Monotonic use of space and computational complexity over abstract structures

Daniel Leivant
October 31, 1989

CMU-CS-89-212

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We use relational pointer machines as a framework for generalized computational complexity. Non-reuse of memory space, dubbed *monotonic* computing, is proposed as a fundamental concept that threads together various abstract generalizations of PTime. Depending on the use of space, relational machines generalize DLogSpace, PTime, NPTime and PSpace, We show that alternating first order machines are equivalent, over all finite structures, to monotonic machines with positive queries, generalizing the Chandra-Kozen-Stockmeyer Theorem $ASpace(f) = DTime(2^f)$, and showing that the latter does not depend on any counting mechanism. We also show that of two generalizations of PTime, deterministic monotonic machines and nondeterministic monotonic positive machines, the former can simulate the latter on all structures, and the latter can simulate the former on enumerated structures. Finally, first order inductive definitions are shown to be equivalent to monotonic pointer machines with random selection.

Research sponsored in part by the Defense Advanced Research Projects Agency (DOD) under Contract F33615-87-C-1499 , ARPA Order No. 4976 (Amendment 20) and monitored by: Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFSC), Wright-Patterson AFB, OH 45433-6543. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the U.S. Government.

Introduction

Models of computing over arbitrary structures have been studied for several decades, but have attracted increased interest only in recent years, in Logic of Programs, Computational Complexity, Database Theory, and Generalizes Recursion Theory. For foundations of computing, computing over arbitrary structures is motivated by the search for unifying principles, analogies between complexity and definitional specifications and hierarchies, relations with finite model theory, and an understanding of the limitations of structures and of computation models.

Generalized complexity theory was initialized in [Fri70] and pursued by Tiuryn, Kfoury, Urzyczyn and others. One goal of the theory is the *generalization* of Turing machine based complexity classes, such as the identification of generalized forms of PTime. These generalizations clarify the nature, importance, and stability of the Turing class considered. If several generalizations of the same Turing class TC (such as PTime) are equivalent in general, the stability of TC is demonstrated. If two generalizations of TC are shown to differ on some structures, then two orthogonal aspects of TC are identified. Dually, if the generalizations of two classes TC_1 and TC_2 are shown to differ on some structure, then we go some of the way towards separating TC_1 and TC_2, and we know that if they coincide as Turing classes then a characteriztic feature of the Turing model, such as counting configurations within the model, is crucial in the equivalence.

In another vein, generalized computational complexity theory might shed light on relations and tradeoffs between various computational complexity measures for specific hardware models (e.g., Turing machines, various models for concurrency), such as time, space, reversals, alternations, number of processes, depth of circuits, extent of communication, etc.

In [Lei87] we proposed the explicit use of higher order objects in computation models, a direction that has already borne additional interesting results [Goe89]. The model defined in [Lei87] is that of *relational pointer machines* (see §2 below). In this paper we study the effects of natural restrictions on such machines: disallowing reuse of space (*monotonic* computing), and disallowing negative information (by requiring queries to be *positive*, i.e. abort on negative response). These restrictions lead to generalizations of deterministic and nondeterministic PTime.

We show that of two generalizations of PTime, deterministic monotonic machines and nondeterministic monotonic positive machines, the former can simulate the latter on all structures, and the latter can simulate the former on enumerated structures. We further show that the most natural generalization of PTime, nondeterministic monotonic positive machines, is equivalent *over all structures* to alternating first order pointer machines, thereby generalizing one of the main results of [CKS81]. This shows that the equivalence between, e.g., alternating logarithmic space and deterministic polynomial time, is in fact independent of the exhaustive enumerability (within the computational model) of the structure elements or of configurations. This should be contrasted with other results on Turing complexity, such as Savitch's Theorem and Immerman-Szelepcsényi Theorem [Imm88, Sze87], which

have so far resisted such generalizations, and may well depend crucially on the ability of the computation model to count configurations.

Our theorem equating alternation with monotonic use of space is related to the equivalence proved in [HK84], between alternating programs and first order inductive definitions. However, the proofs are different, and the present equivalence seems to relate more natural generalization of time and space.

