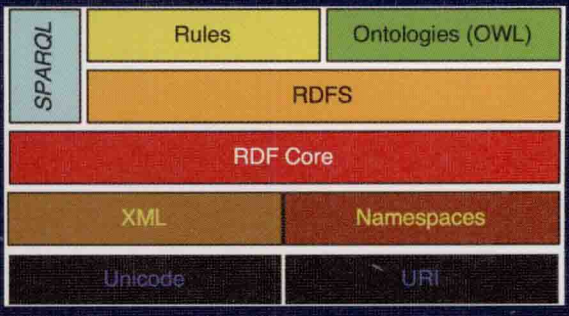
Grigoris Antoniou Uwe Aßmann
Cristina Baroglio Stefan Decker
Nicola Henze Paula-Lavinia Patranjan
Robert Tolksdorf (Eds.)

Tutorial

LNCS 4636

Reasoning Web

Third International Summer School 2007
Dresden, Germany, September 2007
Tutorial Lectures



Springer

Grigoris Antoniou Uwe Aßmann
Cristina Baroglio Stefan Decker
Nicola Henze Paula-Lavinia Patranjan
Robert Tolksdorf (Eds.)

Reasoning Web

Third International Summer School 2007
Dresden, Germany, September 3-7, 2007
Tutorial Lectures

Springer

Volume Editors

Grigoris Antoniou
FORTH, Heraklion, Greece
E-mail: antoniou@ics.forth.gr

Uwe Aßmann
Technische Universität Dresden, Germany
E-mail: uwe.assmann@inf.tu-dresden.de

Cristina Baroglio
Università degli Studi di Torino, Italy
E-mail: baroglio@di.unito.it

Stefan Decker
DERI and National University of Ireland, Galway
E-mail: stefan.decker@deri.org

Nicola Henze
Universität Hannover, Germany
E-mail: henze@kbs.uni-hannover.de

Paula-Lavinia Patranjan
Ludwig-Maximilians-Universität München, Germany
E-mail: paula.patranjan@ifi.lmu.de

Robert Tolksdorf
Freie Universität Berlin, Germany
E-mail: tolk@inf.fu-berlin.de

Library of Congress Control Number: 2007933835

CR Subject Classification (1998): H.4, H.3, C.2, H.5, J.1, K.4, K.6, I.2.11

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-74613-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74613-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12115853 06/3180 5 4 3 2 1 0

Preface

The summer school series Reasoning Web focuses on theoretical foundations, current approaches, and practical solutions for reasoning in a Web of Semantics. It has established itself as a meeting point for experts from research and industry, and students undertaking their PhDs in related fields. This volume contains the tutorial notes of the Reasoning Web summer school 2007, which was held in Dresden, Germany, in September 2007. This summer school was the third school of the Reasoning Web series, following the very successful predecessors held in Malta and Lisbon.

The first part of the 2007 edition is devoted to "Fundamentals of Reasoning and Reasoning Languages" and surveys concepts and methods for rule-based query languages. Further, it gives a comprehensive introduction to description logics and its usage.

Reactive rules and rule-based policy representation are covered in the second part on "Rules and Policies." A thorough discussion on the importance of rule interchange in the Web and promising solution strategies is given together with an overview on current W3C initiatives.

Finally, the third part is devoted to "Applications of Semantic Web Reasoning," and demonstrates practical uses of Semantic Web reasoning. The academics viewpoint is presented by a contribution on reasoning in Semantic Wikis. The industry's viewpoint is presented by contributions that discuss the importance of semantic technologies for search solutions for enterprises, for creating an enterprise knowledge base with Semantic Wiki representations, and Semantic Web Service discovery and selection in B2B scenarios.

We would like to thank all lecturers of the Reasoning Web summer school 2007 for giving interesting and inspiring tutorials. Further, we thank the local organizers in Dresden for their efficient and great work, and Norbert Eisinger from the Ludwig-Maximilians-Universität München and Jan Małuszyński from the University of Linköping, they made our job as Program Committee members very enjoyable and smooth.

