MIT/LCS/TM-204

ON THE EXPRESSIVE POWER OF DYNAMIC LOGIC, II

Joseph Y. Halpern

August 1981

On The Expressive Power of Dynamic Logic, II

Joseph Y. Halpern
M.I.T. Laboratory for Computer Science and
Aiken Computation Laboratory, Harvard University

1. Introduction

In this paper we study the expressive power of nondeterminism in dynamic logic. In particular, we show that first-order regular dynamic logic ***without equality*** (hereafter abbreviated DL) is more expressive than its deterministic counterpart (DDL). This result has already been shown for the quantifier-free case [MW], and for the propositional case [HR]. Berman and Tiuryn have recently extended the present result to the case with equality. By contrast, Meyer and Tiuryn have shown in [MT] that in the r.e. case, deterministic and nondeterministic dynamic logic coincide.

The proof hinges on showing that in a precise sense a deterministic regular program cannot search a full binary tree. Because of this, the truth of a first-order DDL formula, even with first-order quantification, cannot depend on every value in a full binary tree. From this it will follow that DDL is less expressive than DL. The kernel of the proof presented here can already be found in [HR].

2. Syntax and Semantics

We give a brief description of the syntax and semantics of DL and DDL. The reader is referred to [Har] for more details.

Syntax: Just as in first-order predicate calculus, we have predicate symbols P, Q, ... and function symbols f, g, ..., each with an associated arity, variables x, y, z, x_0, x_1, ..., and logical symbols $\exists$, $\neg$, $\vee$, (, and). DL also uses a few special symbols in programs, namely =, ;, *, $\cup$, ?, and $\Diamond$ (pronounce "diamond").

Terms are formed exactly as in first-order predicate calculus. Formulas and programs are defined inductively.

(a) Any formula of first-order predicate calculus is a formula.

(b) <variable> = <term> is a (basic) program.

(c) If p, q are formulas, and a, b are programs, then
$p \vee q$, $\neg p$, $\exists xp$, and $\langle a \rangle p$ are formulas, and
a;b, $a \cup b$, and a^* are programs.

(d) If p is a quantifier-free formula of predicate calculus, p? is a program.

$\forall$ as an abbreviation for $\neg\exists\neg$; similarly, [] (pronounced "box") is an abbreviation for $\neg\Diamond\neg$.

Semantics: A *state* (I,s) consists of two parts: I is a *structure* which consists of a domain, dom(I), and an interpretation of all the function and predicate symbols over this domain, and s is a *valuation* which assigns values in the domain to all the variables.

Given a structure I, ρ_I is a mapping from programs to binary relations on valuations which describes the input-output behavior of programs in structure I, and π_I is a mapping from formulas to sets of valuations, the ones valuations where the formulas is "true". We usually write $(I,s) \models p$ instead of $s \in \pi_I(p)$. We define both ρ_I and π_I inductively.

(a) For p a formula of first-order predicate calculus, $(I,s) \models p$ is defined as usual.

(b) For basic programs of the form x=t, t a term, $\rho_I(x=t) = \{(s,s[x/d])|$ where $d \in \mathrm{dom}(I)$ is the value of the term t in (I,s), and s[x/d] is the valuation such that $s[x/d](y) = s(y)$ if $y \neq x$ and $s[x/d](x) = d\}$

(c) For programs a, b and formula p

$\rho_I(a \cup b) = \rho_I(a) \cup \rho_I(b)$

$\rho_I(a;b) = \rho_I(a) \circ \rho_I(b)$

$\rho_I(a^*) = \cup_{n \geq 0} \rho_I(a^n)$

$\rho_I(p?) = \{(s,s) \mid (I,s) \models p\}$

(d) For formulas p, q and program a

$(I,s) \models \neg p$ iff $(I,s) \not\models p$

$(I,s) \models p \vee q$ iff $(I,s) \models p$ or $(I,s) \models q$

$(I,s) \models \exists x p$ iff for some $d \in \mathrm{dom}(I)$ $(I,s[x/d]) \models p$

$(I,s) \models \langle a \rangle p$ iff for some t with $(s,t) \in \rho_I(a)$ $(I,t) \models p$.