The use of monotonic (i.e. non-reused) space seems to be of independent interest, as an abstract generalization of time complexity, and as a unifying concept for several generalizations of PTime. The concept of non-reused space goes back at least to Hao Wang [Wan57], who showed that non-erasing TM's are as general as unrestricted TM's. Our results suggest that monotonic space is a fundamental concept which manifests itself in a number of guises, and threads together several generalizations of PTime. The significance of monotonic space computing is illustrated by an interesting result in Thermodynamics: Charles Bennett [Ben87] has discovered a surprising and beautiful resolution of the paradoxes of Thermodynamics, such as Maxwell's Demon, in terms of informational entropy: the point where thermal entropy decreases, in all idealized machines that violate the Second Law, is precisely ehere information space is being reclaimed, thereby assuming a gratuitous decrease in information entropy. In another vein, Girard's Linear Logic is based on related observations about the cost of reusing information, and in fact the provably recursive functions of Linear Logic with counters on space reuse are precisely the PTime computable functions [GSS89].

1. Preliminaries on computing over structures

1.1. Abstract machines

The simplest computation model over arbitrary structures are the program schemes of [LP64, Pat68], and their notational variants such as register machine [Fri70] and first order pointer machines [Lei87]. (The concept of programs over arbitrary structures goes back at least to [Ian60].) When the underlying structure is the Turing Tape, the model becomes a multihead read-only Turing machine. We use the concrete syntax of first order pointer machines from [Lei87], to which we refer as 1M's. The basic machine has a finite set of *states*, with a designated initial state and an accepting state, a finite set of *pointers* (to objects), and transition rules of two types: **Valuation**, that places a pointer at the value of a structure function applied to the current values of designated pointers (and alters the state); and **Test**, that branches to one of two states depending on whether a structure relation evaluates as true or false for the current values of designated pointers. The default control flow is nondeterministic. The machine *accepts* its input if there is an accepting computation, with designated pointers initialized to the input. A more general model arises from enriching the control flow to allow *alternation* [CKS81]: the non-accepting states are classified as existential or universal, and *accepting alternating computations* are defined as in [CKS81]. The machine *accepts* a structure if there is an accepting alternating computation for the input.

1.2. Global relations

The issue of interest when computing over finite structures is global (i.e. uniform) computability: a *global relation [function, boolean]* over a class $\mathcal{C}$ of structures is a mapping that assigns to each structure $\mathcal{S} \in \mathcal{C}$ a relation [function, boolean value] in $\mathcal{S}$ [Tar52, Gur87]. Suppose M is a 1M over structures in $\mathcal{C}$, with some canonical ordering $\pi_1 \ldots \pi_q$ of its pointers. For each r M determines a global r-ary (partial) function $[M]_r$ on $\mathcal{C}$, where $[M]_r\mathcal{S}(x_1 \ldots x_r)$ is the value (if there is one) obtained by running M on $\mathcal{S}$, with pointer π_i initialized to x_i for $i = 1 \ldots \min[r, q]$.

The nature of computing over structures is greatly affected by the nature of object nameability. A structure is *denoted* if each element is the value of some closed term of the underlying vocabulary.

If **mc** is a model of computation over structures, a class $\mathcal{C}$ of structures is **mc**-*accessible* if there is an $M \in$ **mc** such that, in each $\mathcal{S} \in \mathcal{C}$, every $a \in |\mathcal{S}|$ is the value of some pointer of M along some computation of M (without input). The pebbling technique of [Fri70] shows that there exists a class of denotable finite structures that is not accessible by a 1M (even allowing nondeterminism or alternation).

The greatest nameability is present in *enumerated* structures $\mathcal{S}$, whose elements come enumerated (ordered) by some structure (or computable) unary function $\mathbf{n}$, starting from a structure constant $\mathbf{0}$: $|\mathcal{S}| = \{\mathbf{0}, \mathbf{n}(\mathbf{0}), \ldots, \mathbf{n}^k(\mathbf{0})=\mathbf{0}\}$. Structures presented on a Turing tape come enumerated by virtue of their presentation.

1.3. Measuring space complexity

A *computation space* traditionally consists of addressable memory locations, where information can be stored, retrieved, and replaced. The computation space need not be the input structure itself; in Turing computability this separation allows making sense of low space measures, and for computing over abstract structures it permits the very definition of space complexity.

