\; \mathsf{False}$
$\Phi^1(\mathsf{False}) \;\equiv\; R(x,u) \wedge \forall z.\neg R(u,z)$
$\Phi^2(\mathsf{False}) \;\equiv\; \Phi(\Phi(\mathsf{False})) \;\equiv\; \Phi(R(x,u) \wedge \forall z.\neg R(u,z)) \equiv$
$R(x,u) \wedge \forall z.(R(u,z) \rightarrow (R(x,z) \wedge \forall z_1.\neg R(z,z_1))))$
$\Phi^3(\mathsf{False}) \;\equiv\; \Phi(\Phi^2(\mathsf{False})) \;\equiv$
$\Phi(R(x,u) \wedge \forall z.(R(u,z) \rightarrow (R(x,z) \wedge \forall z_1.\neg R(z,z_1)) \equiv$
$R(x,u) \wedge \forall z.(R(u,z) \rightarrow R(x,z) \wedge \forall z_2.(R(z,z_2) \rightarrow (R(x,z_2) \wedge \forall z_1.\neg R(z_2,z_1))))$
...

Thus the Löb axiom (12.2) is equivalent to

$$\forall x.\forall u.R(x,u) \rightarrow \bigvee_{i \in \omega} \Phi^i(\mathsf{False}),$$

which expresses that the relation $R$ is transitive and reverse well-founded. Transitivity of $R$ can be seen by unfolding the fixpoint two times and the reverse well-foundness of $R$ then follows from the simplification of $\forall x.\forall u.R(x,u) \rightarrow \mu P(u).[R(x,u) \wedge \forall z.(R(u,z) \rightarrow P(z))]$ to $\forall x.\forall u.R(x,u) \rightarrow \mu P(u).[\forall z.R(u,z) \rightarrow P(z)]$ under the transitivity assumption. ■

**Example 12.8.4** Consider the temporal logic formula

$$\Box(p \rightarrow \bigcirc p) \;\rightarrow\; (p \rightarrow \Box p). \tag{12.3}$$

where $\Box$ should be interpreted as *always* or *henceforth* and $\bigcirc$ as *at the next moment of time*.

Translated:
$\forall P.\forall x.[\forall y.(R_{\Box}(x,y) \rightarrow (P(y) \rightarrow \forall z.(R_{\bigcirc}(y,z) \rightarrow P(z))))] \rightarrow$
$\qquad [P(x) \rightarrow \forall u.(R_{\Box}(x,u) \rightarrow P(u))]$

Negated:
$\exists x.\exists P.[\forall y.(R_{\Box}(x,y) \rightarrow (P(y) \rightarrow \forall z.(R_{\bigcirc}(y,z) \rightarrow P(z))))] \wedge$
$\quad [P(x) \wedge \exists u.(R_{\Box}(x,u) \wedge \neg P(u))]$
Transformed into the conjunctive normal form:
$\exists x.\exists u.\exists P.\forall y.\forall z.(P(z) \vee \neg R_{\Box}(x,y) \vee \neg R_{\bigcirc}(y,z) \vee \neg P(y)) \wedge$
$\quad \forall z.(P(z) \vee x \neq z) \wedge R_{\Box}(x,u) \wedge \neg P(u)$
Transformed into the form required in the Theorem 12.8.1:
$\exists x.\exists u.\exists P.\forall z.[P(z) \vee (x \neq z \wedge \forall y.(\neg R_{\Box}(x,y) \vee \neg R_{\bigcirc}(y,z) \vee \neg P(y))] \wedge$
$\quad R_{\Box}(x,u) \wedge \neg P(u)$
After application of Theorem 12.8.1:
$\exists x.\exists u.R_{\Box}(x,u) \wedge$
$\qquad \nu \neg P(u).(x \neq u \wedge \forall y.(\neg R_{\Box}(x,y) \vee \neg R_{\bigcirc}(y,u) \vee \neg P(y))]$

Unnegated:

$$\begin{aligned}&\forall x.\forall u.R_{\Box}(x,u) \rightarrow\\ &\qquad \mu P(u).(x=u \vee \exists y.(R_{\Box}(x,y) \wedge R_{\bigcirc}(y,u) \wedge P(y)).\end{aligned}$$

