Information and Computation **147**, 145–170 (1998)
Article No. IC982742

Positive Versions of Polynomial Time*

Institut für Informatik FB17, Johannes Gutenberg—Universität Mainz, D-55099 Mainz, Germany

and

I. A. Stewart[†, ‡]

Department of Mathematics and Computer Science, University of Leicester, Leicester LE1 7RH, United Kingdom

We show that restricting a number of characterizations of the complexity class **P** to be positive (in natural ways) results in the same class of (monotone) problems, which we denote by **posP**. By a well-known result of Razborov, **posP** is a proper subclass of the class of monotone problems in **P**. We exhibit complete problems for **posP** via weak logical reductions, as we do for other logically defined classes of problems. Our work is a continuation of research undertaken by Grigni and Sipser, and subsequently Stewart; indeed, we introduce the notion of a positive deterministic Turing machine and consequently solve a problem posed by Grigni and Sipser.

1. INTRODUCTION

The question of monotone vs positive has a number of manifestations in theoretical computer science. Perhaps the best known is in the context of boolean circuits where Razborov [25] proved that there exist monotone boolean functions which are computable by polynomial-size sequences of boolean circuits but not by polynomial-size sequences of boolean circuits in which negation gates do not appear. The question can also be considered in the wider context of complexity classes. Grigni and Sipser [17] considered positive versions of a variety of computational models such as boolean circuits, branching programs, and nondeterministic Turing machines and, in a natural way, made them positive. For a number of complexity classes, such as **REG**, **CFL** and **NP**, they proved that the positive restriction

* An extended abstract of this paper appeared as (1996), On positive P, *in* "Proc. 11th Annual IEEE Symposium on Computational Complexity," IEEE Press, New York.

† Supported by British-German ARC Project 604.

‡ Supported by EPSRC Grant GR/H 81108. Most of the work in this paper was done while the author was at the University of Wales Swansea.

0890-5401/98 $25.00

coincides exactly with the subclass of monotone problems. They also studied classes (mostly contained within **NC**) for which the positive restriction does not, or is not known to, include all monotone problems of the class (this study was extended in [18]). Subsequently, Stewart [29] obtained logical characterizations of the class of monotone problems in **NP**. Finally, Ajtai and Gurevich [1] considered the question in the context of finite model theory and proved that there exist monotone problems which are definable in first-order logic but not by any positive first-order sentence (a simpler example is given in [32]).

In this paper we study a number of possible definitions for a positive version of **P**, obtained by restricting logics, boolean circuits, alternating Turing machines, non-deterministic auxiliary pushdown machines, and deterministic Turing machines. We show that the resulting classes of (monotone) problems are, in fact, identical and we denote this class of problems by **posP**. We also exhibit complete problems for **posP** via weak logical reductions, as we do for other logically defined complexity classes and solve a problem posed in [17] by introducing and developing the notion of a positive deterministic Turing machine.

2. SOME DIFFERENT CHARACTERIZATIONS OF P

We consider naturally defined positive versions of a variety of characterizations of the complexity class **P**, polynomial-time, involving extensions of first-order logic with a built-in successor relation, log-space bounded nondeterministic and alternating machines, deterministic polynomial-time bounded machines, and uniform sequences of boolean circuits. In particular, we prove that naturally defined positive versions of the following all define the same complexity class, and so we arrive at a stable notion of "positive **P**" (more precise definitions follow subsequently):

- LFP*[FO_s]: first-order logic, with a built-in successor relation, extended by positive applications of a least fixed point operator.
- PS*[FO_s]: first-order logic, with a built-in successor relation, extended by positive applications of an operator corresponding to the path-system problem.
- D3S*[FO_s]: first-order logic, with a built-in successor relation, extended by positive applications of an operator corresponding to the degree-at-least-3-subgraph problem.
- BC[*qffo*]: quantifier-free uniform sequences of polynomial-size boolean circuits.
- BC[*poly*]: polynomial-time uniform sequences of polynomial-size boolean circuits.
- ATM[*log*]: alternating Turing machines with logarithmically bounded work tapes.
- NauxPDM[*log*]: nondeterministic auxiliary pushdown machines with logarithmically bounded work tapes.
- DTM[*poly*]: deterministic polynomial-time Turing machines.

With regard to defining positive versions of the above, making LFP*[FO_s], PS*[FO_s], etc., positive (in the logical sense) with regard to the symbols of the underlying signature is the obvious thing to do. As for the machine characterizations, positive nondeterministic or alternating machines can be defined by making the machine random-access and demanding that whenever the machine reads an input 0 it rejects, and we can define a positive version of a deterministic Turing machine by insisting that, informally speaking, after reading a 0 it must not write a 1, unless, in the same situation except reading a 1, it also writes a 1. Of course, boolean circuits are made positive by disallowing negation gates.

Let us give a more detailed account of our formalisms (and also references to further, more complete definitions: general references are [11] for logical concepts and [24] for complexity-theoretic concepts, and we follow, for example, [29] as regards our notation).

2.1. Logical Operators

For any signature σ consisting of relation and constant symbols, by $\mathrm{FO}_s(\sigma)$ we denote the set of all first-order formulas (possibly with free variables) built using relation symbols from $\sigma \cup \{succ\}$, where *succ* is a binary relation symbol (which never appears in σ), and also the constant symbols 0 and *max* (again, which never appear in σ). By $\mathrm{FO}_s^+(\sigma)$ we denote those formulas of $\mathrm{FO}_s(\sigma)$ in which relation symbols from σ occur only positively, i.e., not within the scope of a negation symbol (note that no restriction is placed on the occurrence of any constant symbols of σ). Sentences, i.e., formulas with no free variables, of $\mathrm{FO}_s(\sigma)$ can be used to define problems over σ, i.e., sets of finite σ-structures closed under isomorphism, where a finite structure S over σ, or σ-structure, consists of a universe $|S| = \{0, 1, ..., n-1\}$ and has a relation of arity a over $|S|$ corresponding to every relation symbol of σ of arity a and a constant corresponding to every constant symbol of σ (we always assume that the universe of any structure contains at least 2 elements). We denote the set of all finite σ-structures by $\mathrm{STRUCT}(\sigma)$. Given some σ-structure S of size n, i.e., where $|S| = \{0, 1, ..., n-1\}$, and some sentence $\phi \in \mathrm{FO}_s(\sigma)$, $S \models \phi$ iff S satisfies ϕ in the obvious way except that 0 (respectively *max*) is always interpreted as $0 \in |S|$ (respectively $n-1 \in |S|$) and *succ* is always interpreted as the natural successor relation on $|S|$. We call 0 and *max* built-in constants and *succ* a built-in successor relation.[1] We denote $\bigcup_\sigma \mathrm{FO}_s(\sigma)$ by FO_s (and do likewise for other logics or classes of σ-machines defined later). Note that not every sentence $\psi \in \mathrm{FO}_s$ defines a problem (as it may be the case that the set of structures defined by ψ is not closed under isomorphism, e.g., when σ consists of a unary relation symbol U and when ψ is $\exists x(succ(0, x) \wedge U(x))$). Throughout, we are only ever concerned with sentences of FO_s defining problems, and the same goes for other logics (the reader is referred to [11] where the role of built-in relations such as *succ* is discussed).

The logic FO_s was extended in [21, 34] using the *least fixed point operator* LFP to obtain the logic ($\pm$LFP)* [FO_s], and it was proved that the problems definable

[1] We do not always differentiate between a relation or constant and a relation or constant symbol.

by the sentences of $(\pm\mathrm{LFP})^*[\mathrm{FO}_s]$ are exactly those problems in **P**. (We think of a complexity class variously as consisting of sets of strings over $\{0, 1\}$, decision problems (in the abstract complexity-theoretic sense, see [12]) and sets of finite structures closed under isomorphism. Also, we write, for example, $(\pm\mathrm{LFP})^*[\mathrm{FO}_s]$ to denote the formulas of the actual logic and also the class of problems defined by the sentences of $(\pm\mathrm{LFP})^*[\mathrm{FO}_s]$. Hence, we may write $(\pm\mathrm{LFP})^*[\mathrm{FO}_s] = \mathbf{P}$ and say that $(\pm\mathrm{LFP})^*[\mathrm{FO}_s]$ *captures* **P**.) Moreover, it was proven that the logics $(\pm\mathrm{LFP})^*[\mathrm{FO}_s]$, $\mathrm{LFP}^*[\mathrm{FO}_s]$ (the fragment of $(\pm\mathrm{LFP})^*[\mathrm{FO}_s]$ where no occurrence of the operator LFP in any formula may appear within the scope of a negation symbol), and $\mathrm{LFP}^1[\mathrm{FO}_s]$ (the fragment of $\mathrm{LFP}^*[\mathrm{FO}_s]$ where no nesting of the operator LFP is allowed) define the same class of problems.

- A formula $\psi \in \mathrm{LFP}^*[\mathrm{FO}_s(\sigma)]$ belongs to $\mathrm{LFP}^*[\mathrm{FO}_s^+(\sigma)]$ if all occurrences in ψ of relation symbols from σ are positive, with $\mathrm{LFP}^1[\mathrm{FO}_s^+(\sigma)]$ defined likewise.

An alternative method of capturing complexity classes was introduced in [22] and involves the extension of FO_s using *operators* corresponding to problems (these operators are alternatively known as *vectorized sequences of Lindström quantifiers*, and we shall see some of these operators in action in the subsequent sections). In particular, extending FO_s with positive occurrences of the transitive closure operator TC to obtain the logic $\mathrm{TC}^*[\mathrm{FO}_s]$, as in [22], enables one to capture the complexity class **NL**. Immerman's celebrated result that the class of problems defined by the logic $(\pm\mathrm{TC})^*[\mathrm{FO}_s]$, where we no longer insist that an occurrence of the operator TC does not appear within the scope of a negation symbol, coincides with the class of problems defined by the logic $\mathrm{TC}^*[\mathrm{FO}_s]$, thus yielding $\mathbf{NL} = \mathbf{co\text{-}NL}$, followed later [23] (this corollary was proved independently by Szelepcsényi [33]).

One may take any problem and extend any logic by the corresponding operator. Many complexity classes have been captured by such extensions of FO_s (see, for example, [15, 22, 28, 29]). In particular, it was proven in [29] that $(\pm\mathrm{PS})^*[\mathrm{FO}_s] = \mathrm{PS}^1[\mathrm{FO}_s] = \mathbf{P}$, where PS is the *path-system problem* and is defined as follows. Let $\sigma_{3,1,1} = \langle R, U, V\rangle$, where R is a relation symbol of arity 3 and U and V are unary relation symbols. Any $\sigma_{3,1,1}$-structure S can be regarded as a path-system, in the sense of [12], and $S \in \mathrm{PS}$ iff starting with the set of accessible vertices $\{u \in |S| : U^S(u) \text{ holds}\}$, the *sources*, eventually a vertex of $\{v \in |S| : V^S(v) \text{ holds}\}$, the *sinks*, becomes accessible in the path-system described by S, where the rule for determining accessible vertices is "z becomes accessible if there exist two accessible vertices x and y (not necessarily distinct) such that $R^S(x, y, z)$ holds" (if $R^S(x, y, z)$ holds then we say that (x, y, z) is a *rule*). The problem PS was shown to be complete for **P** via log-space reductions in [9].

