Descriptive Complexity Theories

Joerg FLUM

ABSTRACT: In this article we review some of the main results of descriptive complexity theory in order to make the reader familiar with the nature of the investigations in this area. We start by presenting the characterization of automata recognizable languages by monadic second-order logic. Afterwards we explain the characterization of various logics by fixed-point logics. We assume familiarity with logic but try to keep knowledge of complexity theory to a minimum.

Keywords: Computational complexity theory, complexity classes, descriptive characterizations, monadic second-order logic, fixed-point logic, Turing machine.

Complexity theory or more precisely, *computational complexity theory* (cf. Papadimitriou 1994), tries to classify problems according to the complexity of algorithms solving them. Of course, we can think of various ways of measuring the complexity of an algorithm, but the most important and relevant ones are *time* and *space*. That is, we think we have a type of machine capable, in principle, to carry out any algorithm, a so-called general purpose machine (or, general purpose computer), and we measure the complexity of an algorithm in terms of the time or the number of steps needed to carry out this algorithm. By space, we mean the amount of memory the algorithm uses. Time bounds yield (time) complexity classes consisting of all problems solvable by an algorithm keeping to the time bound. Similarly, space complexity classes are obtained. It turns out that these definitions are quite robust in the sense that, for reasonable time or space bounds, the corresponding complexity classes do not depend on the special type of machine model chosen.

But are the resources time and space tied to the inherent mathematical complexity of a given problem? We can try to classify problems also in terms of the complexity of formal languages or logics that allow to express the problems (e.g., if problem P_2 is expressible in second-order logic but not in first-order logic then it is more complex than a problem P_1 expressible in first-order logic). We speak of *descriptive complexity theory* (cf. Ebbinghaus & Flum 1999, Immerman 1999), if we have in mind this kind of investigations. It is plausible that problems that are harder to check, i.e., problems that are of higher computational complexity, might be harder to express, i.e., are of higher descriptive complexity. What is surprising is how close the relationship between computational and descriptive complexity is. In fact, the most important classes of computational complexity have descriptive characterizations. Hence, the computational complexity of a problem can be measured in terms of the richness of a logic needed to express this problem. These results show that (as Neil Immerman states it in Immerman 1999) "time and space are not model-dependent engineering concepts, they are more fundamental".

However, descriptive characterizations of complexity classes have been criticized for being merely simple translations of the original machine-based definitions of the classes into a logical formalism and therefore, for not providing new insights. Although it is certainly true that the logical characterizations are often very close to the

machine descriptions, we do not agree with this criticism. We think that the descriptive characterizations are important in various respects:

- Characteristic features of the logic may be seen as characteristic features of the complexity class described by it and may add to a better understanding of this class.
- They may help to recognize that a concrete problem is in a given complexity class (by expressing it in the corresponding logic). Moreover, one can view the logics involved as higher programming languages for problems of the corresponding complexity class (since in all concrete descriptive characterizations one easily sees how to convert a sentence of the logic into an algorithm accepting the class of its models and satisfying the required resource restrictions).
- The descriptive characterizations allow to convert problems, methods, and results of computational complexity theory into logic and vice versa, thus widening the methodological possibilities for both sides.

There is a further way to define complexity classes: Starting from a problem P_0, mostly one takes a problem P_0 of central importance in a given context, one considers the class of all problems reducible to P_0, where reducibility is defined in an appropriate way. In fact, various classes of so-called parameterized complexity theory (cf. Downey & Fellows 1999), a new branch of complexity theory, have been defined in this way. For some of these classes, descriptive characterizations (but no machine-based characterization) are known (cf. Flum & Grohe 2002) and have led to isolate and to understand better the essential features of these classes.

Moreover, these results show that not always descriptive characterizations are mere translations of machine-based characterizations, since, as we just remarked, in some cases no simple and natural machine descriptions of these parameterized classes are known.

The purpose of this article is to give an insight into the nature of results of descriptive complexity. We assume some acquaintence with logic but try to keep knowledge of complexity theory to a minimum.

1 Automata

We start with the -historically first- logical characterization of a complexity class. Often, this class is not considered to be a class of computational complexity theory, since it is not defined in terms of time or space restrictions of a general purpose machine, but it is obtained by restricting the "hardware" of the machine: We consider automata. We want to show that the class of problems recognized by automata coincides with the class of problems definable in monadic second-order logic. It turns out that this simple case already reflects some of the crucial aspects encountered in the descriptive characterizations of "real" complexity classes.