Thus formula (12.3) is equivalent to the following one:

$$\begin{aligned}\forall u.\forall x.R_{\Box}(u,x) \Rightarrow\ &\{R_{\Box}(u,u) \wedge [u=x \vee R_{\bigcirc}(u,x) \vee\\ &\textstyle\bigvee_{i\in\omega} \exists v_0 \ldots \exists v_i.(R_{\Box}(u,v_0) \wedge \ldots \wedge R_{\Box}(u,v_i) \wedge\\ &R_{\bigcirc}(u,v_0) \wedge R_{\bigcirc}(v_0,v_1) \wedge \ldots \wedge R_{\bigcirc}(v_{i-1},v_i) \wedge R_{\bigcirc}(v_i,x))]\}.\end{aligned}$$

i.e. this formula states that $R_{\Box}$ is the reflexive and transitive closure of $R_{\bigcirc}$, a property which is not expressible by means of the classical logic but expressible by means of the fixpoint logic. ■

## 12.9 Exercises

1. Let $B(b,x,y)$ means that there is a connection between places $x$ and $y$ by bus $b$ and let $T(t,x,y)$ means that there is a connection between places $x$ and $y$ by train $t$. Formulate the following queries:

   (a) Is there a connection between $x$ and $y$ by a single bus or a single train?

   (b) Is there a connection between $x$ and $y$ by bus lines only?

   (c) Is there a connection between $x$ and $y$ combining arbitrary trains and busses?

2. Consider the following axiom schema, which is sometimes called the *modified Löb Axiom*:

   $$\Box(P \rightarrow \Diamond P) \ \rightarrow\ (\Diamond P \rightarrow \Box\Diamond P).$$

   Find a fixpoint correspondence of the schema.

3. Consider the axiom schema

   $$\Box(\Box P \rightarrow P) \rightarrow P.$$

   Show that no Kripke frame validates the schema. Hint: find the fixpoint correspondence and show that the resulting fixpoint formula is equivalent to $\mathsf{False}$.

4. Design a strictly arithmetically complete proof system for fixpoint calculus.

# Chapter 13

# Rough Concepts

## 13.1 Introduction

In many AI applications, a precise set or relation is either unknown, or to complex to represent. In such cases one often approximates the set/relation.

The lower approximation of a concept represents points that are known to be part of the concept, the boundary region represents points that might or might not be part of the concept, and the complement of the upper approximation represents points that are known not to be part of the concept.

The rough set philosophy is founded on the assumption that we associate some information (data, knowledge) with every object of the universe of discourse. This information is often formulated in terms of attributes about objects. Objects characterized by the same information are interpreted as indiscernible (similar) in view of the available information about them. An indiscernibility relation, generated in this manner from the attribute/value pairs associated with objects, provides the mathematical basis of rough set theory.

Any set of all indiscernible (similar) objects is called an *elementary set*, and forms a basic granule (atom) of knowledge about the universe. Any union of some elementary sets in a universe is referred to as a *crisp (precise) set*; otherwise the set is referred to as being a *rough (imprecise, vague) set*. In the latter case, two separate unions of elementary sets can be used to approximate the imprecise set, as we have seen in the example above.

Consequently, each rough set has what are called *boundary-line cases*, i.e., objects which cannot with certainty be classified either as members of the set or of its complement. Obviously, crisp sets have no boundary-line elements at all. This means that boundary-line cases cannot be properly classified by employing only the available information about objects.

The assumption that objects can be observed only through the information available about them leads to the view that knowledge about objects has granular structure. Due to this granularity, some objects of interest cannot always be discerned given the information available, therefore the objects appear as the same (or similar). As a consequence, vague or imprecise concepts, in contrast to precise concepts, cannot be characterized solely in terms of information about their elements since elements are not always discernable from each other. In the proposed approach, we assume that any vague or imprecise concept is replaced by a pair of precise concepts called the lower and the upper approximation of the vague or imprecise concept. The lower approximation consists of all objects which with certainty belong to the concept and the upper approximation consists of all objects which have a possibility of belonging to the concept.

The difference between the upper and the lower approximation constitutes the boundary region of a vague or imprecise concept. Additional information about attribute values of objects classified as being in the boundary region of a concept may result in such objects being re-classified as members of the lower approximation or as not being included in the concept. Upper and lower approximations are two of the basic operations in rough set theory.

## 13.2 Information Systems and Indiscernibility

One of the basic fundaments of rough set theory is the indiscernibility relation which is generated using information about particular objects of interest. Information about objects is represented in the form of a set of attributes and their associated values for each object. The indiscernibility relation is intended to express the fact that, due to lack of knowledge, we are unable to discern some objects from others simply by employing the available information about those objects. In general, this means that instead of dealing with each individual object we often have to consider clusters of indiscernible objects as fundamental concepts of our theories.

Let us now present this intuitive picture about rough set theory more formally.