Nondeterminism occurs in DL through the constructs * and ∪. We can eliminate the nondeterminism by allowing * and ∪ to appear only in the contexts

$p?;a \cup \neg p?;b$ and

$(p?;a)^*;\neg p?$,

which we abbreviate respectively as **if** p **then** a **else** b **fi** and **while** p **do** a **od**. We leave it to the reader to check that this restriction leaves us with a deterministic set of programs. The restricted language is called DDL.

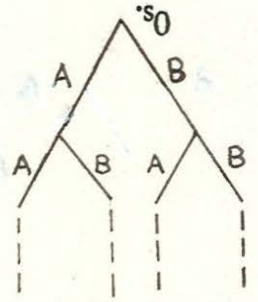
3. Motivation from the Propositional Case

This section reviews results from [HR] which provide motivation for the main result in this paper. While it does provide insight into the ideas, it can be skipped without loss of continuity. We presume familiarity with the relevant definitions (see [HR] for more details).

A *tree model* is one whose graph is a tree. A tree is said to be *sparse* if for some polynomial F, there are $< F(k)$ nodes at depth k. In [HR] it was shown that the truth of a formula of (strict) deterministic PDL depends only on a sparse set of nodes of any tree model. In particular, the proof of Theorem 4.12 of [HR] can be easily modified to show the following:

Theorem: Suppose $M = (S,\pi,\rho)$ is a tree model (where π assigns meaning to the primitive formulas, and ρ assigns meaning to the primitive programs) and $M,s_0 \models q$, where q is an SDPDL formula. Then there exists a sparse subtree $S_q \subseteq S$, such that for any $M' = (S,\pi',\rho)$ such that π' and π agree on S_q (i.e. $\pi|S_q = \pi'|S_q$) we have $M',s_0 \models q$.

As a straightforward application of this theorem, we can show that SDPDL is less expressive than PDL. Let p be the formula $[(A\cup B)^*]P$, and let $M = (S,\pi,\rho)$ be the binary tree model pictured below, with P true at every state:



Clearly $M,s_0 \models p$. But suppose p was equivalent to some SDPDL formula q. Then let $S_q \subseteq S$ be the sparse subtree of the previous theorem. Let $\pi'(P) = S_q$, and let $M' = (S,\pi',\rho)$. Thus $\pi'|S_q = \pi|S_q$, so by the theorem, $M',s_0 \models q$. But since P is not true at every state in M', we have $M',s_0 \models \neg p$, contradicting the equivalence of p and q.

The point here is that we can always "fool" a deterministic formula q which is supposed to be equivalent to $[(A\cup B)^*]P$ at some state where q did not look.

4. DDL is less expressive than DL

By analogy to the formula $[(A \cup B)^*]P$ of the previous section, we consider the formula $[(x_0{=}f(x_0) \cup x_0{=}g(x_0))^*]P(x_0)$, which we call p_0. We will prove the following

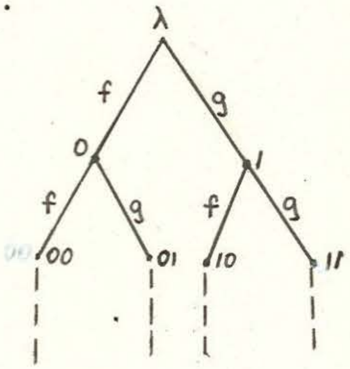
Theorem: p_0 is not equivalent to any formula of DDL

Corollary: DDL is less expressive than DL.

The rest of this paper is devoted to proving this theorem. The structures that we use are analogous to the complete binary trees of the previous section.

Definition: Let $\Sigma=\{0,1\}$ and let $S \subseteq \Sigma^*$. Then I_S is the structure with $\mathrm{dom}(I_S) = \Sigma^*$ and f, g, P interpreted as $f(w) = w0$, $g(w) = w1$, and $P(w)$ iff $w \notin S$. (Thus $P(w)$ holds iff w is *not* an element of S). All other functions and predicates in I_S are trivial; that is, the functions are projections on the first variable, and the predicates are identically true. In most of our applications below, we will take S to be finite. If S is the singleton consisting of $w \in \Sigma^*$, we write I_w instead of $I_{\{w\}}$.

We can think of I_S as the tree pictured below, where f means "go right" and g means "go left".



