Fixpoint Logics, Relational Machines, and Computational Complexity

Serge Abiteboul*
INRIA

Moshe Y. Vardi†
Rice University

Victor Vianu‡
University of California, San Diego

August 11, 1996

Abstract

We establish a general connection between fixpoint logic and complexity. On one side, we have fixpoint logic, parameterized by the choices of 1st-order operators (inflationary or noninflationary) and iteration constructs (deterministic, nondeterministic, or alternating). On the other side, we have the complexity classes between P and EXPTIME. Our parameterized fixpoint logics capture the complexity classes P, NP, PSPACE, and EXPTIME, but equality is achieved only over ordered structures.

There is, however, an inherent mismatch between complexity and logic – while computational devices work on encodings of problems, logic is applied directly to the underlying mathematical structures. To overcome this mismatch, we use a theory of relational complexity, which bridges the gap between standard complexity and fixpoint logic. On one hand, we show that questions about containments among standard complexity classes can be translated to questions about containments among relational complexity classes. On the other hand, the expressive power of fixpoint logic can be precisely characterized in terms of relational complexity classes. This tight, three-way relationship among fixpoint logics, relational complexity and standard complexity yields in a uniform way logical analogs to all containments among the complexity classes P, NP, PSPACE, and EXPTIME. The

*Address: INRIA, BP 105, 78153 Le Chesnay CEDEX, France. E-mail: `Serge.Abiteboul@inria.fr`.

†Part of this work was done while this author was at the IBM Almaden Research Center. Address: Dept. of Computer Science, MS 132, Rice University, Houston, TX 77005-1892, USA. E-mail: `vardi@cs.rice.edu`, URL: `http://www.cs.rice.edu/~vardi`.

‡This author was supported in part by the National Science Foundation under grant IRI-9221268.
Address: CSE C-0114, UC San Diego, La Jolla, CA 92093, USA. E-mail: `vianu@cs.ucsd.edu`.

logical formulation shows that some of the most tantalizing questions in complexity theory boil down to a single question: the relative power of inflationary vs. noninflationary 1st-order operators.

1 Introduction

The *computational complexity* of a problem is the amount of resources, such as time or space, required by a machine that solves the problem. Complexity theory traditionally has focused on the computational complexity of problems. A more recent branch of complexity theory, started by Fagin in [Fag74, Fag75] and developed during the 1980s, focuses on the *descriptive complexity* of problems, which is the complexity of describing problems in some logical formalism [Imm87a]. One of the exciting developments in complexity theory is the discovery of a very intimate connection between computational and descriptive complexity.

This intimate connection was first discovered by Fagin, who showed that the complexity class NP coincides with the class of properties expressible in existential 2nd-order logic [Fag74] (cf. [JS74]). Another demonstration of this connection was shown by Immerman and Vardi, who discovered tight relationships between the complexity class P and inflationary fixpoint logic [Imm86, Var82] and between the class PSPACE and noninflationary fixpoint logic [Var82]; see also [Imm82].[1] The tight connection between descriptive and computational complexity, typically referred to as the connection between "logic and complexity", was then proclaimed by Immerman [Imm87b], and studied by many researchers [Com88, Goe89, Gra84, Gra85, Gur83, Gur84, Gur88, HP84, Imm89, Lei89a, Liv82, Liv83, Lyn82, Saz80b, Saz80a, TU88].[2] See [Imm89] for a survey.

Although the relationship between descriptive and computational complexity is intimate, it is not without its problems, and the "partners" do have some irreconcilable differences. While computational devices work on *encodings* of problems, logic is applied directly to the underlying mathematical structures. As a result, machines are able to enumerate objects that are logically unordered. For example, while we typically think of the set of nodes in a graph as unordered, it does become ordered when it is encoded on the tape of a Turing machine. This "impedance mismatch" does not pose any difficulty in the identification of NP with existential 2nd-order logic (in [Fag74]), since the logic can simply assert the existence of the desired order. The relationship between the class P and inflationary fixpoint logic is, however, more complicated as a result of the mismatch. Although inflationary fixpoint logic can describe P-complete problems, there are some very easy problems in P that are not expressible in inflationary fixpoint logic (e.g., checking whether the cardinality of the structure is even [CH82]). It is only when we assume built-

[1]Inflationary and noninflationary fixpoint logics are defined in Section 2. Vardi actually proved a correspondence between PSPACE and the query language RQL. Abiteboul and Vianu [AV89] later proved the equivalence of RQL and noninflationary fixpoint logic.

[2]The focus here is on the connection between finite-model theory and complexity. The connection between logic and complexity has also a proof-theoretic aspect; see [Bus86, GSS90, Lei91].

in order that we get that P coincides with the class of properties expressible in inflationary fixpoint logic [Imm86, Var82]. Similarly, it is only when we assume a built-in order that we get that PSPACE coincides with the class of properties expressible in noninflationary fixpoint logic [Var82].

A consequence of Fagin's result is that NP=co-NP if and only if existential and universal 2nd-order logic have the same expressive power. This equivalence of questions in computational and descriptive complexity is one of the major features of the connection between the two branches of complexity theory. It holds the promise that techniques from one domain could be brought to bear on questions in the other domain.

Unfortunately, the order issue complicates matters. The results by Immerman and Vardi show that P=PSPACE if and only if inflationary and noninflationary fixpoint logics have the same expressive power over *ordered* structures. We would like, however, to eliminate the restriction to ordered structures for two reasons. The first reason is technical; questions about logical expressiveness over ordered structures are typically much harder than their unordered counterparts; see for example [dR84, Sch94]. The second reason is more fundamental; the restriction to ordered structures seems to be a technical device rather than an intrinsic feature. Because of this restriction, the above equivalence provides a translation of the P vs. PSPACE question to a descriptive complexity question, but does not provide a translation of the inflationary vs. noninflationary question to a computational complexity question. Thus, we would like to obtain an order-free correspondence between questions in computational and descriptive complexity. (See also [IL90] for a discussion of the order issue from another perspective.)

The order issue was partially overcome recently by Abiteboul and Vianu [AV95], who showed that indeed P=PSPACE if and only if inflationary and noninflationary fixpoint logics have the same expressive power. The crux of their result is a description of a certain "internal" order in inflationary fixpoint logic. While this order is much weaker than the total order available to computational devices, it is sufficient to enable the translation of the P=PSPACE question into a logical one [AV95].

Although this result goes a long way in overcoming the order issue, it is not completely satisfying. We now have two very different ways to relate descriptive and computational complexity: Fagin's result, extended later to a correspondence between the polynomial hierarchy and 2nd-order logic [Sto77], enables us to phrase questions about the polynomial hierarchy in terms of second-order logic, and the result of [AV95] enables us to phrase the P vs. PSPACE question in terms of fixpoint logic. Because the two formalisms are so different, it is not clear whether they can be combined to yield a descriptive-complexity analog to the P=NP question.

Our goal in this paper is to extend the results and techniques of [AV95] into a general connection between fixpoint logics and complexity classes between P and EXPTIME. Our results are based on several fundamental ideas. At the heart of our framework is the view of fixpoint logic as 1st-order logic augmented with iteration. While traditionally fixpoint logic was viewed as the extension of 1st-order logic by recursion (cf. [Mos74]), iteration proved to be a more general extension to 1storder logic than recursion [AV89, GS86, Imm82, Lei90]. In both inflationary and noninflationary fixpoint logics, iteration is applied in its simplest form: sequential and deterministic. It turns out, however, that in order to express certain problems in fixpoint logic one seems to require more elaborate forms of iteration, such as nondeterministic or alternating. Indeed, part of this research was motivated by the question whether PSPACE-complete problems such as "quantified Boolean formulas" or "nonuniversality for finite automata" [GJ79] can be described in noninflationary fixpoint logic. By augmenting fixpoint logic with nondeterministic or alternating iteration we find that the connection between fixpoint logic and computational complexity is quite broad; it covers not only the classes P and PSPACE, but also the classes NP and EXPTIME. For example, over ordered structures NP coincides with the class of properties expressible in nondeterministic inflationary fixpoint.

This broad connection between fixpoint logic and computational complexity still suffers from the order mismatch discussed above. To get around this difficulty, we expand a theory of *relational* complexity, initiated in [AV91b, AV95], which bridges the gap between standard complexity and fixpoint logic. In relational complexity theory, computations on unordered structures are modeled by *relational machines*. Relational machines are Turing machines that are augmented by a relational store and the ability to perform relational operations on that store. This idea of extending 1st-order logic with a general computational capability was suggested in [CH80] and pursued in [AV95].[3] Unlike Turing machines, which operate on encodings of problems, relational machines operate directly on the underlying mathematical structures.

For Turing machines, the most natural measure of complexity is in terms of the size of the input. This measure is not the most natural one for relational machines, since such machines cannot measure the size of their input. In fact, relational machines have a limited *discerning* power, i.e., the power to distinguish between different pieces of their input, since they are only able to manipulate their input relationally. The discerning power of relational machines is best understood by viewing them as an effective fragment of a certain infinitary logic, studied recently in [KV92, KV95]. This view yields a precise characterization of the discerning power of relational machines in terms of certain infinite 2-player pebble games. The characterization can then be used by relational machines in order to determine their own discerning power. This suggests that it is natural to measure the size of the input to a relational machine with respect to the discerning power of the machine. The new measure gives rise to a new notion of computational complexity, called *relational complexity*, resulting in classes such as P_r (relational polynomial time) and NP_r (relational nondeterministic polynomial time).

Relational complexity can now serve as a mediator between logical complexity and standard complexity. On one hand, we show that questions about containments among standard complexity classes can be translated to questions about containments among relational complexity classes. On the other hand, the expres-

[3]A closely related idea, of generalizing Turing machines to operate on general structures, goes back to [Fri71] and was investigated extensively in [Lei89a, Lei89b].

sive power of fixpoint logic can be *precisely* characterized in terms of relational complexity classes. This tight, three-way relationship among fixpoint logics, relational complexity and standard complexity yields in a uniform way logical analogs to all containments among the complexity classes P, NP, PSPACE, and EXPTIME. It also enables us to translate known relationships among complexity classes, such as the equality of PSPACE, NPSPACE, and APSPACE, into results about the expressive power of fixpoint logics. This fulfills the promise of applying results from one domain to another domain and shows that some of the most tantalizing questions in complexity theory – P. vs. PSPACE, NP vs. PSPACE, and PSPACE vs. EXPTIME – boil down to *one fundamental issue*: understanding the relative power of inflationary vs. noninflationary 1st-order operators.