**Definition 13.2.1** An *information system* is any pair $\mathcal{A} = \langle U, A \rangle$ where $U$ is a non-empty finite set of *objects* called the *universe* and $A$ is a non-empty finite set of *attributes* such that $a : U \to V_a$ for every $a \in A$. The set $V_a$ is called the *value set* of $a$. By $Inf_B(x) = \{(a, a(x)) : a \in B\}$, we denote the *information signature of $x$ with respect to $B$*, where $B \subseteq A$ and $x \in U$. ■

Note that in this definition, attributes are treated as functions on objects, where $a(x)$ denotes the value the object $x$ has for the attribute $a$.

Any subset $B$ of $A$ determines a binary relation $IND_{\mathcal{A}}(B) \subseteq U \times U$, called an indiscernibility relation, defined as follows.

**Definition 13.2.2** Let $\mathcal{A} = \langle U, A \rangle$ be an information system and let $B \subseteq A$.

By the *indiscernibility relation determined by* $B$, denoted by $IND_{\mathcal{A}}(B)$, we understand the relation

$$IND_{\mathcal{A}}(B) = \{(x, x') \in U \times U : \forall a \in B.[a(x) = a(x')]\}.$$

If $(x, y) \in IND_{\mathcal{A}}(B)$ we say that $x$ and $y$ are *B-indiscernible*. Equivalence classes of the relation $IND_{\mathcal{A}}(B)$ (or blocks of the partition $U/B$) are referred to as *B-elementary sets*. The unions of $B$-elementary sets are called *B-definable sets*. ■

Observe that $IND_{\mathcal{A}}(B)$ is an equivalence relation. Its classes are denoted by $[x]_B$. By $U/B$ we denote the partition of $U$ defined by the indiscernibility relation $IND(B)$.

In the rough set approach the elementary sets are the basic building blocks (concepts) of our knowledge about reality.

The ability to discern between perceived objects is also important for constructing many entities like reducts, decision rules, or decision algorithms which will be considered in later sections. In the classical rough set approach the *discernibility relation*, $DIS_{\mathcal{A}}(B)$, is defined as follows.

**Definition 13.2.3** Let $\mathcal{A} = \langle U, A\rangle$ be an information system and $B \subseteq A$. The *discernibility relation* $DIS_{\mathcal{A}}(B) \subseteq U \times U$ is defined by $xDIS(B)y$ if and only if $not(xIND(B)y)$. ■

We now consider how to express (define) sets of objects using formulas constructed from attribute/value pairs. The simplest type of formula consisting of one attribute/value pair is called an *elementary descriptor*.

**Definition 13.2.4** An *elementary descriptor* is any formula of the form $a = v$ where $a \in A$ and $v \in V_a$. A *generalized descriptor* is any formula of the form $\bigvee_{i=1}^{n} a = v_i$, where $a \in A$ and each $v_i \in V_a$. A *boolean descriptor* is any boolean combination of elementary or generalized descriptors.

Let $S = \{v_1, \ldots v_n\} \subseteq V_a$. We will sometimes use the following notation, $a \doteq S$ or $a \doteq \{v_1, \ldots v_n\}$ as an abbreviation for $\bigvee_{i=1}^{n} a = v_i$. ■

Let $\varphi$ be a boolean descriptor. The meaning of $\varphi$ in $\mathcal{A}$, denoted $\|\varphi\|_{\mathcal{A}}$, is defined inductively as follows:

1. if $\varphi$ is of the form $a = v$ then $\|\varphi\|_{\mathcal{A}} = \{x \in U : a(x) = v\}$ ;
2. $\|\varphi \wedge \varphi'\|_{\mathcal{A}} = \|\varphi\|_{\mathcal{A}} \cap \|\varphi'\|_{\mathcal{A}}$
   $\|\varphi \vee \varphi'\|_{\mathcal{A}} = \|\varphi\|_{\mathcal{A}} \cup \|\varphi'\|_{\mathcal{A}}$
   $\|\neg\varphi\|_{\mathcal{A}} = U - \|\varphi\|_{\mathcal{A}}$.

**Definition 13.2.5** Any set of objects $X \subseteq U$ definable in $\mathcal{A}$ by some formula $\varphi$ (i.e., X=$\|\varphi\|_{\mathcal{A}}$) is referred to as a *crisp* (*precise, exact*) set; otherwise the set is referred to as a *rough* (*imprecise, inexact, vague*) set . ■

Observe that vague concepts cannot be represented only by means of crisp concepts, although they can be *approximated* by crisp concepts.

## 13.3 Approximations and Rough Sets

Let us now define approximations of sets in the context of information systems.