September 2007

Grigoris Antoniou
Uwe Aßmann
Cristina Baroglio
Stefan Decker
Nicola Henze
Paula-Lavinia Pătrânjan
Robert Tolksdorf

Organization

Program Committee

Grigoris Antoniou, Information Systems Laboratory – FORTH, Greece
Uwe Aßmann, Technische Universität Dresden, Germany
Cristina Baroglio, Università degli Studi di Torino, Italy
Stefan Decker, DERI and National University of Ireland, Galway, Ireland
Nicola Henze, Leibniz Universität Hannover, Germany (Chair)
Paula-Lavinia Pătrânjan, Ludwig-Maximilians-Universität München, Germany
Robert Tolksdorf, Freie Universität Berlin, Germany

Local Organization

Michael Schroeder, Transinsight and Technische Universität Dresden, Germany

Applications Chair

Cristina Baroglio, Università degli Studi di Torino, Italy

Proceedings Chair

Paula-Lavinia Pătrânjan, Ludwig-Maximilians-Universität München, Germany

Sponsoring Institutions

Technische Universität Dresden

Network of Excellence REWERSE

Transinsight

Author Index

Table of Contents

Fundamentals of Reasoning and Reasoning Languages

Rules, Rule Languages, and Policies

Applications of Semantic Web Reasoning

Foundations of Rule-Based Query Answering

François Bry[1], Norbert Eisinger[1], Thomas Eiter[2], Tim Furche[1], Georg Gottlob[2,3], Clemens Ley[1], Benedikt Linse[1], Reinhard Pichler[2], and Fang Wei[2]

[1] Institute for Informatics, University of Munich, Oettingenstraße 67, D-80538 München, Germany
http://www.pms.ifi.lmu.de/
[2] Institute of Information Systems, Vienna University of Technology, Favoritenstraße 11/184-3, A-1040 Vienna, Austria
http://www.kr.tuwien.ac.at/, http://www.dbai.tuwien.ac.at/
[3] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, England
http://web.comlab.ox.ac.uk/oucl/people/georg.gottlob.html

Abstract. This survey article introduces into the essential concepts and methods underlying rule-based query languages. It covers four complementary areas: declarative semantics based on adaptations of mathematical logic, operational semantics, complexity and expressive power, and optimisation of query evaluation.

The treatment of these areas is foundation-oriented, the foundations having resulted from over four decades of research in the logic programming and database communities on combinations of query languages and rules. These results have later formed the basis for conceiving, improving, and implementing several Web and Semantic Web technologies, in particular query languages such as XQuery or SPARQL for querying relational, XML, and RDF data, and rule languages like the "Rule Interchange Framework (RIF)" currently being developed in a working group of the W3C.

Coverage of the article is deliberately limited to declarative languages in a classical setting: issues such as query answering in F-Logic or in description logics, or the relationship of query answering to reactive rules and events, are not addressed.

1 Introduction

The foundations of query languages mostly stem from logic and complexity theory. The research on query languages has enriched these two fields with novel issues, original approaches, and a respectable body of specific results. Thus, the foundations of query languages are arguably more than applications of these two fields. They can be seen as a research field in its own right with interesting results and, possibly, even more interesting perspectives. In this field, basic and applied research often are so tightly connected that distinguishing between the two would be rather arbitrary. Furthermore, this field has been very lively since

G. Antoniou et al. (Eds.): Reasoning Web 2007, LNCS 4636, pp. 1–153, 2007.

the late 1970s and is currently undergoing a renaissance, the Web motivating query and rule languages with novel capabilities. This article aims at introducing into this active field of research.

Query languages have emerged with database systems, greatly contributing to their success, in the late 1970s. First approaches to query languages were inspired by mathematical logic. As time went by, query languages offering syntactical constructs and concepts that depart from classical logic were being developed, but still, query languages kept an undeniably logical flavour. The main strengths of this flavour are: compound queries constructed using connectives such as "and" and "or"; rules expressed as implications; declarative semantics of queries and query programs reminiscent of Tarski's model-theoretic truth definition; query optimisation techniques modelled on equivalences of logical formulas; and query evaluators based on methods and heuristics similar to, even though in some cases simpler than, those of theorem provers.