By Σ we always denote a finite alphabet, i.e., a finite set of *letters* or *symbols*. Then, Σ^+ is the set of (finite and nonempty) *words* (or, *strings*) over Σ. If $\Sigma=\{0,1\}$ then 01101

is a word over Σ. If w_1:=010 and w_2:=11011 then we denote by w_1w_2 the concatenation of w_1 and w_2, w_1w_2= 01011011.

A *problem* (or, *language*) P on Σ is a set of words over Σ, $P \subseteq \Sigma^+$. E.g., for Σ={0,1}, $P_{\text{even(0)}}$:={$w \in \Sigma^+ \mid$ w contains an even number of 0's} and $P_{0=1}$:={$w \in \Sigma^+ \mid$ w contains as many 0's as 1's } are problems.

Automata are machines with restricted memory and with restricted reading capabilities, more precisely, an *automaton* $\mathfrak{A}$ over the alphabet Σ is given by a finite set Q, the set of states, an element q_0 of Q, the *starting state*, a subset F of Q, the set of *final* (or, *accepting*) states and a function $\delta: Q \times \Sigma \to Q$, the *transition function*. The intuitive meaning of $\delta(q, a)=q_0$ is: If the automaton is in state q and reads a, then it passes to state q_0.

The function δ induces a function $\tilde{\delta}: Q \times \Sigma^+ \to Q$, the meaning of $\tilde{\delta}(q, w)=q_0$ being: If the automaton is in state q and reads w letter by letter, then it ends in state q_0, more formally, $\tilde{\delta}(q,w)$ is defined by induction on the length of the word w:

If $w=a \in \Sigma$, then $\tilde{\delta}(q,a) := \delta(q,a)$

if $w = va$, then $\tilde{\delta}(q,w) := \delta\left(\tilde{\delta}(q,v)\, a\right)$

An automaton $\mathfrak{A}$ *accepts* the word w if, in state q_0, $\mathfrak{A}$ reads w, it ends in a state of F, i.e., if $\tilde{\delta}(q_0,w) \in F$.

$$P(\mathfrak{A}) := \{ w \in \Sigma^+ \mid \mathfrak{A} \text{ accepts } w \}$$

is the *problem accepted by* $\mathfrak{A}$.

For example, consider the automaton $\mathfrak{A}_o$ with

$$Q := \{q_{\text{even}}, q_{\text{odd}}\},\ q_0 := q_{\text{even}},\ F := \{q_{\text{even}}\}$$

and with

$$\delta(q_{\text{even}},0) := q_{\text{odd}}\quad \delta(q_{\text{even}},1) := q_{\text{even}};$$
$$\delta(q_{\text{odd}},0) := q_{\text{even}}\quad \delta(q_{\text{odd}},1) := q_{\text{odd}}.$$

One easily verifies that $\mathfrak{A}_o$ accepts $P_{\text{even(0)}}$, $P(\mathfrak{A}_o)=P_{\text{even(0)}}$. It can be shown that there is no automaton accepting the problem P_0=1:={$w \in \Sigma^+ \mid$ w contains as many 0's as 1's}, the main reason being that any automaton only has a finite number of states and no writing capability and hence, when reading a {0, 1}-word, it cannot memorize the number of 0's and the number of 1's (or the difference of both numbers).

We aim to show or at least to understand the following result linking the measure "being accepted by an automaton" from computational complexity to the measure "being definable in monadic second-order logic" from descriptive complexity:

> A problem $P \subseteq \Sigma^*$ is accepted by an automaton just in case it is definable in monadic second-order logic.

What does "P is definable in monadic second-order" logic mean? Given Σ we consider the vocabulary τ_Σ:={<, S, $(P_a)_{a\in\Sigma}$, min, max} consisting of the binary relation symbols <, S, the unary relation symbols P_a for $a \in \Sigma$, and the constant symbols min,

max. For a word $w=a_1...a_n$ of length n let $\mathcal{B}_w:=(\{1,...,n\},<^w,S^w,(P_a^w)_{a\in\Sigma}, \min^w, \max^w)$ be the *word mo*del associated with w, i.e., the τ_Σ-structure with universe $\{1,...,n\}$, where $<^w$ denotes the natural ordering on $\{1,...,n\}$, $S^w=\{(i,i+1)\mid 1\leq i\leq n-1\}$ is the successor relation, $\min^w=0$ and $\max^w=n$, and where $P_a^w=\{i\mid a_i=a\}$ is the set of positions of w carrying the letter a.

We denote by FO and by MSO the set of formulas of *first-order logic* and of *monadic second-order logic*. In addition to first-order logic, monadic second order logic contains monadic (unary) relation variables, called set variables.