We will also consider auxiliary structures which look like countably many trees of the form I_S. More formally, for $S^* = (S_0,S_1,S_2,\ldots)$ define the structure I_{S^*} to have domain $\Sigma^* \times \mathbb{N}$, with $f((w,i))=(w0,i)$, $g((w,i))=(w1,i)$, and $P((w,i))$ iff $w \notin S_i$. Again all other functions and predicates are trivial.

Notation: We use s, s_0, s_1, ... for the valuations on structures of the form I_S, and t, t_0, t_1, ... for valuations on I_{S^*}. We reserve s_0 for the valuation which takes x_i to λ (the root of the tree), and t_0 for the valuation which takes x_i to (λ,i) (the root of the i^{th} tree).

Definition: Given $R \subseteq \Sigma^*$ (resp. $\Sigma^* \times \mathbb{N}$), let $R(n) = \{w \in R | \, |w| \leq n\}$ (resp. $\{(w,i) \in R | \, |w| \leq n\}$). R is said to be *sparse* if there is a polynomial F such that for all n, $|R(n)| \leq F(n)$.

Our method of proving the theorem is to show that for a DDL formula q, there is a sparse set $R_q \subseteq \Sigma^*$, such that if $q \equiv p_0$, then for all $w \notin R_q$, $(I_w,s_0) \models q$, which of course is a contradiction since $(I_w,s_0) \models \neg p_0$.

We proceed through a series of four lemmas. The first shows the strong connections between the structures I_S and I_{S^*}. The idea is that since we do not have equality in the language, all that matters about a state is the locations in the tree(s) where P does not hold relative to the values given to the variables.

Definition: For $S \subseteq \Sigma^*$ and $w \in \Sigma^*$, let $w \backslash S = \{u | \, wu \in S\}$.

Lemma 1: Let $S^* = (S_0,S_1,S_2,\ldots)$, $T^* = (T_0,T_1,T_2,\ldots)$. Let s be a valuation on I_S with $s(x_i) = u_i$, let t_1 be a valuation on I_{S^*} with $t_1(x_i) = (v_i,j_i)$, and let t_2

be a valuation on I_{T*} with $t_2(x_i) = (w_i,k_i)$. Let q be a quantifier-free DDL formula with free variables $\subseteq \{x_0,\ldots,x_k\}$. Suppose $u_i\backslash S = v_i\backslash S_{j_i}$ (resp. $v_i\backslash S_{j_i} = w_i\backslash T_{k_i}$) for $i = 0,\ldots,k$. Then $(I_S,s) \models q$ iff $(I_{S*},t_1) \models q$ (resp. $(I_{S*},t_1) \models q$ iff $(I_{T*},t_2) \models q$).

Proof: By a straightforward induction on the structure of formulas. ∎

Next, we want to show that without loss of generality we can make certain special assumptions about DDL formulas interpreted over these structures. The first is that no function and predicate symbols appear other than f, g, and P. This follows immediately the fact that all other functions and predicates get a trivial interpretation in the structures I_S. Secondly we can assume that all basic assignments in programs (the ones of the form x := term) are actually of the form $x_i := x_j$, $x_i := f(x_i)$, or $x_i := g(x_i)$, since any assignment of the form x_i := term can be replaced by a sequence of the assignments above; for example, $x_1 := f(g(x_2))$ can be replaced by $(x_1 := x_2; x_1 := f(x_1); x_1 := f(x_1))$. Thirdly, we can assume that the argument to predicate P is always a variable x_i; for example **while** $P(f(x_1))$ **do** a **od** can be replaced by $x_2 := f(x_1)$; **while** $P(x_2)$ **do** a; $x_2 := x_1$ **od**. (We must be careful to ensure that x_2 is a fresh variable which does not appear anywhere else in the DDL formula.) And finally, we can assume that the formula is of the form $Q_1x_1Q_2x_2\ldots Q_kx_kq$, where q is a quantifier-free DDL formula, and Q_i is either $\forall$ or $\exists$. This follows immediately from the following

Lemma 2: If y does not appear in the DDL program a and q is a DDL formula, then

(a) $\forall y\langle a\rangle q \equiv \langle a\rangle\forall yq$

(b) $\exists y\langle a\rangle q \equiv \langle a\rangle\exists yq$

Proof: We will prove part (a). The proof of (b) is similar and does not require a to be deterministic.