With the advent of the Web in the early 1990s things have changed. Query languages are undergoing a renaissance motivated by new objectives: Web query languages have to access structured data that are subject to structural irregularities – so-called "semi-structured data" – to take into account rich textual contents while retrieving data, to deliver structured answers that may require very significant reorganisations of the data retrieved, and to perform more or less sophisticated forms of automated reasoning while accessing or delivering meta-data. All these issues have been investigated since the mid 1990s and still are. Further issues of considerable relevance, which, up till now, have received limited attention, include: query processing in a distributed and decentralised environment, query languages for search engines, search as a query primitive, and semantical data alignment.

The current query language renaissance both, takes distance from the logical setting of query languages, and builds upon it. On the one hand, recent Web query languages such as XPath and XQuery seem to be much less related to logic than former relational query languages such as SQL and former object-oriented query languages such as OQL. On the other hand, expressly logic-based Web query languages such as the experimental language Xcerpt [28,141,30,29] have been proposed, and Semantic Web query languages such as RQL, RDQL, and SPARQL clearly have logical roots (see [14,73] for surveys on Web and Semantic Web query languages). Furthermore, language optimisers and evaluators of XPath and XQuery exploit techniques formerly developed, thus bringing these languages back to the logical roots of query languages. At the beginning of this ongoing query language renaissance, a principled and summarised presentation of query language foundations surely makes sense.

1.1 What Are Query Languages? Tentative Definitions

A first definition of what query languages are considers what they are used for: they can be defined as specialised programming languages for selecting and retrieving data from "information systems". These are (possibly very large) data repositories such as file systems, databases, and (all or part of) the World Wide

Web. Query languages are specialised inasmuch as they are simpler to use or offer only limited programming functionalities that aim at easing the selection and retrieval of data from information systems.

A second attempt at defining what query languages are is to consider their programming paradigms, i.e., the brand of programming languages they belong to: query languages are declarative languages, i.e., languages abstracting out how (query) programs are to be evaluated. This makes query languages both easier to use – an advantage for the human user – and easier to optimise – an advantage for the computer. The declarativity of query languages is the reason for their close relationship to logic: declarative languages are all in some way or other based on logic.

A third approach to define what query languages are considers their major representatives: SQL for relational databases, OQL for object-oriented databases, XPath and XQuery for HTML and XML data, and RQL, RDQL, and SPARQL for RDF data. Forthcoming are query languages for OWL ontologies. Viewed from this angle, what have query languages in common? First, a separation between query programs and accessed data, requiring to compile query programs without any knowledge at all or with only limited knowledge of the data the compiled query programs are to access. Second, a dedication to data models, many, if not all, of which are strongly rooted in logic.

Query languages, as a research field, can also be defined by the issues being investigated. Central issues in query languages research include:

- query paradigms (e.g., visual, relational, object-oriented, and navigational query languages),
- declarative semantics,
- complexity and expressive power,
- procedural semantics,
- implementations of query evaluators,
- query optimisation (e.g., equivalence of queries).

Further query language issues include: integrity constraints (languages, expressive power, complexity, evaluation, satisfiability, maintenance); incremental or distributed evaluation of queries; evaluation of queries against data streams; storage of large collections of queries (e.g., for publish-subscribe systems); approximate answers; query answering in presence of uncertain information; query answering in presence of inconsistent information; querying special data (e.g., constraints, spatial data, graphic data); algorithms and data structures for efficient data storage and retrieval.

1.2 Coverage of This Survey

This survey article on the foundations of query languages is focused on logic, complexity and expressive power, and query optimisation. The reasons for such an admittedly limited focus are manifold. First, this focus arguably provides with a corner stone for most of the past and current research on query languages. Second, this focus covers a rather large field that could hardly be enlarged in

a survey and introductory article. Third, such a focus provides with a unity of concerns and methods.