- A formula $\psi \in \mathrm{PS}^*[\mathrm{FO}_s(\sigma)]$ belongs to $\mathrm{PS}^*[\mathrm{FO}_s^+(\sigma)]$ if all occurrences in ψ of relation symbols from σ are positive, with $\mathrm{PS}^1[\mathrm{FO}_s^+(\sigma)]$ defined likewise.

We also consider the logics $\mathrm{D3S}^*[\mathrm{FO}_s]$ and $\mathrm{D3S}^1[\mathrm{FO}_s]$, where D3S is the *degree-at-least*-3-*subgraph problem* and is defined as follows (see also [13] where it is referred to as the high-degree-subgraph problem). Let $\sigma_2 = \langle E\rangle$, where E is a binary relation symbol. Any σ_2-structure S can be regarded as an undirected graph,

and $S \in$ D3S iff it contains an induced subgraph with minimum degree at least 3. It was shown in [2] that D3S is complete for **P** via log-space reductions. The logics D3S*[$\mathrm{FO}_s^+(\sigma)$] and D3S^1[$\mathrm{FO}_s^+(\sigma)$] are defined as above.

The reason we focus on PS and D3S is that these problems are monotone in the following sense. Let S_1 and S_2 be σ-structures, for some signature σ. We say that S_2 is a *relational refinement* of S_1 iff $|S_1| = |S_2|$; for every constant symbol C of σ, $C^{S_1} = C^{S_2}$; and for every relation symbol R of σ, of arity a, and for every $\mathbf{u} \in |S_1|^a = |S_2|^a$, if $R^{S_1}(\mathbf{u})$ holds then $R^{S_2}(\mathbf{u})$ holds. Any problem Ω over σ is *monotone* if for any σ-structures S_1 and S_2 with S_2 a relational refinement of S_1, $S_1 \in \Omega$ implies $S_2 \in \Omega$. It is not difficult to prove that any problem defined by a sentence of either of the logics PS*[FO_s^+] or D3S*[FO_s^+] is monotone. As it is our aim to establish a stable notion of "positive **P**," it is essential that all problems in "positive **P**" be monotone (note that all problems in LFP*[FO_s^+] are indeed monotone). It should be mentioned that inflationary fixed point logic, ($\pm$IFP)*[FO_s] (see, for example, [11]), is not relevant in our context. It is a simple exercise to show that even if the relation symbols of the underlying signature are restricted only to appear positively in formulas of ($\pm$IFP)*[FO_s], we are still able to define nonmonotone problems.

It is worthwhile pausing to compare our notion of a monotone problem with the notion of monotonicity commonly encountered in complexity theory. As remarked in the Introduction, much research in computational complexity has centered on whether certain monotone boolean functions can be computed by boolean circuits of restricted size and depth without negation gates; that is, monotonicity is considered in the context of boolean functions. If it is the case that a problem is over a relational signature then the problem can obviously be equated directly with a boolean function, and having constants as part of some structure, which are not subject to montonicity constraints, would seemingly destroy this correspondence. This is not the case, however, as the constants merely allow one to consider parameterized monotone functions: such functions are also regularly considered in the literature (e.g., $\mathrm{CLIQUE}_{k,n}$ and $\mathrm{CONNECT}_n$ [5]). Alternatively, if parameterized monotone functions are to be avoided then one simply forces all signatures to be relational.

2.2. Circuits

It is well known that **P** is the class of languages over $\{0, 1\}$ accepted by polynomial-time uniform sequences of polynomial-size boolean circuits [6]; here, polynomial-time uniform means that there is a polynomial-time algorithm which, when given n (in unary), produces a description of the nth circuit of the sequence. In order that we might talk about uniformity constraints imposed through logical means, we need to say how a boolean circuit $\mathscr{C}_n$ can be considered to be a finite structure over an appropriate signature and how a set of formulas can, for every n, describe such a circuit $\mathscr{C}_n$: moreover, in our context we need only consider positive[2] boolean circuits, i.e., circuits with no negation gates.

[2] Traditionally, such circuits are called "monotone" in the literature. However, we prefer to use "monotone" exclusively as a semantic notion.

We consider a positive boolean circuit with n gates as a structure S of size n over the signature $\sigma_{pc} = \langle G_\wedge, G_\vee, G_0, G_1, I, O, C \rangle$, where $G_\wedge$, $G_\vee$, G_0, G_1, I, and O are unary relation symbols and C is a relation symbol of arity 3, as follows: the gates of the circuit are labeled 0, 1, ..., $n-1$; $G_\wedge^S(x)$ (respectively, $G_\vee^S(x)$) holds iff gate x is a $\wedge$-gate (respectively, $\vee$-gate); $G_0^S(x)$ (respectively, $G_1^S(x)$) holds iff gate x is a gate set at the constant value 0 (respectively, 1); $I^S(x)$ (respectively, $O^S(x)$) holds iff gate x is an input (respectively, the unique output) gate; and $C^S(x, y, z)$ holds iff the input gates connected to gate z are gates x and y, where $x \neq y$ (note that not every structure over σ_{pc} corresponds to a positive boolean circuit). Also, we allow our circuits to have "partially specified" gates, such as gates whose type may not be specified (for example, gates for which $\neg(G_\wedge^S(x) \vee G_\vee^S(x) \vee I^S(x) \vee G_0^S(x) \vee G_1^S(x))$ holds) or gates which do not have a full complement of inputs. Such gates we regard as "broken" and we assume that the output of such gates is always 0. As we shall see, we do this so that we can more easily define circuits using logical formulas otherwise, as our uniformity is provided by "vectorized first-order formulas" (see below), every circuit $\mathscr{C}_n$ in a uniform sequence of circuits $\{\mathscr{C}_n\}$ would necessarily have exactly n^m gates, for some (fixed) m. (We can clearly think of arbitrary boolean circuits as finite structures by including another unary relation symbol in σ_{pc} to cater for negation gates.)

We are now in a position to say what we mean by a quantifier-free uniform sequence of positive boolean circuits. We make all our boolean circuits and machine models take structures, as opposed to strings over $\{0, 1\}$, as inputs. Consequently, whereas which bit of the input string is associated with which input gate of the circuit is not usually an issue in the traditional setup, we do need to say which bit of our input structure is associated with which input gate of our circuit in our logical setting.

• A sequence $\{\mathscr{C}_n\}$ of positive boolean circuits, taking as inputs structures over the signature $\sigma = \langle R_1, R_2, ..., R_r, C_1, C_2, ..., C_c \rangle$, where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol, is *quantifier-free uniform* if there are quantifier-free formulas of FO_s:

$$\phi_\wedge(\mathbf{x}), \phi_\vee(\mathbf{x}), \phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \phi_I(\mathbf{x}), \phi_O(\mathbf{x}), \phi_C(\mathbf{x}, \mathbf{x}', \mathbf{z}), \psi_i(\mathbf{x}, \mathbf{y}_i), \xi_j(\mathbf{x}, w),$$

where $|\mathbf{y}_i| = a_i$ and $|\mathbf{x}| = |\mathbf{x}'| = |\mathbf{z}| = k$, for some k, for $i = 1, 2, ..., r$ and for $j = 1, 2, ..., c$, which involve only symbols from $\{0, max, succ\}$, describing $\{\mathscr{C}_n\}$ in the following sense. For each n, let ε_n be the structure of size n over the empty signature $\sigma_\varnothing$. The gates of $\mathscr{C}_n$ are indexed by k-tuples over $\{0, 1, ..., n-1\}$, and for every $\mathbf{a}, \mathbf{a}', \mathbf{b} \in \{0, 1, ..., n-1\}^k$ and $d \in \{0, 1, ..., n-1\}$:

a is an $\wedge$-gate (respectively, $\vee$-gate) iff $\varepsilon_n \models \phi_\wedge(\mathbf{a})$ (respectively, $\varepsilon_n \models \phi_\vee(\mathbf{a})$)

a is a gate with constant value 0 (respectively, 1) iff $\varepsilon_n \models \phi_0(\mathbf{a})$ (respectively, $\varepsilon_n \models \phi_1(\mathbf{a})$)

a is an input (respectively, output) gate iff $\varepsilon_n \models \phi_I(\mathbf{a})$ (respectively, $\varepsilon_n \models \phi_O(\mathbf{a})$)

a and **a**′ are input gates to gate **b** iff $\varepsilon_n \models \phi_C(\mathbf{a}, \mathbf{a}', \mathbf{b})$

for each $i = 1, 2, ..., r$ and for each $\mathbf{c}_i \in \{0, 1, ..., n-1\}^{a_i}$, the bit $R_i(\mathbf{c}_i)$ of the input structure, of size n, to $\mathscr{C}_n$ is associated with input gate $\mathbf{a}$ iff $\varepsilon_n \models \psi_i(\mathbf{a}, \mathbf{c}_i)$

for each $j = 1, 2, ..., c$, a 1 (respectively, 0) is associated with input gate $\mathbf{a}$ iff C_j is the unique element of $\{0, 1, ..., n-1\}$ such that $\varepsilon_n \models \xi_j(\mathbf{a}, C_j)$ (respectively, $\varepsilon_n \models \neg\xi_j(\mathbf{a}, C_j)$).

We denote the class of quantifier-free uniform sequences of positive boolean circuits taking σ-structures as input by $\mathrm{BC}^+(\sigma)[qffo]$. The notion of the sequence of boolean circuits $\{\mathscr{C}_n\}$ being polynomial-time uniform is defined analogously, and the class of polynomial-time uniform sequences of positive boolean circuits taking σ-structures as input is denoted by $\mathrm{BC}^+(\sigma)[poly]$.

Note that not every set of formulas (as in the above definition) properly defines a sequence of boolean circuits. Also, note that the constant symbols in the signature σ (if present) parameterize the resulting sequence of boolean circuits by determining whether certain input gates take the value 0 or 1, with these inputs independent of the rest of the input structure.

2.3. Machines

We now focus on making our machine models positive. A *random-access* σ-Turing machine works on σ-structures rather than strings over $\{0, 1\}$. It has an address-tape and for each relation symbol R, of arity a, from σ, a special query state q_R. Upon entering the state q_R, the contents of the address-tape are interpreted as an a-tuple $(u_1, u_2, ..., u_a) \in |S|^a$, where S is the input structure. The machine then enters one of two distinguished states, q_1 or q_0, depending on whether $R^S(u_1, u_2, ..., u_a)$ holds or does not hold, respectively (if the contents of the address-tape do not encode an a-tuple over $|S|$ then the machine crashes). The machine initially has access to the size of the input structure S and any constants of S.

Alternating Turing machines, ATMs, were originally introduced and studied in [7], where it was shown that **P** coincides exactly with those languages accepted by log-space ATMs (see also [24]).

