LOGIC AND COMPUTER SCIENCE

Phokion G. Kolaitis
Computer Science Department, University of California, Santa Cruz, CA 95064, USA

Keywords: algorithm, Armstrong's axioms, complete problem, complexity class, computable function, computational complexity, conjunctive query, constraints, data complexity, database query language, database system, database theory, database, Datalog, decidable problem, decision problem, definable query, descriptive complexity, deterministic Turing machine, domain-independent formula, Ehren-feucht-Fraïssé game, existential second-order logic, expression complexity, finite model theory, finite-variable logics, first-order logic, functional dependency, inductive definition, infinitary logic, intractable problem, least fixed-point logic, logarithmic space, many-one reduction, mathematical logic, model checking problem, model theory, NL, non-deterministic logarithmic space, non-deterministic Turing machine, non-deterministic polynomial time, NP, NP-complete, partial fixed-point logic, pebble games, polynomial hierarchy, polynomial space, polynomial time, preservation theorem, preservation under extensions, propositional logic, PSPACE, query, recursive function, recursively enumerable, relational algebra, relational databases, satisfaction relation, satisfiable formula, satisfiability problem, second-order logic, solvable problem, SQL, structured query language, superkey, tautology, tractable problem, Turing machine, undecidable problem, unsolvable problem, zero-one law.

UNESCO – EOLSS SAMPLE CHAPTERS

Contents

1. Introduction
2. Complexity Classes and the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ problem
3. Propositional Logic and Complexity Classes
4. The Complexity of First-Order Logic and Richer Logics
4.1. The Complexity of First-Order Logic
4.2. The Complexity of Existential Second-Order Logic
4.3. Fagin's Theorem and Descriptive Complexity
4.4. Least Fixed-Point Logic and Polynomial-Time
4.5. Partial Fixed-Point Logic and Polynomial Space
5. Finite Model Theory
5.1. Classical Model Theory in the Finite
5.2. Ehrenfeucht-Fraïssé Games and First-Order Logic
5.3. Pebble Games and Fixed-Point Logics
5.4. 0-1 Laws in Finite Model Theory
6. Logic and Databases
6.1. Database Query Languages
6.2. Constraints in Databases
Glossary
Biographical Sketch

Summary

The interaction between logic and computer science has been extensive and continuous. Logic has found numerous uses in several different areas of computer science; furthermore, new areas in the interface between logic and computer science have emerged and have been thoroughly investigated in their own right.

In this chapter, we present selected highlights of the interaction between logic and computer science by focusing on just a few areas. The aim is to explore each of these areas in some depth, at the expense of breadth of coverage.

The first part of the chapter contains an overview of the fundamentals of computational complexity and an exposition of the main connections between logic and computational complexity. The second part focuses on finite model theory, which is the study of first-order logic and richer logics on classes of finite structures; topics covered include the analysis of the expressive power of logics using combinatorial games, and the connections between asymptotic probabilities and logic on finite structures. The third and final part of the chapter contains some of the interactions between logic and databases with emphasis on the use of first-order logic as a database query language and as a formalism for specifying constraints in databases and reasoning about them.

1. Introduction

Over the years an extensive interaction between logic and computer science has taken place. Concepts and methods of logic have permeated computer science to the point that logic has been called "the calculus of computer science". Indeed, logic has found uses and applications to a broad spectrum of areas in computer science, including computer architecture, programming languages, artificial intelligence, computational complexity, database systems, distributed systems, software engineering, and computer-aided verification of hardware design. Furthermore, the interaction between logic and computer science has given rise to new areas in the interface between the two disciplines, such as finite model theory and proof complexity. Overall, logic provides computer science with a unifying foundational framework, a variety of formalisms for expressing and analyzing algorithmic problems, and a rich set of concepts and tools for specifying aspects of computation and reasoning about them.

The goal of this chapter is to present some highlights of the interaction between logic and computer science by focusing on just a few selected areas. Specifically, we will discuss the connections between logic and computational complexity, give a bird's-eye view of finite model theory, and describe some of the interactions between logic and databases. There are two characteristics of the interaction between logic and computer science that will be evident in what follows. First, although infinite structures have a place in logic and computer science, the focus is mainly on finite structures or, more precisely, on classes of finite structures. Second, in addition to propositional logic and first-order logic, numerous other logics play an important role in computer science, including second-order logic and its fragments, various logics with fixed-point operators, finite-variable infinitary logics, and modal and temporal logics.