To make sense of space complexity for computing over a structure $\mathcal{S}$ we consider computation over $\mathcal{S}$ joint with an auxiliary memory, in the form of a Turing Tape. This structure-joining method has appeared in several independent accounts of computing over abstract structures [Fri71, Fen80], and in Turing computability to make sense of sublinear resources [SHL65]. For sets A, B, let $A \oplus B$ denote the disjoint union of A and B. If $\mathcal{S}$, $\mathcal{A}$ are structures over vocabularies σ and α, respectively, then $\mathcal{S} \oplus \mathcal{A}$ is the structure of signature $\sigma \oplus \alpha$ whose universe is $|\mathcal{S}| \oplus |\mathcal{A}|$ (disjoint union), where $\mathbf{F}^{\mathcal{S}\oplus\mathcal{A}} = \mathbf{F}^{\mathcal{S}}$ for each identifier $\mathbf{F} \in \sigma$, and $\mathbf{F}^{\mathcal{S}\oplus\mathcal{A}} = \mathbf{F}^{\mathcal{A}}$ for each $\mathbf{F} \in \alpha$. A $(\sigma \oplus \alpha)$-machine M and structure $\mathcal{A}$ determine r-ary global functions over σ-structures $\mathcal{S}$, $[M]_r^{\mathcal{A}}(\mathcal{S}) =_{Df} [M]_r(\mathcal{S} \oplus \mathcal{A})$.

Our canonical auxiliary structure is the Turing Tape *Tape*, whose elements are pairs

of stacks over the alphabet $\{0, 1, B\}$, with the obvious operations. A computation over $\mathcal{S} \oplus \mathcal{T}ape$ is then *in space k* if it uses only elements of $\mathcal{T}ape$ of length $< k$. An r-ary global function ρ over σ-structures is *Turing computable in space f* if $\rho = [M]_\tau^{\mathcal{T}ape}$ for some M over the vocabulary $\sigma \oplus \tau$ (where τ is the vocabulary of $\mathcal{T}ape$), such that, for a σ-structure $\mathcal{S}$, $[M]_r(\mathcal{S} \oplus \mathcal{T}ape)$ uses $\leq f(|\mathcal{S}|)$ of $\mathcal{T}ape$, for all input. *Turing computability in time f* is defined analogously.

THEOREM I (Har72, Gur87, CKS81) *The global relations over enumerated structures defined by 1M's (deterministic 1M's) are precisely the ones Turing computable in NLogSpace (DLogSpace, respectively). The global relations defined by alternating 1M's are precisely the ones Turing computable in PTime.*

1.4. Relational pointers and transitions

A relational (second order) machine [Lei87], abbreviated 2M, is a 1M enriched with *relational pointers*, each assigned a unique arity, and transitions rules for them: **Store,** that places the value of a vector of first order pointers into a relational pointer P, **Delete,** that removes such a value vector from P, and **Query,** that branches to one of two states on testing the presence of such a vector in P. Thus, a 2M is a 1M enriched with a memory of size polynomially related to the size of the underlying structure, whereas a 1M has only a fixed size "CPU." Relational pointers are initialized to the empty set (more generally, they might be used to allow relational input, i.e. oracles).

THEOREM II (Lei87) *The global functions over enumerated structures definable by a (deterministic or nondeterministic) 2M are precisely the ones Turing computable in PSpace.*

2. Monotonic and positive uses of space

2.1. Monotonic Turing computability

A Turing machine T is *monotonic (non-erasing)* if it never writes a 0 on top of a 1. Hao Wang [Wan57] showed that non-erasing TM's are as general as unrestricted TM's.

THEOREM III *A language $L \subseteq \{0,1\}^*$ is accepted in monotonic DSpace(f) (where $f(n) \geq n$) iff it is accepted in DTime(f). Similarly, L is accepted in monotonic NSpace(f) iff it is accepted in NTime(f).*

Proof. We give the proof for the deterministic case, the nondeterministic case being similar. A TM machine T that runs in monotonic space f will repeat a configuration after at most $k \cdot f(n)$ steps, where k is the number of states, and hence is in DTime(f).

