Reprinted from INFORMATION AND COMPUTATION
Vol. 81, No. 3, June 1989
Printed in Belgium

An Arithmetical Hierarchy in Propositional Dynamic Logic

SORIN ISTRAIL

Department of Mathematics, Wesleyan University, Middletown, Connecticut 06457

We prove that any alternation of modalities in PDL adds to its expressive power. The proof uses Turing machine models where *PDL* formulas define the arithmetical hierarchy of sets. As a by-product, we obtain a theorem of Berman and Paterson.

1. INTRODUCTION

Propositional dynamic logic (PDL), as introduced by Fischer and Ladner (1979) is the propositional part of the dynamic logic of Pratt (1976). Our aim in this paper is to answer a basic question: Does every alternation of modalities add to the expressive power of the logic? We give a positive answer to the question by building models where modalities behave (somewhat) like quantifiers in first-order logic.

The structure of the paper is as follows. The remainder of this section contains some basic definitions. Section 2 introduces our Turing machine models. In Section 3 we present an arithmetical hierarchy of PDL. Section 4 contains conclusions and open problems.

1.1. *Propositional Dynamic Logic*

We briefly define the syntax and semantics of PDL. We refer to (Harel, 1984) for basic facts concerning PDL theory.

Syntax. The basic objects are two sets of primitives: Φ_0 the *basic formulas*, and Δ_0 the *basic programs*.

Programs and *formulas* are defined inductively.

1. θ (the null program) and the basic programs are programs.

2. if α and β are programs and p is a formula then $(\alpha; \beta)$, $(\alpha \cup \beta)$, $(p?)$, (α^*) are programs.

3. *true*, *false*, and the basic formulas are formulas.

0890-5401/89 $3.00

4. if p and q are formulas and α is a program then $p \wedge q$, $\neg p$, $\langle\alpha\rangle p$ are formulas.

The letters p, q, r, ..., are reserved as metavariables for formulas and α, β, γ, ..., as metavariables for programs.

Semantics. A *structure* $\mathscr{A}$ of PDL is a triple $\mathscr{A} = (W, \pi, \rho)$, where $\pi: \Phi_0 \to 2^W$, $\rho: \Delta_0 \to 2^{W \times W}$. The extension of ρ and π to programs and formulas is as follows:

1. $\rho(\theta) = \varnothing$
2. $\rho(\alpha; \beta) = \rho(\alpha) \circ \rho(\beta)$
3. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
4. $\rho(\alpha^*) = (\rho(\alpha))^*$
5. $\rho(p?) = \{(w, w) | w \in \pi(p)\}$
6. $\pi(true) = W$
7. $\pi(\text{false}) = \varnothing$
8. $\pi(p \vee q) = \pi(p) \cup \pi(q)$
9. $\pi(\neg p) = W - \pi(p)$
10. $\pi(\langle\alpha\rangle p) = \{w \in W | \exists v: ((w, v) \in \rho(\alpha) \wedge v \in \pi(p))\}$.

We write $\mathscr{A}, w \models p$ just in case $w \in \pi(p)$. We say that p and q are *equivalent* in $\mathscr{A}$ ($p \equiv q$ in $\mathscr{A}$) if for all $w \in W$,

$$\mathscr{A}, w \models p \quad \text{iff} \quad \mathscr{A}, w \models q;$$

p and q are *equivalent* iff they are equivalent in all structures.

2. Turing Machine Models

Our intention is to introduce a special family of models having as universes finitely generated free monoids. The basic programs and formulas abstract the elementary actions and tests performed by a Turing machine, viewed as a rewriting device.

Let V be a finite alphabet, V^* be the free monoid generated by V and λ be the empty word. Suppose that V contains all the tape symbols and all the state symbols of a certain Turing machine. The machine can be easily seen as a rewriting device. (See (Salomaa, 1973) for details.)

Consider the following examples of transitions of a Turing machine:

(1) $wxqw' \vdash wq'x'w'$, a move to the left

(2) $wqx\# \vdash wx'q'y\#$, a move to the right,

where w, w' are words over V; x, x' are symbols, q, q' are state symbols, and $\#$ is the boundary marker, all elements of V.

The analysis that follows will identify a natural way to construct the models by focusing on the elementary components of the computation process described as a rewriting system. The two basic components that we identify are *actions* and *tests*. They will be the meaning of the syntactic primitives *basic programs* and *basic formulas*, respectively.