2 Fixpoint Logics

Let $\varphi(x_1, \ldots, x_n, S)$ be a 1st-order formula in which S is a n-ary relation symbol (not included in a vocabulary σ) and let $\mathbf{D}$ be a structure over the vocabulary σ. The formula φ gives rise to an operator $\Phi(S)$ from n-ary relations on the universe D of $\mathbf{D}$ to n-ary relations on D, where $\Phi(T) = \{(a_1, \ldots, a_n) : \mathbf{D} \models \varphi(a_1, \ldots, a_n, T)\}$, for every n-ary relation T on D.

Every such operator $\Phi(S)$ generates a sequence of *stages* that are obtained by iterating $\Phi(S)$. The stages Φ^m (also denoted by φ^m), $m \geq 1$, of Φ on $\mathbf{D}$, are defined by the induction: $\Phi^1 = \Phi(\emptyset)$, $\Phi^{m+1} = \Phi(\Phi^m)$. Intuitively, one would like to associate with an operator $\Phi(S)$ the "limit" of its stages. This is possible only when the sequence Φ^m, $m \geq 1$, of the stages "converges", i.e., when there is an integer m_0 such that $\Phi^{m_0} = \Phi^{m_0+1}$ and, hence, $\Phi^{m_0} = \Phi^m$, for all $m \geq m_0$. Notice that in this case Φ^{m_0} is a fixpoint of $\Phi(S)$, since $\Phi^{m_0} = \Phi^{m_0+1} = \Phi(\Phi^{m_0})$. The sequence of stages may not converge. In particular, this will happen if the formula $\varphi(\mathbf{x}, S)$ has no fixpoints.

2.1 Inflationary Fixpoint Logic

A formula $\varphi(x_1, \ldots, x_n, S)$ is *inflationary in* S if $T \subseteq \Phi(T)$ for any n-ary relation T. In particular, φ is inflationary in S if it is of the form $S(x_1, \ldots, x_n) \vee \psi(x_1, \ldots, x_n, S)$. If φ is inflationary in S, then the sequence Φ^m, $m \geq 1$, of stages is increasing. Thus, it must have a "limit". More precisely, if $\mathbf{D}$ is a finite structure with s elements, then there is an integer $m_0 \leq s^n$ such that $\Phi^{m_0} = \Phi^m$ for every $m \geq m_0$. That is, the sequence of stages of $\varphi(\mathbf{x}, S)$ converges to Φ^{m_0}. We write φ^∞ or Φ^∞ to denote the fixpoint Φ^{m_0} of φ. *Inflationary fixpoint logic* (IFP) is 1st-order logic augmented with the inflationary fixpoint formation rule for inflationary formulas[4]. The canonical example of a formula of inflationary fixpoint logic

[4] In the following, we assume w.l.o.g. that each inflationary formula is of the form

$$S(x_1, ..., x_n) \vee \psi(x_1, ..., x_n, S)$$

.

is provided by the inflationary fixpoint $\varphi^\infty(x, y)$ of the 1st-order formula

$$S(x, y) \vee E(x, y) \vee (\exists z)(S(x, z) \wedge S(z, y)).$$

In this case, $\varphi^\infty(x, y)$ defines the *transitive closure* TC of the *edge* relation E. It follows that *connectivity* is a property expressible in inflationary fixpoint logic, but, as is well known (cf. [Fag75, AU79]), not in 1st-order logic.

Remark 2.1: Fixpoints can also be guaranteed to exist when the formula $\varphi(x_1, \ldots, x_n, S)$ is *positive in* S, namely, when every occurrence of S is governed by an even number of negations. In that case, the sequence Φ^m, $m \geq 1$, of stages is also increasing, and consequently has a limit that is a fixpoint. In fact, that limit is the least fixpoint of φ. *Positive fixpoint logic* is 1st-order logic augmented with the least fixpoint formation rule for positive formulas. It is easy to see that IFP is at least as expressive as positive fixpoint logic. Gurevich and Shelah showed that in fact the two logics have the same expressive power [GS86] (see also [Lei90]). ∎

The complexity-theoretic aspects of IFP were studied in [CH82, Imm86, Var82]. (These papers actually focused on positive fixpoint logic, but, as observed above, positive fixpoint logic and IFP have the same expressive power.) It is known that IFP captures the complexity class P, where a logic L is said to *capture* a complexity class C if the following holds:

1. All problems expressible in L are in C, and

2. on ordered structures L can express all problems in C.

(A more natural definition would be to require that a problem is in C iff it is expressible in L. Because of the order mismatch, this requirement would be too stringent.) Note, however, that there are problems in P (e.g., checking whether the cardinality of the structure is even) that are not expressible in IFP [CH82].

2.2 Noninflationary Fixpoint Logic

How can we obtain logics with iteration constructs that are more expressive than inflationary fixpoint logic? A more powerful logic results if one iterates general 1st-order operators, until a fixpoint is reached (which may never happen). In this case, we may have non-terminating computations, unlike inflationary fixpoint logic, where the iteration was guaranteed to converge. Let Φ^m, $m \geq 1$, be the sequence of stages of the operator $\Phi(S)$ associated with a formula $\varphi(x_1, \ldots, x_n, S)$. If there is an integer m_0 such that $\Phi^{m_0} = \Phi^{m_0+1}$, then we put $\varphi^\infty = \Phi^\infty = \Phi^{m_0}$; otherwise, we set $\varphi^\infty = \Phi^\infty = \emptyset$.[5] We call φ^∞ the *noninflationary fixpoint* of φ on $\mathbf{D}$. *Noninflationary Fixpoint Logic* (NFP) is 1st-order logic augmented with the noninflationary fixpoint formation rule for arbitrary 1st-order formulas. Note that NFP fixpoint logic is an extension of IFP. While IFP formulas can be evaluated

[5] Actually, it is shown in [AV90] that there is no loss of generality in restricting attention to converging 1st-order operators.

in polynomial time, NFP can express PSPACE-complete problems, such as the *network convergence* problem [Fed91].

Noninflationary fixpoint logic was introduced by Abiteboul and Vianu [AV91a] (who called it *partial* fixpoint logic). In particular, they observed that noninflationary fixpoint logic coincides with the query language RQL introduced in [CH82] and studied further in [Var82]. It follows that NFP captures the complexity class PSPACE [Var82], but the problem of even cardinality is not expressible in NFP [CH82].

Clearly, if IFP and NFP have the same expressive power, then P=PSPACE. It is not clear, however, that the converse holds because IFP and NFP need order to "fully capture" P and PSPACE, respectively. Nevertheless, P=PSPACE if and only if IFP and NFP have the same expressive power [AV95].

2.3 Nondeterministic Fixpoint Logics

IFP and NFP are obtained by iterating inflationary and noninflationary 1st-order operators, respectively. In both cases, however, the iteration is sequential and deterministic. Certain problems, however, seem to defy description by such iterations.

As an example, consider nonuniversality of finite automata over a binary alphabet, which is known to be PSPACE-complete [GJ79]. That is, we are given a finite automaton over the alphabet $\{0, 1\}$ and we want to know whether there is a word rejected by the automaton. An instance of this problem can be viewed as a structure $(S, S_0, F, \rho_0, \rho_1)$, where S is the set of states, S_0 is the set of initial states, F is the set of accepting states, $\rho_0 \subseteq S^2$ is the transition relation for the symbol 0 and $\rho_1 \subseteq S^2$ is the transition relation for the symbol 1. To check whether there is a word rejected by the automaton, one can simply guess a word and check that it is rejected by the automaton. This can be easily described by a *nondeterministic iteration* of 1st-order operators. Let us define what nondeterministic iteration is.

Let Φ_1 and Φ_2 be 1st-order operators. This pair of operators generates *convergent sequences* of stages that are obtained by successively applying, until convergence is reached, Φ_1 or Φ_2. That is, the pair generates sequences of the form $S_0, S_1, \ldots, S_m$, where $S_0 = \emptyset$, either $S_{i+1} = \Phi_1(S_i)$ or $S_{i+1} = \Phi_2(S_i)$, and $\Phi_1(S_m) = \Phi_2(S_m) = S_m$. In such a convergent sequence, there are m stages. We call S_m a *local nondeterministic fixpoint* of the pair Φ_1, Φ_2. Note that the pair Φ_1, Φ_2 can have more than one local nondeterministic fixpoint or none. We define the *nondeterministic fixpoint* of the pair Φ_1, Φ_2 as the union of all local nondeterministic fixpoints of the pair Φ_1, Φ_2. If no local nondeterministic fixpoint exists, then we define the nondeterministic fixpoint to be the empty set. The underlying intuition is that the outcome of a nondeterministic computation is defined as the disjunction of the outcomes of all possible computations. The number of stages of a nondeterministic fixpoint is taken to be the maximum over all convergent sequences.

Nondeterministic fixpoint logics are obtained by augmenting 1st-order logic with the nondeterministic (inflationary and noninflationary) fixpoint formation rules, under the restriction that negation cannot be applied to nondeterministic fixpoints.

The readers can now convince themselves that the nonuniversality problem can be expressed by a nondeterministic fixpoint of the noninflationary operators Φ_1 and Φ_2 corresponding to the next input symbol being 0 or 1. The problem, however, does not seem to be expressible by a deterministic iteration. Savitch's Theorem [Sav80] tells us how to convert the nondeterministic polynomial-space algorithm into a deterministic polynomial-space algorithm, but this construction assumes a built-in order and it is not obvious how to do that by a deterministic iteration. (It will follow later from our results that this problem is expressible by a deterministic iteration.)

Remark 2.2: The nondeterminism described above resides in the *control*, i.e. the choice of the formula to be applied at each stage. This should be contrasted with the *data nondeterminism* of [AV91a], which allows to nondeterministically choose an arbitrary tuple from a relation. Data nondeterminism is strictly stronger than control nondeterminism. For instance, with data nondeterminism, one can (nondeterministically) order the domain elements and use the constructed order to compute the parity of the set of domain elements. ∎

2.4 Alternating Fixpoint Logics

For an example that motivates another extension of fixpoint logic, consider truth of quantified Boolean formulas, also known to be PSPACE-complete [GJ79]. This problem, which can be trivially solved by an alternating polynomial algorithm [CKS81], is most naturally expressed by an *alternating iteration* of 1st-order operators. Let us define what alternating iteration is.