For the converse, assume that T runs in DTime(f). Simulate T by a TM T' that generates a list of successive "local configurations" of T, i.e. triplets $\langle$*state, scanned symbol, position*$\rangle$, where *position* is the distance of the scanned cell from the first tape cell. T' can compute the next local configuration by scanning already-listed local configurations. For example, suppose that the last local configuration is $\langle q, 0, k\rangle$, and the control of T dictates moving the head (of T) to the right and changing state to q'. To determine the next scanned symbol, T' will serach backwards through the list of local configurations, until a triplet of the form $\langle p, s, k{+}1\rangle$ is encountered. T' then returns to the last local configuration on its own tape, and write $\langle q', s, k{+}1\rangle$ to its right. If the search fails, then T' returns to the end of its tape and writes $\langle q', B, k{+}1\rangle$.
□

2.2. Monotonic relational machines

A 2M is *monotonic* (abbreviated M) if it uses no Deletion transition. Thus, an M2M cannot "reuse its store."

THEOREM IV *A global function over enumerated structures is definable by a deterministic (nondeterministic) M2M iff it is Turing defined in DPTime (NPtime, respectively).*

Proof. This is a simple modification of the proof of Theorem II. The monotonic use of space trivially implies a time bound similar to the space bound. Conversely, if computing is within polynomial time, then the relational pointers used to capture PSpace can be modified to have extra dimensions, in which a time stamp of the computation (using the structure enumeration) can be placed. This renders the computation monotonic.
□

A related result for Turing computability is:

THEOREM V *Let f be a numeric function, $f(n) \geq n$, The global relations over enumerated structures definable by deterministic (non-deterministic) M2M's running in space f are precisely the ones computable in DTime(f) (in NTime(f), respectively).*

2.3. Positive queries

A query is **positive** if on a negative response the machine enters an aborting state. A 2M is *positive* (abbreviated P) if all its queries are positive.

THEOREM VI *Let C be a P2M-accessible class of finite structures (in particular, any class of enumerated structures). If a global function Φ over C is PSpace Turing definable, then it is definable already by a P2M.*

Proof. Suppose Φ is defined by a 2M M. Let M' be a 2M that has the pointers of M plus, for each relational pointer P of M, a fresh pointer $\bar{P}$, with $arity(\bar{P}) = arity(P)$. M' simulates M, while maintaining in $\bar{P}$ the complement of P (this is doable using the structure's accessibility and the presence of both Store and Delete transitions). Then M' simulates an unrestricted query of M, $\hat{\pi} \in_? P$, by nondeterministically branching to a positive query $\hat{\pi} \in_? P$ and to a positive query $\hat{\pi} \in_? \bar{P}$.
□

2.4. Monotonic positive relational machines

We have three types of restriction on 2M's: storage restricted by monotonicity, queries restricted to be positive, and control restricted to deterministic. The conjunction of all three eliminates the computational advantage of relational pointers, except possibly for a quadratic reduction in computing time:

THEOREM VII *Every deterministic MP2M can be simulated by a deterministic 1M. Therefore, these two computation models define the same global functions.*

Proof. Define a *first order* configuration of M as consisting of the state and the value of the first order pointers, disregarding the values of the relational pointers. Note that in a deterministic MP2M, if a first order configuration repeats, then there is divergence.

Suppose M is a deterministic MP2M, say with one relational pointer P, which is unary. Let M' be a deterministic 1M that simulates M, disregarding Store transitions, until a query $\pi \in_? P$ is encountered. M' then suspends the simulation, saves the value v of π and the first order configuration *cfg* of M, and restarts the simulation of M from the initial configuration, this time checking values stored in P against v, until the return of *cfg*. If v has been encountered along the second simulation, M' proceeds as M would with a positive response to the query. Otherwise M' aborts.

This procedure is iterated, except that to simulate the n'th query of M, for $n > 1$, M' reruns as above the computation of M, as if all queries up to the return of *cfg* are answered positively (M' has already verified in previous iterations that these queries have had a positive response). Note that the second phase of that procedure brings M' to the first occurrence of *cfg*, even if simulation was suspended at a repeated occurrence (with possibly a different contents for P); this will produce divergence, but then M diverges too, as pointed out above.
□

2.5. Abstract equivalents of PTime

From Theorem III we know that DM2M is equivalent to PTime over enumerated structures. We show that so is (nondeterministic) MP2M. One containment holds for *arbitrary*

structures:

THEOREM VIII *If a global relation (over finite or infinite structure) is definable by an MP2M, then it is definable by a DM2M.*