- $\mathrm{ATM}(\sigma)[log]$ is the class of log-space random-access alternating σ-Turing machines, and, furthermore, $\mathrm{ATM}^+(\sigma)[log]$ consists of those machines for which q_0 is a rejecting state: such machines are called *positive*.

The nondeterministic auxiliary pushdown machine, NauxPDM, was defined in [8] as a nondeterministic Turing machine with an additional tape operating as a pushdown store, or stack, and it was proved that **P** coincides exactly with the class of languages accepted by log-space nondeterministic auxiliary pushdown machines.

- $\mathrm{NauxPDM}(\sigma)[log]$ is the class of log-space random-access nondeterministic auxiliary pushdown σ-machines, and, furthermore, $\mathrm{NauxPDM}^+(\sigma)[log]$ consists of those machines for which q_0 is a rejecting state: such machines are called *positive*.

Finally, let M be a k-tape deterministic Turing machine, DTM, with input alphabet $\{0, 1\}$. We call tuples $I = (q, \alpha_1, ..., \alpha_k)$, where q is a state of M and each

α_i is a tape symbol of M, *in-tuples* of M. Analogously, we define tuples $O = (q, \alpha_1, ..., \alpha_k, d_1, ..., d_k)$ in which q is a state, each α_i is a tape symbol and each d_j is a direction (say "left," "right," or "stay"), to be *out-tuples*. Hence, we view the transition function δ of M as a mapping from in-tuples to out-tuples. For two in-tuples $I = (q, \alpha_1, ..., \alpha_k)$ and $I' = (q', \alpha'_1, ..., \alpha'_k)$, we write $I \leqslant I'$ iff $q = q'$ and for every i, either $\alpha_i = \alpha'_i$ or both $\alpha_i = 0$ and $\alpha'_i = 1$ (note that the work alphabet may contain other symbols apart from 0 and 1). We define $O \leqslant O'$ for out-tuples O and O' analogously (the respective directions must be identical in both tuples): furthermore, if the state of O is rejecting and the state of O' is accepting then $O \leqslant O'' \leqslant O'$, for every out-tuple O''. This final condition allows us to signal acceptance or rejection of some input string. We call M *positive* if from $I \leqslant I'$, $\delta(I) = O$ and $\delta(I') = O'$, it always follows that $O \leqslant O'$. As usual, a positive DTM takes σ-structures as input, for some signature σ, and we assume that the input is an encoding of the relations of some σ-structure, which are presented for input as tuples over $\{0, 1\}$, with the constants of the σ-structure being given in binary using different symbols from 0 and 1; and so the constants are parameters.

Informally, M being positive means that whether the machine reads a 0 or a 1 does not affect the next state, the head movements or the written symbols different from 0 and 1. However, the transition function of M is not allowed to be such that a 0 is written immediately after reading a 1 when in the same situation except reading a 0 a 1 is then written. Note that a positive deterministic Turing machine is oblivious in a very strong sense: the state and head positions at any time depend only on the length of the input but not on the input itself (except for the time instant at which a computation halts).

As an example, consider the Turing machine M with transition function δ given by

$$\delta(q, 0) = (q, 0, R); \qquad \delta(q, 1) = (q_y, 0, S); \qquad \delta(q, b) = (q_n, b, S),$$

where q_y (respectively, q_n) is the accepting (respectively, rejecting) state, and b is the blank symbol. M is positive and accepts all strings with at least one 1.

- $\mathrm{DTM}^+(\sigma)[poly]$ is the class of polynomial-time positive deterministic Turing σ-machines.

Note that the problem accepted by any positive ATM, NauxPDM, or DTM is always monotone.

2.4. Main Result

Our main result is the equivalence of all the above definitions.

THEOREM 1. *As classes of problems*:

$$\begin{aligned} LFP^*[FO_s^+] &= LFP^1[FO_s^+] = PS^*[FO_s^+] = PS^1[FO_s^+] \\ &= D3S^*[FO_s^+] = D3S^1[FO_s^+] = BC^+[qffo] = BC^+[poly] \\ &= ATM^+[log] = NauxPDM^+[log] = DTM^+[poly]. \end{aligned}$$

We call the class of problems determined by Theorem 1 positive **P** or, for short, **posP**. Of course, all problems in **posP** are monotone. The following three sections are devoted to a proof of Theorem 1.

Before proceeding, let us comment on Theorem 1. Razborov [25] proved that no (uniform or nonuniform) sequence of polynomial-size boolean circuits can solve BPM, the (monotone) problem of deciding whether a bipartite graph has a perfect matching, which is well known to be solvable in polynomial time [24]. Hence, the class $\mathbf{P} \cap \mathbf{mono}$ of monotone problems in **P**, where **mono** is the class of all monotone problems, properly contains **posP**. By rephrasing this result logically as $\mathrm{LFP}^*[\mathrm{FO}_s^+] \subset \mathrm{LFP}^*[\mathrm{FO}_s] \cap \mathbf{mono}$, we can view this result as an analogue to Ajtai and Gurevich's result mentioned earlier [1]. However, a more "logical" analogue would be such a result in the absence of the built-in successor relation.

3. LEAST FIXED POINTS AND PATH SYSTEMS

In this section, we relate "monotone computations" using least fixed points and path systems. The main result is the following.

PROPOSITION 2. *For every signature* σ, $LFP^1[FO_s^+(\sigma)] \subseteq PS^1[FO_s^+(\sigma)]$.

Proof. Let σ be some signature and let $\Phi \in \mathrm{LFP}^1[\mathrm{FO}_s^+(\sigma)]$ be a sentence of the form:

$$\mathrm{LFP}[\lambda \mathbf{x}, X, Q_1 y_1 Q_2 y_2 \cdots Q_r y_r \phi](\mathbf{z}),$$

where $|\mathbf{x}| = m$, X is an m-ary relation symbol not in σ, $\mathbf{z}$ is an m-tuple of constant symbols, each Q_i is the quantifier $\forall$ or $\exists$, and $\phi \in \mathrm{FO}_s^+(\sigma \cup \{X\})$ is quantifier-free. We intend to prove that Φ is logically equivalent to a sentence $\Psi \in \mathrm{PS}^1[\mathrm{FO}_s^+(\sigma)]$ of the form

$$\mathrm{PS}[\lambda \mathbf{x}', \mathbf{y}', \mathbf{z}', \phi_R, \mathbf{x}', \phi_U, \mathbf{x}', \phi_V],$$

where $|\mathbf{x}'| = |\mathbf{y}'| = |\mathbf{z}'| = k$, for some k; and $\phi_R, \phi_U, \phi_V \in \mathrm{FO}_s^+(\sigma)$ are quantifier-free.

The proof (of which the basic idea is not difficult but technically somewhat tedious) takes the form of a reduction: given a σ-structure S, of size n, we construct a path system $P(S)$ such that $S \models \Phi$ iff some sink is accessible from the sources in $P(S)$. We also show that the sources, the sinks, and the accessibility relation of $P(S)$ can be described in terms of S by quantifier-free formulas $\phi_R, \phi_U, \phi_V \in \mathrm{FO}_s^+(\sigma)$.

The construction is as follows. The primary vertices of $P(S)$ represent subformulas of Φ, instantiated with elements of S, and also the formula $X(\mathbf{x})$, instantiated with elements of S (if such instantiations are not already catered for by Φ: henceforth, let us assume for simplicity that they are). We intend that such a vertex $u_\psi(\mathbf{a})$ will be accessible in $P(S)$ iff the corresponding formula $\psi^S(\mathbf{a})$ holds when X is interpreted as the least fixed point. We are interested in the truth or otherwise of the formula $X(\mathbf{z})$ and so the vertex $u_X(\mathbf{z})$ (corresponding to $X(\mathbf{z})$) will be the solitary sink of $P(S)$. As sources, we take the vertices corresponding to those instantiated literals over $\sigma \cup \{succ, =\}$ which are set to true over S. Here, a literal is an atomic

or negated atomic subformula of Φ.[3] We will "connect" these vertices using our accessibility relation in an appropriate way (and using some additional vertices).

More precisely, our vertices are arranged in $r+2$ levels. On level 0, we have all instantiations of $\phi_0 := Q_1 y_1 Q_2 y_2 \cdots Q_r y_r \phi$, i.e., all vertices of the form $u_{\phi_0}(\mathbf{a})$, where $\mathbf{a} \in |S|^m$. The truth of $\phi_0(\mathbf{a})$ (in S) depends directly on the truth of $\phi_1(\mathbf{a}, b_1) := Q_2 y_2 \cdots Q_r y_r \phi(\mathbf{a}, b_1, y_2, ..., y_r)$, for all choices of $b_1 \in |S|$. We arrange all vertices of the form $u_{\phi_1}(\mathbf{a}, b_1)$ on level 1 and connect them with rules to the vertices of the form $u_{\phi_0}(\mathbf{a})$ on level 0. These connections will depend upon whether $Q_1 = \forall$ or $Q_1 = \exists$, and will be explained below. Continuing in this fashion, we get vertices of the form $u_{\phi_i}(\mathbf{a}, b_1, ..., b_i)$ on level i, where $\phi_i := Q_{i+1} y_{i+1} \cdots Q_r y_r \phi$ and where $i = 0, 1, ..., r$, and connections between vertices on adjacent levels. Thus, on level r, all vertices correspond to instantiations of the form $\phi(\mathbf{a}, b_1, b_2, ..., b_r)$. The truth of such a (quantifier-free) formula depends directly upon the truth of instantiations of literals whose corresponding vertices lie on level $r+1$. Again, we connect the vertices of level r with those of level $r+1$ by appropriate rules, as detailed below. Note that, in general, the truth of $\phi(\mathbf{a}, b_1, b_2, ..., b_r)$ depends not only on the truth of instantiated atomic and negated atomic formulas built using the symbols of $\sigma \cup \{succ, =\}$ but also on the truth of instantiated atomic formulas involving the relation symbol X.

Let us assume for the moment that all connections between levels have been made in the correct way and that the sources of our path system are the vertices corresponding to true instantiations of literals (occurring in Φ) involving the symbols of $\sigma \cup \{succ, =\}$. Then, for any $\mathbf{a} \in |S|^m$, the vertex corresponding to $X(\mathbf{a})$ is not accessible, and $u_{\phi_0}(\mathbf{a})$ is accessible iff $\phi_0^S(\mathbf{a})$ is true when X is interpreted as the empty set, i.e., if $\mathbf{a}$ is in the first iteration of the least fixed point computation. In order to simulate the subsequent iterations, as the final step in our construction we connect, for each $\mathbf{a} \in |S|^m$, the vertex $u_{\phi_0}(\mathbf{a})$ on level 0 with the vertex corresponding to $X(\mathbf{a})$ on level $r+1$. In this way, the results of one iteration of the least fixed point computation are fed back into the path system for the simulation of the next iteration. Hence, $S \models \Phi$ iff the sink, $u_X(\mathbf{z})$, is accessible in $P(S)$.