1.3 Structure of This Survey

This survey article is organised as follows. Section 1 is this introduction. Section 2 introduces a few general mathematical notions that are used in later sections. Section 3 is devoted to syntax. It introduces the syntax of classical first-order predicate logic,[1] then of fragments of first-order predicate logic that characterise classes of query languages. This section shows how the syntax of the various practical query languages can be conveyed by the syntax of first-order predicate logic. Section 4 introduces into classical first-order model theory, starting with Tarski model theory, the notion of entailment, Herbrand interpretations, and similar standard notions, explaining the relevance of these notions to query languages. After this main part, the section covers Herbrand model theory and finite model theory, which have a number of interesting and rather surprising properties that are relevant to query languages. Section 5 then treats the adaptations of classical model theory to query and rule languages, covering minimal model semantics and fixpoint semantics and discussing approaches to the declarative semantics of rule sets with negation. A subsection on RDF model theory rounds out this section. Sections 6 and 7 introduce into the operational semantics of query programs, considering positive rule sets and rule sets with non-monotonic negation, respectively. These two sections present (terminating) algorithms for the (efficient) evaluation of query programs of various types. Section 8 is devoted to complexity and expressive power of query language fragments. Section 9 introduces into query optimisation, successively considering query containment, query rewriting, and query algebras.

The purpose of Sections 3 and 4 is to make the article self-contained. Therefore these sections are entirely expository. They should make it possible for readers with limited knowledge of mathematical logic – especially of classical first-order predicate logic – to understand the foundations of query languages. For those readers who already have some of that knowledge and mathematical practice, the sections should help recall notions and state terminologies and notations.

2 Preliminaries

2.1 General Mathematical Notions

By a *function* we mean, unless otherwise stated, a *total function.*

We consider zero to be the smallest *natural number.* The set of natural numbers is $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

Definition 1 (Enumerable). *A set S is called* enumerable*, if there is a surjection $\mathbb{N} \to S$. A set S is called* computably enumerable *(or* recursively enumerable*), if it is enumerable with a surjection that is computable by some algorithm.*

[1] Sometimes called simply "first-order logic", a short form avoided in this article.

Note that any finite set is enumerable and computably enumerable. The infinite set of all syntactically correct C programs is computably enumerable and thus enumerable. Its subset consisting of all syntactically correct C programs that do not terminate for each input is enumerable, but not computably enumerable.

2.2 Logic vs. Logics

The development of logic started in antiquity and continued through mediaeval times as an activity of philosophy aimed at analysing rational reasoning. In the late 19th century parts of logic were mathematically formalised, and in the early 20th century logic turned into a tool used in a (not fully successful) attempt to overcome a foundational crisis of mathematics. The fact that logic is not restricted to analysing reasoning in mathematics became somewhat eclipsed during those decades of extensive mathematisation, but came to the fore again when computer science discovered its close ties to logic. Today, logic provides the foundations in many areas of computer science, such as knowledge representation, database theory, programming languages, and query languages.

Logic is concerned with statements, which are utterances that may be true or false. The key features of logic are the use of formal languages for representing statements (so as to avoid ambiguities inherent to natural languages) and the quest for computable reasoning about those statements. *"Logic"* is the name of the scientific discipline investigating such formal languages for statements, but any of those languages is also called *"a logic"* – logic investigates logics.

3 Syntax: From First-Order Predicate Logic to Query Language Fragments of First-Order Predicate Logic

This section introduces the syntax of *first-order predicate logic*, which is the most prominent of logics (formal languages) and occupies a central position in logic (the scientific discipline) for several reasons: it is the most widely used and most thoroughly studied logic; it is the basis for the definition of most other logics; its expressive power is adequate for many essential issues in mathematics and computer science; its reasoning is computable in a sense to be made precise in Section 4; it is the most expressive logic featuring this kind of computability [110].

Practical query and rule languages depart from first-order predicate logic in many respects, but nonetheless they have their roots in and can conveniently be described and investigated in first-order predicate logic.

Subsection 3.1 below contains the standard definitions of first-order predicate logic syntax. The second subsection 3.2 discusses fragments (or sublanguages) of first-order predicate logic that correspond to common query or rule languages. The last subsection 3.3 discusses several modifications of the standard syntax that are used in some areas of computer science.

3.1 Syntax of First-Order Predicate Logic

First-order predicate logic is not just a single formal language, because some of its symbols may depend on the intended applications. The symbols common to

all languages of first-order predicate logic are called *logical symbols*, the symbols that have to be specified in order to determine a specific language are called the *signature* (or *vocabulary*) of that language.