These variables can be quantified, universally or existentially, then ranging over all subsets of the given universe. Now, we can give a precise formulation of the result above-mentioned:

Theorem 1.1 (Büchi 1960, Elgot 1961)

Let Σ *be an alphabet and* $P\subseteq\Sigma^+$.

Then the following are equivalent:

(i) *There is an automaton* $\mathfrak{A}$ accepting P, i.e. $P(\mathfrak{A})=P$.

(ii) *There is a monadic second-order sentence* φ of vocabulary τ_Σ such that for every $w\in\Sigma^+$,

$$w\in P\Leftrightarrow\mathcal{B}_w\models\varphi.$$

Proof: We sketch a proof. First assume that there is an automaton $\mathfrak{A}$ accepting P. We just describe in MSO that there is an accepting computation of $\mathfrak{A}$ (nearly in all characterizations to come, one argues similarly for this direction): For every state $q\in Q:=\{q_0,..., q_m\}$ of $\mathfrak{A}$ we use a monadic second-order variable X_q for the subset of the universe corresponding to the positions where the automaton is in state q before reading the letter at this position. The sentence we aim at has the form

$$\exists X_{q_0}...\exists X_{q_m}\psi(X_{q_0},...,X_{q_m}),$$

where $\psi(X_{q_0},...,X_{q_m})$ is a formula (with no second-order quantifiers) expressing that $\mathfrak{A}$ starts in q_0, behaves according to its transition function, and finally ends in an accepting state. Let us write down this formula explicitly for the automaton $\mathfrak{A}_e$ accepting $P_{\text{even}(0)}$: $\psi(X_{q_{\text{even}}},X_{q_{\text{odd}}})$ is the following formula (we write X_e for $X_{q_{\text{even}}}$ and X_0 for $X_{q_{\text{odd}}}$):

$$\begin{aligned}\forall x(X_e x\leftrightarrow\neg X_o x)\wedge X_e\min\wedge\forall x\forall y(Sxy\rightarrow((X_e x\wedge P_o x)\rightarrow X_o y)\\ \wedge(X_e x\wedge P_1 x)\rightarrow X_e y)\\ \wedge(X_o x\wedge P_0 x)\rightarrow X_e y)\\ \wedge(X_o x\wedge P_1 x)\rightarrow X_o y)\\ \wedge((X_o\max\wedge P_o\max)\vee(X_e\max\wedge P_1\max))\end{aligned}$$

(the last line takes care of the last letter of the given word). Now it should be clear that for $w\in\Sigma^+$,

$$w\in P_{\text{even}}\Leftrightarrow\mathcal{B}_w\models\exists X_e\exists X_0\psi(X_e, X_o).$$

For the other direction we need some notation. For $\varphi \in \mathrm{MSO}$ we denote by $\mathrm{qr}(\varphi)$ the *quantifier rank* of φ, i.e., the maximal number of nested quantifiers in φ; e.g., $\mathrm{qr}(\varphi)=2$ for $\varphi := \exists x(\forall X X x \wedge \exists z Q z x)$.

For $w \in \Sigma^+$, and $r \in \mathbb{N}$ let

$$\mathrm{MSO}_r(w) := \{\varphi \text{ MSO-sentence} \mid \mathrm{qr}(\varphi) \le r,\ \mathcal{B}_w \models \varphi\}$$

be the set of sentences of MSO of quantifier rank $\le r$ valid in $\mathcal{B}_w$, the r-*theory* of w.

We use the following two facts (the first one can be proved using, for example, the Ehrenfeucht-Fraïssé method, a model-theoretic method (e.g. compare Ebbinghaus & Flum 1999); the second one holds, since for every r, up to logical equivalence, there are only finitely many sentences of MSO of quantifier rank $\le r$, a fact that is easily shown by induction on r):

(1) If $\mathrm{MSO}_r(w)=\mathrm{MSO}_r(w')$ and $a \in \Sigma$, then $\mathrm{MSO}_r(wa)=\mathrm{MSO}_r(w'a)$.

(2) For fixed r the set $T_r := \{\mathrm{MSO}_r(w) \mid w \in \Sigma^+\}$ of r-theories of words over Σ is finite.

