Logical Characterisations of Complexity Classes*

Martin Grohe
RWTH Aachen University

Mathematics Subject Classification

68Q19; 68Q15; 03C13

Short Definition

The complexity of a computational problem, originally defined in terms of the computational resources required to solve the problem, can be characterised in terms of the language resources required to describe the problem in a logical system. This yields logical characterisations of all standard complexity classes.

Description

It was realised from the beginnings of computability theory in the 1930s that there is a close connection between logic and computation. Indeed, various degrees of computability have natural characterisations in terms of logical definability. For example, the recursively enumerable sets of natural numbers are precisely the sets definable by an existential formula of first-order predicate logic in the language of arithmetic.

Descriptive Complexity Theory

Descriptive complexity may be viewed as a natural continuation of these results of computability theory in the realm of computational complexity. It provides characterisations of most standard complexity classes in terms of

*To appear in *Enyclopedia of Applied and Computational Mathematics*

logical definability. Arguably the most important of these characterisations are given by the following two theorems:

Fagin's Theorem (Fagin 1974). *A property of finite structures is decidable in nondeterministic polynomial time* NP *if and only if it is definable in existential second order logic* ∃SO*. (Short:* ∃SO *captures* NP*.)*

Immerman-Vardi Theorem (Immerman 1982, Vardi 1982). *A property of ordered finite structures is decidable in polynomial time* P *if and only if it is definable in least fixed-point logic* LFP*. (Short:* LFP *captures* P *on ordered structures.)*

To explain these two theorems, we need to review the basic framework of computational complexity theory and some logic. Complexity classes are usually defined as classes of problems that can be solved with restricted resources such as time or space. To turn this into a precise mathematical definition, we need to fix a machine model and a coding scheme for representing computational problems as inputs. Typically, multitape Turing machines are used as machine model. Without much loss of generality, we can focus on decision problems (i.e., problems with a yes/no-answer) and represent them by languages over the binary alphabet $\{0, 1\}$, i.e., as sets of strings of zeroes and ones. Obviously, complexity classes defined this way depend on both the machine model and the representation scheme, but fortunately most classes are robust enough so that they end up the same for any "reasonable" machine model and representation.

Yet the instances of most computational problems are not naturally modelled as strings over a finite alphabet, but rather by richer mathematical structures. For example, instances of a network connectivity problem are naturally modelled as directed graphs, and so are the instances of many combinatorial optimisation problems. Boolean circuits can be modelled by labelled directed graphs. The standard relational database model represents databases by a collection of finite relations, i.e., a finite relational structure. Of course the instances of some problems, such as problems on the natural numbers (in binary representation) or pattern matching problems, are most naturally described by finite strings, but strings can also be viewed as specific finite structures. If we adopt finite structures as flexible models of the instances of computational problems, then decision problems become properties of finite structures, or equivalently, classes of finite structures closed under isomorphism. This is the point of view taken in descriptive complexity

theory.

Logics express, or define, properties of structures. The logics considered in descriptive complexity theory are extensions of first-order predicate logic FO. Instead of going through formal definitions, we give three examples of logics and graph properties defined in these logics.

Example 1 (First-Order Logic). *The diameter of a graph is the maximum distance between any two vertices of the graph. The following sentence of first-order logic in the language of graphs defines the property of a graph having diameter at most* 2*:*

$$\forall x \forall y \Big(x = y \vee Exy \vee \exists z \big(Exz \wedge Ezy \big) \Big).$$

Here the variables x, y, z range over the vertices of a graph, and Exy expresses that the vertices interpreting x, y are adjacent.

It has turned out that first-order logic is too weak to express most properties that are interesting from a computational point of view. Second-order logic SO is much more powerful; actually it is too powerful to stay in the realm of efficient computation. Hence various fragments of SO are studied in the context of descriptive complexity theory. In SO, we not only have "individual variables" ranging over the vertices of a graph, but also "set variables" ranging over sets of vertices and, more generally, "relation variables" ranging over relations between vertices. Existential second-order logic $\exists$SO is the fragment of SO consisting of all formulas that only use existential quantification over set and relation variables and where no existential quantifier binding a relation variable appears in the scope of a negation symbol.