First note that if y does not appear in a, then a does not affect the value of y during its computation. Thus if I is any structure, and s_1, s_2 are any valuations, then for any $d \in dom(I)$, $(s_1,s_2) \in \rho_I(a)$ iff $(s_1[y/d],s_2[y/d]) \in \rho_I(a)$. Moreover, since a is *deterministic*, if $(s_1[y/d],s_3) \in \rho_I(a)$, then we must have $s_3 = s_2[y/d]$. Thus we get

$(I,s) \models \langle a \rangle \forall y q$

iff for some t, $(s,t) \in \rho_I(a)$ and $(I,t) \models \forall y q$

iff for some t, $(s,t) \in \rho_I(a)$ and for all $d \in dom(I)$, $(I,t[y/d]) \models q$

iff for all $d \in dom(I)$, $(I,s[y/d]) \models \langle a \rangle q$

iff $(I,s) \models \forall y \langle a \rangle q$ ∎

For the next two lemmas we will concentrate on quantifier-free DDL formulas.

Definition: $R \subseteq \Sigma^*$ is said to *set q for S at s* iff for all S′ such that $R \cap S = R \cap S'$, we have $(I_S,s) \models q$ iff $(I_{S'},s) \models q$. Thus R sets q for S at s if all that matters for the truth of q in (I_S,s) is the value of P on R. We can similarly define what it means for $R \subseteq \Sigma^* \times \mathbb{N}$ to set q for S^* at t.

Definition: Let Φ_m be the set of sequences $S^* = (S_0, S_1, S_2, \ldots)$ such that each S_i is empty or is the singleton $\{w_i\}$ where $|w_i| \leq m$.

Lemma 3: Let q be a quantifier-free formula of DDL.

(a) For all finite $S \subseteq \Sigma^*$, and all valuations s, there is a sparse set $R_{q,S,s}$ which sets q for S at s. Moreover, $|R_{q,\emptyset,s}(n)| \leq c_q n$ and $|R_{q,w,s}(n)| \leq c_q(|w|+2)^k n$, where c_q is a constant depending only on the formula q, and k is the number of free variables in q.

(b) For all sequences $S^* = (S_0, S_1, S_2, \ldots)$, where each $S_i \subseteq \Sigma^*$ is finite, there is a sparse set $R_{q,S^*,t}$ which sets q for S^* at t. Moreover, $|R_{q,(\emptyset,\emptyset,\ldots),s}(n)| \leq c_q n$ and for $S^* \in \Phi_m$, $|R_{q,S^*,s}(n)| \leq c_q(m+2)^k n$, where c_q and k are as above.

Proof: Deferred to the appendix.

Note that from Lemma 3(a) it already follows that p_0 is not equivalent to any quantifier-free DDL formula q. For if q is equivalent to p_0, then $(I_\emptyset, s_0) \models q$. Let $w \in \Sigma^*-R_{q,\emptyset,s_0}$ (such a w exists since $R_{q,\emptyset,s_0}$ is sparse). Then by Lemma 3(a), $(I_w, s_0) \models q$. But $(I_w, s_0) \models \neg p_0$, and this contradicts the equivalence of q and p_0.

Definition: For a quantifier-free DDL formula q, let $R_{q,n} = \cup_{S^* \in \Phi_n} R_{q,S^*,t_0}$, and let $R_q = \{w \mid (w,i) \in R_{q,|w|}\}$.

Lemma 4: R_q is sparse.

Proof: It is sufficient to find a polynomial F_q such that $|R_{q,n}(n)| \leq F_q(n)$, since clearly $|R_q(n)| \leq |R_{q,n}(n)|$. We will do this for the case that the free variables of q are x_0, x_1, x_2. It will be clear that the proof extends to q's with arbitrarily many free variables.

Note that from Lemma 1, it follows that since the free variables of q are x_0, x_1, and x_2, R_{q,S^*,t_0} depends only on S_0, S_1, and S_2. That is, given another sequence $T^* = (T_0, T_1, T_2, \ldots)$, if $S_i = T_i$ for $i = 0,1,2$, then $R_{q,S^*,t_0} = R_{q,T^*,t_0}$. Thus we need only concern ourselves with elements of Φ_n of the form $(S_0, S_1, S_2, \emptyset, \emptyset, \ldots)$, which we abbreviate as (S_0, S_1, S_2).