All that remains is to explain the different connections between levels and to give the formulas ϕ_R, ϕ_U, and ϕ_V. The formulas ϕ_U and ϕ_V will have the $k = m + r + t$ free variables $v_1, v_2, ..., v_t, x_1, x_2, ..., x_m, y_1, y_2, ..., y_r$, where t denotes the number of subformulas of Φ: let us call these subformulas $\psi_1, \psi_2, ..., \psi_t$. The formula ϕ_R will have $3k$ free variables. Each k-tuple over S is to be interpreted as a vertex of the path system. The first t components encode the subformula of Φ to which the vertex corresponds and we encode ψ_i as the t-tuple $(0^{i-1}, n-1, 0^{t-i})$. The last $m+r$ components are the instantiations of the free variables of ψ_i, the formula encoded by the first t components. Thus, for example, $(0^{i-1}, n-1, 0^{t-i}, \mathbf{a}, \mathbf{b})$ encodes the vertex $u_{\psi_i}(\mathbf{a}, \mathbf{b})$. Note that not every variable of $\mathbf{x}$ and $\mathbf{y}$ need occur free in ψ_i and so the encoding process might yield numerous tuples encoding the vertex $u_{\psi_i}(\mathbf{a}, \mathbf{b})$ of $P(S)$. In such cases, we consider the tuple encoding $u_{\psi_i}(\mathbf{a}, \mathbf{b})$ to be that obtained by inserting 0 for every component $j \in \{t+1, t+2, ..., k\}$ for which the corresponding variable does not appear in ψ_i. The tuples $(0^{i-1}, j, 0^{t-i}, \mathbf{a}, \mathbf{b})$, for

[3] Note that since $\Phi \in \mathrm{LFP}^1[\mathrm{FO}_s^+(\sigma)]$, only atomic formulas involving *succ* or $=$ may be negated.

$j \neq n-1$, encode additional vertices (used, as mentioned above, to connect vertices on different levels).

Thus, we can define the sources of the path system $P(S)$ by the formula $\psi_U(\mathbf{v}, \mathbf{x}, \mathbf{y})$ defined as

$$\bigvee_i \left(v_i = max \wedge \psi_i(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{j \neq i} v_j = 0 \wedge \bigwedge_h x_h = 0 \wedge \bigwedge_l y_l = 0 \right),$$

where i ranges over the indices of all formulas ψ_i which are literals involving symbols from $\sigma \cup \{succ, =\}$ and where h (respectively, l) ranges over all indices where x_h (respectively, y_l) does not appear in ψ_i. Similarly, $\psi_V(\mathbf{v}, \mathbf{x}, \mathbf{y})$ is defined as

$$v_i = max \wedge \bigwedge_{j \neq i} v_j = 0 \wedge \mathbf{x} = \mathbf{z} \wedge \mathbf{y} = \mathbf{0},$$

where i is the index of the formula ψ_i which is $X(\mathbf{x})$. Henceforth, for the sake of ledgibility, instead of $(0^{i-1}, n-1, 0^{t-i}, \mathbf{a}, \mathbf{b})$ we shall write $(\psi_i, \mathbf{a}, \mathbf{b})$.

It remains for us to define ψ_R, the formula describing the rules connecting the vertices of different levels. Let us begin with those rules connecting the vertices on levels 0 and $r+1$. These are of the form $\langle (\phi_0, \mathbf{a}, \mathbf{0}), (\phi_0, \mathbf{a}, \mathbf{0}), (\psi_i, \mathbf{a}, \mathbf{0}) \rangle$, where i is the index of the formula ψ_i which is $X(\mathbf{x})$. These rules are easily described using a quantifier-free formula of $\mathrm{FO}_s^+(\sigma)$.

Next, consider the rules connecting levels i and $i-1$, for $i = 1, 2, ..., r$. Here, the construction depends on the quantifier Q_i. If $Q_i = \exists$ then the formula $\phi_{i-1}^S(\mathbf{a}, b_1, ..., b_{i-1})$ is true iff for some $b \in |S|$, $\phi_i^S(\mathbf{a}, b_1, ..., b_{i-1}, b)$ is true. In order to describe this dependency, we introduce the rules

$$\langle (\phi_i, \mathbf{a}, b_1, ..., b_{i-1}, b, \mathbf{0}), (\phi_i, \mathbf{a}, b_1, ..., b_{i-1}, b, \mathbf{0}), (\phi_{i-1}, \mathbf{a}, b_1, ..., b_{i-1}, \mathbf{0}) \rangle,$$

for all $b \in |S|$. Clearly, these rules can be described by a quantifier-free formula of $\mathrm{FO}_s^+(\sigma)$. If, on the other hand, $Q_i = \forall$, then the construction is more involved. Now, $\phi_{i-1}^S(\mathbf{a}, b_1, ..., b_{i-1})$ is true iff $\phi_i^S(\mathbf{a}, b_1, ..., b_{i-1}, b)$ is true, for all $b \in |S|$. In order to connect the vertices of the form $u_{\phi_i}(\mathbf{a}, b_1, b_2, ..., b_{i-1}, b)$ with those of the form $u_{\phi_{i-1}}(\mathbf{a}, b_1, b_2, ..., b_{i-1})$, we make use of the additional vertices $u_{\phi_{i-1}}^c(\mathbf{a}, b_1, b_2, ..., b_{i-1})$ encoded by tuples $(0^{j-1}, c, 0^{t-j}, \mathbf{a}, b_1, b_2, ..., b_{i-1}, \mathbf{0})$, where j is the index of ϕ_{i-1} (as a subformula of Φ) and $c \in |S| \setminus \{0, n-1\}$. The connections are given by (encodings of) the rules

$$\langle u_{\phi_i}(\mathbf{a}, b_1, ..., b_{i-1}, 0, \mathbf{0}), u_{\phi_i}(\mathbf{a}, b_1, ..., b_{i-1}, 1, \mathbf{0}), u_{\phi_{i-1}}^1(\mathbf{a}, b_1, ..., b_{i-1}, \mathbf{0}) \rangle$$

$$\langle u_{\phi_i}(\mathbf{a}, b_1, ..., b_{i-1}, c+1, \mathbf{0}), u_{\phi_i}^c(\mathbf{a}, b_1, ..., b_{i-1}, \mathbf{0}), u_{\phi_{i-1}}^{c+1}(\mathbf{a}, b_1, ..., b_{i-1}, \mathbf{0}) \rangle,$$

$$\text{for} \quad c \in |S| \setminus \{0, n-2, n-1\}$$

$$\langle u_{\phi_i}(\mathbf{a}, b_1, ..., b_{i-1}, n-1, \mathbf{0}), u_{\phi_i}^{n-2}(\mathbf{a}, b_1, ..., b_{i-1}, \mathbf{0}), u_{\phi_{i-1}}(\mathbf{a}, b_1, ..., b_{i-1}, \mathbf{0}) \rangle.$$

Again, these rules can be described by a quantifier-free formula of $\mathrm{FO}_s^+(\sigma)$.

Finally, we describe the connections between levels r and $r+1$. These connections express the fact that the truth of ϕ depends directly only on the truth of literals. We can assume that ϕ is built from atomic and negated atomic formulas using only $\vee$ and $\wedge$. Then ϕ is of the form $\chi \vee \theta$ or $\chi \wedge \theta$, where χ and θ are either literals or, in turn, of the form $\chi' \vee \theta'$ or $\chi' \wedge \theta'$. All these subformulas occur in the list $\psi_1, \psi_2, ..., \psi_t$ and consequently have an associated index. If $\theta = \chi' \wedge \theta'$ then we include (the encoding of) the rule

$$\langle u_{\chi'}(\mathbf{a}, \mathbf{b}), u_{\theta'}(\mathbf{a}, \mathbf{b}), u_{\theta}(\mathbf{a}, \mathbf{b})\rangle,$$

and if $\theta = \chi' \vee \theta'$ then we include (encodings of) the rules

$$\langle u_{\chi'}(\mathbf{a}, \mathbf{b}), u_{\chi'}(\mathbf{a}, \mathbf{b}), u_{\theta}(\mathbf{a}, \mathbf{b})\rangle$$
$$\langle u_{\theta'}(\mathbf{a}, \mathbf{b}), u_{\theta'}(\mathbf{a}, \mathbf{b}), u_{\theta}(\mathbf{a}, \mathbf{b})\rangle,$$

for every $\mathbf{a} \in |S|^m$ and $\mathbf{b} \in |S|^r$. This concludes the construction. ▮

Note also that it follows from the proof of Proposition 2 that any constant symbol in the tuple $\mathbf{z}$ which does not appear in ϕ, only appears in ϕ_V.

The proof of the following is straightforward.

LEMMA 3. *Let σ be some signature and let $\Psi \in PS^1[FO_s^+]$ be of the form*

$$PS[\lambda \mathbf{x}, \mathbf{y}, \mathbf{z}, \phi_R, \mathbf{x}, \phi_U, \mathbf{x}, \phi_V],$$

where $|\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| = k$, for some k, and $\phi_R, \phi_U, \phi_V \in \mathrm{FO}_s^+(\sigma)$ are all quantifier-free. Then Ψ is logically equivalent to the formula $\Phi \in \mathrm{LFP}^1[\mathrm{FO}_s^+(\sigma)]$ of the form

$$\exists \mathbf{x}(LFP[\lambda \mathbf{z}, X, (\phi_U(\mathbf{z}) \vee \exists \mathbf{x}\, \exists \mathbf{y}(X(\mathbf{x}) \wedge X(\mathbf{y}) \wedge \phi_R(\mathbf{x}, \mathbf{y}, \mathbf{z})))](\mathbf{x}) \wedge \phi_V(\mathbf{x})),$$

where X is a k-ary relation symbol not in σ. ▮

COROLLARY 4. *$LFP^*[FO_s^+] = LFP^1[FO_s^+] = PS^1[FO_s^+] = PS^*[FO_s^+]$, and any problem in $LFP^*[FO_s^+]$ can be defined by a sentence of the form*

$$LFP[\lambda \mathbf{x}, X, \exists y_1\, \exists y_2 ... \exists y_r \phi](max, max, ..., max),$$

for some r, where $|\mathbf{x}| = m$, for some m, X is an m-ary relation symbol not in σ, and $\phi \in FO_s^+$ is quantifier-free.

Proof. The result follows immediately from Proposition 2, Lemma 3, and Corollary 4.8 of [29].[4] ▮

Note that the normal form for $\mathrm{LFP}^*[\mathrm{FO}_s^+]$ in Corollary 4 is identical with that exhibited for $\mathrm{LFP}^*[\mathrm{FO}_s]$ in Corollary 4.4 of [30] (except $\phi \in \mathrm{FO}_s^+$ as opposed to $\phi \in \mathrm{FO}_s$). We postpone a more detailed analysis of the proof of Proposition 2 until later when we see that yet more information is forthcoming.

[4] The problem PS was defined slightly differently in [29] but the results still apply here.

4. CIRCUITS AND MACHINES

We now establish the equivalences of Theorem 1 involving circuits and machines.

PROPOSITION 5. $LFP^*[FO_s^+] \subseteq BC^+[qffo]$.