Example 2 (Existential Second-Order Logic). *A graph is 3-colourable if its vertices can be coloured with three colours in such a way that no two adjacent vertices get the same colour. The following sentence of existential second-order logic defines the property of a graph being 3-colourable:*

$$\begin{aligned} \exists R \exists B \exists G \Big(& \forall x \big(Rx \vee Bx \vee Gx \big) \\ & \wedge \forall x \forall y \big(Exy \rightarrow \big(\neg(Rx \wedge Ry) \wedge \neg(Bx \wedge By) \wedge \neg(Gx \wedge Gy) \big) \big) \Big). \end{aligned}$$

Here the variables R, B, G are set variables representing the three colours, and x, y are individual variables. Rx expresses that the vertex interpreting x is contained in the set interpreting R.

Fixed-point logics are extensions of FO with a more algorithmic flavour than SO. They allow it to formalise inductive definitions, as illustrated by the following example.

Example 3 (Least Fixed-Point Logic). *Suppose we want to define the transitive closure T of the edge relation of a graph $G = (V, E)$. It admits the following inductive definition: We let $T_1 := E$, and for all i we let T_{i+1} be the set of all pairs (u, v) of vertices such that there is a vertex w with $(v, w) \in T_i$ and $(w, u) \in T_i$. Then T is the union of all the T_i. Equivalently, we may define T as the least fixed-point of the (monotone) operator*

$$X \mapsto \Big\{(v, w) \;\Big|\; (v, w) \in E \vee \exists z\big((v, z) \in X \wedge (z, w) \in X\big)\Big\}.$$

In least fixed-point logic LFP*, we can form a formula*

$$\mathrm{lfp}\,\big(Xxy \leftarrow Exy \vee \exists z(Xxz \wedge Xzy)\big)(v, w)$$

to define this least fixed-point (and thus the transitive closure). If we call this formula $\psi(v, w)$, then the LFP*-sentence $\forall v \forall w\big(v = w \vee \psi(v, w)\big)$ defines connectedness of (undirected) graphs.*

To connect the properties of structures defined in our logics with complexity classes, we need to fix an encoding scheme for structures. It is common to use a generalisation of the adjacency-matrix encoding of graphs to encode structures by binary strings. Unfortunately, a graph has different adjacency matrices, obtained by associating the vertices with the rows and columns of the matrix in different orders, and among these there is no distinguished canonical one that we could use as "the" encoding of the structure. This observation generalises to arbitrary structures. Only if a structure B comes with a linear order of its elements, that is, it has a distinguished binary relation $\leq^B$ that is a linear order of its elements, then we can fix a canonical binary string $\langle B \rangle$ encoding B. We call such structures *ordered structures*, or we say that they have a *built-in order*. With each property $\mathcal{Q}$ of ordered structures we associate the language $\mathcal{L}(\mathcal{Q}) := \{\langle B \rangle \mid B \text{ has property } \mathcal{Q}\}$. With a structure A without built-in order we can only associate a language $\mathcal{L}(A)$ consisting of all encodings of A. Equivalently, we may view $\mathcal{L}(A)$ as the set of all strings $\langle B \rangle$ for all ordered expansions B of A. For a property $\mathcal{P}$ of structures we let $\mathcal{L}(\mathcal{P})$ be the union of all $\mathcal{L}(A)$ for structures A that have property $\mathcal{P}$. Now we say that a logic L *captures* a complexity class K if for each property $\mathcal{P}$ of structures, there is an L-sentence that defines $\mathcal{P}$ if

and only if $\mathcal{L}(\mathcal{P}) \in \mathsf{K}$. We say that L *captures* K *on ordered structures* if for each property $\mathcal{Q}$ of ordered structures, there is an L-sentence that defines $\mathcal{Q}$ if and only if $\mathcal{L}(\mathcal{Q}) \in \mathsf{K}$.