Call this subset $\Phi_n(2)$.

Define an equivalence relation on $\Phi_n(2)$ via $(S_0,S_1,S_2) \equiv (T_0,T_1,T_2)$ iff $R_{q,(S_0,S_1,S_2),t_0} = R_{q,(T_0,T_1,T_2),t_0}$. We will show that there are less than $G_q(n)$ equivalence classes, for some polynomial G_q depending on q. Then we can take $F_q(n)$ to be $c_q(n+2)^3nG_q(n)$, since by Lemma 3(b) $|R_{q,(S_0,S_1,S_2),t_0}| \leq c_q(n+2)^3n$ for $(S_0,S_1,S_2) \in \Phi_n(2)$.

We now proceed to count equivalence classes. Note that $(w,\emptyset,\emptyset) \equiv (\emptyset,\emptyset,\emptyset)$ unless $(w,0) \in R_{q,(\emptyset,\emptyset,\emptyset),t_0}$. Similarly for $(\emptyset,w,\emptyset)$ and $(\emptyset,\emptyset,w)$.
Thus there are at most $1+nc_q$ equivalence classes of the form $(w,\emptyset,\emptyset)$, $(\emptyset,w,\emptyset)$, or $(\emptyset,\emptyset,w)$ in $\Phi_n(2)$, since $|R_{q,(\emptyset,\emptyset,\emptyset),t_0}(n)| \leq c_qn$ by Lemma 3(b).

Now to get an equivalence class of the form $(w_1,w_2,\emptyset)$ distinct from those of the form $(w,\emptyset,\emptyset)$, $(\emptyset,w,\emptyset)$, or $(\emptyset,\emptyset,w)$, we must either have

(a) $(w_1,1) \in R_{q,(\emptyset,\emptyset,\emptyset),t_0}$ and $(w_2,2) \in R_{q,(w_1,\emptyset,\emptyset),t_0}$, or

(b) $(w_2,2) \in R_{q,(\emptyset,\emptyset,\emptyset),t_0}$ and $(w_1,1) \in R_{q,(\emptyset,w_2,\emptyset),t_0}$, or

(c) $(w_1,1), (w_2,2) \in R_{q,(\emptyset,\emptyset,\emptyset),t_0}$.

Again, since $|w_1|, |w_2| \leq n$, it is easy to check, using Lemma 3(b), that there $\leq c_q^2(n+2)^3n^2$ new equivalence classes satisfying each of conditions (a) and (b), and $\leq c_q^2n^2$ satisfying condition (c). Thus we get $O(n^5)$ new equivalences classes of the form $(w_1,w_2,\emptyset)$, $(w_1,\emptyset,w_2)$, or $(\emptyset,w_1,w_2)$ for $w_1, w_2 \in \Sigma^*(n)$. A similar analysis can be used to show we get $O(n^9)$ new equivalence classes of the form (w_1,w_2,w_3). Thus we get polynomially many equivalence classes, as desired. (Similar arguments show that in general, the polynomial G_q will have degree $O(k^2)$, where k is the number of variables in q.) ∎

We are (finally!) ready to prove our theorem. The proof is by contradiction. Suppose the DDL formula q is equivalent to p_0. By Lemma 2 we can assume q is of the form $Q_1x_1...Q_kx_kq'$, where q' is a quantifier-free formula of DDL. We will assume for ease of exposition that q is of the form $\forall x_1 \exists x_2 q'$, where the free variables of q' are x_0, x_1, and x_2, but it should be clear that the proof will work for arbitrary sequences of quantifiers and for a q' with arbitrarily many free variables.