Regarding *actions*, we remark that basically there are symbol transformation ($x \rightsquigarrow q', q \rightsquigarrow x'$) and symbol construction (extending the workspace) ($\lambda \rightsquigarrow y$). Concerning *tests*, the machine tests membership in some simple sets: V^*wqV^* in (1) and $V^*qx\#$ in (2). We might choose various classes of test sets. We can take (a) the simplest class containing the above sets. The least class containing the alphabet symbols and closed under $\cdot$, $\cup$, $*$, i.e., is class of *regular sets*. On the other hand, (b) we can consider the most general class for which the tests are effective, namely the *recursive sets*. It happens that both these classes are suitable for our separation results.

In our development we are going to consider option (b) the recursive sets; the results obtained by considering option (a) are similar. Let us remark that the "programs" we can write using our actions and tests are lightly more general that the ones obtained by the exact simulation of Turing machines, but certainly do not lead outside the computational power of the Turing machines.

2.1. *Defining the Models*

Let $N = \{x_1, x_2, ...,\}$ be an infinite alphabet:

1. The set of basic formulas Φ_0 is the collection of all formulas p_R such that R is a recursive set over a finite subset of N.

2. The set of basic programs Δ_0 consists of all programs of the forms: $x_i \rightsquigarrow x_j$ $i, j \geqslant 1$, or $\lambda \rightsquigarrow x_i$ $i \geqslant 1$.

A *Turing machine model* is defined to be $\mathscr{A} = (K^*, \pi, \rho)$, where K is a finite subset of N and π, ρ are given for Φ_0 and respectively Δ_0 as follows:

$$\pi(p_R) = \text{if } R \subset K^* \text{ then } R \text{ else } \varnothing$$
$$\rho(x_i \rightsquigarrow x_j) = \{(w, w') \mid w, w' \in K^*, w = w_1 x_i w_2, w' = w_1 x_j w_2\}$$
$$\rho(\lambda \rightsquigarrow x_i) = \{(w, w') \mid w, w' \in K^*, w = w_1 w_2, w' = w_1 x_i w_2\}.$$

AN EXAMPLE. We are going to define a formula and explain the behaviour of the programs involved. Let V be a finite subset of N and $\bar{V} = \{\bar{a} \mid a \in V\}$. Let α, β, γ be the programs:

$$\alpha = (\lambda \rightsquigarrow \bar{\#}); (\bigcup_{a \in V} (\lambda \rightsquigarrow \bar{a}))^*$$
$$\beta_i = ((\#V^*)^{i-1} \bar{\#} \bar{V}^* (\#V^*)^{m-i})?$$
$$\gamma = (\bigcup_{a \in V} (\bar{a} \rightsquigarrow a))^*$$

and the formula q_R, where $R = (\#V^*)^m$.

The program α on a certain input word u non-deterministically inserts one $\bar{\#}$ symbol and several $\bar{a}$ for $a \in V$. The program β_i is a test; the intention is to apply β_i to the output provided by α and to control the position where α inserts. The outputs of α that will pass the test β_i are the ones that have the barred symbols in the ith group $\#V^*$. The role of γ is to get rid of the bars. The formula q_R is true only for bar-free symbols. Consider the formula $p = \langle \alpha; \beta_i; \gamma \rangle q_R$. We claim that

$$\mathscr{A}, u \models p \quad \text{iff} \quad u \in (\#V^*)^{m-1}.$$

Indeed,

$$\mathscr{A}, u \models p \quad \text{iff} \quad u \alpha u_1, u_1 \beta_i u_1, u_1 \gamma u_2 \text{ and } \mathscr{A}, u_2 \models q_R$$

iff

$$u_2 = \#w_1 \# \cdots \#w_m \text{ and}$$
$$u_1 = \#w_1 \# \cdots \#w_{i-1} \bar{\#} \bar{w}_i \#w_{i+1} \cdots \#w_m \text{ and}$$
$$u = \#w_1 \# \cdots \#w_{i-1} \#w_{i+1} \# \cdots \#w_m \text{ iff}$$
$$u \in (\#V^*)^{m-1}.$$

3. The Arithmetical Hierarchy

We analyse the expressive power of PDL in the Turing machine models defined in the previous section. It turns out that the PDL formulas define exactly the arithmetical sets. The following defines an arithmetical-like hierarchy within the PDL formulas. The result of Theorem 1 shows that the resulting classes form a strict hierarchy, that is, alternation of modalities adds to the expressive power of PDL.