We come to a proof of the direction of (ii) to (i). So we assume that for the monadic second-order sentence φ of vocabulary τ_Σ we have for every $w \in \Sigma^+$:

$$w \in P \Leftrightarrow \mathcal{B}_w \models \varphi.$$

Set $r := qr(\varphi)$. Let $\mathfrak{A}$ be the automaton with $Q := T_r \dot{\cup} \{q_0\}$. So, the set of states is the set of r-theories together with an additional state q_0, which is the starting state of $\mathfrak{A}$. The transition function δ is given by

$$\delta(q_0,a) := \mathrm{MSO}_r(a)$$
$$\delta(\mathrm{MSO}_r(w),a) := \mathrm{MSO}_r(wa)$$

δ is well-defined by (1). Finally, we set $F := \{\mathrm{MSO}_r(w) \mid \varphi \in \mathrm{MSO}_r(w)\}$. An easy induction (on the length of w) shows that $\hat{\delta}(q_0,w)=\mathrm{MSO}_r(w)$ and thus,

$$\begin{aligned} w \in P \quad &\Leftrightarrow \mathcal{B}_w \models \varphi \\ &\Leftrightarrow \varphi \in \mathrm{MSO}_r(w) \\ &\Leftrightarrow \mathrm{MSO}_r(w) \in F \\ &\Leftrightarrow \hat{\delta}(q_0,w) \in F \\ &\Leftrightarrow w \in P(\mathfrak{A}), \end{aligned}$$

i.e., $P=P(\mathfrak{A})$. ÿ

The characterization of the previous theorem can be used to get the decidability of the monadic second-order theory of words (using results that more or less are trivial for automata). On the other hand these investigations also led to so-called automata on infinite words and automata on infinite trees; they allowed to study the monadic second-order theory of corresponding structures and also led to decidability results for some interesting mathematical theories and for several logics of programs.

2 Time-bounded algorithms

In the following discussion the reader can everywhere replace "algorithm" by Turing machine (or program for a Turing machine) or by any other general purpose machine model he is familiar with. Indeed, the concepts to be introduced are robust with respect to the type of (general purpose) machine model chosen.

Let $P \subseteq \Sigma^+$ be a problem. Suppose that an algorithm takes $2^{2^{|w|}}$ steps to decide whether $w \in P$. Then, even for quite short w, this algorithm cannot be performed in practice even with the fastest computers. Thus, it is conceivable that a problem P is "theoretically", but not "practically" solvable by an algorithm, since all algorithms deciding membership in P are too costly, since they need too many computation steps or too much memory. In fact, even for $\Sigma = \{0, 1\}$, one can show the existence of such problems.

Denote by PTIME all problems that are *solvable in polynomial time*, i.e., all problems P such that there is an algorithm $\mathbb{A}$ and a polynomial $p(x)$ such that for all words w,

1. $w \in P \Leftrightarrow \mathbb{A}$ accepts w;

2. the algorithm $\mathbb{A}$, on input w, stops after $\leq p(|w|)$ steps.

Here, $|w|$ denotes the length of the word w.

The importance of PTIME resides in the opinion that P coincides with the class of problems that can be realistically solved by computers. Therefore, one often identifies the "practically solvable" problems with the problems in PTIME. This "Church's Thesis of practical computability" can only be justified to a certain extent: note, for example, that no restriction is imposed on the degree or the coefficients of the polynomials. But experience shows that, as far as problems in practical applications (or problems that arise naturally in mathematics) are concerned, essentially the existence of an algorithm executable in practice corresponds to the existence of a polynomially bounded algorithm. This experience is summarized in (Downey & Fellows 1999) as follows: "When we can do anything clever at all, we can usually achieve polynomial-time complexities with small exponent polynomials. This can be regarded as a fascinating empirical fact about the 'natural' world of computational complexity".

Clearly, any problem P accepted by an automaton is in PTIME: In fact, an automaton $\mathfrak{A}$ with $P(\mathfrak{A}) = P$ is an algorithm deciding $w \in P$ in $|w|$ steps. For many important problems it is still open if they are in PTIME. We give an example. Let $\mathcal{G} = (G, E^G)$ be a (finite) graph, that is, $\mathcal{G}$ is a $\tau := \{E\}$-structure, where E is a binary relation symbol; hence, $E^G \subseteq G \times G$. The elements of G are the vertices of the graph. If $E^G ab$, we say that there is an *arc* from a to b. A graph is 3-colorable if we can color the vertices of G such that no adjacent vertices a, b, i.e., no a, b with $E^G ab$, are similarly colored. Is 3-colorability in PTIME? To get a problem in the sense of the previous section we must encode structures by words over some alphabet. This can be done in a straightforward way. Since we already saw that words correspond to structures, it is consistent with

our exposition so far, if in the rest of the paper we view problems as classes of finite structures[1].