Definition 2 (Logical symbol). *The* logical symbols *of first-order predicate logic are:*

symbol class		*symbols*	*pronounced*
punctuation symbols		,) (	
connectives	0-*ary*	$\bot$	*bottom, falsity symbol*
		$\top$	*top, truth symbol*
	1-*ary*	$\neg$	*not, negation symbol*
	2-*ary*	$\wedge$	*and, conjunction symbol*
		$\vee$	*or, disjunction symbol*
		$\Rightarrow$	*implies, implication symbol*
quantifiers		$\forall$	*for all, universal quantifier*
		$\exists$	*exists, existential quantifier*
variables		$u\ v\ w\ x\ y\ z\ \ldots$ *possibly subscripted*	

The set of variables is infinite and computably enumerable.

Definition 3 (Signature). *A* signature *or vocabulary for first-order predicate logic is a pair* $\mathcal{L} = (\{Fun^n_{\mathcal{L}}\}_{n\in\mathbb{N}}, \{Rel^n_{\mathcal{L}}\}_{n\in\mathbb{N}})$ *of two families of computably enumerable symbol sets, called* n*-ary* function symbols *of* $\mathcal{L}$ *and* n*-ary* relation symbols *or predicate symbols of* $\mathcal{L}$*. The* 0*-ary function symbols are called* constants *of* $\mathcal{L}$*. The* 0*-ary relation symbols are called* propositional *relation symbols of* $\mathcal{L}$*.*

Note that any of the symbol sets constituting a signature may be empty. Moreover, they need not be disjoint. If they are not, the signature is called *overloaded*. Overloading is usually uncritical, moreover it can be undone by annotating each signature symbol with its symbol class (*Fun* or *Rel*) and arity whenever required.

First-order predicate logic comes in two versions: *equality* may or may not be built-in. The version with built-in equality defines a special 2-ary relation symbol for equality, written $=$ by some authors and written $\doteq$ or differently by authors who want to avoid confusion with the same symbol at the meta level. In this article we assume first-order predicate logic without equality, unless built-in equality is explicitly mentioned.

Definition 4 ($\mathcal{L}$-term). *Let* $\mathcal{L}$ *be a signature. We define inductively:*

1. *Each variable* x *is an* $\mathcal{L}$*-term.*
2. *Each constant* c *of* $\mathcal{L}$ *is an* $\mathcal{L}$*-term.*
3. *For each* $n \geq 1$*, if* f *is an* n*-ary function symbol of* $\mathcal{L}$ *and* $t_1, \ldots, t_n$ *are* $\mathcal{L}$*-terms, then* $f(t_1, \ldots, t_n)$ *is an* $\mathcal{L}$*-term.*

Definition 5 ($\mathcal{L}$-atom). *Let* $\mathcal{L}$ *be a signature. For* $n \in \mathbb{N}$*, if* p *is an* n*-ary relation symbol of* $\mathcal{L}$ *and* $t_1, \ldots, t_n$ *are* $\mathcal{L}$*-terms, then* $p(t_1, \ldots, t_n)$ *is an* $\mathcal{L}$-atom *or atomic* $\mathcal{L}$*-formula. For* $n = 0$ *the atom may be written* $p()$ *or* p *and is called a* propositional $\mathcal{L}$-atom.

Definition 6 ($\mathcal{L}$-formula). *Let $\mathcal{L}$ be a signature. We define inductively:*

1. *Each $\mathcal{L}$-atom is an $\mathcal{L}$-formula.* *(atoms)*
2. *$\bot$ and $\top$ are $\mathcal{L}$-formulas.* *(0-ary connectives)*
3. *If φ is an $\mathcal{L}$-formula, then $\neg\varphi$ is an $\mathcal{L}$-formula.* *(1-ary connectives)*
4. *If φ and ψ are $\mathcal{L}$-formulas, then $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$ and $(\varphi \Rightarrow \psi)$ are $\mathcal{L}$-formulas.* *(2-ary connectives)*
5. *If x is a variable and φ is an $\mathcal{L}$-formula, then $\forall x\varphi$ and $\exists x\varphi$ are $\mathcal{L}$-formulas.* *(quantifiers)*

In most cases the signature $\mathcal{L}$ is clear from context, and we simply speak of terms, atoms, and formulas without the prefix "$\mathcal{L}$-". If no signature is specified, one usually assumes the conventions:

$p, q, r, \ldots$ are relation symbols with appropriate arities.
$f, g, h, \ldots$ are function symbols with appropriate arities $\neq 0$.
$a, b, c, \ldots$ are constants, i.e., function symbols with arity 0.