**Definition 13.3.1** Let $\mathcal{A} = \langle U, A\rangle$ be an information system, $B \subseteq A$ and $X \subseteq U$. The *B-lower approximation* and *B-upper approximation* of $X$, denoted by $X_{B^+}$ and $X_{B^\oplus}$ respectively, are defined by $X_{B^+} = \{x : [x]_B \subseteq X\}$ and $X_{B^\oplus} = \{x : [x]_B \cap X \neq \emptyset\}$. ■

The $B$-lower approximation of $X$ is the set of all objects which can be classified with certainty as belonging to $X$ just using the attributes in $B$ to discern distinctions.

**Definition 13.3.2** The set consisting of objects in the *B-lower approximation* $X_{B^+}$ is also called the *B-positive region of* $X$. The set $X_{B^-} = U - X_{B^\oplus}$ is called the *B-negative region of* $X$. The set $X_{B^\pm} = X_{B^\oplus} - X_{B^+}$ is called the *B-boundary region of* $X$. ■

Observe that the positive region of $X$ consists of objects that can be classified with certainty as belonging to $X$ using attributes from $B$. The negative region of $X$ consists of those objects which can be classified with certainty as not belonging to $X$ using attributes from $B$. The $B$-boundary region of $X$ consists of those objects that cannot be classified unambiguously as belonging to $X$ using attributes from $B$.

The size of the boundary region of a set can be used as a measure of the quality of that set's approximation (relative to $U$). One such measure is defined as follows.

**Definition 13.3.3** The *accuracy of approximation* is defined in terms of the following coefficient,

$$\alpha_B(X) = \frac{|X_{B^+}|}{|X_{B^\oplus}|},$$

where $|X|$ denotes the cardinality of $X \neq \emptyset$. ■

It is clear that $0 \leq \alpha_B(X) \leq 1$. If $\alpha_B(X) = 1$ then $X$ is crisp with respect to $B$ ($X$ is precise with respect to $B$); otherwise, if $\alpha_B(X) < 1$ then $X$ is rough with respect to $B$ ($X$ is vague or imprecise with respect to $B$).

## 13.4 Decision Systems

Rough set techniques are often used as a basis for supervised learning using tables of data. In many cases the target of a classification task, that is, the family of concepts to be approximated, is represented by an additional attribute called a decision attribute. Information systems of this kind are called decision systems.

**Definition 13.4.1** Let $\langle U, A\rangle$ be an information system. A *decision system* is any system of the form $\mathcal{A} = \langle U, A, d\rangle$, where $d \notin A$ is the *decision attribute* and $A$ is a set of *conditional attributes*, or simply *conditions*. ■

Let $\mathcal{A} = \langle U, A, d\rangle$ be given and let $V_d = \{v_1, \ldots, v_{r(d)}\}$. Decision $d$ determines a partition $\{X_1, \ldots, X_{r(d)}\}$ of the universe $U$, where $X_k = \{x \in U \ : \ d(x) = v_k\}$ for $1 \leq k \leq r(d)$. The set $X_i$ is called the $i$-th *decision class* of $\mathcal{A}$. By $X_{d(u)}$ we denote the decision class $\{x \in U \ : \ d(x) = d(u)\}$, for any $u \in U$.

One can generalize the above definition to the case of decision systems of the form $\mathcal{A} = \langle U, A, D\rangle$ where the set $D = \{d_1, ...d_k\}$ of decision attributes and $A$ are assumed to be disjoint. Formally this system can be treated as the decision system $\mathcal{A} = \langle U, C, d_D\rangle$ where $d_D(x) = (d_1(x), ..., d_k(x))$ for $x \in U$.

A decision table can be identified as a representation of raw data (or training samples in machine learning) which is used to induce concept approximations in a process known as supervised learning. Decision tables themselves are defined in terms of decision systems. Each row in a decision table represents one training sample. Each column in the table represents a particular attribute in $A$, with the exception of the 1st column which represents objects in $U$ and selected columns representing the decision attribute(s).

The formulas consisting only of descriptors containing condition (decision) attributes are called *condition formulas* of $\mathcal{A}$ (*decision formulas* of $\mathcal{A}$).

Any object $x \in U$ belongs to a *decision class* $\| \bigwedge_{a \in D} a = a(x) \|_{\mathcal{A}}$ of $\mathcal{A}$. All decision classes of $\mathcal{A}$ create a partition of the universe $U$.