It is not known if the class of 3-colorable graphs is in PTIME. But it is in NPTIME, the class of problems solvable by a nondeterministic algorithm in polynomial time. In certain states such nondeterministic algorithms may have the choice between more than one possible behaviours (actions). By definition, a nondeterministic algorithm *accepts* a given input, if there is at least one run accepting the input (that is, in any nondeterministic step there is at least one choice of behaviour such that eventually the algorithm accepts the input). In a precise framework, Turing machines correspond to algorithms and nondeterministic Turing machines to nondeterministic algorithms. Nondeterministic Turing machines are obtained by relaxing the notion of Turing machine similarly as the notion of algorithm is generalized in order to get nondeterministic algorithms. Nondeterministic Turing machines are an unrealistic but theoretically important model of computation The class of 3-colorable graphs is in NPTIME: The algorithm starts with a series of nondeterministic steps where it guesses a coloring of all the vertices (say, to every vertex it assigns 0, 1, or 2) and then it (deterministically) checks whether adjacent vertices have distinct colors.

Clearly, PTIME⊆NPTIME. The question whether PTIME=NPTIME, the so-called $\mathcal{P}$=$\mathcal{NP}$-problem, is still open. It is considered to be one of the most challenging problems of complexity theory, yet even of mathematics.

The class of 3-colorable graphs is axiomatized by the monadic second-order sentence

$$\exists X\exists Y\exists Z(\forall x(Xx\vee Yx\vee Zx)\wedge\forall x(\neg(Xx\wedge Yx)\wedge\neg(Yx\wedge Zx)\wedge\neg(Xx\wedge Zx))\wedge\forall x\forall y(Exy\rightarrow(\neg(Xx\wedge Xy)\wedge\neg(Yx\wedge Yy)\wedge\neg(Zx\wedge Zy)))).$$

This is a Σ^1_1-sentence, i.e., a second-order sentence of the form

$$\exists X_1...\exists X_\ell\,\psi$$

where ψ has no second-order quantifiers and $X_1,...,X_\ell$ are (not necessarily monadic) relation variables. The following theorem generalizes this observation. It was the starting point for the descriptive characterization of complexity classes.

Theorem 2.1 (Fagin 1974)

A class of structures is in NPTIME *just in case it is axiomatizable by a* Σ^1_1*-sentence.*

But what about PTIME? We obtain a characterization of PTIME by adding to first-order logic a so-called fixed-point operator (which resembles the μ-operator of classical recursion theory). We explain this operator by an example.

Consider a graph $\mathcal{G}$= (G, E^G). The sequence $a_0,...,a_n$ is a *path from a to b of length n*, if $a=a_0$, $a_n=b$, and $E^Ga_0a_1,...,E^Ga_{n-1}a_n$. We write $d(a, b)=n$, if there is a path from a to b of

[1] The reader familiar with descriptive complexity theory will realize that throughout the paper we do not address the problems that arise in absence of an order relation.

length n and there is no path from a to b of length k with $k<n$. In particular, $d(a, a)=0$. We write $d(a, b)=\infty$, if there is no path from a to b.

Consider the formula

$$\varphi(x,y,X):=(x=y\vee\exists z(Xxz\wedge Ezy)).$$

It gives rise to a sequence F_0, F_1,... (more precisely, $F_0(\varphi,\mathcal{G})$, $F_1(\varphi,\mathcal{G})$...) of subsets of $G\times G$ defined by

$$F_0:=\emptyset \text{ and } F_{n+1}:=\{(a,b)\in G\times G \mid \mathcal{G}\models\varphi[a,b,F_n]\},$$

i.e., F_{n+1} is the set of ordered pairs of elements of G that fulfill φ in $\mathcal{G}$, if we interprete X by F_n. An easy induction shows that

$$F_n:=\{(a,b)\in G\times G \mid d(a,b)<n\}.$$

In particular,

$$F_0\subseteq F_1\subseteq F_2\subseteq.... \qquad (1)$$

We denote by

$$F_+:=\{(a,b)\in G\times G \mid \exists n_0\forall n\geq n_0:(a,b)\in F_n\}.$$

By (1), $F^+=\bigcup_{n\in\mathbb{N}}F_n$, i.e.,

$$F_+:=\{(a,b)\in G\times G \mid d(a,b)<\infty\}.$$

Since $F_n=F_{n+1}$ implies $F_n=F_{n+1}=F_{n+2}=...=F_+$ and since $F_n\subseteq G\times G$ for every n, we see that the fixed-point of the sequence is reached after at most $|G|^2$ steps, i.e., after a number of steps which is polynomial in the size of $\mathcal{G}$. Now, by definition, the formula of "fixed-point logic"

$$\forall x\forall y[\mathrm{FP}_{xy,X}\varphi(x,y,X)]\,xy$$

expresses that all pairs are in the fixed-point, i.e.,

$$\mathcal{G}\models\forall x\forall y[\mathrm{FP}_{xy,X}\varphi(x,y,X)]xy \quad \Leftrightarrow \text{ for all } a, b\text{: } d(a,b)<\infty$$
$$\Leftrightarrow \mathcal{G} \text{ is connected.}$$