Fagin's Theorem and the Immerman-Vardi Theorem give logics capturing the complexity classes NP and P, respectively, the latter only on ordered structures. There are similar logical characterisations for most other complexity classes (for background and references, we refer the reader to the textbooks (Ebbinghaus and Flum 1995, Immerman 1999, Grädel et al. 2007). For the standard space complexity classes, we have the following characterisations: Deterministic Transitive Closure Logic DTC captures L ("logarithmic space") on ordered structures; transitive Closure Logic TC captures NL ("nondeterministic logarithmic space") on ordered structures; and partial Fixed-Point Logic PFP captures PSPACE ("polynomial space") on ordered structures. While these characterisations use various extensions of first-order logic by fixed-point operators or similar "generalised quantifiers", we also have characterisations of various complexity classes by restrictions and extensions of second-order logic: Second-order logic SO captures PH (the "polynomial hierarchy"). The "Krom-fragment" of second-order logic captures NL on ordered structures, and the "Horn-fragment" of second-order logic captures P on ordered structures. The extension of second-order logic with a (second-order) transitive closure operator captures PSPACE. There are also logical characterisations of complexity below L, but in addition to a built-in order, these require structures to have *built-in arithmetic*. For example, first-order logic FO captures dlogtime-uniform AC^0 on structures with built-in arithmetic.

Note that for the class P and smaller classes such as L and NL we only have logical characterisations on ordered structures. Indeed, it is a major open problem whether there are logical characterisations for these classes on arbitrary (not necessarily ordered) structures. Only partial results characterising P on restricted classes of structures ar known (the most powerful in (Grohe 2011)).

Function Algebras and Implicit Computational Complexity

An alternative way of characterising complexity classes is inspired by the characterisations of the computable functions as recursive functions and by the λ-calculus. The idea is to describe the functions in a complexity class as an algebra of functions. We extend complexity classes K to classes of

functions on binary strings and speak of K-functions. We usually think of of K-functions as functions on the natural numbers (via a binary encoding). The classical result in this area is Cobham's characterisation of the polynomial time computable functions using the following restricted version of primitive recursion: A $(k+1)$-ary function f on the natural numbers is defined from functions g, h_0, h_1, b by *bounded primitive recursion on notation* if for all $\bar{x}$ we have $f(\bar{x},0) = g(\bar{x})$ and $f(\bar{x},2y+i) = h_i(\bar{x},y,f(\bar{x},y))$ for $i = 0, y > 0$ and $i = 1, y \geq 0$, provided that $f(\bar{x},y) \leq b(\bar{x},y)$ for all $\bar{x},y$. The addition "on notation" refers to the fact that this definition is most naturally understood if one thinks of natural numbers in binary notation.

Cobham's Theorem (Cobham 1962). *The class of* P*-functions is the closure of the basic functions* $x \mapsto 0$ *("constant 0"),* $(x_1,\ldots,x_k) \mapsto x_i$ *for all* $i \leq k$ *("projections"),* $x \mapsto 2x$ *and* $x \mapsto 2x+1$ *("successor functions"), and* $(x,y) \mapsto 2^{|x|\cdot|y|}$ *("smash function"), where* $|x|$ *denotes the length of the binary representation of* x*, under composition and bounded primitive recursion on notation.*

Similar characterisations are known for other complexity classes.

What is slightly unsatisfactory about Cobham's characterisation of the P-functions is the explicit time bound b in the bounded primitive recursion scheme. Bellantoni and Cook (1992) devised a refined primitive recursion scheme that distinguishes between different types of variables and how they may be used and characterise the P-functions without an explicit time bound. This is the starting point of the area of "implicit computational complexity" ((Hofmann 2000) is a survey). While Bellantoni and Cook's recursion scheme is still fairly restrictive, in the sense that the type system excludes natural definitions of P-functions by primitive recursion, subsequently, researchers have developed a variety of full (mostly functional) programming languages with very elaborate type systems guaranteeing that precisely the K-functions (for many of the standard complexity classes K) have programs in this language. The best-known of these is Hofmann's functional language for the P-functions with a type system incorporating ideas from linear logic (Hofmann 2003).

Proof Theory and Bounded Arithmetic

There is yet another line of logical charactarisations of complexity classes. It is based on provability in formal system rather than just definability. Again,

these characterisations have precursors in computability theory, in particular the characterisation of the primitive recursive functions as precisely those functions that are Σ_1-definable in the fragment $\mathsf{I}\Sigma_1$ of Peano Arithmetic.