Proof. The key idea is that in MP2M's the order of relational-type transitions is immaterial, because Deletion is absent and all queries are positive. Let M be an MP2M with r object pointers. Construct a DM2M M' that simulates M, as follows. M' has the relational pointers of M, plus, for each state q of M, an r-ary relational pointer $\tilde{q}$, intended to accumulate value vectors $\hat{x}$ such that the first order configuration $(q, \hat{x})$ is accessible in M from the initial configuration. M' iterates a process C, where each run of C cycles deterministically through all transitions of M, and for each such transition, with target state q say, stores in $\tilde{q}$ the target values of the object pointers of M. Also, if such transition is a Store, then M' simulates that Store, and if the transition is a Query, then M' tests (fully, not merely positively) for the query, and updates the appropriate $\tilde{q}$ only on positive response. M' accepts when a value is stored in $\tilde{q}_{\text{acc}}$ where q_{acc} is the accepting state of M.
□

Lemma 1 *If a global relation over enumerated structures is definable by a DM2M, then it is definable by an MP2M.*

Proof. Let M be a DM2M, say with one relational pointer P. The monotonicity implies that there is a polynomial f that bounds the length of repetition free computation sequences. Let $\hat{\tau}$ be an r-ary vector of fresh object pointers that can serve, on enumerated structures of size $\leq n$, as a counter up to $f(n)$.

Define an MP2M M' that simulates M, as follows. M' keeps in $\hat{\tau}$ count of the number of queries encountered so far by M. M' also has fresh r-ary relational pointers L and R, with $\hat{t} \in L$ intended to indicate that the $\hat{t}$'th query was answered positively, and $\hat{t} \in R$ that it was answered negatively.

On a query $\hat{\pi} \in_? P$ of M, with target states q^+ and q^-, M' branches nondeterministically to processes Y and N, after incrementing $\hat{\tau}$. Process Y queries $\hat{\pi} \in_? P$, aborting on negative response, or storing, on positive response, the current value $\hat{t}$ of $\hat{\tau}$ in L, and proceeding to simulate M from q^+. Process N saves the current values $\hat{v}$ of $\hat{\pi}$ and $\hat{t}$ of $\hat{\tau}$, and restarts a simulation of M from its initial configuration, using nondeterministic guessing for membership in L or R to decide on proper branching at queries, and counting queries until reaching the $\hat{t}$'th (i.e. the suspended) query. If, during that rerun, $\hat{v}$ is one of the values stored in P, then N aborts. Otherwise N proceeds with simulating M from q^-, after storing $\hat{t}$ in R.
□

Combining Theorems IV and VIII, and Lemma 2..1, we have

THEOREM IX *The global relations over enumerated structures defined by MP2M's are precisely the ones Turing computable in PTime.*

2.6. The spectrum of relational machines

We summarize the classification obtained above for pointer machines, as generalizations of five natural Turing complexity classes. Writing $C_1 \supseteq C_2$ if every global function (over finite structures) definable in computation model C_2 is definable in computation model C_1, and $C_1 \equiv C_2$ if both $C_1 \supseteq C_2$ and $C_2 \supseteq C_1$, we have

THEOREM X

Turing class	*Generalizations*
PSpace	$2M \supseteq \begin{matrix} P2M \\ D2M \end{matrix} \supseteq DP2M$
NPTime	M2M
PTime	DM2M $\supseteq$ MP2M
NLogSpace	1M
DLogSpace	DMP2M $\equiv$ D1M

3. Monotonic space and alternation

The following theorem is a generalization to arbitrary structures of the Chandra-Kozen-Stockmeyer Theorem on the equivalence of alternating space f and deterministic time 2^f [CKS81, Theorem 3.4]. Note that the proof is not an adaptation of the proof in [CKS81], which uses the exhaustive enumerability of configurations. The formulation of the theorem relates generalizations of ALogSpace and DPTime, but the general form of the CKS Theorem falls out by considering, in place of the given structure, the structure already expanded with a workspace (by joining an auxiliary structure as in §1).