We come to the general case. Let $\varphi(x_1,..., x_r,X)$ be an arbitrary formula of first-order logic of vocabulary τ with a second-order variable X of arity r. Suppose $\mathcal{A}$ is a τ-structure. Then φ gives rise to the sequence $F_0(=F_0(\varphi,\mathcal{A}))$, F_1,... defined by

$$F_0:=\emptyset \text{ and } F_{n+1}:=\{(a_1,...,a_r)\in A_r \mid \mathcal{A}\models\varphi[\hat{a},F_n]\}$$

(here $\hat{a}$ abbreviates the sequence $a_1,...,a_r$). The set

$$F_+:=\{(a_1,...,a_r)\in A_r \mid \exists n_0\forall n\geq n_0:(a_1,...,a_r)\in F_n\}$$

is called the (generalized) *fixed-point* of φ in $\mathcal{A}$.

Fixed-point logic consists of all sentences of the form $\forall y_1...\ \forall y_r[\mathrm{FP}_{x_1,...,x_r,X}\varphi\ ]y_1...\ y_r$ or of the form $\exists y_1...\exists y_r[\mathrm{FP}_{x_1,...,x_r,X}\varphi\ ]y_1...y_r$. By definition, in the structure $\mathcal{A}$ they express that $F_+=A^r$ and that $F_+\neq\emptyset$, respectively[2].

[2] Usually, fixed-point logic is defined as the set of all formulas obtained from atomic ones by closing under the first-order operations and under the fixed-point operation just introduced; but it turns out

If φ is positive in X, that is, no occurrence of X in φ is in the scope of a negation symbol (and we only use the connectives $\neg$, $\wedge$, $\vee$ and the quantifiers $\exists$, $\forall$), then a simple induction shows that $F_0 \subseteq F_1 \subseteq F_2 \subseteq \ldots$ and that $F_+ = F_{|A|^r}$. So in this case we only need a number of steps polynomial in the size of the structure to reach the fixed-point. We call the corresponding fragment of fixed-point logic the *positive fixed-point logic*[3]. Indeed we have:

Theorem 2.2 (Immerman 1986, Vardi 1982)

A class of structures is contained in PTIME *if and only if it is axiomatizable in positive fixed-point logic.*

Theorem 2.1 and Theorem 2.2 yield the following purely logical reformulation of the $\mathcal{P}=\mathcal{NP}$-problem:

Corollary 2.3

PTIME=NPTIME If and only if (on ordered finite structures) every property expressible by a Σ^1_1*-sentence is expressible by a sentence of positive fixed-point logic.*

3 Space-bounded algorithms

In Theorem 2.2 we only considered sentences of fixed-point logic that were positive in the fixed-point variable. What is the expressive power of full fixed-point logic? It turns out that it corresponds to the complexity class PSPACE ("polynomial space"). A problem P is in PSPACE, if there is an algorithm $\mathbb{A}$ and a polynomial p such that for every input the algorithm decides membership in P and the memory used by $\mathbb{A}$ (number of cells of the memory or the number of tape squares of a corresponding Turing machine) is bounded by p (size of the input).

Again consider a formula $\varphi(x_1,\ldots,x_n,X)$ and a structure $\mathcal{A}$. Denote $F_n(\varphi,\mathcal{A})$ by F_n. Since $F_n \subseteq A^r$ and the power set of A^r has $2^{(|A|^r)}$ elements, there are $m,\ell \leq 2^{(|A|^r)}$ with $\ell \geq 1$ such that $F_m = F_{m+\ell}$. An easy induction shows that $F_k = F_{k+\ell}$ holds for all $k \geq m$. Hence,

$$F_+ := F_{2^{(|A|^r)}} \cap F_{2^{(|A|^r)}+1} \cap \ldots \cap F_{2^{(|A|^r)}+2^{(|A|^r)}}. \qquad (2)$$

For any n, to calculate F_{n+1} we just need F_n and not the whole sequence $F_0,\ldots,F_n$. Hence, as $|F_n| \leq |A|^r$, we only need space polynomial in the size of $\mathcal{A}$ in order to calculate F_+ using (2)[4]. This is one direction of:

---

that every such sentence is equivalent to a sentence of the form above-mentioned. In the literature, fixed-point logic is often called *partial fixed-point logic*.

[3] Since then the fixed-point F_+ is the least fixed-point of the corresponding operation, this fragment is called least *fixed-point logic* in the literature.

[4] Note that the time will be exponential in the size of $\mathcal{A}$.

Theorem 3.1 (Abiteboul & Vianu 1989)

A class of structures is contained in PSPACE *if and only if it is axiomatizable by a sentence of fixed-point logic.*

As a corollary of Theorem 2.2 and Theorem 3.1 we obtain the following logical reformulation of an open problem of complexity theory:

Corollary 3.2 (Abiteboul & Vianu 1991)

PTIME=PSPACE *if and only if positive fixed-point logic and fixed-point logic have the same expressive power.*

We give an axiomatization in fixed-point logic of a concrete class. For this purpose, we consider a version of the Game of Life. Let $\tau:=\{E,S\}$ with binary E and unary S. Let $\mathcal{G}=(G,E^G,S^G)$, where (G,E^G) is a undirected graph, i.e.,

$$(G,E^G)\models\forall x\neg Exx\wedge\forall x\forall y(Exy\rightarrow Eyx).$$

We interprete S^G as the set of vertices hosting a live cell (of some species). Set $S_0=\emptyset$, $S_1=S^G$, and for $n>1$, define S_n, the "nth generation", by the following "reproduction rule": For $a\in G$, $a\in S_n$ iff (i) or (ii) holds, where

(i) $a\in S_{n-1}$ and, in S_{n-1}, a has exactly two neighbors hosting a live cell;
(ii) in S_{n-1}, a has at least three neighbors hosting a live cell.

Then, $S_n=F_n(\varphi,\mathcal{G})$ for the following formula $\varphi(x,Y)$ where the first two lines correspond to (i), the third one to (ii), and the last one gives the right value to the first generation:

$$\begin{aligned}&(Yx\wedge\exists y\exists z((y\neq z\wedge Exy\wedge Exz\wedge Yy\wedge Yz)\\&\qquad\qquad\wedge\forall u((Exu\wedge Yu)\rightarrow(u=y\vee u=z)))\\&\vee\exists y\exists z\exists u(y\neq z\wedge u\neq z\wedge y\neq u\wedge Exy\wedge Exz\wedge Exu\wedge Yy\wedge Yz\wedge Yu)\\&\qquad\qquad\vee(\neg\exists xYx\wedge Sx).\end{aligned}$$

The statement "there is a vertex that eventually hosts a live cell" can be expressed by the sentence of fixed-point logic:

$$\exists x[\mathrm{FP}_{x,Y}\varphi(x,Y)]x.$$

Hence, this sentence axiomatizes the class of structures $\mathcal{G}=(G,E^G,S^G)$ that contain a vertex that eventually hosts a live cell. By Theorem 3.1, this class is in PSPACE.

For our last characterization of a complexity class we come back to the formula

$$\varphi(x,y,X):=(x=y\vee\exists z(Xz\wedge Ezy)).$$

we analyzed at the beginning of Section 2, where we saw that the sentence $\forall x\forall y[\mathrm{FP}_{xy,X}\varphi(x,y,X)]xy$ expresses connectivity in graphs. The formula $\varphi(x,y,X)$ can equivalently be rewritten as

$$(x=y\vee\exists u\exists z(Xuz\wedge u=x\wedge Ezy)),$$

which we abbreviate as

$$(x=y\vee\exists uz\in X(u=x\wedge Ezy));$$

hence, the corresponding fixed-point sentence expressing connectivity has the form

$$\forall x\forall y[\mathrm{FP}_{xy,X}(\psi_1(x,y)\vee\exists uz\in X\psi(u,z,x,y))]xy, \qquad (3)$$

where ψ_1 and ψ do not contain the variable X. Each tuple in a new stage of the fixed-point process in (3), say in F_{n+1}, is already witnessed by a single tuple of the preceding stage F_n. Hence, $(x,y)\in F_k$ if and only if there is a sequence $(x_1,y_1),...,(x_k,y_k)$ with $(x_k,y_k)=(x,y)$ and

$$\psi_1(x_1,y_1) \text{ and } \psi(x_i,y_i,x_{i+1},y_{i+1}) \text{ for } i<k.$$

This yields a nondeterministic algorithm $\mathbb{A}$ that, given a structure $\mathcal{B}$ as input, checks whether $\mathcal{B}\models\forall x\forall y[\mathrm{FP}_{xy,X}(\psi_1(x,y)\vee\exists u\bar{z}\in X\psi(u,\bar{z},x,y))]xy$: Since $F_0\subseteq F_1\subseteq...$, we have $F_+ = F_{|B|^2}$. For every $a,b\in B$, the algorithm $\mathbb{A}$, step by step, guesses the elements of a sequence $(a_1,b_1),...,(a_k,b_k)$ with $k\leq|B|^2$, $(a_k,b_k)=(a,b)$, and with

$$\mathcal{B}\models\psi_1(a_1,b_1) \text{ and } \mathcal{B}\models\psi(a_i,b_i,a_{i+1},b_{i+1}) \text{ for } i<k.$$