Proof. The proof follows by amending the proof of Proposition 2 (whose terminology we now adopt) so that we build a circuit as opposed to a path system, and by using Corollary 4. As the construction stands, the iterations of the least fixed point computation are performed by "re-using" vertices of the path system in the sense that the vertices on level $r+1$ are updated after each iteration, and rules are rechecked to see whether more can be applied. A boolean circuit is necessarily acyclic and so the capability for reuse is absent. However, as at most n^m iterations are required, we can "unfold" the construction so that we obtain a polynomial-size (in n) positive boolean circuit by replacing rules with gates:

- we replace any rule (α, β, γ) by a $\wedge$-gate with inputs α and β and output γ
- we add extra circuitry to ensure that once a gate takes the value 1 then all "unfolded copies" also take the value 1.

Note that by proceeding as above, the rules (α, β, γ) and $(\alpha', \beta', \gamma)$ would introduce a $\wedge$-gate, corresponding to γ, with more than 2 inputs, which would not be what was intended. However, we can include extra circuitry, involving polynomially many binary $\vee$-gates, so that a collection of such path system rules can be modeled in our circuit (analogously to how we handled the quantifier $\forall$ in the proof of Proposition 2).

Consequently, the boolean circuit consists of essentially n^m copies, stacked one on top of the other, of the basic circuit corresponding to levels 0 through $r+1$ of the path system $P(S)$ (without the rules connecting levels 0 and $r+1$), with extra circuitry added to join consecutive copies together (to cater for the rules connecting levels 0 and $r+1$) and to ensure that corresponding gates in copies higher up the circuit are set to 1 if the gate lower down is (that is, in the unfolding process every vertex of $P(S)$ on level $r+1$ corresponds to a gate in each of the n^m circuit copies, and if any one of these gates is set to 1 then all corresponding gates in higher copies must be set to 1: this mimics the fact that in the computation of the least fixed point, once $X(\mathbf{u})$ holds, for some $\mathbf{u} \in |S|^m$, then it holds henceforth). Just as with the path system $P(S)$, our positive boolean circuit can be defined using quantifier-free first-order formulas, and the result follows. ∎

As any first-order formula can be evaluated in polynomial time, we clearly have that $\mathrm{BC}^+[qffo] \subseteq \mathrm{BC}^+[poly]$.

PROPOSITION 6. $BC^+[poly] \subseteq ATM^+[log]$.

Proof. The construction of a random-access ATM to evaluate a boolean circuit is standard (see, for example, Theorem 4.4 of [3]). Beginning with the output gate, the ATM M recursively verifies that the current gate g evaluates to 1 as follows:

- if g is a $\vee$-gate then M guesses two gates, g_1 and g_2, and verifies that they are both input gates to g and that one of them evaluates to 1

- if g is a $\wedge$-gate then M guesses two gates, g_1 and g_2, and verifies that they are both input gates to g and that they both evaluate to 1
- if g is an input gate then M reads the corresponding input bit.

In order to verify that g_1 and g_2 are the input gates to g, the machine branches universally: on one branch it proceeds under the assumption that, indeed, g_1 and g_2 are the input gates to g, and on the other branch it tests whether they really are. Since the sequence of circuits is polynomial-time uniform, and $\mathbf{P} = \mathrm{ATM}[log]$, this can be done in alternating log-space without reading any input bits. Also, since every circuit is positive, only positive input information is required for acceptance, and so we may use a positive ATM. ▮

PROPOSITION 7. $ATM^+[log] \subseteq NauxPDM^+[log]$.

Proof. Again, this is a standard construction. Let M be a positive random-access alternating σ-Turing machine. A positive NauxPDM N that simulates M works as follows. To start with, N pushes M's initial configuration onto its stack, and then N repeatedly pops the topmost configuration I off its stack, copies it onto its work tape and

- if I is a universal configuration then N generates I's two successor configurations and pushes them onto the top of the stack
- if I is an existential configuration then N guesses one of I's successor configurations and pushes it onto the top of the stack
- if I is a query configuration then N executes the query and rejects if the result is 0: if the result is 1 and the stack is empty then N accepts, otherwise it proceeds with the topmost configuration.

It is not hard to see that N accepts its input iff M does. ▮

PROPOSITION 8. $NauxPDM^+[log] \subseteq LFP^*[FO_s^+]$.

Proof. The proof is based on the one given by Cook in [8] where it was shown how a $s(n)$-space-bounded NauxPDM can be simulated by a $2^{O(s(n))}$-time-bounded deterministic Turing machine.

Define a surface configuration of the positive NauxPDM N to be a configuration in which only the topmost symbol of the pushdown store is represented. There are only a polynomial number, in n, of surface configurations, for some given input structure S of size n. A pair (c, d) of such surface configurations is called realizable if on input S, N can move from c to d by a (partial) computation which never looks at stack entries below the current top entry, and at the end of which the stack has the same height as at the beginning. In particular, N accepts S iff (c_0, d_0) is realizable, where c_0 (respectively, d_0) is the start (respectively, unique accepting) configuration (we assume that the stack is emptied before acceptance).

In [8], Cook proved that the set of realizable surface configuration pairs can be defined inductively as follows:

- every pair (c, c) is realizable
- if (c_1, d_1) and (c_2, d_2) are realizable then the pair (c_1, d_3) is realizable if either:

$d_1 = c_2$ and N can move from d_2 to d_3 in one step, without moving the stack head

or

d_1 and d_3 have the same pushdown symbol, and N can move from d_1 to c_2 in one step, pushing one symbol, and from d_2 to d_3 in one step, popping one symbol.

It will be convenient for us to restrict the above definition to nonrejecting configurations.

There is a polynomial number of different surface configurations and so we can encode a surface configuration as a k-tuple, for some fixed k, over the input σ-structure S, and define $\mathrm{LFP}^*[\mathrm{FO}_s(\sigma_\varnothing)]$-formulas to describe properties of such configurations: this is a standard construction (see, for example, [11] for details). For instance, there is a formula $\phi_0(\mathbf{x})$ that holds precisely for nonrejecting configurations, where the configuration is represented using the k-tuple of variables $\mathbf{x}$, and for each arity a of a relation symbol in σ, there is a formula $\psi_a(\mathbf{x}, \mathbf{z})$, where $|\mathbf{z}| = a$, that holds for a surface configuration c, represented by the k-tuple $\mathbf{x}$, and where $\mathbf{z} = \mathbf{u} \in |S|^a$ iff the contents of the query tape in the surface configuration c encode $\mathbf{u}$.

Now we can formalize the relations between surface configurations which are needed to describe realizability as above. First, there are formulas ϕ_{start} and ϕ_{halt}, which define c_0 and d_0, respectively. We also have a formula $\phi_{sym}(\mathbf{x}, \mathbf{y})$, where $|\mathbf{x}| = |\mathbf{y}| = k$, that holds for a pair of (tuples representing) surface configurations (c, d) iff c and d have the same pushdown symbol. Similarly, there exists a formula ϕ_{push} (respectively, ϕ_{pop}, ϕ_{nil}) that holds for (c, d) iff c is not in the query state and d is reached from c in one step involving a push (respectively, a pop, no stack operation at all). All these relations between surface configurations are independent of the input and can therefore be defined in $\mathrm{LFP}^*[\mathrm{FO}_s(\sigma_\varnothing)]$.

The only situation in which the transition of N depends on the input is when N is in the query state and the answer to the query is positive (remember, in case of a negative answer, N enters the rejecting state). For each relation symbol $R \in \sigma$, of arity a, the formula $\chi_R(\mathbf{x}, \mathbf{y}) \in \mathrm{LFP}^*[\mathrm{FO}_s(\sigma_\varnothing)]$, where $|\mathbf{x}| = |\mathbf{y}| = k$, holds for (c, d) iff c is a query configuration with state q_R, and d is the same configuration but with state q_1; i.e., d is the successor configuration of c after a positive answer to the query. The transition from c to d can now be described by the following formula $\phi_R(\mathbf{x}, \mathbf{y}) \in \mathrm{LFP}^*[\mathrm{FO}_s^+(\sigma)]$:

$$\chi_R(\mathbf{x}, \mathbf{y}) \wedge \exists \mathbf{z}(\psi_a(\mathbf{x}, \mathbf{z}) \wedge R(\mathbf{z})).$$

It is now straightforward to construct a formula of $\mathrm{LFP}^*[\mathrm{FO}_s^+(\sigma)]$ which tests whether the pair (c_0, d_0) is realizable. ∎

PROPOSITION 9. $LFP^*[FO_s^+] \subseteq DTM^+[poly]$.

Proof. Let the problem $\Omega \in \mathrm{LFP}^*[\mathrm{FO}_s^+]$ be defined by a sentence as in Corollary 4. The positive DTM calculates the least fixed point by repeatedly evaluating the least fixed point formula. For all internal calculations, the machine uses symbols different from 0 and 1; except for the representation of the intermediate stages of the least fixed point. Beginning with the empty relation, intermediate stages of the least fixed point are stored as strings over $\{0, 1\}$. Here, a 1 is never changed to a 0, and since both the least fixed point relation symbol and the relation symbols of the signature of the input structure appear only positively in the formula, a 0 is never changed to 1 as a result of reading a 0. ▌

PROPOSITION 10. $DTM^+[poly] \subseteq LFP^*[FO_s^+]$.

Proof. In the general (nonpositive) setting, the respective proof proceeds as follows (see [11] or [21]). A n^k-time-bounded computation of some DTM M is encoded into several $2k$-ary relations $\{R_i^\gamma : \gamma$ a work tape symbol of $M\}$ and $\{K_i : i = 1, 2, ..., m$, where m is the number of tapes of $M\}$ and k-ary relations $\{Q^q : q$ is a state of $M\}$. These relations are amalgamated into one $(2k + c)$-ary relation, for some c, which is "filled in" by an LFP-formula. Numbers between 0 and $n^k - 1$ are encoded as k-tuples. The relations are such that:

- $R_i^\gamma(\mathbf{t}, \mathbf{x})$ holds iff after step $\mathbf{t}$ of the computation there is a symbol γ at position $\mathbf{x}$ of tape i
- $K_i(\mathbf{t}, \mathbf{x})$ holds iff after step $\mathbf{t}$ of the computation the head of tape i is positioned at cell $\mathbf{x}$
- $Q^q(\mathbf{t})$ holds iff after step $\mathbf{t}$ of the computation the state is q.

The first order part of the LFP-formula expresses the fact that the start configuration will be in the least fixed point, and also that if the configuration at time $\mathbf{t}$ is in the least fixed point then so is the configuration at time $\mathbf{t} + 1$. The only part of the whole formula where negation occurs is where $R_0^0(\mathbf{0}, \mathbf{x})$ is built from the input relations (R_0^0 describes the 0s on the input tape). Normally, the formula says that $(\mathbf{0}, \mathbf{x}) \in R_0^0$ iff $\mathbf{x}$ is not in the corresponding input relation.