Definition 1. We define the classes of formulas $\sum_n^0$ and $\prod_n^0$, $n \geqslant 0$ as follows:

1. $\sum_0^{\text{PDL}} = \prod_0^{\text{PDL}} =$ the class of PDL formulas which are modality-free.

2. $\sum_{n+1}^{\text{PDL}} =$ the class of formulas of the form $\langle \alpha \rangle p$, where α is a modality-free program, and p is in $\prod_n^{\text{PDL}}$, $n \geqslant 0$.

3. $\prod_{n+1}^{\mathrm{PDL}}$ = the class of formulas of the form $[\alpha]p$, where α is a modality-free program, and p is in $\sum_n^{\mathrm{PDL}}$, $n \geqslant 0$.

We are referring to (Rogers, 1967) for the definition of the arithmetical hierarchy of sets. A first-order formula (predicate) defining a set in a class $\mathscr{C}$ of the arithmetical hierarchy is said a $\mathscr{C}$-formula (predicate). For an n-ary predicate δ defined on $(V^*)^n$, consider its *encoding* $L_\delta = \{\sharp w_1 \sharp \cdots \sharp w_n \mid \delta(w_1, ..., w_n)\}$, where $\sharp \notin V$. It is easy to remark the relation between δ and L_δ.

LEMMA 1. *δ is a $\sum_n^0$ ($\prod_n^0$) predicate iff L_δ belongs to $\sum_n^0$ ($\prod_n^0$).*

The next lemma gives the starting point for defining the arithmetical sets by PDL formulas. A program (formula) is called *modality-free* if it contains no $\diamond$ or $\square$ symbols.

LEMMA 2. *For every modality-free program α, its input–output prddicate $R_\alpha(X, Y)$ defined by*

$$R_\alpha(w, w') \qquad \textit{iff} \quad w\alpha w'$$

is recursive, i.e., a $\sum_0^0$-predicate.

Proof. We can observe that the program rules are "non-decreasing," so we can accept L_{R_α} with a Turing machine in linear space. The simulation of α can be done backwards. To see if a word w' is in L_{R_α} we perform on w' the reverse transformations given by α and then check if the result is w. ∎

LEMMA 3. *For every r.e. set L, (i.e., $L \in \sum_1^0$) there exist a $\sum_1^{\mathrm{PDL}}$-formula p, ($p = \langle\alpha\rangle q$, having α and q modality-free), and a structure $\mathscr{A}$, such that in $\mathscr{A}$, $\pi(p) = L$. Conversely, every $\sum_1^{\mathrm{PDL}}$-formula defines in every structure an r.e. set.*

Proof. Let T be a Turing machine accepting L. T has V as tape alphabet, Q as the state set, $q_0 \in Q$ as the initial state, $\# \in Q$ as the initial state, $\# \in V$ as the boundary marker, Q_1 as the final state set, and F as the set of rules.

Members of F are rules of the forms:

1. $qa \to q'b$
2. $qac \to aq'c$
3. $qa\# \to aq'd\#$
4. $cqa \to q'ca$
5. $\#qa \to \#q'da$.

The language accepted by T is $L(T) = \{w \in V^* \mid \# q_0 w \# \Rightarrow^* \# w_1 q_1 w_2 \#, q_1 \in Q_1, w_1, w_2 \in (V - \{\#\})^*\}$.

The structure, we are interested in, is $\mathscr{A} = (V^*, \pi, \rho)$. For each rule r, we shall associate the programs $\alpha_r, \beta_r, \gamma_r$ such that $\alpha_r; \beta_r; \gamma_r$ will implement r. We consider case 3, and denote the rule $r3$; the other cases are similar. Let

$$\alpha_{r3} = q \rightsquigarrow \bar{a}; a \rightsquigarrow q'; \# \rightsquigarrow \bar{d}; \lambda \rightsquigarrow \bar{\#}$$

$$\beta_{r3} = (V^* \bar{a} \bar{q} \bar{d} \bar{\#} V^*)?$$

$$\gamma_{r3} = \bar{a} \rightsquigarrow a; \bar{q}' \rightsquigarrow q'; \bar{d} \rightsquigarrow d; \bar{\#} \rightsquigarrow \#.$$

Consider now the formula $p = \langle ((\bigcup_{r \in F} (\alpha_r; \beta_r; \gamma_r))^*; q_R? \rangle \; true$, where $R = \#(V - \{\#\})^* Q_1 (V - \{\#\})^* \#$.