Let Φ and Ψ be 1st-order operators. This pair of operators generates *convergent trees* of stages that are obtained by successively applying, until convergence is reached, either one of Φ and Ψ or both of Φ and Ψ. More formally, a convergent tree is a labeled binary tree such that:

1. The root is labeled by the empty relation,
2. if a node x with label S_x is at an odd level of the tree, then x has one child x' labeled by $\Phi(S_x)$ or $\Psi(S_x)$,
3. if a node x with label S_x is at an even level of the tree, then x has two children x_1 and x_2 labeled by $\Phi(S_x)$ and $\Psi(S_x)$, respectively, and
4. if x is a leaf with label S_x, then $\Phi(S_x) = \Psi(S_x) = S_x$.

We take the intersection of the labels of the leaves of a convergent stage tree to be a *local alternating fixpoint* of the pair Φ, Ψ. For instance, a convergent stage tree is represented in Figure 1. Its local fixpoint is:

$$\Phi(\Phi(\emptyset)) \cap \Phi(\Psi(\Psi(\Phi(\emptyset)))) \cap \Psi(\Psi(\Psi(\Phi(\emptyset)))).$$

The underlying intuition is that in a computation tree of an alternating machine all the existential choices have already been factored out. Thus, one needs to take

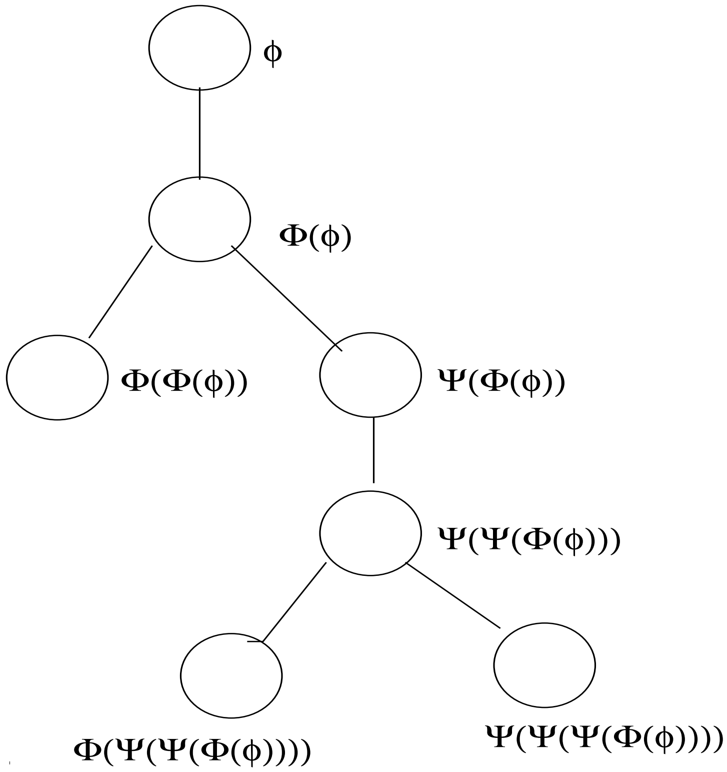

Figure 1: A convergent stage tree

the conjunction of all the universal choices and then take a disjunction over all computation trees. The number of stages in a convergent tree is the length of the longest branch.

Note that the pair Φ, Ψ can have more than one local alternating fixpoint or none. We define the *alternating fixpoint* of the pair Φ, Ψ as the union of all local alternating fixpoints of the pair Φ, Ψ. This union is essentially a disjunction over all the existential choices that were factored out in each convergent tree. If no local alternating fixpoint exists, then we define the alternating fixpoint to be the empty set. The number of stages of a nondeterministic fixpoint is taken to be the maximum over all convergent trees.

The discussion so far shows that fixpoint logics can be parameterized along two dimensions: the power of their iteration construct, *d*eterministic vs. *n*ondeterministic vs. *a*lternating, and the power of their 1st-order operators, *i*nflationary vs. *n*on-inflationary. This gives rise to six fixpoint logics. We use the notation $FP(\alpha, \beta)$ to refer to a fixpoint logic with iteration of type α and operators of type β. Thus, the logics IFP and NFP will be denoted $FP(D,i)$ and $FP(D,n)$, respectively, and $FP(A,n)$ denotes alternating noninflationary fixpoint logic.

Let α, β be as above. *Formulas* in $FP(\alpha, \beta)$ are obtained starting from atoms, by repeated applications of first-order operators $(\neg, \vee, \wedge, \exists, \forall)$ and the fixpoint operator. The fixpoint formation rule in $FP(\alpha, \beta)$ for α in D is the familiar one. For α in $\{N, A\}$, the formation rule is as follows. Let $\varphi_1(S)$ and $\varphi_2(S)$ be formulas in $FP(\alpha, \beta)$ with n free variables where, S is an n-ary predicate. Then $FP(\alpha, \beta)(\varphi_1(S), \varphi_2(S), S)(\vec{t})$ is a formula, where $\vec{t}$ is a sequence of n variables or constants. When $\beta = i$, we assume that $\varphi_1(S)$ and $\varphi_2(S)$ are inflationary. When $\alpha = N$, we further require that negation not be applied to a formula containing a fixpoint. The semantics of the formulas is defined inductively as described above.

The obvious containments among these logics are described by the following diagram:

$$\begin{array}{ccc} FP(A,i) & \subseteq & FP(A,n) \\ \cup\!\mid & & \cup\!\mid \\ FP(N,i) & \subseteq & FP(N,n) \\ \cup\!\mid & & \cup\!\mid \\ FP(D,i) & \subseteq & FP(D,n) \end{array}$$

Remark 2.3: One can prove for these various fixpoint logics standard results such as the closure under composition and complement (except for $\alpha = N$). The collapse of the nesting of fixpoints (in other words, one fixpoint operator per formula suffices) and the simultaneous induction (one carrier suffices) hold for all cases. These results were shown for $FP(D,i)$ in [GS86] and for $FP(D,n)$ in [AV91a]. The proof is essentially the same for the new logics. ∎

2.5 Fixpoint Logics and Standard Complexity

Let us now examine the complexity-theoretic aspects of the family of fixpoint logics. We already know that $FP(D,i)$ captures P and $FP(D,n)$ captures PSPACE. One would expect nondeterministic and alternating iterations to capture nondeterministic and alternating computations. This is indeed the case. Recall that capturing a complexity class C by a language L has two aspects. The first is that all problems expressible in L are in C. The second is that, on ordered structures, L expresses all problems in C. We next consider these aspects separately.

Lemma 2.4: *For X in $\{D, N, A\}$,*

1. *$FP(X, i) \subseteq$ XPTIME,*
2. *$FP(X, n) \subseteq$ XPSPACE.*

Proof: As an illustration we show that $FP(A, n) \subseteq$ APSPACE. We have to show that we can compute the alternating fixpoint of a pair (φ, ψ) of arbitrary first-order formulas in alternating polynomial space. Suppose that φ and ψ have k free variables. Since the number of k-ary tuples over the domain D is polynomial, for a fixed k, it suffices to show that we can check in alternating polynomial space whether a given tuple $(a_1, \ldots, a_k)$ is in the alternating fixpoint of (φ, ψ). The algorithm stores a k-ary relation r, which is initially empty. The algorithm then alternates between existential and universal states. In an existential state, the algorithm branches existentially, choosing between computing $\Phi(r)$ and computing $\Psi(r)$. In a universal state, the algorithm branches universally, computing both $\Phi(r)$ and $\Psi(r)$. The algorithm terminates when $r = \Phi(r) = \Psi(r)$, and it accepts if $(a_1, \ldots, a_k) \in r$ Note that computing $\Phi(r)$ and $\Psi(r)$ can be done in polynomial space. Thus, the whole computation can be done in polynomial space. ∎

To see that the fixpoint logics can express on ordered structures all problems in the corresponding complexity classes, we first show that, when the number of stages is polynomialy bounded, noninflationary computation can be simulated in an inflationary manner *on ordered structures*. The technique that we use essentially generalizes the "timestamping" technique of [AV91a].

Lemma 2.5: *Let X be in $\{D, N, A\}$ and f in $FP(X, n)$ be such that on each ordered structure, the number of stages of f is bounded by a polynomial in the size of the structure. Then there exists f' in $FP(X, i)$ equivalent to f on ordered structures.*

Proof: We prove the statement for $X = A$. (The other cases are simpler.) Let σ be an input type including a binary predicate $\leq$ and

$$f = FP(A, n)(\varphi_1, \varphi_2, S)(\vec{t})$$

Suppose that there exists an integer p such that for each ordered structure $\mathbf{D}$ over σ, the depths of the converging stage trees are bounded by $|\mathbf{D}|^p$. We construct an equivalent fixpoint formula in $FP(A, i)$.

Let $\mathbf{D}$ be a structure over σ, ordered by $\leq$. Since $\mathbf{D}$ is ordered, we can use p-tuples to denote integers in $[0, |\mathbf{D}|^p - 1]$. The carrier of the induction will be $\widehat{S}$

such that $arity(\widehat{S}) = arity(S) + p$. Instead of φ_1, φ_2 we use inflationary formulas ψ_1, ψ_2 defined as follows. For each i, ψ_i is:

$$\widehat{S}(\vec{u}, y_1, ..., y_p) \vee \exists y'_1, ..., y'_p(current(y'_1, ..., y'_p) \wedge next(y'_1, ..., y'_p, y_1, ..., y_p) \wedge \varphi'_i)$$

where

- $y_1, ..., y_p, y'_1, ..., y'_p$ are new, distinct variables;
- $\vec{u}$ is the vector of free variables of φ_i;
- φ'_i is obtained from φ_i by replacing each atom $S(\vec{v})$ by $\widehat{S}(\vec{v}, y'_1, ..., y'_p)$;
- *current* is a first order formula stating that the tuple $\langle y'_1, ..., y'_p \rangle$ is the largest p-tuple (in the ordering over p-tuples induced by $\leq$) belonging to the projection of $\widehat{S}$ on its last p coordinates; and
- *next* is a first order formula stating that the tuple $\langle y_1, ..., y_p \rangle$ follows $\langle y'_1, ..., y'_p \rangle$ is the ordering over p-tuples induced by $\leq$.