THEOREM XI *A global relation over finite structures is definable by an alternating 1M iff it is definable by a (nondeterministic) MP2M.*

Proof. Forward direction. Let M be an alternating 1M with states $q_1 \ldots q_k$, where q_k is the accepting state. Let M' be a MP2M with r-ary relational pointers $\tilde{q}_1, \ldots, \tilde{q}_k$. The intention is to store in $\tilde{q}_i$ the r-ary tuples of values $\hat{x}$ for which $\langle q_i, \hat{x} \rangle$ is an accepting configuration of M. M' iterates the following process S, until the initial value vector is stored in pointer $\tilde{q}_k$. S simulates nondeterministically some execution sequence of M. At any time, S may nondeterministically choose to cease simulation and to check for acceptance of the current configuration $(q_j, \hat{u})$: if q_j is accepting, the check is successful; if q_j is existential, M' chooses some applicable transition, with target configuration $(q_i, \hat{x})$ say, and considers the check successful if either $i = k$ or the query $\hat{x} \in ?\ \tilde{q}_i$ succeeds; if q_j is universal, M' considers the check successful if, for every target configuration $(q_i, \hat{x})$, either $i = k$ or $\hat{x} \in ?\ \tilde{q}_i$

is answered positively. If the check of the current configuration is successful, S stores $\hat{u}$ in $\tilde{q}_j$, and returns control to M' (otherwise S aborts). M' accepts the input iff M accepts.

Backward direction. Let M be a deterministic M2M. Suppose, without loss of generality, that the relational pointers of M consist of a single, r-ary, pointer P. We define an alternating 1M M' that simulates M. The basic process S of M' takes as input first order configurations cfg_0 and cfg_1 of M and a value v, and determines whether cfg_1 is accessible in M from cfg_0, by a computation sequence that stores v in P. S simulates M from cfg_0, comparing v to each value stored in P. When a (positive) query $\pi \in ? P$ is encountered, on a first order configuration cfg_2 say, S branches universally to S for input $(cfg_0, cfg_2, value(\pi))$, and to S for input (cfg_2, cfg_1, v). S_0 is a variant of S with no search for a value v (though such searches may be activated by the spawned processes).

Without loss of generality, M has a unique accepting first order configuration. To determine whether M accepts its input it remains for M' to run S_0 with, as input, the initial configuration and the accepting first order configuration of M.
□

Bounded computational formulas are defined in [Lei87]. A corollary of Theorem XI is:

THEOREM XII *A global relation is definable by an M2M iff it is definable by a bounded computational formula.*

Alternating transitive closures are defined in [Imm87]. Another corollary of Theorem XI is:

THEOREM XIII *A global relation is definable by an M2M iff it is definable by the alternating transitive closure of an existential first order formula.*

4. Relational machines and PTime

4.1. Equivalents of monotonic relational machines

MP2M's are a simple computational model that captures exactly DPTime over enumerated structures. In this section we list other methods of defining global functions that are equivalent, over *all* finite structures, to MP2M's.

Papadimitriou observed [Pap85] that pure uninterpreted logic programs, without negation on defined relations, define exactly the DPTime global relations over enumerated structures. This computation model, which arose first as a database query language [HN84], is basically the same as our MP2M's (the latter refers also to functions, but these can be simulated by their graphs). Hence the result of [Pap85] is the same as Theorem IX above. Note that

computability by logic programs without negations is trivially equivalent to definability by positive fixpoints over Σ_1 formulas, which therefore define again the same global relations as MP2M's. (Compare [Fit81, BG85]).

Dually, one can consider structures with functions only. The evaluation rules for functional computing can be simulated by monotonic and positive relational programs. Theorem IX is therefore another guise of the theorem of Sazonov and Gurevich [Saz80, Gur83], that global functions over enumerated structures are in PTime iff they are definable by recursion equations.

Two aspects of our models make them a particularly attractive generalization of PTime. First, the use of machine-like terminology allows a clear statement, in a single framework, of control mechanisms (deterministic, nondeterministic, alternating). In addition, we have explicit forms of interaction between relations and objects, allowing a single framework for describing use and reclaim of memory.

A result related to Theorem XI is Harel and Kozen's [HK84], where it is shown (by arguments different from ours) that alternating programs with random assignments are equivalent to first order inductive definitions. MP2M's and 1M's seem, however, to be more straightforward generalizations of PTime and ALogSpace.