It is not difficult to see that in $\mathscr{A}$,

$$w \models p \quad \text{iff} \quad w \in L(T).$$

For the $\prod_1^0$ case, replace $\diamond$ with $\square$ and *true* with *false* in p.

The proof of the converse is trivial. ∎

COROLLARY 1 (Berman–Paterson theorem). *PDL is weaker without tests.*

Proof. PDL without tests, denoted in (Berman, Paterson, 1981), PDL_0, is PDL restricted to ?-free formulas. It is easy to observe that in our models such formulas define only regular sets. On the other hand, with tests, the expressive power increases, by Lemma 3, to $\sum_1^0 \cup \prod_1^0$. ∎

LEMMA 4 (The quantifier–modality correspondence). *Let V be a finite alphabet and $L \subseteq V^*$ be a set in $\sum_n^0 (\prod_n^0)$, $n \geqslant 0$. Then there exist a structure $\mathscr{A}$, and $\sum_n^{\mathrm{PDL}} (\prod_n^{\mathrm{PDL}})$-formula p_L such that $\pi_{\mathscr{A}}(p_L) = L$. (I.e., there are modality-free programs $\alpha_1, \ldots, \alpha_n$ such that*

$$w \in L \quad \textit{iff} \quad w \models p_L = \langle \alpha_1 \rangle [\alpha_2] \cdots \{\alpha_n\} p$$

$$(w \in L \quad \textit{iff} \quad w \models p_L = [\alpha_1] \langle \alpha_2 \rangle \cdots \{\alpha_n\} p),$$

where $\{\}$ is $\langle\rangle$ or $[]$ depending on the parity of n.

Proof. Let $\mathscr{A} = ((V \cup \{\sharp\})^*, \pi, \rho)$ and $p_L = \langle \alpha_1 \rangle [\alpha_2] \cdots \{\alpha_n\} p$ be the formula given for $\sum_n^0$; similar notation for $\prod_n^0$.

We prove by induction on n that for every $L \subseteq (V \cup \sharp)^*$, if $L \in \sum_n^0 (\prod_n^0)$ then there exists a formula p_L of the above form, such that $\pi_{\mathscr{A}}(p_L) = L$.

Lemma 3 gives us the case $n = 1$. Indeed, the extension to $L \subseteq (V \cup \sharp)^*$ is trivial.

Assume the lemma for n: for every $L \in \sum_n^0$ ($\prod_n^0$), there exists p_L with $w \models p_L$ iff $w \in L$:

- The case $\exists$. Let $L' \in \sum_{n+1}^0$. Then there is a $\prod_n^0$-predicate $\delta(X_1, X_2)$ such that

$$\delta(X_1) = \exists X_2 \delta(X_1, X_2).$$

Lemma 1 gives us that $L_\delta = \{\sharp w_1 \sharp w_2 \mid \delta(w_1, w_2)\} \in \prod_n^0$. By induction, there exists $p_{L_\delta} = [\alpha_1]\langle\alpha_2\rangle \cdots \{\alpha_n\}\, p$ such that $\pi(p_{L_\delta}) = L_\delta$.

We construct now a formula p_L by using p_{L_δ} and the programs α, β, γ described in the example of Section 2.1. Take $p_L = \langle(\lambda \rightsquigarrow \sharp); \alpha; \beta; \gamma\rangle p_\delta$. Because $\pi(p_{L_\delta}) = \{\sharp w_1 \sharp w_2 \mid \delta(w_1, w_2)\}$, we can get, similarly as in the example,

$$w \models p_L \quad \text{iff} \quad \sharp w \in \{\sharp w_1 \mid \exists w_2 : \delta(w_1, w_2)\}.$$

The programs α, β, γ are modality-free. Therefore, the formula

$$p_L = \langle(\lambda \rightsquigarrow \sharp); \alpha; \beta; \gamma\rangle[\alpha_1]\langle\alpha_2\rangle \cdots \{\alpha_n\}\, p$$

is an alternating formula, of the desired form, for the set L.

- The case $\forall$. Take formula p_L as $[(\lambda \rightsquigarrow \sharp); \alpha; \beta; \gamma; (V \cup \sharp)^*?]p_\delta$. ∎

LEMMA 5. *For any PDL formula p and any structure $\mathscr{A}$, $\pi_{\mathscr{A}}(p)$ is an arithmetic set.*

Proof. We use structural induction on formulas. Define $l(p)$ as follows:

- $l(false) = l(true) = l(p) = 0$, for any basic formula p.
- $l(p \vee q) = 1 + l(p) + l(q)$
- $l(\neg p) = 1 + l(p)$
- $l(\langle\alpha\rangle p) = l([\alpha]\, p) = 1 + l(p) + \sum \{l(q) \mid q \text{ a subprogram in } \alpha\}$.