Intuitively, this is a "timestamping" technique, where the value of S at each stage of the fixpoint is identified by a timestamp of the stage (a p tuple).

Let $\widehat{f}$ be the $FP(A, i)$ formula

$$\widehat{f} = \exists z_1 \dots z_p(FP(A, i)(\varphi'_1, \varphi'_2, \widehat{S})(\vec{t}, z_1, ..., z_p) \wedge current(z_1, \dots, z_p)),$$

where $z_1, \dots, z_p$ are new, distinct variables.

Clearly, $\widehat{f}$ is equivalent to f. ∎

Lemma 2.6: *For X in DNA, on ordered structures,*

1. XPTIME $\subseteq FP(X, i)$,
2. XPSPACE $\subseteq FP(X, n)$.

Proof: First consider (1). The result is known for $X = D$ [Imm86, Var82]. Let $X = A$. Let σ be an input type including a binary predicate $\leq$. Let f be a boolean function on structures over σ ordered by $\leq$, such that f is in APTIME. Then there exists an alternating Turing machine M in APTIME, which, given an encoding of an ordered structure $\mathbf{D}$ over σ, accepts iff $f(\mathbf{D})$. We first show how to simulate M in $FP(A, n)$ (then use Lemma 2.5).

The simulation is similar to that used in [Imm86, Var82] to show that $FP(D, n)$ ($FP(D, i)$) can simulate polynomial-space (-time) Turing machines. The configurations and moves of M are represented similarly. The difference lies in the simulation of the control. We elaborate on this aspect. We can assume w.l.o.g. that a move of M always takes a universal state to an existential one, or conversely; and that for each state q, and tape symbol u, there are at most two possible moves, say Moves 1 and 2. As in [Imm86, Var82], we can construct a 1st-order formula φ_1 defining the transitions corresponding to Moves 1 and another 1st-order formula φ_2 for Moves

2. It now suffices to alternatingly iterate φ_1 and φ_2 a polynomial number of times. This can be done in $FP(A,n)$. However, we wish to simulate M in an inflationary manner. To see that this can be done, observe the convergent stage trees of φ_1, φ_2. Since M is in $APTIME$, the depth of these trees is bounded by a polynomial in the size of the input. Thus, by Lemma 2.5, f can be expressed in $FP(A,i)$. This demonstrates that APTIME $\subseteq FP(A,i)$. For X = N, it suffices to assume that M has no alternative when in a universal state. For (2), the proof is similar. ∎

Theorem 2.7:

1. *$FP(N,i)$ captures* NP.
2. *(a) $FP(A,i)$, and (b) $FP(N,n)$ capture* PSPACE.
3. *$FP(A,n)$ captures* EXPTIME.

Proof: The result is now immediate using PSPACE = NPSPACE = APTIME and EXPTIME = APSPACE. Indeed,

1. By Lemmas 2.4, 2.6 (1) ($X = N$), $FP(N,i)$ captures NPTIME, i.e. NP.

2(a). By Lemmas 2.4, 2.6 (1) ($X = A$), $FP(A,i)$ captures APTIME = PSPACE.

2(b). By Lemmas 2.4, 2.6 (2) ($X = N$), $FP(N,n)$ captures NPSPACE = PSPACE.

3. By Lemmas 2.4, 2.6 (2) ($X = A$), $FP(A,n)$ captures APSPACE = EXPTIME.

∎

Theorem 2.7 says that the connection between fixpoint logics and the classes P and PSPACE discovered in [Imm86, Var82] is just one half of the picture; the complete picture is a broad connection between fixpoint logics and complexity classes between P and EXPTIME. Theorem 2.7 is strong enough to yield a separation between $FP(D,i)$ and $FP(A,n)$; this follows from the fact that P is strictly contained in EXPTIME.

The above broad connection between fixpoint logic and computational complexity still suffers from the order mismatch. Thus, we cannot translate containments between complexity classes into containments between fixpoint logics. For example, from the fact that $FP(D,n)$, $FP(N,n)$, and $FP(N,i)$ all capture PSPACE we can only infer that they have the same expressive power over *ordered* structures. This does not tell us anything about the relationship between these logics in general. To get around this difficulty, we use a theory of *relational* complexity, which mediates between standard complexity and fixpoint logics.

3 Relational Machines

3.1 The Model

A relational machine is a Turing machine augmented with a *relational store*. The relational store consists of a set of relations of certain arities. Some of these relations are designated as input relations and some are designated as output relations. The *type* of a relational machine is the sequence of arities of its relations. The *arity* of the machine is the maximal arity of the relations in its store. The tape of the machine is a work tape and is initially empty. Transitions depend on the current state, the content of the current tape cell, and a test of emptiness of a relation of the relational store. Transitions involve the following actions: move right or left on the tape, overwrite the current tape cell with some tape symbol, and replace a relation of the store with the result of an algebraic operation on relations of the store. The algebraic operations are: boolean operations $(\cap, \cup, -)$, $\pi_{i_1 \ldots i_m}$ (project the tuples in a relation on the coordinates $i_1 \ldots i_m$ in the specified order), $\times$ (cross-product of two relations), and $\sigma_{i=j}$, $\sigma_{i=a}$ (resp., select from relation tuples whose i-th coordinate coincides with j-th one, or select those tuples whose i-th coordinate coincides with domain element a), see [Ull89]. For example, the machine can have instructions such as:

> If the machine is in state s_3, the head is reading the symbol 1, and relation R_1 is empty, then change the state to s_4, replace the symbol 1 by 0, move the head to the right, and replace R_2 by $R_2 \cap R_3$.

A *configuration* of a relational machine consists of the state, tape content and head position, and the content of the relational store. For a configuration l, we denote by $l(store)$ the content of the store in configuration l. The formal definitions of relational machine, configuration, and move are straightforward but lengthy, and are omitted.

Relational machines model the embedding of a relational database query language [Cod72] in a general purpose programming language[6]. They are essentially the *loosely coupled generic machines* (GM^{loose}), introduced in [AV95]. A negligible difference is that unlike relational machines, GM^{loose} can apply general 1st-order transformations to the relational store. However such transformations can be simulated by the above algebraic operations in a constant number of steps, see [Ull89].

Unlike Turing machines, which get their input spread on the input tape, relational machines get their input in a more natural way. For example, if the input is a graph, then the input to the machine is the set of nodes and the set of edges. (The type of the machine is $\langle 1, 2 \rangle$ and the arity of the machine is 2.) Thus, the order issue does not come up between relational machines and fixpoint logics. We will come back later to the connection between relational machines and fixpoint logics.

[6] Relational machines are closely related to the 2nd-order pointer machines of [Lei89a]. Pointer machine manipulate their store by means of tagging and untagging operations, while relational machines manipulate their store by means of relational operations.

Relational machines give rise to three types of computational devices. First, we can think of a relational machine as an acceptor of a *relational language*, i.e., a set of structures. In that case, we assume that there is a single 0-ary output relation; the machine accepts if the output relation consists of the 0-ary tuple and rejects if the output relation is empty. We can also think of a relational machine as a *relational function*, computing an output structure for a given input structure. Finally, we can think of relational machine as a *mixed function*, i.e., a function from structures to strings, where the output is written on the machine's tape.

In analogy with nondeterministic and alternating Turing machines, we can define nondeterministic and alternating relational machines. We will use nondeterministic and alternating relational machines viewed as acceptors or relational functions (but not as mixed functions, for which deterministic machines only will be used). For acceptors, the definitions are completely analogous to those for Turing machines. For machines viewed as relational functions, they are slightly more complicated. The output of a nondeterministic relational machine is defined as follows. A tuple is in an output relation R if it belongs to R at the end of *some* computation of the machine on its input. The intuition is that the result of a nondeterministic computation taken to be the (relation-wise) union of the results of all possible computations. The output of an alternating relational machine is defined as follows. A finite computation tree is a tree whose nodes are configurations of the machine, such that:

1. the root is the start configuration;
2. if the state of a configuration in the tree is existential, it has a single child which is one of the possible next configurations;
3. if the state of a configuration in the tree is universal, its children are all possible next configurations;
4. all leaves are halting configurations.

Let $\mathcal{T}_{\mathcal{M}}$ be the set of finite computation trees of an alternating relational machine M on input $\mathbf{D}$. The output of M on $\mathbf{D}$ is:

$$\bigcup_{T\in\mathcal{T}_{\mathcal{M}}} \bigcap \{l(store) \mid l \text{ is a leaf of } T\}.$$

(Union and intersection are relation-wise.) Here the intuition is that we have to take a union over all existential choices, and for each term in the union we have to take an intersection of all universal choices.

3.2 Discerning Power and Relational Complexity

We would like to define complexity classes using relational machines. As noted in [AV95], using as measure of the input its size is not appropriate for relational machines. This is because, unlike Turing machines, relational machines have limited access to their input, and in particular cannot generally compute its size. Indeed, relational machines are less "discerning" than Turing machines. As in [AV95], we say that a relational machine M cannot *discern* between k-tuples $\mathbf{u}$ and $\mathbf{v}$ over an input $\mathbf{D}$, if for each k-ary relation R in its store and throughout the computation of M over $\mathbf{D}$, we have that $\mathbf{u}$ is in R precisely when $\mathbf{v}$ is in R. For example, if there is an automorphism on $\mathbf{D}$ that maps $\mathbf{u}$ to $\mathbf{v}$, then $\mathbf{u}$ and $\mathbf{v}$ cannot be discerned by M over $\mathbf{D}$. The discerning power of relational machines is best understood by viewing them as an effective fragment of the logic $L^{\omega}_{\infty\omega}$. This is an infinitary logic with a finite number of variables studied recently in [KV92, KV95]. The connection between relational machines and $L^{\omega}_{\infty\omega}$ is developed in [AVV95]; it suffices here to say that it yields a characterization of the discerning power of relational machines in terms of certain infinite 2-player pebble games.