A *decision rule* for $\mathcal{A}$ is any expression of the form $\varphi \Rightarrow \psi$, where $\varphi$ is a condition formula, $\psi$ is a decision formula, and $\|\varphi\|_{\mathcal{A}} \neq \emptyset$. formulas $\varphi$ and $\psi$ are referred to as the *predecessor* and the *successor* of the decision rule $\varphi \Rightarrow \psi$. Decision rules are often called "$IF \ldots THEN \ldots$" rules.

A decision rule $\varphi \Rightarrow \psi$ is *true* in $\mathcal{A}$ if and only if $\|\varphi\|_{\mathcal{A}} \subseteq \|\psi\|_{\mathcal{A}}$; otherwise, one can measure its *truth degree* by introducing some inclusion measure of $\|\varphi\|_{\mathcal{A}}$ in $\|\psi\|_{\mathcal{A}}$. For example, one such measure which is widely used, is called a *confidence coefficient* and is defined as,

$$\frac{|\|\varphi \wedge \psi\|_{\mathcal{A}}|}{|U|}.$$

Another measure of non-classical inclusion called *support of the rule*, is defined as,

$$\frac{|\|\varphi\|_{\mathcal{A}}|}{|U|}.$$

Each object $x$ in a decision table determines a decision rule,

$$\bigwedge_{a \in C} a = a(x) \Rightarrow \bigwedge_{a \in D} a = a(x).$$

Decision rules corresponding to some objects can have the same condition parts but different decision parts. Such rules are called *inconsistent* (*nondeterministic*, *conflicting*, *possible*); otherwise the rules are referred to as *consistent* (*certain*, *sure*, *deterministic*, *nonconflicting*) rules. Decision tables containing inconsistent decision rules are called *inconsistent* (*nondeterministic*, *conflicting*); otherwise the table is *consistent* (*deterministic*, *nonconflicting*).

In machine learning and pattern recognition it is often necessary to induce concept approximations from a set of learning samples together with decision attribute values for each sample. The concept approximations generated can be used as the basis for additional algorithms which classify previously unobserved objects not listed in the original decision tables. Classifiers in such algorithms can be represented by sets of decision rules together with mechanisms for resolving conflicts between decision rules which provide us with different decisions for the same object.

Numerous methods based on the rough set approach combined with boolean reasoning techniques have been developed for decision rule generation. When a set of rules have been induced from a decision table containing a set of training examples, they can be inspected to determine if they reveal any novel relationships between attributes that are worth pursuing further. In addition, the rules can be applied to a set of unseen cases in order to estimate their classification power.

## 13.5 Dependency of Attributes

An important issue in data analysis is to discover dependencies between attributes. Intuitively, a set of attributes $D$ depends totally on a set of attributes $C$, denoted $C \Rightarrow D$, if the values of attributes from $C$ uniquely determine the values of attributes from $D$. In other words, $D$ depends totally on $C$, if there exists a functional dependency between values of $C$ and $D$.

Formally, a dependency between attributes can be defined as follows.

**Definition 13.5.1** Let $\mathcal{A} = \langle U, A \rangle$, and $D$ and $C$ be subsets of $A$. We say that $D$ *depends on* $C$ to *degree* $k$ $(0 \leq k \leq 1)$, denoted by $C \Rightarrow_k D$, if

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|},$$

where

$$POS_C(D) = \bigcup_{X \in U/D} X_{C^+},$$

called a *positive region of the partition* $U/D$ with respect to $C$, is the set of all elements of $U$ that can be uniquely classified to blocks of the partition $U/D$, by means of $C$. The coefficient $k$ is called the *degree of the dependency.*

If $k = 1$ we say that $D$ *depends totally* on $C$, and if $k < 1$, we say that $D$ *depends partially* (to *degree* $k$) on $C$. ■

The coefficient $k$ expresses the ratio of all elements of the universe which can be properly classified to blocks of the partition $U/D$ employing attributes $C$, to all elements of the universe.

If $D$ depends totally on $C$ then $IND(C) \subseteq IND(D)$. This means that the partition generated by $C$ is finer than the partition generated by $D$. Notice, that the concept of dependency discussed above corresponds to that considered in relational databases. Definition 13.5.1 is a central concept and is used to define the notion of reduct in the next section.

In summary, $D$ is totally (partially) dependent on $C$, if all (some) elements of the universe $U$ can be uniquely classified to blocks of the partition $U/D$, employing $C$.

## 13.6 Reduction of Attributes

A question often arises as to whether one can remove some data from a data table and still preserve its basic properties, that is, whether a table containing superfluous data can be optimized by removal of redundant data. Let us express this more precisely.

**Definition 13.6.1** Let $\mathcal{A} = (U, A, D)$ be a decision system and $C \subseteq A$, $E \subseteq D$ be sets of condition and decision attributes, respectively. We will say that $C' \subseteq C$ is a *E-reduct* (reduct with respect to $E$) of $C$, if $C'$ is a minimal subset of $C$ such that $\gamma(C, E) = \gamma(C', E)$.

The intersection of all $E$-reducts is called a *E-core* (core with respect to $E$). ■

Essentially, the definition states that if we disregard some subset of the attributes of $C$, this will not allow us to classify more or less objects as being in $E$ equivalence classes in the partition $U/E$.

Because the core is the intersection of all reducts, it is included in every reduct, i.e., each element of the core belongs to some reduct. Thus, in a sense, the core is the most important subset of attributes, since none of its elements can be removed without affecting the classification power of attributes.

One can consider preserving less restrictive constraints than $\gamma(C, E) = \gamma(C', E)$ when trying to reduce conditional attributes . For example, one can require $|\gamma(C, E) - \gamma(C', E)| < \epsilon$, for a given threshold $\epsilon > 0$. The reduction of conditional attributes preserving such constraints results in reduct approximation.

Many other kinds of reducts and their approximations are discussed in the literature. It turns out that they can be efficiently computed using heuristics based on Boolean reasoning techniques.

## 13.7 Representing Rough Concepts in Predicate Calculus

Let us now discuss the concept of rough sets from the point of view of the logical calculus we use in other parts of the book.

In order to construct a language for dealing with rough concepts, we introduce the following relation symbols for any rough relation $R$:

- $R^+$ – represents the positive facts known about the relation. $R^+$ corresponds to the lower approximation of $R$. $R^+$ is called the *positive region (part)* of $R$.

- $R^-$ – represents the negative facts known about the relation. $R^-$ corresponds to the complement of the upper approximation of $R$. $R^-$ is called the *negative region (part)* of $R$.

- $R^\pm$ – represents the unknown facts about the relation. $R^\pm$ corresponds to the set difference between the upper and lower approximations to $R$. $R^\pm$ is called the *boundary region (part)* of $R$.

- $R^\oplus$ – represents the positive facts known about the relation together with the unknown facts. $R^\oplus$ corresponds to the upper approximation to $R$. $R^\oplus$ is called the *positive-boundary region (part)* of $R$.

- $R^\ominus$ – represents the negative facts known about the relation together with the unknown facts. $R^\ominus$ corresponds to the lower approximation of the complement of $R$. $R^\ominus$ is called the *negative-boundary region (part)* of $R$.

From the logical point of view, elementary sets can be represented by means of logical formulas or primitive relations, assuming their extensions form a partition of the universe. Assume we are given elementary sets defined by formulas $\{\alpha_1(\bar{x}_1), \ldots, \alpha_n(\bar{x}_n)\}$. Any relation can now be approximated as follows:

$$R^+(\bar{x}) \stackrel{\text{def}}{\equiv} \bigvee\{\alpha_i : 1 \leq i \leq n \wedge \forall\bar{x}\forall\bar{x}_i.(\alpha_i(\bar{x}_i) \to R(\bar{x}))\}$$
$$R^\oplus(\bar{x}) \stackrel{\text{def}}{\equiv} \bigvee\{\alpha_j : 1 \leq j \leq n \wedge \exists\bar{x}\exists\bar{x}_j.(R(\bar{x}) \wedge \alpha_j(\bar{x}_j))\}.$$

# 13.8 Rough Deductive Databases

## 13.8.1 The Language of Extensional Databases

The extensional database consists of positive and negative facts. We thus assume that the language of the extensional database is a set of literals, i.e. formulas of the form $R(\bar{c})$ or $\neg R(\bar{c})$, where $R$ is a relation symbol and $\bar{c}$ is a tuple of constant symbols.

## 13.8.2 The Language of Intensional Databases

The intensional database is intended to infer new facts, both positive and negative via application of intensional rules to the EDB. The rules have the form,

$$\pm P(\bar{x}) \leftarrow \pm P_1(\bar{x}_1), \ldots \pm P_k(\bar{x}_k), \tag{13.1}$$

where ± is either the empty string or the negation symbol ¬ and any variable that appears in a head of a rule (i.e. any variable of $\bar{x}$ in a rule of the form (13.1)) appears also in the rule's body (i.e. among variables of $\bar{x}_1, \ldots \bar{x}_k$ in the rule).

The rules can be divided into two layers, the first for inferring positive and the second for inferring negative facts. The first layer of rules (called the *positive IDB rule layer*), used for inferring positive facts, has the form,

$$P(\bar{x}) \leftarrow \pm P_1(\bar{x}_1), \ldots \pm P_k(\bar{x}_k) \tag{13.2}$$

while the second layer of rules (called the *negative IDB rule layer*), used for inferring negative facts, has the following form,

$$\neg P(\bar{x}) \leftarrow \pm P_1(\bar{x}_1), \ldots \pm P_k(\bar{x}_k) \tag{13.3}$$

## 13.8.3 The Semantics of Extensional Databases

The semantics of the extensional database is given by rough sets of tuples. Let $R()$ be a relational symbol appearing in the extensional database. Then $R()$ is interpreted as the rough set whose positive part contains all tuples $v(\bar{c})$ for which literal $R(\bar{c})$ is in the database and the negative part contains all tuples $v(\bar{c})$ for which literal $\neg R(\bar{c})$ is in the database. All other tuples are in the boundary region of $R()$.

$$\begin{aligned}
&EDB \Vvdash R(\bar{a}) \text{ iff } R(\bar{a}) \in EDB^{+}(R),\\
&EDB \Vvdash \neg R(\bar{a}) \text{ iff } \neg R(\bar{a}) \in EDB^{-}(R),
\end{aligned}$$

where $R()$ is a relation of the EDB and $\bar{a}$ is a tuple of constants.

Rough relations for the EDB are then defined as follows:

$$\begin{aligned}
R^{+}_{EDB} &= \{v(\bar{a}) : EDB \Vvdash R(\bar{a})\}\\
R^{-}_{EDB} &= \{v(\bar{a}) : EDB \Vvdash \neg R(\bar{a})\}\\
R^{\pm}_{EDB} &= \{v(\bar{a}) : EDB \not\Vvdash R(\bar{a}) \text{ and } EDB \not\Vvdash \neg R(\bar{a})\}.
\end{aligned}$$

It is important to observe here, that consistency is not required here.

### 13.8.4 The Semantics of Intensional Databases

The semantics of the intensional database is given by rough sets of tuples after application of the intensional rules to the extensional database.

In order to provide the semantics of IDB we require the definition of so-called Feferman-Gilmore translation.

**Definition 13.8.1** By a *Feferman-Gilmore translation of formula $\alpha$*, denoted by $FG(\alpha)$, we mean the formula obtained from $\alpha$ by replacing all negative literals of the form $\neg R(\bar{y})$ by $R^{-}(\bar{y})$ and all positive literals the form $R(\bar{y})$ by $R^{+}(\bar{y})$. $\blacksquare$

Let $\bar{S} = (S_1, \ldots, S_p)$ contain all relation symbols of the form $R^{+}$ and $R^{-}$, where $R$ is a relation symbol occurring in an IDB rule. For any relation $S_i$, all rules with $S_i^{+}$ (respectively $S_i^{-}$) in their heads should be gathered into a single formula of the form

$$\forall \bar{y}_i.[S_i^{\pm}(\bar{y}_i) \leftarrow \alpha_i(\bar{y}_i)]$$

where

$$\alpha_i(\bar{y}_i) \equiv \bigvee_j \exists \bar{z}_j.\beta_{ij}(\bar{z}_j)$$

where $\beta_{ij}(\bar{z}_j)$ denotes the bodies of all the appropriate rules and ± stands for + or −, respectively.

Define $\bar{S}_{IDB} \equiv \mu\bar{S}.[FG(\alpha_1), \ldots, FG(\alpha_p)]$, where $\mu$ denotes the least simultaneous fixpoint operator. In some cases the IDB might appear inconsistent. This happens when there is a relation $R()$ such that $R^+ \cap R^- \neq \emptyset$. In what follows we require that the IDB is consistent, i.e. for all IDB relations $R()$ we have that $R^+ \cap R^- = \emptyset$. This consistency criterion can be verified in time polynomial in the size of the database.

The semantics of IDB rules are then defined as follows:

$$\begin{aligned} &IDB \Vdash R(\bar{a}) \text{ iff } \bar{a} \in EDB^+(R) \cup IDB^+(R),\\ &IDB \Vdash \neg R(\bar{a}) \text{ iff } \bar{a} \in EDB^-(R) \cup IDB^-(R),\end{aligned}$$

where $R()$ is a relation in the EDB or in the head of an intensional rule, $\bar{a}$ is a tuple of constants, and $IDB^+(R)$ and $IDB^-(R)$ are computed from the simultaneous fixpoint definition $\bar{S}_{IDB}$ defined above.