In the simulation of positive DTMs we modify this construction in the following way: instead of a relation R_0^0 we make use of a relation $R^{0 \vee 1}$ which says that the symbol at the position in question is a 0 or a 1. It is straightforward that with this replacement an LFP-formula without any negations can then describe the computation. ▌

5. COMPLETE PROBLEMS FOR posP

In this section, we complete the proof of Theorem 1, and also prove that **posP** has complete problems via positive quantifier-free projections.[5] Positive quantifier-free projections are restricted versions of first-order translations: they were defined

[5] Positive quantifier-free projections were called monotone quantifier-free projections in [27] and [29] but in keeping with our philosophy in this paper we rename them.

in [27] and it was proved in [29] that PS is complete for $(\pm\mathrm{PS})^*[\mathrm{FO}_s^+]$ via positive quantifier-free projections.

In more detail, let $\sigma = \langle R_1, R_2, ..., R_r, C_1, C_2, ..., C_c\rangle$ and σ' be signatures, where each R_i is a relation symbol of arity a_i, and each C_j is a constant symbol. Let k be some positive integer and let $\phi_i(\mathbf{x}_i), \psi_j(\mathbf{y}_j) \in \mathrm{FO}_s(\sigma')$, where $|\mathbf{x}_i| = ka_i$, for $i = 1, 2, ..., r$, $|\mathbf{y}_j| = k$, for $j = 1, 2, ..., c$, and, additionally, for every σ'-structure S' and for every $j \in \{1, 2, ..., c\}$, there exists exactly one $\mathbf{u} \in |S'|^k$ such that $\psi_j^S(\mathbf{u})$ holds. Given any σ'-structure S', define the σ-structure S as follows:

- $|S| = |S'|^k$
- for each $i = 1, 2, ..., r$ and for each $\mathbf{u}_i \in |S|^{ka_i}$, $R_i^S(\mathbf{u}_i)$ holds iff $\phi_i^{S'}(\mathbf{u}_i)$ holds (here, we think of the ka_i-tuple $\mathbf{u}_i$ as an a_i-tuple of k-tuples)
- for each $j = 1, 2, ..., c$, $C_j^S = \mathbf{u}$ iff $\mathbf{u} \in |S'|^k$ is the unique k-tuple such that $\psi_j^{S'}(\mathbf{u})$ holds.

Then S is the *first-order translation* of S' w.r.t. $\{\phi_1, \phi_2, ..., \phi_r, \psi_1, \psi_2, ..., \psi_c\}$. We say that some problem Ω, over the signature σ, is a *first-order translation* of some problem Ω', over the signature σ', w.r.t. $\{\phi_1, \phi_2, ..., \phi_r, \psi_1, \psi_2, ..., \psi_c\}$ if for every σ'-structure S', $S' \in \Omega'$ iff the first-order translation S of S' w.r.t. $\{\phi_1, \phi_2, ..., \phi_r, \psi_1, \psi_2, ..., \psi_c\}$ is in Ω. We clearly also have the notion of a *quantifier-free translation.*

A quantifier-free formula $\phi \in \mathrm{FO}_s(\sigma)$ is a *quantifier-free projection* (see [22]) if ϕ is of the form

$$(\alpha_1 \wedge \beta_1) \vee (\alpha_2 \wedge \beta_2) \vee \cdots \vee (\alpha_m \wedge \beta_m),$$

for some m, where each α_i involves no relation symbol from σ (that is, only the relation symbol *succ* and the constant symbols from $\{0, max\} \cup \sigma$ appear), each β_i is atomic or negated atomic; and for $i \neq j$, α_i and α_j are mutually exclusive. Further, ϕ is a *positive quantifier-free projection* if each β_i is atomic. We clearly have the notion of one problem being a (respectively, *positive*) *quantifier-free projection* of another.

PROPOSITION 11. *There is a positive quantifier-free projection from PS to D3S.*

Proof. Let the $\sigma_{3,1,1}$-structure S be some path system of size n. If there exists some vertex x such that $U^S(x) \wedge V^S(x)$ holds then let $\rho(S)$ be a clique of size at least 4. Otherwise, define the graph $\rho(S)$ as follows. The vertices of $\rho(S)$ are arranged into rows and columns with the rows numbered 0 to $N = n^4 - 1$ (the rows can be thought of as being indexed by the elements of $|S|^4$) and the columns numbered 0 to $n-1$. For each row $i \neq 0$ and for each column j, there are vertices $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, and $d_{i,j}$. For each column j of row 0, there is a vertex $d_{0,j}$, and each column j of row 0 such that $U^S(j)$ holds, also contains vertices $a_{0,j}$, $b_{0,j}$, $c_{0,j}$, and $e_{0,j}$.

The edges of $\rho(S)$ are of two types: the first depend only on n and the second depend on the relation R^S. The edges of the first type are as follows. For every row $i \neq 0$ and for every column j, there are edges as in Fig. 1a, and for every $j \in |S|$ such that $U^S(j)$ holds there are edges as in Fig. 1b.

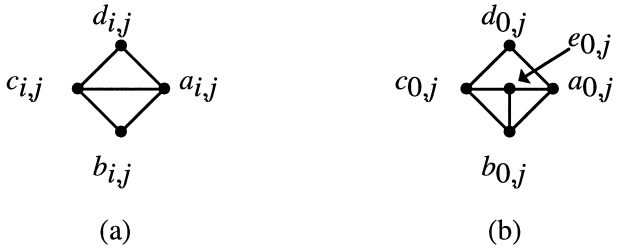

FIG. 1. The component graphs of $\rho(S)$.

Without loss of generality, we may assume that $I = \{x \in |S| : \neg V^S(x)\ h\text{olds}\} = \{0, 1, ..., p\}$ and that $J = \{y \in |S| : V^S(y)\ h\text{olds}\} = \{p+1, p+2, ..., n-1\}$. There are edges

$$\{(d_{N,i}, b_{N,i+1}) : i = 0, 1, ..., p-1\} \cup \{(b_{N,i}, d_{N,i+1}) : i = p+1, p+2, ..., n-2\} \cup \{(d_{N,p}, d_{N,p+1})\}.$$

Finally, the vertices in any column are joined via for every row $i \neq N$ and for every column j, there is an edge $(d_{i,j}, b_{i+1,j})$.

The edges of the second type are as follows. We write $m = (h, i, j, k)$ if $(h, i, j, k) \in |S|^4$ corresponds to $m \in \{0, 1, ..., N\}$. For every $m \in \{0, 1, ..., N-1\}$, set $m = (h, i, j, k)$ and:

- if $R^S(i, j, k)$ holds and $i \neq j \neq k \neq i$ then there is a new vertex such that this new vertex is joined to $a_{m,i}$, $a_{m,j}$, and $b_{m+1,k}$
- if $R^S(i, i, k)$ holds then there is an edge joining vertex $a_{m,i}$ to vertex $b_{m+1,k}$.

The graph $\rho(S)$ can be pictured as in Fig. 2.

Suppose that $\rho(S)$ has a subgraph H such that every vertex has degree at least 3. Note that for every row $m \neq N$, where $m = (h, i, j, k)$:

- $a_{m+1,l} \in H$ iff either $a_{m,l} \in H$ or $l = k$, $R^S(i, j, k)$ holds and $a_{m,i}$ and $a_{m,j}$ are both in H (it may be the case that $i = j$)
- if $a_{m,l} \in H$ then $a_{m+1,l} \in H$.

Then $(d_{N,p}, d_{N,p+1}) \in H$ and consequently $(b_{N,p+i}, d_{N-1,p+i})$ is an edge of H, for some $i \in \{1, 2, ..., n-1-p\}$. Hence, vertex $p+i$ is accessible in the path system S (as $d_{0,p+i}$ has degree 1 in $\rho(S)$).

Conversely, suppose that some vertex $p+i$ is accessible in the path system S. Then applying the rules $(0, 0, 0), (0, 0, 1), (0, 0, 2), ..., (n-1, n-1, n-2)$ in this order (if they exist) and repeating this sequence of applications $n-1$ times will certainly witness this fact. The structure of $\rho(S)$ is such as to mimic this application of rules (that is, row $m = (h, i, j, k)$ corresponds to the hth application of rule (i, j, k), if it exists), and so there is a subgraph of $\rho(S)$ of degree at least 3. As $\rho(S)$ can clearly be defined in terms of S using a positive quantifier-free projection then the result follows (see, for example, [29] for some descriptions of reductions as (positive) quantifier-free projections). ▮

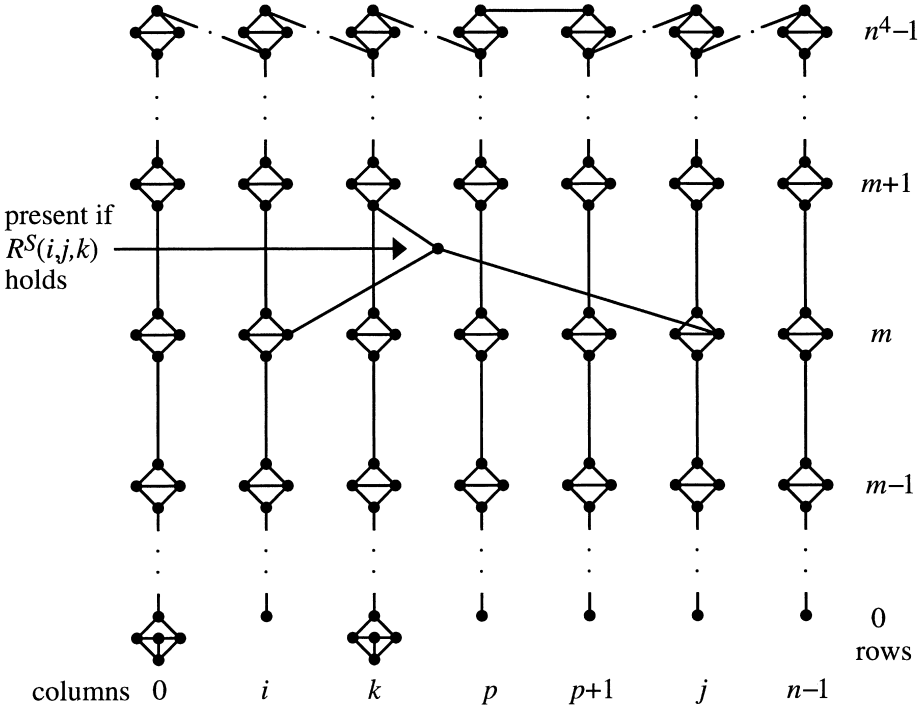

FIG. 2. The graph $\rho(S)$.

PROPOSITION 12. $D3S \in DTM^+[poly]$.

Proof. Let G be a graph with n vertices. The idea of the algorithm is to compute subgraphs $G_0, G_1, ..., G_n$ of G such that:

- $G_0 = G$
- G_{i+1} is the subgraph of G_i induced by those vertices of G_i with degree at least 3.

Clearly, G is in D3S iff G_n is not empty.

The positive DTM M successively computes strings $\omega_0, \omega_1, ..., \omega_n$ over $\{0, 1\}$, where the jth bit of ω_i is 1 iff vertex j is in G_i. M accepts G iff ω_n contains at least one 1. This last stage is easily computed by a positive DTM (using the fact that an out-tuple whose associated state is the accepting state is greater than or equal to, in the ordering on out-tuples, any other out-tuple). Also, it is easy to compute ω_0 since it depends only on the length of the input.

We now describe how M computes ω_{i+1} from ω_i. The input tape of M contains the characteristic string of the adjacency matrix of G. M makes use of five additional tapes. Tape 1 contains ω_i and tape 2 is used for writing ω_{i+1}. From each of the tapes 3, 4, and 5, only one single cell is needed. These cells c_3, c_4, and c_5 are used to count the number of neighbors of a given vertex up to the value 3.

The algorithm works as follows. For every $j \leqslant n$, M does the following. First, M resets c_3, c_4, and c_5 to 0, which signifies that no neighbors of vertex j have so far been found. For every $k \leqslant n$, whenever (j, k) is an edge of G and both j and k are

vertices of G_i, one more of the cells c_3, c_4, and c_5 is updated to 1 to signify this fact, except that if 3 neighbors of vertex j have been found then no updating is done. If, after all vertices have been tested, $c_3 = c_4 = c_5 = 1$ then vertex j is included in G_{i+1} with this fact signified in ω_{i+1}. Note that M must cycle through all neighbors of j, even if j is not in G_i, as M is strongly oblivious. For the same reason, for any vertex j, M must cycle through all remaining vertices even if 3 neighbors of j in G_i have already been found. ∎

As PS is complete for PS*[FO_s^+] via positive quantifier-free projections [29], Propositions 11 and 12 yield that D3S is complete for PS*[FO_s^+] via positive quantifier-free projections, and consequently Theorem 1 follows.

Let Ω be complete for **posP** via positive quantifier-free projections. In particular, there is a positive quantifier-free projection from PS to Ω. By [29], there is a quantifier-free projection from any problem in **P** to PS, and as the notion of quantifier-free projection is transitive [31], Ω must also be complete for **P** via quantifier-free projections. Hence, any problem complete for **posP** via positive quantifier-free projections is necessarily complete for **P** via quantifier-free projections.

Remark. Our positive DTM model seems to be able to characterize numerous complexity classes. For example, for all deterministic space classes **CC** above **LIN** (the class of problems accepted by a linear-space DTM), it characterizes exactly the monotone problems in **CC**. To see this, given an input string ω of length n, cycle through all strings of length n, using symbols different from 0 and 1, and accept whenever the original machine accepts a string $\omega' \leqslant \omega$. Also, Grigni [16] observed that for **L**, our model characterizes the class **monL** as it was defined in [18], hence solving an open problem posed in [17].

6. SOME LOGICAL CONSEQUENCES

The proof of Proposition 2 yields further results regarding logical definability, especially in the absence of a built-in successor relation; but first let us derive another complete problem for **posP** via positive quantifier-free projections. Let the signature σ_{pcv} be defined as $\sigma_{pc} \cup \langle M \rangle$, where M is a unary relation symbol, and let S be some σ_{pcv}-structure of size n which encodes a positive boolean circuit, with a unique output gate (see Section 2), together with an input string from $\{0,1\}^n$ via $M^S(i)$ holds iff the input value associated with gate i is a 1 (if gate i is not an input gate then this information is inconsequential). We would like to define the positive circuit value problem as

$$\{S \in \mathrm{STRUCT}(\sigma_{pcv}) \text{: the positive boolean circuit encoded by } S \text{ accepts the input string encoded by } M^S\}$$

(this problem was first shown to be complete for **P** via logspace reductions by Goldschlager [14]). However, this problem, as defined, is not monotone. Hence, our version of the positive circuit value problem, PCVP, is

$$\{S \in \mathrm{STRUCT}(\sigma_{pcv})\text{: the positive boolean circuit encoded by } S \text{ accepts the input string encoded by } M^S\} \cup \{S \in \mathrm{STRUCT}(\sigma_{pcv})\text{: } S \text{ does not encode a positive boolean circuit}\}.$$

The problem PCVP is monotone and can be defined in $\mathrm{LFP}^1[\mathrm{FO}_s^+]$.

COROLLARY 13. *The problem PCVP is complete for* **posP** *via positive quantifier-free projections.*

Proof. The result follows by Theorem 1, Corollary 4, the proof of Proposition 5, and the observation that the formulas defining the circuits in the proof of Proposition 5 can be made positive quantifier-free projections. ▮

Corollary 13 strengthens Goldschlager's result [14] mentioned earlier (note also that the more preferable, but nonmonotone, version of PCVP is hard for **posP** via positive quantifier-free projections, and complete for **P** via quantifier-free projections).

As remarked above, of more interest to us is the proof of Proposition 2 in the absence of a built-in successor relation. In the original proof, the successor relation is used to cater for the quantifier $Q_i = \forall$. Consequently, in the absence of a built-in successor relation we obtain weaker results. For any logic $\mathscr{L}$, let $\exists\mathscr{L}$ denote its existential fragment.

COROLLARY 14. $\exists LFP^*[FO^+] = \exists LFP^1[FO^+] = \exists PS^1[FO^+] = \exists PS^*[FO^+]$, *and PS is complete for* $\exists LFP^*[FO^+]$ *via positive quantifier-free projections without successor* (*moreover, there is an analogous result with the superscript* $^+$ *and the word "positive" omitted throughout*). *However, PS is not complete for* $LFP^*[FO]$ *via quantifier-free translations without successor, and consequently PS is not complete for* $LFP^*[FO^+]$ *via positive quantifier-free translations without successor.*

Proof. Let Φ be as in the proof of Proposition 2 except that $\phi \in \mathrm{FO}^+(\sigma \cup \{X\})$ and each $Q_i = \exists$. The proof of Proposition 2 can easily be amended to work without using a built-in successor relation and the constants 0 and *max* (note that 0 and *max* are used simply to denote two distinct elements of a structure). Hence, $\exists\mathrm{LFP}^1[\mathrm{FO}^+] \subseteq \exists\mathrm{LFP}^*[\mathrm{FO}^+] \subseteq \exists\mathrm{PS}^*[\mathrm{FO}^+]$.

Consider the proof of Theorem 4.2 in [29]. In particular, the proof of Case (iv) holds in the absence of a built-in successor relation and the constants 0 and *max*, and consequently $\exists\mathrm{PS}^*[\mathrm{FO}^+] \subseteq \exists\mathrm{PS}^1[\mathrm{FO}^+] \subseteq \exists\mathrm{LFP}^1[\mathrm{FO}^+]$ (the latter inclusion follows by Lemma 3).

The fact that the problem PS is complete for $\exists\mathrm{LFP}^*[\mathrm{FO}^+]$ via positive quantifier-free projections without successor, i.e., where a built-in successor relation and the constant symbols 0 and *max* are not used, follows from the observation that the proof of Theorem 4.2 of [29] goes through in the absence of a built-in successor relation for every case apart from Case (iii), and additionally for the case where the formula Φ (in the proof of Theorem 4.2 of [29]) is of the form $\Phi \equiv \Phi_1 \wedge \Phi_2$.

Note that every sentence of $\exists\mathrm{LFP}^*[\mathrm{FO}]$ is preserved under extensions [4], which is patently not true for $\mathrm{LFP}^*[\mathrm{FO}]$, and so the result follows. ▮

The study of the existential fragment of least-fixed point logic, $\exists$LFP*[FO], was initiated in [4] and has since been studied by Grohe [19]. In [19], a complete problem for $\exists$LFP*[FO] via quantifier-free translations was derived. Our complete problem, PS, is a much more "natural" complete problem than that presented in [19].

Let us now turn to possible analogues of Corollary 5 in the absence of a built-in successor relation. In order to transform the construction in the proof of Proposition 2 into one yielding a boolean circuit rather than a formula of PS*[FO_s^+], among other things we unfolded the path system. This unfolding seems to require the use of a built-in successor relation. So, not only do we have to get by without using the successor relation in the proof of Proposition 2, which, as remarked above, we have failed to do, we must also be able to unfold without using the successor relation (note that the transformation from a formula to a boolean circuit uses the successor relation to cope with both existential and universal quantifiers).

Suppose we relax our definition of a boolean circuit to allow unbounded fan-in for our gates. Then clearly in this model we can dispense with the successor relation to cope with both existential and universal quantifiers; but this still leaves unfolding. For obvious reasons, acyclicity is necessary for general boolean circuits. When dealing with positive boolean circuits, however, these circuits need not be acyclic so long as we define that a gate is initialized at the start of the computation as having the value 0: let us refer to such circuits as *cyclic*.

Consider the signature $\sigma_{ps} = \langle G_\wedge, G_\vee, G_0, G_1, I, in \rangle$, where all symbols are unary relation symbols except for *in* which is a relation symbol of arity 2. A *positive unbounded fan-in cyclic circuit* can be equated with a structure S over σ_{ps} as before except that $in^S(x, y)$ holds iff gate x is an input to gate y. Also, as before, we extend σ_{ps} to cater for which bits of some input structure are associated with which input gates, and we define σ_{psv} by including the extra relation symbol M of arity 1 in σ_{ps} so as to consider positive unbounded fan-in cyclic circuits together with input assignments. We define the positive unbounded fan-in cyclic circuit value problem, PUSCVP, as

$$\{S \in \mathrm{STRUCT}(\sigma_{psv}) : \text{the positive unbounded fan-in cyclic circuit encoded by } S \text{ accepts the input encoded by } M^S\}.$$

Note that PUSCVP is not monotone, and that the trick applied to "monotonize" PCVP does not work here.

Nonpositive cyclic circuits actually make sense if we insist that any input to any $\neg$-gate must be an input gate. We equate (nonpositive) cyclic circuits with structures over the signature $\sigma_s = \sigma_{ps} \cup \langle G_\neg \rangle$, where $G_\neg$ is a unary relation symbol, via $G_\neg^S(x)$ holds iff gate x is a $\neg$-gate, and for any $\neg$-gate y, $in^S(x, y)$ holds iff gate x is the input to gate y. The signature σ_s is extended to cater for which bits of some input stucture are associated with which input gates, as before, and the signature σ_{sv} and the unbounded fan-in cyclic circuit value problem, USCVP, are defined as expected.

COROLLARY 15. (i) *Any problem in LFP*[FO] (respectively, LFP*[FO$^+$]) can be recognized by a quantifier-free-uniform sequence $\{\mathscr{C}_n\}$ of polynomial-size unbounded fan-in (respectively, positive) cyclic circuits.*

(ii) *The problem PUSCVP is hard for LFP*[FO$^+$] via positive quantifier-free projections without successor, and the problem USCVP is complete for LFP*[FO] via quantifier-free projections without successor.*

Proof. By [21], any formula of LFP*[FO] is logically equivalent to a sentence of the form

$$\mathrm{LFP}[\lambda \mathbf{x}, X, \phi(\mathbf{x}, X)](\mathbf{y}),$$

where $|\mathbf{x}| = |\mathbf{y}| = k$; X is a relation symbol of arity k not in the underlying signature, and $\phi \in \mathrm{FO}$. Hence, the result follows by Corollary 14, Lemma 3, and by proceeding as in Proposition 5. ▮

The result that the problem USCVP is complete for LFP*[FO] via quantifier-free translations without successor was independently proven by Imhof [20].