4.2. Selection transitions

The elements of a structure need not be denotable: in a graph no element is denotable, and in a non-standard model of arithmetic the denotable elements are the standard numbers. Such elements become computationally accessible by **Selection** transitions, that nondeterministically assign random values to value pointers. The most natural semantics of Selection transitions is existential nondeterministic, but they may equally well be given a universal nondeterministic semanticsemantics. With the former, Selection can be used to simulated control nondeterminism (over structures with at least two elements). Vice versa, Selection can be simulated in accessible structures by control nondeterminism.

The use of Selection is essential in characterizing computationally global relations defined by formulas with quantifiers. We have:

THEOREM XIV *The following models define the same global relations (over arbitrary structures):*

1. *MP2M with (existential) Selection transitions.*
2. *Alternating 1M with Selection transitions.*
3. *Positive first order inductive definitions [Mos69, Acz77].*

Proof. The equivalence of (1) and (2) is similar to Theorem XI. The equivalent of (1) and (3) is straightforward; alternatively, the equivalence of (2) and (3) is proved in [HK84], since (2) is equivalent to [HK84]'s language IND of alternating programs (Selection nondeterminism enables control nondeterminism).
□

The equivalence of first order inductive definability and PTime Turing computability for defining global relations over enumerated structures is due to Immerman and Vardi [Imm82, Var82].

References

Acz77 Peter Aczel, *An introduction to inductive definitions,* in Jon Barwise (editor), **Handbook of Mathematical Logic**, North-Holland, Amsterdam, 1977, pp.739–782.

Ben87 Charles H. Bennett, *Demons, engines and the Second Law,* **Scientific American**, November 1987, 108–116.

BG85 Andreas Blass and Yuri Gurevich, *Existential fixed-point logic.*

CKS81 Ashok Chandra, Dexter Kozen and Larry Stockmeyer, *Alternation,* **Journal of the ACM 28** (1981), 114–133.

Fen80 Jens E. Fenstad, **General Recursion Theory**, Springer-Verlag, Heidelberg and New York, 1980.

Fit81 Melvin Fitting, **Fundamentals of Generalized Recursion Theory**, North Holland, Amsterdam, 1981.

Fri70 Harvey Friedman, *Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory,* in R.O. Gandy and C.M.E. Yates (eds.), **Logic Colloquium '69**, North-Holland, Amsterdam, 1970. 361–389.

Fri71 Harvey Friedman, *Axiomatic recursive function theory,* in **Logic Colloquium '69** (R.O. Gandy and C.E.M. Yates, editors), North-Holland, Amsterdam, 1971, 113–137.

Goe89 Andreas Goerdt, *Characterizing complexity classes by higher type primitive recursive definitions,* **Proceedings of the Fourth Annual Symposium on Logic in Computer Science**, IEEE Computer Society Press, Washington DC, 1989, 364–374.

GSS89 Jean-Yves Girard, Andre Scedrov and Philip Scott, *Bounded Linear Logic I: A modular approach to polynomial time computability,* Preliminary report, Feasible Mathematics Workshop, Cornell, June 1989.

Gur83 Yuri Gurevich, *Algebras of feasible functions,* **Twenty Fourth Symposium on Foundations of Computer Science**, IEEE Computer Society Press, 1983, 210–214.

Gur87 Yuri Gurevich, *Logic and the challenge of Computer Science,* in **Current Trends in Theoretical Computer Science,** (Egon Borger, editor), Computer Science Press, 1987.

Har72 Juris Hartmanis, *On nondeterminacy in simple computing devices,* **Acta Informatica 1** (1972) 336-344.

HK84 David Harel and Dexter Kozen, *A programming language for the inductive sets, and applications,* **Information and Control 63** (1984) 118–139.

HN84 L.J. Henschen and S.A. Naqvi, *On compiling queries in recursive first order databases,* **JACM 31** (1984) 47–85.

Imm82 Neil Immerman, *Relational queries computable in polynomial time*, **Information and Control 68** (1986) 86–104. Preliminary report in **Fourteenth ACM Symposium on Theory of Computing**, 1982, 147–152.

Imm87 Neil Immerman, *Languages which capture complexity classes*, **Siam Journal of Computing 16** (1987) 760–778.

Imm88 Neil Immerman, *Nondeterministic space is closed under complement*, **IEEE Structure in Complexity Theory Third Annual Conference**, IEEE Computer Society, Washington DC, 1988, 112–115.