Let us fix a structure $\mathscr{A}$. We are going to construct for any formula p an arithmetic predicate $F_p(X)$ such that for every w:

$$w \in \pi_{\mathscr{A}}(p) \quad \text{iff} \quad F_p(w).$$

Consider formula p with $l(p) = 0$, and the structure $\mathscr{A}$. We have $F_{\text{true}}(X) = (X = X)$, $F_{\text{false}}(X) = \neg(X = X)$. For a basic formula p defining a regular set R in $\mathscr{A}$, which is ceertainly recursive, take $F_p(X)$ as a recursive predicate describing R.

Consider now $F_p(X)$ defined for any formula p' with $l(p') \leqslant n$ and consider p a level $n+1$ formula:

Cases $\vee$ and $\neg$. If $p = q \vee r$ then take $F_p(X) = F_q(X) \vee F_r(X)$. In case $p = \neg q$, we put $F_p(X) = \neg F_q(X)$.

Case $\langle\ \rangle$. Lemma 2 states that every modality-free program has its input-output predicate $R_\alpha(X, Y)$ recursive. Programs contain modalities as part of their tests. Let us consider a test, i.e., a program of the form $r?$, with $l(r) \leqslant n$. By the induction hypothesis, $F_r(X)$ exists, so we can define $R_{r?}(X, Y) = F_r(X)) \vee Equal(X, Y)$, where $Equal(w_1, w_2)$ holds iff $w_1 = w_2$. To handle ; and $\bigcup$ we consider the equations:

$$R_{\beta;\gamma}(X, Y) = \exists Z(R_\beta(X, Z) \wedge R_\gamma(Z, Y))$$

$$R_{\beta \cup \gamma}(X, Y) = R_\beta(X, Y) \vee R_\gamma(X, Y).$$

In order to obtain the *-equation, we shall adapt the construction from (Salomaa, 1973, p. 115). We start by defining several predicates: B, E, S, D, T:

- $B(w_1, w_2)$ holds iff $\exists w_3$, such that $w_2 = \# w_1 \# w_3$
- $E(w_1, w_2)$ holds iff $\exists w_3$, such that $w_2 = w_3 w_1$
- $S(w_1, w_2)$ holds iff w_1 is a subword in w_2
- $D(w)$ holds iff $w \neq \# w'$
- $T(X, Z) = B(\# X \#, Z) \wedge D(Z) \wedge E(\#, Z) \wedge (\forall Y)(\forall W)$
  $[\neg S(\# Y \# W \#, Z) \vee S(\#, Y) \vee S(\#, Z)$
  $\vee (\exists Y_1)(\exists Y_2)(\exists Y_3)(\exists W_2)$
  $(Y = Y_1 Y_2 Y_3) \wedge (W = Y_1 W_2 Y_3) \wedge R_\beta(Y_2, W_2)]$.

Informally, $T(w, u)$ holds iff $u = \# w_0 \# w_1 \# \cdots \# w_m \#$, $m \geqslant 1$, $w_0 = w$, and $R_\beta(w_i, w_{i+1})$, for all i, $0 \leqslant i \leqslant m - 1$. Now define

$$R_{\beta+}(X, Y) = \exists Z(T(X, Y) \wedge E(\# Y \#, Z) \wedge \neg S(\#, Y)),$$

$$R_{\beta*}(X, Y) = Equal(X, Y) \vee R_{\beta+}(X, Y).$$

As a consequence, for any program containing modalities, its input–output predicate is recursive. To complete the induction, if $p = \langle\alpha\rangle q$, then take $F_p(X) = \exists Y(R_\alpha(X, Y) \wedge F_q(Y))$.

The case $p = [\alpha]\, q$ is similar. ∎

COROLLARY 2. *Let p be a $\sum_n^{\mathrm{PDL}}$ ($\prod_n^{\mathrm{PDL}}$) formula. Then, for every structure $\mathscr{A}$, $\pi_{\mathscr{A}}(p)$ belongs to $\sum_n^0$ ($\prod_n^0$).*

THEOREM 1. *The following relations hold:*

1. $\sum_n^{\mathrm{PDL}} \prec \sum_{n+1}^{\mathrm{PDL}}$, $n \geqslant 0$
2. $\prod_n^{\mathrm{PDL}} \prec \prod_{n+1}^{\mathrm{PDL}}$, $n \geqslant 0$
3. $\sum_n^{\mathrm{PDL}} \not\equiv \prod_n^{\mathrm{PDL}}$, $n > 0$.