At every step of the computation the algorithm $\mathbb{A}$ only has to store the counter i, the previous guess (a_i,b_i), the actual guess (a_{i+1},b_{i+1}), and the target tuple (a,b). Since we may assume that $B=\{1,...,|B|\}$ we need $\log|B|$ bits to name the elements of B in binary. Altogether, $\mathbb{A}$ needs $\leq c\cdot\log|B|$ cells of memory to check if $\mathcal{B}$ satisfies the fixed-point formula in (3).

This can be generalized. Let τ be an arbitrary vocabulary and consider a formula of the form

$$\forall y_1...\forall y_r\left[FP_{x_1....x_r,X}\left(\psi_1\left(\bar{x}\right)\vee\exists\bar{z}\in X\psi\left(\bar{x},\bar{z}\right)\right)\right]\bar{y}. \tag{4}$$

In particular, neither ψ_1 nor ψ contains X. Again, each tuple in a new stage of the fixed-point process, say in F_{n+1}, is already witnessed by a single tuple of the preceding stage F_n. Therefore, as in the preceding example, we see that the class of structures axiomatized by the sentence in (4) is in NLOGSPACE. By definition, a problem P is in NLOGSPACE, if there is an algorithm $\mathbb{A}$ just accepting the elements of P and if there is a constant $c\in\mathbb{N}$ such that for every input the memory used by $\mathbb{A}$ (besides the memory used to store the input) is bounded by $c\cdot\log$(size of the input).

Our previous observation shows the "easy" part of the following result.

Theorem 3.3 (Immerman 1987)

A class of structures is in NLOGSPACE *if and only if it is axiomatizable in fixed-point logic by a formula of the form*

$$\forall y_1...\forall y_r\left[FP_{x_1....x_r,X}\left(\psi_1\left(\bar{x}\right)\vee\exists\bar{z}\in X\psi\left(\bar{x},\bar{z}\right)\right)\right]\bar{y}.$$

Immerman (Immerman 1988) used this characterization to obtain:

Corollary 3.4

NLOGSPACE *is closed under complements.*

BIBLIOGRAPHY

Abiteboul, S., Vianu, V. (1989) "Fixpoint extensions of first-order logic and Datalog-like languages" in *Proc. 4th IEEE Symp. on Logic in Computer Science*, 71–79.

Abiteboul, S., Vianu, V. (1991) "Generic computation and its complexity" in *Proc. 23rd ACM Symp. on Theory of Computing*, 209–219.

Büchi, J. R. (1960) "Weak second-order arithmetic and finite automata", *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 6, 66–92.

Downey, R. G., Fellows, M. R.. (1999) *Parameterized Complexity*, Springer-Verlag.

Ebbinghaus, H.-D., Flum, J. (1999) *Finite Model Theory*, Springer-Verlag, 2nd edition.

Elgot, C. C. (1961) "Decision problems of finite automata design and related arithmetics", *Trans. Amer. Math. Soc.* 98, 21–52.

Fagin, R. (1974) "Generalized first–order spectra and polynomial–time recognizable sets" in R. M. Karp (ed.), *Complexity of Computation*, SIAMAMS Proceedings, Vol. 7, 43–73.

Flum, J., Grohe, M. (2002) "Describing parameterized complexity classes" in *Proceedings of STACS'02, Lecture Notes in Computer Science 2285*, Springer-Verlag, 359-371.

Immerman, N. (1986) "Relational queries computable in polynomial time", *Information and Control* 68, 86–104.

Immerman, N. (1987) "Languages that capture complexity classes", *SIAM Journal on Computing* 16, 760–778.

Immerman, N. (1988) "Nondeterministic space is closed under complement", *SIAM Journal on Computing* 17, 935–938.

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

Papadimitriou, C. H. (1994) *Computational Complexity*, Addison-Wesley.

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

Joerg FLUM is Professor of Mathematical Logic at the Albert-Ludwigs-Universität at Freiburg. He has worked on various topics in model theory and theoretical computer science including infinitary languages, topological model theory, descriptive complexity, and parameterized complexity. He is co-author of a textbook in mathematical logic and of monographs on topological model theory and on finite model theory

Address: Institut für Mathematische Logik, Eckerstr. 1, 79104 Freiburg, Germany. Email: Joerg.Flum@mathematik.uni-freiburg.de