Lei87 Daniel Leivant, *Characterization of complexity classes in higher order logic*, **Proceedings of the Second Annual Conference on Structure in Complexity Theory**, IEEE, New York, 1987, pp. 203–217. Revised version to appear in the **JCSS**.

LP64 David Luckham and David Park, *The undecidability of the equivalence problem for program schemata*, Report 1141, Bolt, Beranek & Newman Inc., 1964.

Mos69 Yiannis Moschovakis, *Abstract first order computability II*, **Trans. American Mathematical Society 138** (1969) 465–504.

Pap85 Christos Papadimitriou, *A note on the expressive power of PROLOG*, **Bull. EATCS 26** (June 1985) 21–23.

Pat68 Michael Paterson, *Program schemata*, **Machine Intelligence 3** (1968) 19–31.

Saz80 Vladimir Sazonov, *Polynomial computability and recusivity in finite domains*, **Electronische Informationsverarbeitung und Kybernetik 7** (1980) 319–323.

SHL65 R.E. Stearns, J. Hartmanis and P.M. Lewis, *Hierarchies of memory limited computations*, **Conference Record on Switching Circuits Theory and logic Design**, IEEE Press, 1965, 179–190.

Sze87 Robert Szelepcsenyi, *The method of forcing for nondetreministic automata*, **Bull. EATCS 33** (1987) 96-100.

Tar52 Alfred Tarski, *Some notions and methods on the borderline of algebra and metamathematics*, **Proceedings of the International Congress of Mathematicians** (1950), Volume I, American Mathematical Society, 1952, 705–720.

Var82 Moshe Vardi, *Complexity and relational query languages*, **Fourteenth Symposium on Theory of Computing**, 1982, 137–146.

Wan57 Hao Wang, *A variant to Turing's theory of computing machines*, **JACM 4** (1957) 63–92.

December 18, 1989

Allocation of Secretarial Resources
School of Computer Science

Name	Ext.	Faculty Distribution
Lesly Adkins-Shellie	6246	M. Erdmann, C. Seger, B. Vander Zanden, A. Witkin
Kathleen Cywinski	3853	E. Cooper, R. Dannenberg, E. Rollins, D. Touretzky, J. Wing
Lydia DeFilippo	3063	D. McKeown, D. Scott
Colleen Everett	7674	M. Accetta, D. Adams, F. Alleva+, R. Taylor+, H. Wactlar
Laura Forsyth	2619	M. Barbacci+, H. Berliner, D. Siewiorek
Barbara Grandillo	7550	S. Brookes, H. T. Kung, D. O'Hallaron+, R. Sansom
Maria Intrieri	3342	B. Clark, Introductory Lecturers, P. Miller
Michelle Jackson	5025	A. Forin+, P. Lee, F. Pfenning, J. Reynolds, M. Satyanarayanan, B. Scherlis (loa), M. Shaw
Jennifer Jones	7660	J. Carbonell, E. Clarke
Annamarie Mackuliak	3779	S. Burks, D. Giuse+, R. Kannan, G. Miller (Visitor), D. Sleator, D. Tygar
Joan Maddamma	7656	J. Lehoczky (Statistics), D. Leivant, J. Morris, H. Tokuda, M. Tomita
Pat Miller	3825	J. Bates, B. Bruegge, M. Herlihy (LOA), R. Maxion, B. Myers, A. Rudnicky
Phyllis Pomerantz	2592	N. Habermann
Marjorie Profeta	7675	B. John, D. Miller+, A. Newell, A. Waibel
Barbara Sandling	8860	R. Bryant, C. Chien, S. Fahlman, K. Lee, D. Lindsay+, M. Perlin, P. Steenkiste
Jean Scavincky	3802	M. Mason, T. Mitchell, R. Simmons, J. Schlimmer
Cleah Schlueter	7884	G. Blelloch, J. Lehman, R. Harper, A. Kutay (Rob.)+, B. Lerner, S. Rudich
Marian Sodini	7665	R. Bisiani, T. Gross, R. Rashid, S. Young
Terri Stankus	3731	A. Fisher, M. Furst
Teralyn Thompson(Rob.)	3838	M. Hebert (Rob.), K. Ikeuchi, S. Shafer, C. Thorpe (Rob.), J. Webb

+answer telephone only