We can also characterize the logic LFP*[FO] (respectively, LFP*[FO$^+$]) in terms of Petri nets as opposed to (respectively, positive) cyclic circuits. For us, a *Petri net* is a finite set of *places* P together with a (disjoint) finite set of *transitions* T. Each transition $t \in T$ has an associated set of *input* places $in(t)$ and an associated set of *output* places $out(t)$. A *marking* of a Petri net is an assignment m of a natural number to every place, and a marking is *unary* if all natural numbers assigned are either 0 or 1. A transition $t \in T$ is *enabled* for some marking m if $m(p) > 0$, for every $p \in in(t)$. If the transition t is enabled for the marking m then t can *occur*, and if t occurs then the resulting marking $m'(p)$ is given by:

- $m(p) - 1$, if $p \in in(t) \backslash out(t)$
- $m(p) + 1$, if $p \in out(t) \backslash in(t)$
- $m(p)$, otherwise,

for every place p. The *coverability problem* for Petri nets is defined as follows: given a Petri net and two markings m_1 and m_2, is there a *finite occurrence sequence* (i.e., a sequence of transition occurrences) so that starting at the marking m_1 the Petri net reaches a marking m_3 for which $m_3 \geqslant m_2$; that is, $m_3(p) \geqslant m_2(p)$, for all places p? (A general reference for Petri nets is [26].)

We also insist that our Petri nets are such that every input place of every transition is also an output place: let us call these Petri nets *all-in-out* Petri nets. Define the signature $\sigma_p = \langle in, out, M_i, M_o \rangle$, where *in* and *out* are relation symbols of arity 2 and M_i and M_o are relation symbols of arity 1. We can equate any all-in-out Petri net, together with two unary markings, with a structure S, of size n, over σ_p via the set of places is $\{0, 1, ..., n-1\}$, the set of transitions is also $\{0, 1, ..., n-1\}$, $in^S(p, t)$ holds iff the place p is an input place of transition t, $out^S(p, t)$ holds iff the place p is an output place of transition t, and the markings M_i and M_o are defined via $M_i(p) = 1$ (respectively, 0) iff $M_i^S(p)$ (respectively, $\neg M_i^S(p)$) holds, with M_o

defined similarly. The *unary coverability problem for all-in-out Petri nets*, UCPAPN, is defined as:

$$\{S \in \mathrm{STRUCT}(\sigma_p)\colon S \text{ defines an all-in-out Petri net of size } n \text{ and the unary marking } M_o \text{ is coverable from the unary marking } M_i\}.$$

Again, note that this problem is not monotone, nor can it be "monotonized" as before. The following result is immediate from Corollary 15.

COROLLARY 16. *The problem UCPAPN is hard for LFP*[FO$^+$] via positive quantifier-free projections without successor, and complete for LFP*[FO] via quantifier-free projections without successor.* ∎

While Corollary 16 follows trivially from Corollary 15, so trivially in fact that one might be tempted to omit it, we prefer to include it so as to provoke further examinations of Petri nets within the logical framework.

Originally, Dahlhaus [10] showed that LFP*[FO] has a complete problem via quantifier-free translations: the problem involves playing games on structures. Also, Grohe [19] exhibited a complete problem for LFP*[FO] via quantifier-free translations: his problem mirrors the construction of the least fixed point relation. Our completeness results do not rely on those of Dahlhaus and Grohe, and both Dahlhaus' and Grohe's problems are less natural than ours (PSCVP and UCPAPN): moreover, we show completeness via (the more restricted) quantifier-free projections without successor. We have been unable to exhibit a complete problem for LFP*[FO$^+$] via positive quantifier-free projections without successor (or even via any less restricted logical reductions) and we leave this as an open problem.

Finally, let us consider how we might bridge the gap from **posP** to $\mathbf{P} \cap \mathbf{mono}$. We have failed to capture $\mathbf{P} \cap \mathbf{mono}$ by making known characterizations of **P** positive. However, this does not rule out the existence of a syntactic definition of $\mathbf{P} \cap \mathbf{mono}$. Since BPM, the (monotone) problem of deciding whether a bipartite graph has a perfect matching, is not contained in **posP**, a natural contender for such a definition is the extension of FO_s^+ by the operator BPM, BPM*[FO_s^+]. Clearly, BPM*[FO_s^+] defines a subclass of $\mathbf{P} \cap \mathbf{mono}$. However, since BPM is in **RNC** (see [24]) and **RNC** is closed under positive first-order translations, it seems unlikely that BPM*[FO_s^+] contains any **P**-complete problems, such as PS. Thus BPM*[FO_s^+] is unlikely to contain **posP**. On the other hand, BPM*[FO_s^+] contains **posNL**, which we define here as the positive fragment of transitive closure logic (see [11, 22]).

PROPOSITION 17. $TC^*[FO_s^+] \subseteq BPM^*[FO_s^+]$.

Proof. We show that CONN, the class of all directed graphs on the vertices $\{0, 1, ..., n-1\}$ in which there is a path from u to v, can be reduced to BPM by a positive quantifier-free projection.

Let G be a directed graph on $V = \{0, 1, ..., n-1\}$. Construct a bipartite graph H as follows:

• for every vertex $x \neq u, v$ of G, H has two vertices, x_{in}, x_{out}, which are connected by an edge

• for $x = u$, H contains only u_{out}, and for $x = v$ only v_{in}

• for every directed edge (x, y) in G, H has an edge (x_{out}, y_{in}).

Clearly, this construction can be described by a positive quantifier-free projection.

Assume that $(u = x^0, x^1), (x^1, x^2), ..., (x^r, x^{r+1} = v)$ is a path in G. Then the set $\{(x^i_{out}, x^{i+1}_{in}): i = 0, 1, ..., r\} \cup \{(x_{in}, x_{out}) : x \in V \setminus \{x_0, x_1, ..., x_{r+1}\}\}$ is a perfect matching of H. On the other hand, if M is a perfect matching of H, then the set $\{(x, y) : (x_{out}, y_{in}) \in M\}$ contains a path from u to v in G. ▮

Received December 9, 1996; final manuscript received April 23, 1998

REFERENCES

1. Ajtai M., and Gurevich, Y. (1987), Monotone versus positive, *J. Assoc. Comput. Mach.* **34**, 1004–1015.
2. Anderson, A., and Mayr, E. (1984), A P-complete problem and approximations to it, Stanford University Tech. Rep. STAN-CS-84-1014.
3. Balcázar, J., Díaz, J., and Gabarró, J. (1992), "Structural Complexity," Vol. II, Springer-Verlag, Berlin.
4. Blass, A., and Gurevich, Y. (1987), Existential fixed-point logic, *in* "Lecture Notes in Computer Science," Vol. 270, pp. 20–36, Springer-Verlag, Berlin.
5. Boppana, R. B., and Sipser, M. (1990), The complexity of finite functions, *in* "Handbook of Theoretical Computer Science" Vol. A (J. van Leeuwen, Ed.), pp. 757–804, Elsevier, Amsterdam.
6. Borodin, A. (1977), On relating time and space to size and depth, *SIAM J. Comput.* **6**, 733–744.
7. Chandra, A. K., Kozen, D. C., and Stockmeyer, L. J. (1981), Alternation, *J. Assoc. Comput. Mach.* **28**, 114–133.
8. Cook, S. A. (1971), Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18**, 4–18.
9. Cook, S. A. (1974), An observation on time-storage trade off, *J. Comput. System Sci.* **9**, 308–316.
10. Dahlhaus, E. (1987), Skolem normal forms concerning the least fixed point, *in* "Lecture Notes in Computer Science," Vol. 270, Springer-Verlag, Berlin.
11. Ebbinghaus, H. D., and Flum, J. (1995), "Finite Model Theory," Springer-Verlag, Berlin.
12. Garey, M. R., and Johnson, D. S. (1979), "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, New York.
13. Greenlaw, R., Hoover, H. J., and Ruzzo, W. L. (1995), "Limits to Parallel Computation," Oxford Univ. Press, London.
14. Goldschlager, L. M. (1977), The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9**, 25–29.
15. Gottlob, G. (1995), Relativized logspace and generalized quantifiers over finite structures, *in* "Proc. 10th Ann. IEEE Symp. on Logic in Computer Science," pp. 65–78.
16. Grigni, M. (1996), Personal communication.
17. Grigni, M., and Sipser, M. (1992), Monotone complexity, *in* "Boolean Function Complexity" (M. S. Paterson, Ed.), pp. 57–75, Cambridge Univ. Press, Cambridge, UK.
18. Grigni, M., and Sipser, M. (1995), Monotone separation of logarithmic space from logarithmic depth, *J. Comput. System Sci.* **50**, 433–437.

19. Grohe, M. (1994), "The Structure of Fixed-Point Logics," Ph.D. thesis, Albert-Ludwigs Universität, Freiburg i. Br.
20. Imhof, H. (1996), "Fixed Points Logics and Generalized Quantifiers in Descriptive Complexity," Ph.D. thesis, Albert-Ludwigs Universität, Freiburg i. Br.
21. Immerman, N. (1986), Relational queries computable in polynomial time, *Inform. and Control* **68**, 86–104.
22. Immerman, N. (1987), Languages which capture complexity classes, *SIAM J. Comput.* **16**, 760–778.
23. Immerman, N. (1988), Nondeterministic space is closed under complementation, *SIAM J. Comput.* **17**, 935–938.
24. Papadimitriou, C. H. (1994), "Computational Complexity," Addison–Wesley, Reading, MA.
25. Razborov, A. A. (1985), A lower bound on the monotone network complexity of the logical permanent, *Mat. Zametki* **37**, 887–900. [In Russian. English translation in 1985, *Math. Notes* **37**, 485–493].
26. Reisig, W. (1985), Petri nets, *in* "EATCS Monographs on Theoretical Computer Science," Vol. 4, Springer-Verlag, Berlin.
27. Stewart, I. A. (1991), Complete problems involving boolean labelled structures and projection translations, *J. Logic Computat.* **1**, 861–882.
28. Stewart, I. A. (1992), Using the Hamiltonian path operator to capture NP, *J. Comput. System Sci.* **45**, 127–151.
29. Stewart, I. A. (1994), Logical description of monotone NP problems, *J. Logic Computat.* **4**, 337–357.
30. Stewart, I. A. (1994), Context-sensitive transitive closure operators, *Ann. Pure App. Logic* **66**, 277–301.
31. Stewart, I. A. (1995), Completeness of path problems via logical reductions, *Inform. Computat.* **121**, 123–134.
32. Stolboushkin, A. (1995), Finite monotone properties, *in* "Proc. 10th Ann. IEEE Symp. on Logic in Computer Science," pp. 324–330.
33. Szelepcsényi, R. (1988), The method of forced enumeration for nondeterministic automata, *Acta Informat.* **26**, 279–284.
34. Vardi, M. Y. (1982), The complexity of relational query languages, *in* "Proc. 14th Ann. ACM Symp. on Theory of Computing," pp. 137–146.