UNESCO – EOLSS SAMPLE CHAPTERS

2. Complexity Classes and the $\mathrm{P} \overset{?}{=} \mathrm{NP}$ problem

As described in *Computability and Complexity*, computability theory is concerned with the study of the boundary between solvability and unsolvability. Thus, one of the main aims of computability theory is to determine whether a given algorithmic problem of interest is solvable or unsolvable.

Computational complexity can be viewed as the evolution of computability theory where the focus is on solvable problems and the aim is classify these problems according to the resources required to solve them. For this reason, computational complexity has been aptly described by J. Hartmanis in his 1993 Turing Award Lecture as "the quantitative study of solvability". Indeed, the main goal of computational complexity is to characterize the inherent difficulty of solvable decision problems by placing them into classes according to the time resources or space resources required to solve them in some model of computation, which usually is either the (deterministic) Turing machine or the non-deterministic Turing machine.

The following major complexity classes will be of interest to us, where in the table below TM stands for (deterministic) Turing machine and NTM stands for non-deterministic Turing machine.

Class	Model	Resource Bound
NL	NTM	Nondeterministic Logarithmic Space
P	TM	Polynomial Time
NP	NTM	Nondeterministic Polynomial Time
PSPACE	TM	Polynomial Space

Figure 1. Complexity Classes

Time Complexity Classes. The class P consists of all decision problems solvable by a Turing machine in time bounded by some polynomial in the size of the input (see also *Computability and Complexity*). In the early 1960s, A. Cobham and J. Edmonds brought P to center stage by arguing that polynomial-time computability captures the informal concept of a *tractable* decision problem, that is, a problem for which a "good" algorithm exists. The intuition is that polynomial-time algorithms are "good", while on the contrary exponential-time algorithm are not, because the running time of an exponential algorithm dominates the running time of every polynomial-time algorithm, as the size of the input increases,

The class NP consists of all decision problems solvable by a non-deterministic Turing machine in time bounded by some polynomial in the size of the input. More precisely and as described in *Computability and Complexity*, the computation of a non-deterministic Turing machine M on an input x can be visualized as a *computation tree* whose paths represent the possible sequences of choices made by M while processing x.

UNESCO – EOLSS SAMPLE CHAPTERS

The input x is *accepted* by M if at least one path in the computation tree leads to an accepting state; furthermore, the *time* required by M to accept x is the minimum of the lengths of all accepting paths for x. Thus, a decision problem is in NP if there is a non-deterministic Turing machine M and a polynomial $p(n)$ such that an input x is a "yes" instance of the decision problem if and only if the time required by M to accept x is at most $p(|x|)$, where $|x|$ is the size of x.

An equivalent description of NP is that it consists of the class of all decision problems for which all "yes" instances have *succinct certificates*, that is, a proof that an input is a "yes" instance of the decision problem under consideration can be guessed and verified in time polynomial in the size of the input. For example, consider 3-Colorability, which asks: given a graph $\mathbf{G} = (V, E)$, can we assign one of three colors to each node so that no two nodes joined by an edge have the same color? If a graph is 3-colorable, then we can guess an assignment of colors to the nodes (this takes time linear in the size of $\mathbf{G}$) and verify in quadratic time that it is indeed a 3-coloring of $\mathbf{G}$. If effect, the paths of the computation tree of a polynomial-time non-deterministic Turing machine for 3-COLORABILITY are all possible such assignments of colors.

It is clear that $\mathrm{P} \subseteq \mathrm{NP}$. Thus, it is natural to ask whether P is properly contained in NP or, equivalently, whether P is different from NP.

Problem: Is P = NP?

The $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question is regarded as the central mathematical problem in theoretical computer science. It is a fundamental question about the comparative computational power of two different models of computation utilizing the same time resources. Another reason for the prominent status of $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ is that NP contains numerous important decision problems for which only exponential-time algorithms are known. Usually, these problems can be solved in exponential-time via *exhaustive search* through a space of candidate solutions each of which is of size polynomial in the size of the input. Thus, these problems belong to NP, since an actual solution is a succinct certificate that can be guessed and verified in polynomial time; 3-COLORABILITY is a case in point.