The *k-pebble game* between the Spoiler and the Duplicator on the l-tuples ($l \leq k$) $\mathbf{u}$ and $\mathbf{v}$ of a structure $\mathbf{D}$ has the following rules. Each of the players has k pebbles, say $p_1, \ldots, p_k$ and $q_1, \ldots, q_k$, respectively. The game starts with the Spoiler choosing one of the tuples $\mathbf{u}$ or $\mathbf{v}$, placing $p_1, \ldots, p_l$ on the elements of the chosen tuple (i.e., on $u_1, \ldots, u_l$ or $v_1, \ldots, v_l$), and placing the pebbles $p_{l+1}, \ldots, p_k$ on some elements of $\mathbf{D}$. The Duplicator responds by placing the pebbles $q_1, \ldots, q_l$ on the elements of the other tuple, and placing the pebbles $q_{l+1}, \ldots, q_k$ on some elements of $\mathbf{D}$. In each following round of the game, the Spoiler moves some pebble p_i (or q_i) to another element of $\mathbf{D}$, and the Duplicator responds by moving the corresponding pebble q_i (or p_i).

Let a_i (resp., b_i), $1 \leq i \leq k$, be the elements of $\mathbf{D}$ under the pebbles p_i (resp., q_i), $1 \leq i \leq k$, at the end of a round of the game. If the mapping h with $h(a_i) = b_i$, $1 \leq i \leq k$, is not an isomorphism between the substructures of $\mathbf{D}$ with universes $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$ respectively, then the Spoiler wins the game. The Duplicator wins the game if he can continue playing "forever", i.e. if the Spoiler can never win the game.

If the Duplicator wins the k-pebble game on the tuple $\mathbf{u}$ and $\mathbf{v}$ of a structure $\mathbf{D}$, then we say that $\mathbf{u}$ and $\mathbf{v}$ are k-equivalent, denoted $\mathbf{u} \equiv_k \mathbf{v}$, over $\mathbf{D}$. The relation $\equiv_k$ characterizes the discerning power of k-ary relational machines.

Proposition 3.1: *The tuples $\mathbf{u}$ and $\mathbf{v}$ are k-equivalent over a structure $\mathbf{D}$ if and only if no k-ary relational machine M can discern between $\mathbf{u}$ and $\mathbf{v}$ over $\mathbf{D}$.*

Proof: By [Ba77, Imm82], $\mathbf{u}$ and $\mathbf{v}$ are k-equivalent with respect to $\mathbf{D}$ iff they are $L^k_{\infty\omega}$-equivalent, i.e. for each $\varphi \in L^k_{\infty\omega}$ with k free variables, $\mathbf{D} \models \varphi(\mathbf{u})$ iff $\mathbf{D} \models \varphi(\mathbf{v})$. By [KV92], over finite structures, $L^k_{\infty\omega}$-equivalence is the same as L^k-equivalence, where L^k denotes the set of first-order formulas with k variables. Suppose $\mathbf{u}$ and $\mathbf{v}$ are not L^k-equivalent with respect to $\mathbf{D}$. Then there exists an L^k formula φ such that $\mathbf{D} \models \varphi(\mathbf{u})$ and $\mathbf{D} \not\models \varphi(\mathbf{v})$. It is easily seen that there exists a k-ary relational machine M_φ that evaluates φ. Thus, $\mathbf{u}$ and $\mathbf{v}$ are distinguished by M_φ. Conversely, suppose $\mathbf{u}$ and $\mathbf{v}$ are distinguished by some k-ary relational machine M over input $\mathbf{D}$. An easy induction on the number of steps in a computation of M shows that the content of each relation in the store at any time in the computation is defined by a formula in L^k. Thus, $\mathbf{u}$ and $\mathbf{v}$ are distinguished by a formula in L^k. ∎

Since k-ary relational machines cannot discern among k-equivalent tuples, they cannot measure the size of their input. Instead, computations of k-ary relational machines are determined by the k-equivalence classes. Intuitively, relational machines are complete on the equivalence classes of tuples they can distinguish. This is best understood by looking at the extremes. As shown in [AV95], relational machines are complete on ordered inputs (where all distinct tuples can be distinguished from each other), but they collapse to first-order logic on unordered sets (where the number of k-equivalence classes is bounded by a constant independent on the size of the input). Thus, it seems natural to measure the input size with respect to $\equiv_k$. Let the *k-size* of a structure $\mathbf{D}$, denoted $size_k(\mathbf{D})$, be the number of $\equiv_k$-classes of k-tuples over $\mathbf{D}$. In the spirit[7] of [AV95], we propose to measure the time or space complexity of k-ary relational machines as a function of the k-size of their input. This measure, however, can reasonably serve as a basis for measuring complexity only if it can be calculated by relational machines. The techniques of [AV95] can now be used to show that relational machines *can* measure k-size.

We will use the following result from[8] [AV95].

Lemma 3.2: [AV95] *For each input type σ and $k > 0$ there exists an IFP formula φ over σ, with $2k$ free variables $x_1, ..., x_k, y_1, ..., y_k$ such that for each input $\mathbf{D}$ of type σ, φ defines a partial order $\prec$ over k-tuples, and $x_1, ..., x_k$ is incomparable (via $\prec$) with $y_1, ..., y_k$ iff $x_1, ..., x_k \equiv_k y_1, ..., y_k$.*

In other words, there exists an IFP query computing an order on the equivalence classes of $\equiv_k$.

Proposition 3.3: *For each $k > 0$ and input type σ, there is a relational machine that outputs on its tape, for an input structure $\mathbf{D}$ of type σ, a string of length $size_k(\mathbf{D})$ in time polynomial in $size_k(\mathbf{D})$.*

Proof: By the above lemma, there exists an IFP formula φ over σ defining, for each input $\mathbf{D}$ of type σ, the equivalence classes of $\equiv_k$, in some order. By considering the proof of the lemma, one can see that φ has a number of stages which is polynomial in $size_k(\mathbf{D})$. Since each evaluation of a first-order formula is simulated by a relational machine in a constant number of steps, φ can be evaluated by such a machine in time which is still polynomial in $size_k(\mathbf{D})$. Suppose the result is placed in some $2k$-ary relation $R^k_\prec$. Finally, to produce a string of length $size_k(\mathbf{D})$ on the tape, it is sufficient to step through the k-equivalence classes in the order given by $R^k_\prec$ and write a symbol on the tape for each equivalence class. Clearly, the entire computation takes a number of steps polynomial in $size_k(\mathbf{D})$. ∎

From now on we measure the complexity of k-ary relational machines in terms of the k-size of their input. We can now define relational complexity classes in

[7] In [AV95], the measure of the input is based on equivalence classes which are more tightly connected to the particular machine. In contrast, the k-size used here is uniform for all relational machines of arity k.

[8] An alternate proof of the result, which makes an explicit connection with the pebble games, was recently given in [DLW95].

terms of deterministic (resp., nondeterministic, alternating) relational time or relational space, e.g., $\mathrm{DTIME}_r(f(n))$, $\mathrm{NSPACE}_r(f(n))$, and so on. Thus, we can define the relational complexity class P_r (relational polynomial time), $\mathrm{NPSPACE}_r$, etc. Proposition 3.3 guarantees the effective enumerability of these classes. In contrast, it is not clear how to get effective enumerability if we define relational complexity in terms of the actual size of the input. Note that it is not obvious whether known relationships between deterministic, nondeterministic, and alternating complexity classes, such as PSPACE=NPSPACE=APTIME, hold also for relational complexity classes, since these relationships are typically the result of simulations that use order (although we show below that they do hold).

To simplify references to complexity classes, we use the notation *Class*(*Resource*, *Control*, *Bound*) and $Class_r$(*Resource*, *Control*, *Bound*), where *Resource* can be time or space, *Control* can be deterministic, nondeterministic, or alternating, and *Bound* is the bounding function or family of functions. Thus, *Class*(*time*, *nondeterministic*, *poly*) is NP, and $Class_r(space, deterministic, poly)$ is PSPACE_r. We will always assume that our bounds are at least linear. Typically, *Bound* will be a *polynomialy closed* set of functions, i.e., a set of functions that contains the linear functions and contains $p(f(n))$ whenever it contains $f(n)$, for all polynomials $p(x)$. Furthermore, we assume that all bounds are fully time and space constructible. Note that most commonly used functions, including the logarithmic, linear, polynomial and exponential bounds, are fully time and space constructible (see [HU79]). Also, the fully time and space constructible functions are polynomialy closed.

4 Relational vs. Standard Complexity

What is the relationship between relational and standard complexity? It is easy to see that the k-size of a structure $\mathbf{D}$ is always bounded above by a polynomial ($O(n^k)$) in the size of $\mathbf{D}$, for all $k > 0$. Furthermore, relational machines are strictly weaker than Turing machines because they cannot check that the cardinality of their input is even. The latter follows from the fact that the logic $L^\omega_{\infty\omega}$ has a 0-1 law, so it cannot express the evenness property [KV92]. We thus obtain the following:

Proposition 4.1: *Let Φ be a polynomialy closed set of functions. Then*

$$Class_r(resource, control, \Phi) \subset Class(resource, control, \Phi),$$

for any resource and control.

Proof: Each standard complexity class $Class(resource, control, \Phi)$, where Φ is polynomialy closed, contains the evenness property, which is not in the relational complexity class $Class_r(resource, control, \Phi)$. ∎

While relational machines are in some sense weaker than standard machines, the weakness is only due to the lack of order. This weakness disappears in the presence of order. Let w be string, say over the alphabet $\{0, 1\}$, of length n. We can encode w by a structure $rel(w)$ consisting of a total order on the elements

$\{0, \ldots, n-1\}$ and a unary predicate on these elements, giving the positions of 1's in the string. Note than the k-size of $rel(w)$ is bounded by n^k. For a language L, let $rel(L) = \{rel(w) \mid w \in L\}$. Since $rel(w)$ is ordered for each w, a relational machine can simulate a machine for L.