Choose $w \in \Sigma^* - R_{q'}$ (we can do this since $R_{q'}$ is sparse). Then

$(I_{(\varnothing,w,w)},t_0) \models p_0$

$\Rightarrow$ $(I_{(\varnothing,w,w)},t_0) \models q$ (since q is equivalent to p_0)

$\Rightarrow$ $(I_{(\varnothing,w,w)},t_0) \models \forall x_1 \exists x_2 q'$

$\Rightarrow$ $\forall (w_1,i_1) \exists (w_2,i_2)(I_{(\varnothing,w,w)},t_0[x_1/(w_1,i_1),x_2/(w_2,i_2)]) \models q'$

$\Rightarrow$ $\forall w_1 \exists w_2 (I_{(\varnothing,w_1\backslash w,w_2\backslash w)},t_0) \models q'$ (this follows from Lemma 1)

$\Rightarrow$ $\forall w_1 \exists w_2 (I_{(w,w_1\backslash w,w_2\backslash w)},t_0) \models q'$

(since $(\varnothing,w_1\backslash w,w_2\backslash w) \in \Phi_{|w|}(2)$ and $w \notin R_{q'}$, so $(w,0) \notin R_{q',|w|}$)

$\Rightarrow$ $\forall w_1 \exists w_2 (I_w,s_0[x_1/w_1,x_2/w_2]) \models q'$ (by Lemma 1 again)

$\Rightarrow$ $(I_w,s_0) \models \forall x_1 \exists x_2 q'$

$\Rightarrow$ $(I_w,s_0) \models q$

$\Rightarrow$ $(I_w,s_0) \models p_0$ (since q is equivalent to p_0).

But this is clearly a contradiction, since $(I_w,s_0) \models \neg p_0$.

This completes the proof of the theorem. ∎

Appendix

Proof of Lemma 3:

We will prove part (a). The proof of (b) is similar. We first need to simplify the form of quantifier-free DDL formulas.

Definition: We call a quantifier-free formula *elementary* if it is of the form $\{a_1\}_1 \ldots \{a_k\}_k p$, where $\{\}_i$ is either $\langle\rangle$ or $[]$ and p is either $P(x)$ or $\neg P(x)$.

Lemma: Any quantifier-free DDL formula can be written in "disjunctive normal form", i.e. as a disjunction of a conjunction of elementary formulas.

Proof: The proof is by induction on the structure of formulas, and follows immediately from the fact that for DDL programs a,

$\{a\}(p \wedge q) \equiv \{a\}p \wedge \{a\}q$ and

$\{a\}(p \vee q) \equiv \{a\}p \vee \{a\}q.$ ∎

It clearly suffices to prove Lemma 3(a) for elementary formulas, since for an arbitrary quantifier-free DDL formula q, we can take $R_{q,S,s} = \cup_i R_{q_i,S,s}$, where the q_i range over the elementary formulas that appear in the "disjunctive normal form" of the previous lemma.

Now the truth of an elementary formula $q = \{a_1\} \ldots \{a_m\}p$ with free variables $\subseteq \{x_1,\ldots,x_k\}$, in (I_S,s) depends only on the truth values of P at those values in Σ^* assigned to $x_1,\ldots,x_k$ as we run the program $a_1;\ldots;a_m$ starting in state (I_S,s) (remember we are assuming that all tests are of the form $P(x_i)$ or $\neg P(x_i)$).

Since $a_1;\ldots;a_m$ is a regular program, it can be represented as a finite flowchart. We can construct a finite state register machine with an oracle for S which acts as an interpreter for this program. The number of states in the machine depends only on $a_1;...;a_m$ and hence on q. The machine will have k registers, one for each of the variables appearing in the program. The registers are initialized to $s(x_1),...,s(x_k)$. In what follows, we will deliberately confuse x_i with "the contents of register i".

Claim: The machine only consults the oracle on a sparse set of values.

Once we prove this claim we are done. We simply take this sparse set to be $R_{q,S,s}$ and note that if S′ is any set with $S' \cap R_{q,S,s} = S \cap R_{q,S,s}$ the machine with oracle S′ would go through the same sequence of states as the one with oracle S. Thus $(I_S,s) \models q$ iff $(I_{S'},s) \models q$. (The formal proof of this last statement is a messy but straightforward induction on the structure of q.)