The set of terms is a formal language for representing individuals about which statements can be made. The set of formulas is the formal language for representing such statements. For example, constants a and b might represent numbers, function symbol f an arithmetic operation, and relation symbol p an arithmetic comparison relation. Then the term $f(a, f(a, b))$ would also represent a number, whereas the atomic formula $p(a, f(a, b))$ would represent a statement about two numbers.

Unique Parsing of Terms and Formulas. The definitions above, in particular the fact that parentheses enclose formulas constructed with a binary connective, ensure an unambiguous syntactical structure of any term or formula. For the sake of readability this strict syntax definition can be relaxed by the convention that $\wedge$ takes precedence over $\vee$ and both of them take precedence over $\Rightarrow$. Thus, $q(a) \vee q(b) \wedge r(b) \Rightarrow p(a, f(a, b))$ is a shorthand for the fully parenthesised form $((q(a) \vee (q(b) \wedge r(b))) \Rightarrow p(a, f(a, b)))$. Likewise, one usually assumes that $\wedge$ and $\vee$ associate to the left and $\Rightarrow$ associatés to the right. As a further means to improve readability, some of the parentheses may be written as square brackets or curly braces.

Definition 7 (Subformula). *The* subformulas *of a formula φ are φ itself and all subformulas of* immediate subformulas *of φ.*

- *Atomic formulas and $\bot$ and $\top$ have no immediate subformulas.*
- *The only immediate subformula of $\neg\psi$ is ψ.*
- *The immediate subformulas of $(\psi_1 \wedge \psi_2)$ or $(\psi_1 \vee \psi_2)$ or $(\psi_1 \Rightarrow \psi_2)$ are ψ_1 and ψ_2.*
- *The only immediate subformula of $\forall x\psi$ or $\exists x\psi$ is ψ.*

Definition 8 (Scope). *Let φ be a formula, Q a quantifier, and $Qx\psi$ a subformula of φ. Then Qx is called a* quantifier for x. *Its* scope *in φ is the subformula ψ except subformulas of ψ that begin with a quantifier for the same variable x.*

Each occurrence of x in the scope of Qx is bound *in φ by Qx. Each occurrence of x that is not in the scope of any quantifier for x is a* free *occurrence of x in φ.*

Example 9 (Bound/free variable). Let φ be $\big(\forall x[\exists x p(x) \wedge q(x)] \Rightarrow [r(x) \vee \forall x s(x)]\big)$. The x in $p(x)$ is bound in φ by $\exists x$. The x in $q(x)$ is bound in φ by the first $\forall x$. The x in $r(x)$ is free in φ. The x in $s(x)$ is bound in φ by the last $\forall x$.

Let φ' be $\forall x\big([\exists x p(x) \wedge q(x)] \Rightarrow [r(x) \vee \forall x s(x)]\big)$. Here both the x in $p(x)$ and the x in $r(x)$ are bound in φ' by the first $\forall x$.

Note that being bound or free is not a property of just a variable occurrence, but of a variable occurrence relative to a formula. For instance, x is bound in the formula $\forall x\, p(x)$, but free in its subformula $p(x)$.

Definition 10 (Rectified formula). *A formula φ is* rectified, *if for each occurrence Qx of a quantifier for a variable x, there is neither any free occurrence of x in φ nor any other occurrence of a quantifier for the same variable x.*

Any formula can be rectified by consistently renaming its quantified variables. The formula $(\forall x[\exists x p(x) \wedge q(x)] \Rightarrow [r(x) \vee \forall x s(x)])$ from the example above can be rectified to $(\forall u[\exists v p(v) \wedge q(u)] \Rightarrow [r(x) \vee \forall w s(w)])$. Note that rectification leaves any free variables free and unrenamed. Another name for rectification, mainly used in special cases with implicit quantification, is *standardisation apart*.

Definition 11 (Ground term or formula, closed formula). *A* ground term *is a term containing no variable. A* ground formula *is a formula containing no variable. A* closed formula *or sentence is a formula containing no free variable.*

For example, $p(a)$ is a ground atom and therefore closed. The formula $\forall x\, p(x)$ is not ground, but closed. The atom $p(x)$ is neither ground nor closed. In Example 9 above, the formula φ is not closed and the formula φ' is closed.

Definition 12 (Propositional formula). *A* propositional formula *is a formula containing no quantifier and no relation symbol of arity > 0.*

Propositional vs. Ground. Propositional formulas are composed of connectives and 0-ary relation symbols only. Obviously, each propositional formula is ground. The converse is not correct in the strict formal sense, but ground formulas can be regarded as propositional in a broader sense:

If $\mathcal{L}$ is a signature for first-order predicate logic, the set of ground $\mathcal{L}$-atoms is computably enumerable. Let $\mathcal{L}'$ be a new signature defining each ground $\mathcal{L}$-atom as a 0-ary relation "symbol" of $\mathcal{L}'$. Now each ground $\mathcal{L}$-formula can also be read as a propositional $\mathcal{L}'$-formula.

Note that this simple switch of viewpoints works only for ground formulas, because it cannot capture the dependencies between quantifiers and variables.

Definition 13 (Polarity). *Let φ be a formula. The* polarities *of occurrences of its subformulas are positive or negative as follows:*

- *The polarity of φ in φ is positive.*
- *If ψ is $\neg\psi_1$ or $(\psi_1 \Rightarrow \psi_2)$ and occurs in φ,*
  the polarity of ψ_1 in φ is the opposite of the polarity of ψ in φ.

– *In all other cases, if ψ is an occurrence in φ of a subformula with immediate subformula ψ', the polarity of ψ' in φ is the same as the polarity of ψ in φ.*

The polarity counts whether an occurrence of a subformula is within the scope of an even or odd number of negations. The left-hand immediate subformula of an implication counts as an implicitly negated subformula.

Definition 14 (Universal formula). *A formula φ is* universal, *iff each occurrence of $\forall$ has positive and each occurrence of $\exists$ has negative polarity in φ.*

For instance, $\forall x\,([\neg\forall y\, p(x,y)] \Rightarrow [\neg\exists z\, p(x,z)])$ is a universal closed formula, whereas $\forall x\,([\neg\forall y\, p(x,y)] \Rightarrow [\neg\forall z\, p(x,z)])$ is not universal.

Definition 15 (Prenex form). *A formula φ is in* prenex form, *iff it has the form $Q_1 x_1 \ldots Q_n x_n\, \psi$ where $n \geq 0$ and the Q_i are quantifiers and ψ contains no quantifier. The quantifier-free subformula ψ is called the* matrix *of φ.*

Obviously, a formula in prenex form is universal iff it does not contain $\exists$. Each formula can be transformed into an equivalent formula in prenex form (equivalent in the sense of $\models\!\!\models$ from Section 4).

Notation 16 (Term list notation). *Let $\boldsymbol{u} = t_1, \ldots, t_k$ be a list of terms, let f and p be a k-ary function and relation symbol. Then $f(\boldsymbol{u})$ is a short notation for the term $f(t_1, \ldots, t_k)$ and $p(\boldsymbol{u})$ for the atom $p(t_1, \ldots, t_k)$.*

Let $\boldsymbol{x} = x_1, \ldots, x_n$ be a list of variables and φ a formula. Then $\forall \boldsymbol{x}\varphi$ is a short notation for $\forall x_1 \ldots \forall x_n\, \varphi$ and $\exists \boldsymbol{x}\varphi$ for $\exists x_1 \ldots \exists x_n\, \varphi$. In the case $n = 0$ both $\forall \boldsymbol{x}\varphi$ and $\exists \boldsymbol{x}\varphi$ stand for φ.

Definition 17 (Universal/existential closure). *Let φ be a formula. Let $\boldsymbol{x}$ be the list of all variables having a free occurrence in φ. The* universal closure $\forall^*\varphi$ *is defined as $\forall \boldsymbol{x}\varphi$ and the* existential closure $\exists^*\varphi$ *as $\exists \boldsymbol{x}\varphi$.*

Technically, a quantifier-free formula such as $((p(x,y) \wedge p(y,z)) \Rightarrow p(x,z))$ contains free variables. It is fairly common to use quantifier-free notations as shorthand for their universal closure, which is a closed universal formula in prenex form, in this case $\forall x \forall y \forall z((p(x,y) \wedge p(y,z)) \Rightarrow p(x,z))$.

3.2 Query and Rule Language Fragments of First-Order Predicate Logic

Notation 18 (Rule). *A* rule $\psi \leftarrow \varphi$ *is a notation for a not necessarily closed formula $(\varphi \Rightarrow \psi)$. The subformula φ is called the* antecedent *or body and ψ the* consequent *or head of the rule. A rule $\psi \leftarrow \top$ may be written $\psi \leftarrow$ with empty antecedent. A rule $\bot \leftarrow \varphi$ may be written $\leftarrow \varphi$ with empty consequent.*

Implicit Quantification. Typically, a rule is a shorthand notation for its universal closure. The set of free variables in a rule $\psi \leftarrow \varphi$ can be partitioned into the variables $\boldsymbol{x}$ that occur in ψ and the variables $\boldsymbol{y}$ that occur in φ but not in ψ.

Then the universal closure $\forall \boldsymbol{x} \forall \boldsymbol{y}(\psi \leftarrow \varphi)$ is equivalent to $\forall \boldsymbol{x}(\psi \leftarrow \exists \boldsymbol{y} \varphi)$ in the sense of $\models\!\!\models$ (Section 4). Thus, the free variables occurring only in the rule antecedent can be described as implicitly *universally* quantified *in the entire rule* or implicitly *existentially* quantified *in the rule antecedent.* The two alternative descriptions mean the same, but they can be confusing, especially for rules with empty consequent.

Definition 19 (Literal, complement). *If A is an atom, both A and $\neg A$ are* literals. *The literal A is* positive, *the literal $\neg A$ is* negative, *and the two are a pair of* complementary *literals. The* complement *of A, written $\overline{A}$, is $\neg A$, the complement of $\neg A$, written $\overline{\neg A}$, is A.*

Definition 20 (Clause). *A* clause *is a disjunction of finitely many literals. A clause is written $A_1 \vee \ldots \vee A_k \leftarrow L_1 \wedge \ldots \wedge L_n$ in rule notation, which stands for the disjunction $A_1 \vee \ldots \vee A_k \vee \overline{L_1} \vee \ldots \vee \overline{L_n}$ with atoms A_i and literals L_j, $k \geq 0$, $n \geq 0$. A clause represents its universal closure.*

Any formula of first-order predicate logic can be transformed into a finite set of clauses with essentially the same meaning (see Section 4).

3.2.1 Logic Programming

Logic programming considers clauses with non-empty consequent as *programs* and clauses with empty consequent as *goals* used for program invocation. The operational and declarative semantics of logic programs depend on whether the antecedent is a conjunction of atoms or of arbitrary literals and whether the consequent is just a single atom or a disjunction of several atoms.

Definition 21 (Clause classification). *Let $k, n \in \mathbb{N}$, let A, A_j, B_i be atoms and L_i be literals. The following names are defined for special forms of clauses.*

	Name	*Form*	
Horn clause	definite clause	$A \leftarrow B_1 \wedge \ldots \wedge B_n$	$k = 1, n \geq 0$
	unit clause[2]	$A \leftarrow$	$k = 1, n = 0$
	definite goal	$\leftarrow B_1 \wedge \ldots \wedge B_n$	$k = 0, n \geq 0$
	empty clause[3]	$\leftarrow$	$k = 0, n = 0$
normal clause		$A \leftarrow L_1 \wedge \ldots \wedge L_n$	$k = 1, n \geq 0$
normal goal		$\leftarrow L_1 \wedge \ldots \wedge L_n$	$k = 0, n \geq 0$
disjunctive clause		$A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_n$	$k \geq 0, n \geq 0$
general clause		$A_1 \vee \ldots \vee A_k \leftarrow L_1 \wedge \ldots \wedge L_n$	$k \geq 0, n \geq 0$

A finite set of definite clauses is called a *definite program.* Definite programs invoked by definite queries represent a fragment of first-order predicate logic with especially nice semantic properties. In the context of the programming language

[2] Unit clauses are also called *facts*, which is meant in a purely syntactic sense although the word suggests a semantic sense.

[3] The empty clause is usually denoted $\square$ in the literature on automated deduction.