Proposition 4.2: *For each resource, control, polynomialy closed bound and a language L in $Class(resource, control, bound)$, the relational language $rel(L)$ is in $Class_r(resource, control, bound)$.*

Proof: Let σ be the type of $rel(L)$ (a binary and a unary relation). Given an input $\mathbf{D}$ over σ, a k-ary deterministic relational machine ($k > 1$) can check in constant time that $\mathbf{D}$ is of the form $rel(w)$ for some w. If $\mathbf{D}$ is not of the form $rel(w)$, the machine rejects. Otherwise, the machine can write w on the tape, in time polynomial in $size_k(\mathbf{D})$ (which equals $|w|^k$). Lastly the relational machine runs the Turing machine recognizing L, which takes time/space bounded by $f(|w|)$, so by $f(size_k(\mathbf{D}))$, for some function $f \in$ *bound*. Thus a relational machine can recognize $rel(L)$ with the specified control, using *resource* $c + f(size_k(\mathbf{D}))$ for some constant c. It follows that $rel(L)$ is in $Class_r(resource, control, bound)$. ∎

Combining Propositions 4.1 and 4.2, we get:

Corollary 4.3: *Let Φ_1 and Φ_2 be polynomialy closed sets of functions and let $resource_1$, $resource_2$, $control_1$, $control_2$ be resources and controls, respectively. Then we have that*

$$Class(resource_1, control_1, \Phi_1) \subseteq Class(resource_2, control_2, \Phi_2)$$

if

$$Class_r(resource_1, control_1, \Phi_1) \subseteq Class_r(resource_2, control_2, \Phi_2).$$

Proof: Suppose

$$Class_r(resource_1, control_1, \Phi_1) \subseteq Class_r(resource_2, control_2, \Phi_2).$$

Let L be in $Class(resource_1, control_1, \Phi_1)$. By Proposition 4.2, $rel(L)$ is in $Class_r(resource_1, control_1, \Phi_1)$, so in $Class_r(resource_2, control_2, \Phi_2)$. By Proposition 4.1,

$$Class_r(resource_2, control_2, \Phi_2) \subset Class(resource_2, control_2, \Phi_2).$$

It follows that there exists a Turing machine $M_{rel(L)}$ which accepts a standard encoding of $rel(w)$ with control $control_2$ and with the resource $resource_2$ bounded by some function in Φ, iff $w \in L$. Since a standard encoding of $rel(w)$ can obviously be obtained from w in time/space polynomial in $|w|$, it follows that there exists a Turing machine M_L which accepts L, with the same control and resource bound as $M_{rel(L)}$. Thus, $Class(resource_2, control_2, \Phi_2)$ contains L. ∎

It follows from Corollary 4.3 that separation results among standard complexity classes translate into separation results among relational complexity classes. For example, it follows that P_r is strictly contained in $\mathrm{EXPTIME}_r$.

To further understand the relationship between standard and relational complexity, we introduce the notion of *reduction* from a relational language to a standard language. We say that a relational language L of type σ is *relationally reducible in polynomial time* to a standard language L' if there exists a deterministic polynomial-time relational machine T acting as a mixed function from σ-structures to strings, such that for each structure $\mathbf{D}$ of type σ we have that $\mathbf{D} \in L$ if and only if $T(\mathbf{D}) \in L'$.

One of the technical results[9] of this paper is that every relational language can be reduced to a standard language. We will need some notation and a definition.

Let M be a k-ary relational machine. For each $m \leq k$, let $\equiv_k^m$ denote k-equivalence of m-tuples. From Proposition 3.1 it easily follows that at any time in the computation of M on some input, the content of an m-ary relation of the store is a union of classes of $\equiv_k^m$. Consider the action of the algebraic operations on the store. We would like to summarize their effect in terms of the equivalence classes alone. The boolean operations are no problem. Consider the relational algebraic operations $\pi, \times, \sigma$. These operations distribute over union. Therefore their effects are determined by their effects on individual equivalence classes of appropriate arity. These are described by "action tables", defined next. They are the analog to algebraic operations of the action tables for conjunctive queries, defined in [AV95].

Definition 4.4: Let o be an operation among $\{\pi, \times, \sigma\}$, with arguments $R_1, ..., R_n$ of arities $i_1, ..., i_n$ ($n \leq 2$), and result of arity i_{n+1}, $i_j \leq k$. The *action table* of o on a structure $\mathbf{D}$ is a relation R_o of arity $i_1 + ... + i_{n+1}$ such that

$$R_o = \bigcup\{\delta_1 \times \ldots \times \delta_n \times \delta_{n+1} \mid \delta_j \text{ is a class of } \equiv_k^{i_j}, \delta_{n+1} \subseteq o(\delta_1, ..., \delta_n)\}$$

∎

We are now ready to show that each relational language is polynomialy reducible to a standard language.

Theorem 4.5: *Let Φ be a polynomialy closed set of fully time/space constructible functions and let L be a relational language in $Class_r(\rho, \kappa, \Phi)$ for some resource ρ, control κ, and bound Φ. Then L is relationally reducible in polynomial time to a language L' in $Class(\rho, \kappa, \Phi)$.*

Proof: Suppose L is accepted by a relational machine M of arity k in $Class_r(\rho, \kappa, \Phi)$. The reduction from a relational input $\mathbf{D}$ of type σ to a word proceeds as follows. Let $\sigma_\prec$ be the type consisting of the relations $R^m_\prec$ ($m \leq k$) and R_o for each relational operation o among $\{\pi, \times, \sigma\}$ used by M. As in the normal form of [AV95], we first map $\mathbf{D}$ to a structure $\mathbf{D}_\prec$ over $\sigma_\prec$, which can be viewed as an ordered structure. First, the classes of $\equiv_k^m$, $m \leq k$, are computed in some order, and stored in relations $R^m_\prec$. From Lemma 3.2 it follows that the $R^m_\prec$ can be computed by an IFP formula in a number of stages polynomial in $size_k(\mathbf{D})$, so by a deterministic relational machine in time polynomial in $size_k(\mathbf{D})$. Next, the action tables of

[9] This result is implicit in [AV95, DLW95].

the relational algebraic operations used by the relational machine are produced. The construction involves stepping through all sequences $\delta_1, \ldots \delta_{n+1}$ of equivalence classes in the lexicographic order induced by the $R^m_\prec$; this takes a number of steps polynomial in $size_k(\mathbf{D})$.

The structure $\mathbf{D}_\prec$ can be viewed as an ordered structure as follows. Let $abs(\sigma_\prec)$ be the type obtained from $\sigma_\prec$ by replacing each relation $R^m_\prec$ by a binary relation with the same name, and R_o (where o is an operation with n arguments, $n \leq 2$) by a relation of arity $n+1$. Let $abs(\mathbf{D}_\prec)$ be the ordered structure over $abs(\sigma_\prec)$ obtained from $\mathbf{D}_\prec$ as follows: each tuple $\langle x_1, \ldots, x_m, y_1, \ldots, y_m\rangle$ in $R^m_\prec$ is replaced by $\langle i, j\rangle$, where i, j are the ranks of $\langle x_1, \ldots, x_m\rangle$ and $\langle y_1, \ldots, y_m\rangle$ in the partial order on m-tuples given by $R^m_\prec$; and each tuple $\langle \vec{x}_1, ..., \vec{x}_{n+1}\rangle \in R_o$ (where o has n arguments of arities $i_1, \ldots i_n$ and result of arity i_{n+1}, and each $\vec{x}_j$ has arity i_j) is replaced by the $n+1$-tuple $\langle k_1, ..., k_{n+1}\rangle$, where k_j is the rank of $\vec{x}_j$ in the partial order $R^{i_j}_\prec$. Note that $abs(\mathbf{D}_\prec)$ is an ordered structure and the number of elements in $abs(\mathbf{D}_\prec)$ is $size_k(\mathbf{D})$.

Putting together the above, it is clear that a relational machine T can produce a standard encoding of $abs(\mathbf{D}_\prec)$ on its tape in a number of steps polynomial in $size_k(\mathbf{D})$. Let $enc(abs(\mathbf{D}_\prec))$ denote that standard encoding, which uses some alphabet Σ.

Let $f \in \Phi$ be such that M is in $Class_r(\rho, \kappa, \{f\})$. By the same technique used in showing the normal form in [AV95], it can be seen that there exists a Turing machine M', with *control* κ, that accepts $enc(abs(\mathbf{D}_\prec))$ iff M accepts $\mathbf{D}$ with *resource* ρ bounded by a polynomial in $f(size_k(\mathbf{D}))$. Basically, M' simulates the moves of M one by one, using the information provided in $enc(abs(\mathbf{D}_\prec))$. The content of a relation R of arity m in the store is represented by the integers corresponding to the classes of $\equiv^m_k$ currently in R; the algebraic operations on the store are simulated using the action tables. Thus, each step of M is simulated by M' in a number of steps polynomial in $|w|$, so in $size_k(\mathbf{D})$.

We now define the language L'.

Let M'' be a Turing machine with control κ that on input $w \in \Sigma^*$ does the following:

(1) construct a string of length $f(|w|)$;

(2) simulate M' on w as long as M' uses only $f(|w|)$ *resource*, or until it exhausts the resource; in the latter case, reject; otherwise, accept or reject as M' does.

Note that (1) can be done without exceeding the bound on *resource* ρ, because f is fully ρ-constructible.

Let L' consist of all words over Σ that are accepted by M''. Then $L' \in Class(\rho, \kappa, \Phi)$ and for each $\mathbf{D}$ over σ, $\mathbf{D}$ is accepted by M iff $T(\mathbf{D}) \in L'$. Thus, L is relationally reducible in polynomial time to L'. ∎

According to the proof of the theorem, the reduction from L to L' depends only on the type σ of L and the arity k of its relational acceptor M. We denote the relational machine that does the reduction for type σ and arity k by $T_{\sigma,k}$, and call it the *relational reduction* machine of input type σ and arity k.

Combining Propositions 4.1 and 4.2, Corollary 4.3, and Theorem 4.5, we get that the relationships among relational complexity classes are analogous to the relationships among standard complexity classes.

Corollary 4.6: *Let Φ_1 and Φ_2 be polynomialy closed sets of fully time/space constructible functions and let $resource_1$, $resource_2$, $control_1$, $control_2$ be kinds of resources and controls, respectively. Then*

$$Class(resource_1, control_1, \Phi_1) \subseteq Class(resource_2, control_2, \Phi_2)$$

if and only if

$$Class_r(resource_1, control_1, \Phi_1) \subseteq Class_r(resource_2, control_2, \Phi_2).$$

Proof: The if part is Corollary 4.3. For the only-if part, suppose

$$Class(resource_1, control_1, \Phi_1) \subseteq Class(resource_2, control_2, \Phi_2)$$

and let L be a relational language of type σ in $Class_r(resource_1, control_1, \Phi_1)$, accepted by some relational machine of arity k.