The setup is fairly complicated, and we will only be able to scratch the surface; for a thorough treatment we refer the reader to the survey (Buss 1998) and the textbook (Cook and Nguyen 2010). Our basic logic is first-order logic in the language of arithmetic, consisting of the standard symbols $\leq$ (order), $+$ (addition), $\cdot$ (multiplication), 0, 1 (constants 0 and 1), and possibly additional function symbols. In the *standard model of arithmetic* N, all these symbols get their standard interpretations over the natural numbers. A *theory* is a set of first-order sentences that is closed under logical consequence. For example, $\mathsf{Th}(N)$ is the set of all sentences that are true in the standard model N. It follows from Gödel's First Incompleteness Theorem that $\mathsf{Th}(N)$ has no decidable axiom system. A decidable, yet still very powerful theory contained $\mathsf{Th}(N)$ is Peano Arithmetic PA. It is axiomatised by a short list of basic axioms making sure that the basic symbols are interpreted right together with induction axioms of the form $\big(\phi(0) \wedge \forall x(\phi(x) \to \phi(x+1))\big) \to \forall x \phi(x)$ for all first-order formulas ϕ. Here, we are interested in fragments $\mathsf{I}\Phi$ of PA obtained by restriction the induction axioms to formulas $\phi \in \Phi$ for sets Φ of first-order formulas. Δ_0 denotes the set of all bounded first-order formulas, that is, formulas where all quantifications are of the form $\exists x \leq t$ or $\forall x \leq t$ for some term t that does not contain the variable x. Almost everything relevant for complexity theory takes place within Δ_0, but let us mention that Σ_1 is the set of all first-order formulas of the form $\exists x \phi$, where ϕ is a Δ_0-formula.

We say that a function f on the natural numbers is *definable in a theory* T if there is a formula $\phi(x, y)$ such that the theory T proves that for all x there is exactly one y such that $\phi(x, y)$ and for all natural numbers m, n the standard model N satisfies $\phi(m, n)$ if and only if $f(m) = n$. For example, it can be shown the functions in the linear time hierarchy LTH are precisely the functions that are Δ_0-definable in the theory $\mathsf{I}\Delta_0$.

To characterise the classes P and NP and the other classes of the polynomial hierarchy, Buss introduced a hierarchy of very weak arithmetic theories S^i_2. They are obtained by even restricting the use of bounded quantifiers in Δ_0-formulas, defining a hierarchy of Σ^b_i-formulas within Δ_0, but at the same time using an extended language that also contains functions symbols like $\#$ (for the "smash" function $(x, y) \mapsto 2^{|x| \cdot |y|}$) and $|\ |$ (for the binary length).

Buss's Theorem (Buss 1986). *For all $i \geq 1$, the functions Σ^b_i-definable*

in S_2^i *are precisely the* Σ_{i-1}^P*-functions, where* $\Sigma_0^P = \mathsf{P}$*,* $\Sigma_1^P = \mathsf{NP}$*, and* Σ_i^P *is the* i*th level of the polynomial hierarchy.*

References and Recommended Reading

Bellantoni, S. and Cook, S. (1992). A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110.

Buss, S. (1986). *Bounded arithmetic*. Bibliopolis.

Buss, S. (1998). First-order proof theory of arithmetic. In Buss, S., editor, *Handbook of Proof Theory*, pages 79–147. Elsevier.

Cobham, A. (1962). The intrinsic computational difficulty of functions. In *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam.

Cook, S. and Nguyen, P. (2010). *Logical Foundations of Proof Complexity*. Perspectives in Logic. Cambridge University Press.

Ebbinghaus, H.-D. and Flum, J. (1995). *Finite Model Theory*. Springer-Verlag.

Fagin, R. (1974). Generalized first–order spectra and polynomial–time recognizable sets. In Karp, R., editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73.

Grädel, E., Kolaitis, P., Libkin, L., Marx, M., Spencer, J., Vardi, M., Venema, Y., and Weinstein, S. (2007). *Finite Model Theory and Its Applications*. Springer-Verlag.

Grohe, M. (2011). From polynomial time queries to graph structure theory. *Communications of the ACM*, 54(6):104–112.

Hofmann, M. (2000). Programming languages capturing complexity classes. *ACM SIGACT News*, 31(1):31–42.

Hofmann, M. (2003). Linear types and non-size-increasing polynomial time computation. *Information and Computation*, 183:57–85.

Immerman, N. (1982). Relational queries computable in polynomial time (extended abstract). In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 147–152.

Immerman, N. (1999). *Descriptive Complexity*. Springer-Verlag.

Vardi, M. (1982). The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146.