A possible approach to separating P from NP is to show that there is a structural property possessed by one of the two, but not by the other. Since P is a deterministic complexity class, it is *closed under complements*, which means that if a problem Q is P , then so is the complement $\bar{Q}$ of Q. In contrast, NP is not known to be closed under complements. In particular, it is not known whether NON-3-COLORABILITY is in NP.

Problem: Is NP closed under complements? Equivalently, is NP = coNP?

Space Complexity Classes. Let us now consider the space complexity classes NL and PSPACE. When defining space complexity classes, it is understood that the Turing machines considered (whether deterministic or non-deterministic) have a read-only tape that holds the input and one or more separate work tapes; the space-bound is imposed

UNESCO – EOLSS SAMPLE CHAPTERS

only on the number of different cells in the work tape(s) used during the computation. Thus, NL is the class of all decision problems solvable by a nondeterministic Turing machine using work-space bounded by the logarithm of the size of the input, while PSPACE is the class of all decision problems solvable by a (deterministic) Turing machine using work-space bounded by some polynomial of the size of the input. We could also consider the class NPSPACE of all decision problems solvable by a non-deterministic Turing machine in polynomial space. As shown by Savitch in 1970, however, this class turns out to be the same as PSPACE.

Theorem 2.1. [Savitch] PSPACE = NPSPACE.

This result is actually a special case of a more general tradeoff between deterministic and non-deterministic complexity classes. Specifically, Savitch showed that if $f(n) \geq \log n$ is a "well-behaved" function, then

$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n)),$$

where NSPACE($f(n)$) is the class of decision problems solvable by a non-deterministic Turing machine in space $f(n)$ and SPACE($f^2(n)$) is the class of decision problems solvable by a (deterministic) Turing machine in space $f^2(n)$. Thus, the space trade-off between deterministic and non-deterministic Turing machines is much better understood than the time trade-off.

Relationships Between Time and Space Complexity Classes. Although a multitude of complexity classes can be defined by considering different time and space bounds, the complexity classes NL,P, NP, and PSPACE are arguably the most prominent ones in computational complexity as many natural decision problems encountered in various areas of computer science tend to belong to one of them. Using the definitions, Savitch's Theorem 2.1, and simulations between deterministic and nondeterministic Turing machines, it is not too difficult to derive the following relationships between these complexity classes.

Theorem 2.2. *The following containments hold:*

$$\mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}.$$

It is conjectured and widely believed that each of the above immediate containments is a proper one, but proving this remains the key open problem in computational complexity to date. A separation, however, has been established between the two extremes.

Theorem 2.3. NL *is properly contained in* PSPACE, *that is,* $\mathrm{NL} \neq \mathrm{PSPACE}$.

This separation can be derived by combining two results. The first is that $\mathrm{NL} \subseteq \mathrm{SPACE}(\log^2(n))$, which is a consequence of the aforementioned general space trade-off results by Savitch. The second is a *Hierarchy Theorem* by Hartmanis and Stearns, which informally asserts that if there is a sufficiently large gap between the

space bounds used to define two deterministic complexity classes, then one is properly contained in the other. This result is proved using a *delayed diagonalization* technique.

Note that, since $\mathrm{NL} \neq \mathrm{PSPACE}$, at least one of the three immediate containments in Theorem 2.2 must be a proper one, which means that at least one of the separations $\mathrm{NL} \neq \mathrm{P}, \mathrm{P} \neq \mathrm{NP}$, and $\mathrm{NP} \neq \mathrm{PSPACE}$ holds. Yet, to this date it is not known which of these actually hold, even though, as indicated above, it is believed that all three hold.

It had been conjectured that NL is *not* closed under complements, which would have implied that $\mathrm{NL} \neq \mathrm{P}$. This conjecture remained open for quite a long time, until it was eventually refuted by N. Immerman and, independently, R. Szelepcsényi in 1986.

Theorem 2.4. [Immerman, Szelepcsényi] NL *is closed under complements*.

Although the exact relationship between the major complexity classes is far from being understood, there has been progress in understanding the internal structure of these classes, as we will discuss next. Moreover, concepts and methods of computability theory have been a source of inspiration for this work.