Consider the relational reduction machine $T_{\sigma,k}$. By Theorem 4.5, there exists a language L' in $Class(resource_1, control_1, \Phi_1)$ such that for each $\mathbf{D}$ of type σ, $\mathbf{D} \in L$ iff $T_{\sigma,k}(\mathbf{D}) \in L'$. Since L' is in $Class(resource_1, control_1, \Phi_1)$, it is in $Class(resource_2, control_2, \Phi_2)$. Let M' be a relational machine that, on input $\mathbf{D}$ computes $T_{\sigma,k}(\mathbf{D})$ (in time polynomial in $size_k(\mathbf{D})$), then runs on $T_{\sigma,k}(\mathbf{D})$ the Turing machine that accepts L' with control $control_2$ using resource $resource_2$ bounded by some function in Φ. Then M' accepts L, uses control $control_2$ and resource $resource_2$ bounded by some function in Φ. Thus L is in $Class_r(resource_2, control_2, \Phi_2)$. ∎

In particular, it follows from Corollary 4.6, that the known relationships between deterministic, nondeterministic, and alternating complexity classes, such as PSPACE=NPSPACE=APTIME, *do* hold for relational complexity classes, i.e. PSPACE_r=NPSPACE_r=APTIME_r. Also, the open questions about standard complexity classes translate to questions about relational complexity classes, e.g., P=NP if and only if P_r=NP_r.

Further insight into the relationship between standard and relational complexity can be obtained from a closer look at the relationship between size and k-size. Since k-size can be much smaller than size, relational complexity can be much higher than standard complexity. As a result, relational complexity is in some sense orthogonal to standard complexity.

To understand this better, let us re-examine the translation between strings and structures. On one hand, the mapping rel maps strings to structures. It is easy to see that if w is a string, then the k-size of $rel(w)$ is polynomialy related to the size of w. On the other hand, according to Theorem 4.5, we have mappings from structures to strings. Let $T_{\sigma,k}$ be a relational reduction machine mapping structures of the type σ of $rel(w)$ to strings. It can be checked that for a given string w, the size of $T_{\sigma,k}(rel(w))$ is polynomialy related to the size of w. It turns out, however, that using a result of Lindell about the k-size of trees [Lin91], we can encode strings by structures in a way that blows up the size without blowing up the k-size.

Proposition 4.7: *For any function $f : N \rightarrow N$ there is a mapping rel_f from strings to structures such that:*

1. *if w is a string of length n, then $rel_f(w)$ is of size $(f(n)^n - 1)/(f(n) - 1)$, and*

2. *the k-size of $rel_f(w)$ is polynomial in n.*

Proof: Let $w = x_1 \ldots x_n$ be a string over $\{0, 1\}$ of length n. Let $rel_f(w)$ be a structure of type $\langle 2, 1\rangle$, where the binary relation is a complete tree of depth n and fan-out $f(n)$, and the unary relation contains all nodes in the tree at level i such that $x_i = 1$.

The size of $rel_f(w)$ is then $(f(n)^n - 1)/(f(n) - 1)$. Consider the k-size of $rel_f(w)$. Each k-tuple $\langle a_1, \ldots, a_k\rangle$ of elements in $rel_f(w)$ is identified up to automorphism by the sequence of levels in the tree of the least upper bound of the pairs a_i, a_j, $1 \leq i, j \leq k$ (see [Lin91]). Thus, the number of automorphism classes of k-tuples is bounded by n^{k^2}. It follows that the k-size of $rel_f(w)$ is polynomial in n. ∎

For example, by taking $f(n) = 2$, we can blow up the size exponentially. We denote this encoding by rel_2.

Proposition 4.7 implies that we can find languages of different relational complexity but the same standard complexity. In particular, we get the following consequences:[10]

Corollary 4.8:

1. $\mathrm{P}_r = \mathrm{NP}_r \cap \mathrm{P}$ *iff* P=NP.

2. $\mathrm{P}_r = \mathrm{PSPACE}_r \cap \mathrm{P}$ *iff* P=PSPACE.

3. $\mathrm{P}_r \subset \mathrm{EXPTIME}_r \cap \mathrm{P}$.

Proof: Consider (1). Suppose P = NP. By Corollary 4.3, $\mathrm{P}_r = \mathrm{NP}_r$, so $\mathrm{P}_r \cap \mathrm{P} = \mathrm{NP}_r \cap \mathrm{P}$. Since $\mathrm{P}_r \subset \mathrm{P}$, it follows that $\mathrm{P}_r = \mathrm{NP}_r \cap \mathrm{P}$. Conversely, suppose $\mathrm{P}_r = \mathrm{NP}_r \cap \mathrm{P}$. Let $L \in \mathrm{NP}$. First, note that $rel_2(L)$ is in NP_r. Indeed, consider the relational machine M which, given a structure $\mathbf{D}$ of the type $\langle 2, 1\rangle$ does the following:

1. check if $\mathbf{D}$ is of the form $rel_2(w)$ for some w; if not, reject;

2. output w on the tape;

[10]The equivalence of $\mathrm{P}_r = \mathrm{PSPACE}_r \cap \mathrm{P}$ and P=PSPACE was shown in [AV95].

3. run the nondeterministic Turing machine M' recognizing L in NP; accept or reject as M' does.

Clearly, (1) can be done by an IFP query, so by a relational machine in P_r. The translation of $rel_2(w)$ to w is also in P_r. Lastly, (3) is in NP_r. It follows that $rel_2(L)$ is in NP_r. Next, recall that for a word w, $rel_2(w)$ has size exponential in the length of w. Thus, $rel_2(L)$ is in P. Then $rel_2(L)$ is in $\mathrm{NP}_r \cap \mathrm{P}$, so in P_r. It was shown in [AV95] that $\mathrm{P}_r = IFP$. Finally, a result of Lindell [Lin91] states that, if $rel_2(L)$ is definable in IFP, then L is in P. It follows that NP = P, and (1) is proven. The proof of (2) is similar and was done in [AV95]. Consider (3). By the same technique as above, it can be shown that $\mathrm{P}_r = \mathrm{EXPTIME}_r \cap \mathrm{P}$ iff P = EXPTIME. However, the time hierarchy says that P $\subset$ EXPTIME. It follows that $\mathrm{P}_r \subset \mathrm{EXPTIME}_r \cap \mathrm{P}$. ∎

Corollary 4.8 tells us that questions about separation among standard complexity classes can be expressed as questions about relational complexity classes of the *same* standard complexity. The first clause says that P and NP separate if and only iff P_r and NP_r separate *inside* P. Similarly, the second clause says that P and PSPACE separate if and only if P_r and PSPACE_r separate inside P. The third clause in the corollary says that $\mathrm{EXPTIME}_r$ is stronger than P_r not only because $\mathrm{EXPTIME}_r$ contains problems that cannot be solved in polynomial time, but also because it contains problems that cannot be solved in relational polynomial time even though they can be solved in standard polynomial time. See [DLW95] for related results.

5 Relational Machines and Fixpoint Logics

Fixpoint logics involve iterations of 1st-order formulas. Since relational algebra has the expressive power of 1st-order logic, it is clear that relational machines with the appropriate control can simulate all fixpoint logics. For example, $FP(D,i) \subseteq \mathrm{P}_r$ and $FP(A,n) \subseteq \mathrm{APSPACE}_r$.

To find the precise relationship between fixpoint logics and relational machines consider again Theorem 4.5. According to that theorem, every relational language of a certain relational complexity can be reduced in relational polynomial time to a standard language of the analogous standard complexity. For example, a relational language in NP_r can be reduced in relational polynomial time to a language in NP. According to Theorem 2.7, the fixpoint logic $FP(N,i)$ captures NP. Thus, the only gap now between NP_r and $FP(N,i)$ is the relational polynomial time reduction. This gap is now bridged by the following theorem, which uses the normal-form theorem of [AV95].

Theorem 5.1: *Let $T_{\sigma,k}$ be the relational reduction machine of input type σ and arity k. There is a fixpoint formula $\varphi_{\sigma,k}$ in $FP(D,i)$ such that $\varphi_{\sigma,k}(\mathbf{D}) = rel(T_{\sigma,k}(\mathbf{D}))$.*

Proof: Let $\mathbf{D}$ be a structure of type σ. Using the notation in the proof of Theorem 4.5, recall that there exists an $FP(D,i)$ formula φ_1 defining $\mathbf{D}_\prec$ of type $\sigma_\prec$. Clearly, the query mapping $abs(\mathbf{D}_\prec)$ to $rel(T_{\sigma,k}(\mathbf{D}))$ is in P. Since $abs(\mathbf{D}_\prec)$ is an ordered structure and $FP(D,i)$ expresses P on ordered structures [Imm86, Var82], it follows that there exists an $FP(D,i)$ formula φ_2 defining that mapping. An $FP(D,i)$ formula φ_2' mapping $\mathbf{D}_\prec$ to $rel(T_{\sigma,k}(\mathbf{D}))$ is obtained by replacing variables representing a class of $\equiv_k^m$ by a corresponding tuple of m variables, and equality by a test of equivalence of m-tuples. Finally, the composition of φ_1 with φ_2' yields the desired $FP(D,i)$ formula $\varphi_{\sigma,k}$. ∎

Theorem 5.1 supplies the missing link between relational machines and fixpoint logics. Combining this with Theorem 4.5 we get:[11]

Theorem 5.2:

1. $FP(D,i) = \mathrm{P}_r$
2. $FP(N,i) = \mathrm{NP}_r$
3. $FP(A,i) = FP(D,n) = FP(N,n) = \mathrm{PSPACE}_r$
4. $FP(A,n) = \mathrm{EXPTIME}_r$

Proof: To simulate a relational machine by a fixpoint logic, one first uses $\varphi_{\sigma,k}$ to obtain $rel(T_{\sigma,k}(\mathbf{D}))$, and then uses the logic to simulate a standard machine. This is possible because $rel(T_{\sigma,k}(\mathbf{D}))$ is an ordered structure and, by Theorem 2.7, the fixpoint logics express the corresponding standard complexity classes on ordered structures. ∎

Theorem 5.2 should be contrasted with Theorem 2.7. While Theorem 2.7 talks about fixpoint logic *capturing* complexity classes, Theorem 5.2 provides a precise characterization of the expressive power of fixpoint logics in terms of relational complexity classes.