To prove the claim, we first define an equivalence relation on tuples $(w_1,...,w_k)$ via $(w_1,...,w_k) \equiv (u_1,...u_k)$ iff $w_i \backslash S = u_i \backslash S$ for $i = 1,...,k$. Note that since S is finite, $w_i \backslash S = \emptyset$ for all but finitely many $w \in \Sigma^*$, so there are only finitely many equivalence classes. Note also that the equivalence class and the state of the machine completely determine the sequence of basic instructions (those of the form $x_i := x_j$, $x_i := f(x_i)$, or $x_i := g(x_i)$) performed by the machine, since for any two equivalent tuples the oracle for S always gives the same answers at every step in the computation.

Thus, if the machine does not halt after a finite number of steps, there must be two distinct times in its computation when it is in the same state and the contents of the registers are in the same equivalence class. Hence, if the machine does not halt after a finite number of steps, it must repeat the same sequence of basic instructions over and over again; i.e. the instruction sequence is of the form $a;b^\omega$, where a and b are sequences of basic instructions.

Suppose the contents of the registers at some point in time are $(x_1^0,\ldots,x_k^0)$ and let $(x_1^m,\ldots,x_k^m)$ be the contents of the registers after we run b^m from this point. It is easy to see that there exists $j_i \leq k$ and $w_i \in \Sigma^*$ (which depend only on b) such that

$$x_i^1 = x_{j_i}^0 w_i \text{ for } i = 1,\ldots,k.$$

Now suppose $j_1,\ldots,j_k$ are all distinct. (The argument for the general case is similar.) Then it is easy to see that for some $u_i \in \{w_1,\ldots,w_k\}^{k!}$,

$$x_i^{k!} = x_i^0 u_i, \text{ and thus}$$
$$x_i^{k!h} = x_i^0 (u_i)^h.$$

If we let $y_1,\ldots,y_N$ be the contents of the registers at all times as we run $b^{k!}$ starting from $(x_0^0,\ldots x_k^0)$, we can again find $v_1,\ldots,v_N \in \Sigma^*$, $m_1,\ldots,m_N \leq k$ such that

$$y_i = x_{m_i}^0 v_i \quad i = 1,\ldots,N.$$

Thus, at any time as we run b^ω on initial input $(x_0^0,\ldots x_k^0)$ the contents of the registers $\subseteq \{x_{m_i}^0 (u_i)^h v_i | \; i = 1,\ldots,N, \; h \geq 0\}$. Call this set A. Note that for any n, A has at most N elements of length n (at most one of the form $x_{m_i}^0 (u_i)^h v_i$ for each $i = 1,\ldots,N$). If we let B = {contests of the registers at all times as we run a on input $(s(x_0),\ldots,s(x_k))$}, let $(x_0^0,\ldots,x_k^0)$ be the contents of the registers after running program a on input $(s(x_0),\ldots,s(x_k))$, and take $R_{q,S,s} = A \cup B$, we are done.

Note that $N \leq$ the number of steps (assignments) performed by $b^{k!} = k! \times$ (the number of steps performed by program b), while $|B| \leq$ the number of steps performed by program a. Moreover, the number of steps performed by a;b is $\leq$ the number of equivalence classes. If $S = \emptyset$, there is only one equivalence class, and if $S = \{w\}$, there are at most $(|w|+2)^k$ equivalence classes, since there are $|w|+2$ values for $u\backslash w$. These observations give us the second half of Lemma 3(a). ∎

Acknowledgments

I would like to thank Albert Meyer for suggesting the problem, Karl Winklmann for his initial insights, and Neil Immerman and Mike Sipser for pointing out why some of my ideas led to dead ends.

This research was partially supported by a grant from the National Sciences and Engineering Research Council of Canada and by NSF Grant MCS80-10707.

References

[HR] J. Y. Halpern and J. Reif, The Propositional Dynamic Logic of Deterministic, Well-Structured Programs, MIT/LCS/TM-198, March, 1981; to appear *in* "Proceedings of the 22nd Annual Symposium on the Foundations of Computer Science", 1981.

[Har] D. Harel, *First-Order Dynamic Logic*, Lecture Notes in Computer Science, 68, Springer-Verlag, N.Y., 1979.

[MT] A. R. Meyer and J. Tiuryn, The equivalence of several logics of programs, to appear.

[MW] A. R. Meyer and K. Winklmann, On the expressive power of dynamic logic, MIT/LCS/TM-157, February, 1980. A preliminary report appears in "Proceedings of the 11th Annual ACM Conference on Theory of Computing", May, 1979.