Complete Problems. The complexity classes P, NP, NL, and PSPACE contains problem that are *complete* for the class, that is, problems that embody the intrinsic computational difficulty of the class at hand. More precisely, let *C* be a complexity class and *Q* a decision problem. We say that *Q* is *C-complete* if *Q* is in *C* and *Q* is *C-hard* , which means that for every $Q' \in C$, there is a "suitable" *many-one reduction f* of Q' to *Q*, so that for every input *x*

$$x \in Q' \Leftrightarrow f(x) \in Q.$$

Intuitively, by "suitable" reduction, we mean that *f* can be computed using fewer resources than the ones in the definition of the class *C*. More precisely, if *C* is the class NL or the class P, then "suitable" means that *f* is computable by a (deterministic) Turing machine in logarithmic space. For NP , PSPACE and other larger classes, "suitable" means that *f* is computable by a (deterministic) Turing machine in polynomial time.

The significance of complete problems is twofold. To begin with, complete problems for a class *C* hold the secret of whether or not *C* collapses to a complexity class C' contained in *C*, provided that C' is closed under the "suitable" reductions. In particular, for every NP-complete problem *Q*, the following two statements are equivalent:

(1) P = NP.
(2) $Q \in \mathrm{P}$.

Thus, NP-complete problems are the prime candidates for establishing the separation of P from NP. Furthermore, until this separation is established, showing that a decision problem *Q* is NP-complete is regarded as strong evidence that *Q* is not in Pand, thus, *Q* is computationally intractable.

UNESCO – EOLSS SAMPLE CHAPTERS

Note that if P = NP, then every problem in NP is easily seen to be NP-complete. If, however, P = NP (which is believed to be the case), then the structure of NP is quite complex as it contains problems of intermediate complexity between NP-complete and P. The following result, was obtained by R. Ladner in 1975 using a delicate diagonalization method.

Theorem 2.5. [Ladner's Theorem] If $\mathrm{P} \neq \mathrm{NP}$, *then there is a decision problem Q having the following properties:*

(1) $\mathrm{Q} \in \mathrm{NP} - \mathrm{P}$.

(2) *Q is not* NP-*complete.*

Problems in NP that are neither NP-complete nor in P can be viewed as analogous to r.e. sets that are neither recursive nor r.e.-complete (see the discussion about Post's Problem in *Computability and Complexity*).

Although the existence of such r.e. sets was shown by Friedberg and Muchnik without any additional hypotheses, the existence of NP-problems of intermediate complexity between NP-complete and P has so far been shown only under the hypothesis that $\mathrm{P} \neq \mathrm{NP}$.

Clearly, proving outright that such problems exist would also prove that $\mathrm{P} \neq \mathrm{NP}$.

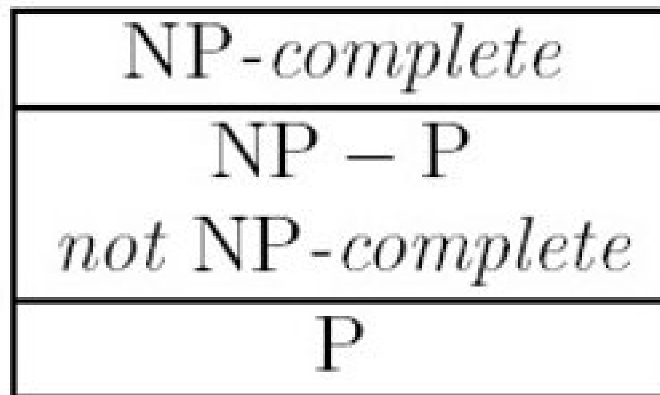

Figure 2. The Fine Structure of NP

It should be noted that the existence of complete problems for a complexity class cannot be taken for granted. As a matter of fact, there are complexity classes that provably contain no complete problems.

Nonetheless, all four classes NL, P, NP, and PSPACE contain numerous complete problems that arise naturally in several different areas of mathematics and computer science.

In particular, as we will see next, propositional logic provides some of the prototypical complete problems for each of these classes.

UNESCO – EOLSS SAMPLE CHAPTERS

-
-
-

TO ACCESS ALL THE **49 PAGES** OF THIS CHAPTER,
Visit: http://www.eolss.net/Eolss-sampleAllChapter.aspx

Biographical Sketch

Phokion G. Kolaitis is a Professor of Computer Science at the University of California, Santa Cruz. Trained as a logician, he has worked primarily in Logic in Computer Science, Finite Model Theory, Complexity and Database Theory. He has received a Guggenheim Fellowship and has had a continuing association with the IBM Almaden Research Center, where he has been serving since 2004 as Manager of the Computer Science Principles and Methodologies Group.

UNESCO – EOLSS
SAMPLE CHAPTERS