Rough relations for the IDB are then defined as follows:

$$\begin{aligned} R^+_{IDB} &= \{v(\bar{a}) : IDB \Vdash R(\bar{a})\}\\ R^-_{IDB} &= \{v(\bar{a}) : IDB \Vdash \neg R(\bar{a})\}\\ R^{\pm}_{IDB} &= \{v(\bar{a}) : IDB \nVdash R(\bar{a}) \text{ and } IDB \nVdash \neg R(\bar{a})\}.\end{aligned}$$

Observe that,

$$\begin{aligned} &EDB \Vdash R(\bar{a}) \text{ implies} IDB \Vdash R(\bar{a})\\ &EDB \Vdash \neg R(\bar{a}) \text{ implies} IDB \Vdash \neg R(\bar{a}).\end{aligned}$$

**Example 13.8.2** Consider the following set of rules:

$$\begin{aligned} &A_1.Q^+(n)\\ &A_2.R^+(n)\\ &A_4.P^-(x) \leftarrow A_3.P^-(x)\\ &A_4.P^-(x) \leftarrow A_2.R^+(x), \neg A_3.P^+(x)\\ &A_4.P^+(x) \leftarrow A_3.P^+(x)\\ &A_5.P^-(x) \leftarrow A_3.P^-\\ &A_5.P^+(x) \leftarrow A_3.P^+(x)\\ &A_5.P^+(x) \leftarrow A_1.Q^+(x), \neg A_3.P^-(x),\end{aligned}$$

where it is assumed that relations are distributed among agents and that $A.R$ denotes relation $R$ coming from agent $A$.

The relations $A_1.Q^+$, $A_2.R^+$, $A_4.P^-$, $A_4.P^+$, $A_5.P^-$, and $A_5.P^+$, occurring in the heads of the above rules, are defined by the fixpoint formula given by

$$\mathsf{lfp}\ A_1.Q^+, A_2.R^+, A_4.P^-, A_4.P^+, A_5.P^-, A_5.P^+.\mathcal{R} \tag{13.4}$$

where $\mathcal{R}$ denotes the conjunction of the bodies of rules.

Applying the fixpoint computation procedure, we can compute the relations characterized by the simultaneous fixpoint formula (13.4):

$$\begin{aligned}&\{\}\\ &\{A_1.Q^+(n), A_2.R^+(n)\}\\ &\{A_1.Q^+(n), A_2.R^+(n), A_4.P^-(n), A_5.P^+(n)\}.\end{aligned}$$

Assume that the query of interest is $P(n)$. Agent $A_4$ answers False and agent $A_5$ answers True to the query. ■

## 13.9 Exercises

1. Consider the following decision table:

| $a_1$ | $a_2$ | $a_3$ | $d_1$ | $d_2$ |
|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |

   - Find all $\{d_1\}$-reducts and $\{d_1\}$-core.
   - Find all $\{d_2\}$-reducts and $\{d_2\}$-core.

2. Create decision rules for $d_1$ and $d_2$ based on the decision table of exercise 1.

3. Define a structure of rough database and exemplary rules for recognizing small cars.

# Bibliography

[BDRS95] L. Bolc, K. Dziewicki, P. Rychlik, and A. Szałas. *Reasoning in non-classical logics. Theoretical Foundations.* Academic Pub. PLJ, Warsaw, In Polish, 1995.

[BDRS98] L. Bolc, K. Dziewicki, P. Rychlik, and A. Szałas. *Reasoning in non-classical logics. Automated Methods.* Academic Pub. PLJ, Warsaw, In Polish, 1998.

[DŁS97] P. Doherty, W. Łukaszewicz, and A. Szałas. Computing circumscription revisited. *Journal of Automated Reasoning*, 18(3):297–336, 1997.

[DŁSS02] P. Doherty, W. Łukaszewicz, A. Skowron, and A. Szałas. *Knowledge Engineering. A Rough Sets Approach.* Springer Physica Verlag, to appear, 2002.

[EFT94] H-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Springer-Verlag, Heidelberg, 1994.

[NOS99] A. Nonnengart, H.J. Ohlbach, and A. Szałas. Elimination of predicate quantifiers. In H.J. Ohlbach and U. Reyle, editors, *Logic, Language and Reasoning. Essays in Honor of Dov Gabbay, Part I*, pages 159–181. Kluwer, 1999.

[Sza92] A. Szałas. *Introduction to automated deduction.* Academic Pub. RM, Warsaw, In Polish, 1992.

[Sza95] A. Szałas. Temporal logic: a standard approach. In L. Bolc and A. Szałas, editors, *Time and Logic. A Computational Approach*, pages 1–50, London, 1995. UCL Press Ltd.

# Index