An immediate consequence of Theorem 5.2 is that $FP(A,i)$, $FP(D,n)$ $FP(N,n)$ all have the same expressive power. It follows that noninflationary fixpoint logic can express the problems of nonuniversality for finite automata and truth of quantified Boolean formula, albeit in a very non-straightforward fashion. Recall that Theorem 2.7 yielded the separation of $FP(D,i)$ and $FP(A,n)$. Both of these results demonstrate for the first time how complexity theory can yield *unconditional* results about expressive power of logics over *unordered* structures.[12]

We wish to use relational complexity as a mediator between fixpoint logic and standard complexity. Corollary 4.6 relates standard complexity to relational complexity, while Theorem 5.2 relates relational complexity to fixpoint logic. Together,

[11] The equalities $FP(D,i) = \mathrm{P}_r$ and $FP(D,n) = \mathrm{PSPACE}_r$ were shown in [AV95].

[12] An example where complexity theory yields *conditional* results about expressive power of logics over unordered structures appeared in [Fag74]. Fagin showed that existential and universal second-order logics coincide iff Hamiltonicity is expressible in universal second-order logic. Examples where complexity theory yields unconditional results about expressive powers of logics over ordered structures appeared in [Imm82, Imm87b]. Immerman showed that over ordered structures logarithmic iteration of first-order formulas is weaker than exponential iteration of first-order formulas. (See also [Str94].)

they bridge the gap between standard complexity and fixpoint logic – the open questions about the complexity classes between P and EXPTIME can be translated to questions about fixpoint logic.[13]

Corollary 5.3:

1. P = NP *iff* $FP(D,i) = FP(N,i)$
2. P = PSPACE *iff* $FP(D,i) = FP(D,n)$
3. NP = PSPACE *iff* $FP(N,i) = FP(N,n)$
4. PSPACE = EXPTIME *iff* $FP(A,i) = FP(A,n)$.

Thus, by the last three equivalences, some of the most tantalizing questions in complexity theory boil down to one fundamental issue: the relative power of inflationary vs. noninflationary 1st-order operators.

6 Concluding remarks

We established a general connection between fixpoint logic and complexity classes from P to EXPTIME. On one side, we have fixpoint logic, parameterized by the choices of 1st-order operators and iteration constructs. On the other side, we have the complexity classes between P and EXPTIME. While this connection is intimate, it is hampered by the order mismatch. Our parameterized fixpoint logics do capture all the complexity classes between P and EXPTIME, but equality is achieved only over ordered structures

To bridge this mismatch, we used and expanded a theory of relational complexity, which bridges the gap between standard complexity and fixpoint logic. On one hand, we show that questions about containments among standard complexity classes can be translated to questions about containments among relational complexity classes. On the other hand, the expressive power of fixpoint logic can be precisely characterized in terms of relational complexity classes. This tight three-way relationship among fixpoint logics, relational complexity and standard complexity yields in a uniform way logical analogs to all containments among the complexity classes P, NP, PSPACE, and EXPTIME. The logical formulation shows that some of the most tantalizing questions in complexity theory boil down to a single question: the relative power of inflationary vs. noninflationary 1st-order operators.

It is important to note that our framework does not extend to complexity classes below P, since our construction of order on the $\equiv_k$-classes takes polynomial time, which, by the results in [Gro96], seems unavoidable. In fact, many logics that capture complexity classes below P have been separated; see [Imm87a, GM92, GM92a].

[13]The equivalence of P=PSPACE and $FP(D,i) = FP(D,n)$ was shown in [AV95].

References

[AU79] A. V. Aho and J. D. Ullman. Universality of data retrieval languages. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 110–117, 1979.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and Datalog-like languages. In *Proc. 4th IEEE Symp. on Logic in Computer Science*, pages 71–79, 1989.

[AV90] S. Abiteboul and V. Vianu. Non-deterministic languages to express deterministic transformations. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 218–229, 1990.

[AV91a] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.

[AV91b] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 209–219, 1991.

[AV95] S. Abiteboul and V. Vianu. Computing with first-order logic. *Journal of Computer and System Sciences* 50:309–335, 1995.

[AVV95] S. Abiteboul, Moshe Y. Vardi, and V. Vianu. Computing with infinitary logic. *Theoretical Computer Science* 149:101–128, 1995.

[Ba77] J. Barwise. On Moschovakis closure ordinals. *Journal of Symbolic Logic*, 42:292–296, 1977.

[Bus86] S. R. Buss. *Bounded Arithmetics.* Bibliopolis, 1986.

[CH80] A. Chandra and D. Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[CH82] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[Cod72] E. F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Database Systems*, pages 33–64. Prentice-Hall, 1972.

[Com88] K. J. Compton. An algebra and a logic for NC^1. In *Proc. 3rd IEEE Symp. on Logic in Computer Science*, pages 12–21, 1988.

[DLW95] A. Dawar, S. Lindell, and S. Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation* 119:160–175, 1995.

[dR84] M. de Rougemont. Uniform definability on finite structures with successor. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 409–417, 1984.

[Fag74] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[Fag75] R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[Fed91] T. Feder. *Stable networks and product graphs.* PhD thesis, Stanford University, 1991.

[Fri71] H. Friedman. Axiomatic recursive function theory. In R. O. Gandy and C. E. M. Yates, editors, *Logic Colloquium '69*, pages 113–137. North Holland, 1971.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., 1979.

[Goe89] A. Goerdt. Characterizing complexity classes by higher-type primitive-recursive definitions. In *Proc. 4th IEEE Symp. on Logic in Computer Science*, pages 364–374, 1989.

[Gra84] E. Grandjean. The spectra of first-order sentences and computational complexity. *SIAM Journal on Computing*, 13:356–373, 1984.

[Gra85] E. Grandjean. Universal quantifiers and time complexity of random access machines. *Mathematical System Theory*, 13:171–187, 1985.

[GM92] E. Grädel and G. McColm. Deterministic versus nondeterministic transitive closure logic. *Proc. 7th IEEE Symp. on Logic in Computer Science*, pages 58–63, 1992.

[GM92a] E. Grädel and G. McColm. Hierarchies in transitive closure logic, stratified datalog and infinitary logic. *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 167–176, 1992.

[Gro96] M. Grohe. To appear in Proc. 37th IEEE Symp. on Foundations of Computer Science, 1996.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[GSS90] J-Y. Girard, A. Scedrov, and P. SCott. Bounded linear logic: a modular approach to polynomial time computability. In S. R. Buss and P. Scott, editors, *Feasible Mathematics*, pages 195–207. Birkhauser, 1990.

[Gur83] Y. Gurevich. Algebras of feasible functions. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 210–214, 1983.

[Gur84] Y. Gurevich. Toward logic tailored for computational complexity. In M. M. Ricther et al., editor, *Computation and Proof Theory, Lecture Notes in Mathematics 1104*, pages 175–216. Springer-Verlag, 1984.

[Gur88] Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current trends in theoretical computer science*, pages 1–57. Computer Science Press, 1988.

[HP84] D. Harel and D. Peleg. Static logics, dynamic logics, and complexity classes. *Information and Control*, pages 86–102, 1984.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.

[IL90] N. Immerman and E. S. Lander. Describing graphs: a first-order approach to graph canonization. In A. Selman, editor, *Complexity Theory Retrospective*, pages 59–81. Springer-Verlag, 1990.

[Imm82] N. Immerman. Upper and lower bounds for first-order expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[Imm87a] N. Immerman. Expressibility as a complexity measure: results and directions. In *Second Structure in Complexity Conference*, pages 194–202, 1987.

[Imm87b] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[Imm89] N. Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Computational Complexity Theory, Proc. Symp. Applied Math., Vol. 38*, pages 75–91. American Mathematical Society, 1989.

[JS74] N. G. Jones and A.L. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.

[KV92] Ph. G. Kolaitis and M. Y. Vardi. Infinitary logics and 0–1 laws. *Information and Computation* 98:258–294, 1992.

[KV95] Ph. G. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: tools and a case study. *Journal of Computer and System Sciences*, 51:110–134, 1995.

[Lei89a] D. Leivant. Descriptive characterization of computational complexity. *Journal of Computer and System Sciences*, 39:51–83, 1989.

[Lei89b] D. Leivant. Monotonic use of space and computational complexity over abstract structures. Unpublished manuscript, 1989.

[Lei90] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95–108, 1990.

[Lei91] D. Leivant. A foundational delineation of computational feasibility. In *Proc. 6th IEEE Symp. on Logic in Computer Science*, pages 2–11, 1991.

[Lin91] S. Lindell. An analysis of fixed point queries on binary trees. *Theoretical Computer Science*, 85:75–95, 1991.

[Liv82] A. B. Livchak. The relational model for systems of automatic testing. *Automatic Documentation and Mathematical Linguistics*, 4:17–19, 1982.

[Liv83] A. B. Livchak. The relational model for process control. *Automatic Documentation and Mathematical Linguistics*, 4:27–29, 1983.

[Lyn82] J. F. Lynch. Complexity classes and theories of finite models. *Mathematical System Theory*, 15:127–144, 1982.

[Mos74] Y. N. Moschovakis. *Elementary Induction on Abstract Structures.* North Holland, 1974.

[Sav80] W. J. Savitch. Relational between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4:177–192, 1980.

[Saz80a] V. Yu. Sazonov. A logical approach to the problem of P=NP. In *Proc. 9th Conf. on Math. Found. of Computer Science*, pages 561–575. Springer-Verlag, Lecture Notes in Computer Science 88, 1980.

[Saz80b] V. Yu. Sazonov. Polynomial computability and recursivity in finite domains. *Elektronische Informationverarbeitung und Kybernetik*, 16:319–323, 1980.

[Sch94] T. Schwentick, Graph connectivity and monadic NP, *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 614–622, 1994.

[Sto77] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Str94] H. Straubing. *Finite automata, formal logic, and circuit complexity.* Birkhauser, Boston, 1994.

[TU88] J. Tiuryn and P. Urzyczyn. Some relationships between logic of programs and complexity theory. *Theoretical Computer Science*, 60:83–108, 1988.

[Ull89] J. D. Ullman. *Database and Knowledge-Base Systems, Volumes I and II.* Computer Science Press, 1989.

[Var82] M. Y. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.