Proof. 1. Pick $L \in \sum_{n+1}^{0} - \sum_{n}^{0}$. Such a set exists because the arithmetic hierarchy of sets is strict (Rogers, 1967). By Lemma 4, we can find a formula p_L in $\sum_{n+1}^{\mathrm{PDL}}$ defining L. By Corollary 2, no formula of $\sum_{n}^{\mathrm{PDL}}$ can define L, because such formulas have their expressive power restricted to $\sum_{n}^{0}$. The proofs of parts 2 and 3 are similar. They are based on corresponding properties of the arithmetic hierarchy of sets (Rogers, 1967). ▮

4. Concluding Remarks

We have shown that alternation of modalities in PDL adds to the expressive power of the logic. Our results concern *horizontal* alternation of modalities. The syntax of the logic also provides with the posibility of constructing alternation of modalities *vertically*. For example,

$$\langle [\gamma; (\langle ([\alpha] p?); \beta \rangle q?)] r \cup \varepsilon \rangle s$$

has 3 levels of vertical alternation. Using equivalences like $\langle \alpha \rangle p \equiv \langle \alpha; p? \rangle$ *true* and $[\alpha] p \equiv [\alpha; p?]$ *true* we can obtain the same separation result for vertical alternation.

We saw that $\langle\ \rangle$ and $[\ ]$ enjoy certain quantifier-like properties, binding some subparts of formulas and adding to the expressive power, as they are supplied alternatively. Somewhat close to the above description are $\cdot$, $\bigcup$, $*$. So we ask two questions about them:

1. Does addition of $\cdot$, $\bigcup$, $*$ operators increase the expressive power of the logic? (e.g., Does the classification of PDL formulas by nesting levels of $*$ yield a strict hierarchy?)

2. Does the Hennessy–Milner logic (HML) (Bloom, Istrail, Meyer, 1988) possess a similar result: alternation adds to the expressive power? In this logic, programs are restricted to be basic, i.e., no ?, $\cdot$, $\bigcup$, $*$ are allowed.

A *HML-formula* is given by the following grammar

$$\varphi ::= tt \mid ff \mid \langle a \rangle \varphi \mid [\alpha] \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi,$$

where a is a basic program.

The interest in HML stands from its use in the theory of concurrent processes. Separation results are important for classifying concurrent processes mechanisms, as well as pointing out fundamental limitations of them (see Bloom, Istrail, Meyer, 1988 for one such application).

ACKNOWLEDGMENTS

Thanks to the referees for valuable detailed suggestions which significantly improved the paper. One of the referees' reports came in the moment when the author was about to "change worlds," a process that was "dynamic" but not always "logical." The author wants to thank that anonymous referee, for the enthusiastic report, which provided with support in a difficult moment. I want to thank David Harel, which in his quality as editor, provided me with continuous support and patience. Last but not least, I grateful to Albert Meyer and Rohit Parikh for comments and suggestions about the paper.

RECEIVED August 1982; ACCEPTED March 10, 1988

REFERENCES

BERMAN, F., AND PATERSON, M. (1981), Propositional dynamic logic is weaker without tests, *Theoret. Comput. Sci.* **16**, 321–328.

BLOOM, B., ISTRAIL, S., AND MEYER, A. R. (1988). Bisimulation can't be traced: Preliminary Report, *in* "Proceedings, 15th ACM Symp. Principles of Programming Languages, POPL 88," pp. 229–239.

FISCHER, M., AND LADNER, R. (1979), Propositional dynamic logic of programs, *J. Comput. System Sci.* **18** No. 2, 194–211.

HAREL, D. (1984), Dynamic logic, *in* "Handbook of Philosophical Logic" (D. Gabbay and F. Guenthner, Eds.), Vol. II, pp. 497–604, Reidel, Dordrecht.

PRATT, V. R. (1976), Semantical considerations on Floyd–Hoare logic, *in* "Proceedings, 17th IEEE Symp. on Theory of Computing, STOC 76," pp. 109–121.

ROGERS, JR., H. (1967), "Theory of Recursive Functions and Effective Computability," McGraw–Hill, New York.

SALOMAA, A. (1973), "Formal Languages," Academic Press, New York.

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium