SIAM J. COMPUT.
Vol. 13, No. 4, November 1984

011

IS THE INTERESTING PART OF PROCESS LOGIC UNINTERESTING?: A TRANSLATION FROM PL TO PDL*

R. SHERMAN†, A. PNUELI† AND D. HAREL†

Abstract. With the (necessary) condition that atomic programs in process logic (PL) be binary, we present an algorithm for the translation of a PL formula p into a program $\zeta(p)$ of propositional dynamic logic (PDL) such that a finite path satisfies p iff it belongs to $\zeta(p)$. This reduction has two immediate corollaries: 1) validity in this PL can be tested by testing validity of formulas in PDL; 2) all state properties expressible in this PL are expressible in PDL. The translation, however, is of nonelementary time complexity.

The significance of the result to the search for natural and powerful logics of programs is discussed.

Key words. process logic, propositional dynamic logic, temporal logic

1. Introduction. The formalism of dynamic logic [Pr1] has been successfully proposed as a unifying framework for the formal reasoning about programs. It generalizes, and at the same time simplifies, previous systems such as Hoare's axiomatic system [Ho], Dijkstra's predicate transformers [D], etc. It appears that as long as it is the input-output relation of a program (in contrast with its ongoing behavior) that we wish to study, dynamic logic provides us with a mathematically complete and elegant system of reasoning.

However, it was soon pointed out that if one is interested in the continuous behavior of programs and not only their in-out behavior, then dynamic logic seems inadequate. The need for reasoning about such behavior arises naturally in the study of nonterminating programs such as operating systems, and in the investigation of concurrent systems. One logic proposed in response to this need, temporal logic, has been used successfully in the analysis of concurrent systems [MP]. However, there is a desirable property, compositionality, which temporal logic does not enjoy, but which dynamic logic and other formalisms, such as Hoare logic and Dijkstra's predicate transformers, do. The principle of compositionality decrees that the formalism be syntax directed in the sense that it should derive its treatment of well-structured programs from its treatment of their immediate components. In contrast, the current formalism of temporal logic refers to instructions and labels in a fixed program and requires the analysis of the program as a whole without the possibility of studying its subparts. An additional drawback of temporal logic is its inability to express combined properties of many programs.

In view of this apparent dichotomy—a compositional system which does not deal with mid-execution properties, and a noncompositional system which does—there were many attempts to extend one or the other to yield a system, generically called *process logic*, enjoying the advantages of both. Pratt's original process logic [Pr2], Parikh's SOAPL [PA] and Nishimura's language [N], were preliminary efforts in this direction. A recently proposed system which seems to have unified the basic concepts of both dynamic and temporal logic is the system of process logic (PL) presented in [HKP]. It borrows the program constructs and modal operators [ ] and ⟨ ⟩ from dynamic logic, and the temporal connectives **f** and **suf** from temporal logic and combines them into a single system.

* Received by the editors June 4, 1982, and in final revised form August 19, 1983. The research reported here was supported in part by a grant of the Israeli Academy of Science, Basic Research Foundation.

† Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel.

The declared purpose of this new system is to enable compositional reasoning about continuous behavior of programs. As such, one would expect it to be able to express properties on the propositional level inexpressible by either PDL or TL, the propositional versions of dynamic and temporal logic, respectively. Indeed, the PL expression $[\alpha]$**some**p, for example, states that in every execution of the program α there must be at least one state satisfying p. It can be shown [H] that this property cannot be expressed in PDL. If p is some property of a second program, say $\langle\beta\rangle q$, then it is not expressible in TL either.

Having demonstrated this greater expressibility, PL certainly becomes an attractive system for study. It is shown in [HKP] that validity in (the propositional version of) PL is decidable by reduction to *SnS*, the second order theory of n successors [R]. This yields a nonelementary decision procedure in general, and it is still unknown whether an alternative elementary decision procedure exists. It has also been shown that various small extensions to the system lead to undecidability [CHMP], [S]. So much for background.

The investigation reported upon here was prompted by the following simple observation. Let us consider the PL statement $[\alpha]$**some**p. As mentioned, it is inexpressible in PDL for an abstract α. But suppose we knew the internal structure of α: for example, let $\alpha = (ab)^*c$ where a, b, c are *atomic* instructions. By assuming that they are atomic, we imply that there are no observable states *during* the execution of any of them. Thus if p, a state property, ever arises during a computation of α it may arise either before or after an atomic instruction but never during one. Similarly, if p holds before and after every atomic instruction of α it may be considered as holding continuously throughout the execution of α. Thus the property $[(ab)^*c]$**some**p, stating that p must hold somewhere in every execution of $\alpha = (ab)^*c$ is equivalent to the PDL statement:

$$[((\neg p)?a(\neg p)?b)^*(\neg p)?c(\neg p)?]\textbf{false}.$$

This PDL formula states that there can be no computation such that p is false before and after each of its atomic instructions. We immediately note the difference between the two modes of expressing this property. In PL we say that somewhere within α, p is realized; in the equivalent PDL formalization we have to explicitly state that it is not true that p is not realized at any of the locations possible during α.

In this paper we show how to apply this basic idea systematically to produce a translation algorithm from PL to PDL. However, no such translation is possible without ensuring the compatibility between models of PL and PDL. To this end one must adopt the *locality* of atomic formulas, i.e., that they are basically state formulas, true at states rather than on paths. Locality was adopted in [HKP] too and, indeed, by [CHMP], [S], the decidability of PL is lost without it. Moreover, one must adopt the *atomicity* of atomic programs as discussed above, i.e., that they consist of binary relations rather than arbitrary paths. Atomicity too is necessary since without it by [H] no such translation from PL to PDL exists. Accordingly we consider BPL, for *binary process logic*, the formalism obtained from PL by adopting the above restrictions. One then notices that the same mathematical objects serve as models for both PDL and BPL, the difference being only in the way satisfiability is extended from atomic formulas to general ones.

Our translation, to be specific, assigns to each formula p of BPL a PDL program $\zeta(p)$ such that for all finite paths x in any model, x satisfies p in BPL iff x is a possible computation path of $\zeta(p)$ in PDL, i.e., $x \in \zeta(p)$. Note that we treat finite paths only, and indeed we consider only finite paths in the notion of validity in BPL. It would

perhaps be possible to consider infinite paths if there were a programming construct, say α^{ω}, which would generate in PDL infinite computations from atomic programs. However, we feel that very little pragmatic power is lost due to the locality, atomicity and finiteness assumptions.

An additional technical matter concerns the presence in PL of paths which do not arise as computations of any program, in contrast to PDL where paths are implicitly present only as program computations. To overcome this incompatibility $\zeta(p)$ will employ a new atomic program u, understood to stand for the *universal* program which connects any two states. This idea involves no real loss of generality since, after all, the paths one is usually interested in are ones which arise from executing programs. Moreover, the two corollaries to our result, which we discuss next, make no use of this understanding regarding u. Also, it is easy to show that the addition of u to either BPL or PDL does not destroy their decidability.

The translation of BPL formulas to PDL programs can be utilized in the following two ways. First, we show how validity (over finite paths) in BPL may be reduced to validity in PDL; this is done by showing that p is satisfiable in BPL iff $\langle\zeta(p)\rangle\mathbf{true}$ is satisfiable in PDL. Secondly, we show that in a certain precise sense BPL and PDL are actually equivalent in expressive power. Specifically, if formulas of BPL are evaluated in zero-length paths, i.e., ones consisting of a single state, then BPL formulas express precisely the properties expressible by PDL, provided all states are connected by programs.

Returning to the title of the paper, we believe that in spite of the restrictions imposed on BPL, it retains all the important features of PL, rendering it an advanced system for reasoning compositionally about the continuous behavior of programs. Yet, we have succeeded in showing that properties expressible in BPL are in fact expressible in PDL too. Does this fact therefore detract from our interest in process logic?

Our argument is that rather than detract from it, the existence of such a translation should even enhance our interest in systems such as PL. One reason for this is the fact that the translation actually emphasizes the difference in modes of expression in the two logics. As already pointed out above, we simply state $[\alpha]\mathbf{some}p$ for the natural utterance: "in all executions of α, p is true somewhere". To state the same in PDL we must have full information about the structure of α and p, and in expressing the statement we must exhaust all possible ways of partitioning them. This need for detailed information about the structure of α and p, violates the principles of encapsulation and information hiding, and implies that the PDL style of expressing this property is by necessity a low level nonuniform one in comparison with the PL style.

Section 2 contains the definitions of PDL and BPL, § 3, the main one of the paper, contains our technical results, and in § 4 we discuss the issue of complexity. The proofs of some of the lemmas in § 3 are tedious but straightforward manipulations of finite automata considered as path acceptors, and are therefore gathered in an Appendix. They do not seem, however, to follow from standard closure properties of finite automata. The reader should be able to understand the idea of the proof of the theorem without having to resort to these detailed proofs, which are provided for completeness.

2. PDL and BPL.

Notation. Φ_0—the set of atomic formulas, Π_0—the set of atomic programs.

Syntax and semantics of PDL ([FL]):

A model is a triple $(S, \models, \rho)$ where:

S—is a set of states.

$\models$—is a satisfiability relation for atomic formulas.

$\rho : \Pi_0 \mapsto 2^{S\times S}$—is an interpretation for atomic programs.

The set of PDL formulas Φ, the set of PDL programs Π and their extended interpretations $\models$ and ρ are defined by:

1. $\Phi_0 \subseteq \Phi$
   true $\in \Phi$, $\forall s \in S$, $s \models$ **true**
   false $\in \Phi$, $\forall s \in S$, $s \not\models$ **false**
2. If $p, q \in \Phi$ then $p \vee q \in \Phi$, $\neg p \in \Phi$
   $s \models p \vee q$ iff $s \models p$ or $s \models q$
   $s \models \neg p$ iff $s \not\models p$
3. If $\alpha \in \Pi$, $p \in \Phi$ then $\langle \alpha \rangle p \in \Phi$
   $s \models \langle \alpha \rangle p$ iff $\exists t$, $(s, t) \in \rho(\alpha)$ and $t \models p$
4. $\Pi_0 \subseteq \Pi$,
   $\theta \in \Pi$, $\rho(\theta) = \varnothing$, i.e., the empty set
   $u \in \Pi$, $\rho(u) = S \times S$, i.e., the universal program
5. If $\alpha, \beta \in \Pi$ then $\alpha \cup \beta \in \Pi$, $\alpha\beta \in \Pi$ and $\alpha^* \in \Pi$
   $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
   $\rho(\alpha\beta) = \{(s, t) | \exists t', (s, t') \in \rho(\alpha), (t', t) \in \rho(\beta)\}$
   $\rho(\alpha^*) = \cup_{i \geqq 0} \rho(\alpha^i)$ $(\rho(\alpha^0) = \{(s, s) | s \in S\}$, $\alpha^{i+1} = \alpha\alpha^i)$
6. If $p \in \Phi$ then $p? \in \Pi$
   $\rho(p?) = \{(s, s) | s \models p\}$

Syntax and semantics of BPL:

A model is a triple, $(S, \models, R)$ where:

S, $\models$—as before.

R—the interpretation for atomic programs is an assignment of sets of paths of length one (two states) to atomic programs.

Note that a model for PDL, i.e., a triple $(S, \models, \rho)$ is a priori also a model for BPL. R is taken to be simply ρ itself.

A path in a model is a finite sequence of states, with repetitions allowed. For a path $x = (x_0, \cdots, x_k)$, y is a *proper suffix of* x, denoted $y < x$, if $y = (x_i, \cdots, x_k)$ for some $i \geqq 1$. We extend $\models$ to a satisfiability relation over paths, denoted by $\models_p$, and define R—the interpretation of BPL programs. R assigns a set of paths R_α to each BPL program, α, i.e., the set of all paths corresponding to α computations. Note that while PDL formulas are interpreted over *states*, BPL formulas are interpreted over *paths*. We will use $\models$ to denote $\models_p$ when there is no danger of confusion.

The set of BPL formulas $\bar{\Phi}$, the set of BPL programs $\bar{\Pi}$ and their extended interpretations $\models_p$, R_α are defined by:

1. $\Phi_0 \subseteq \bar{\Phi}$,
   For a path x and atomic formula $P \in \Phi_0$, $x \models_p P$ iff $x_0 \models P$ where x_0 is the first state of x.
   true $\in \bar{\Phi}$, $\forall x$, $x \models_p$ **true**
   false $\in \bar{\Phi}$, $\forall x$, $x \not\models_p$ **false**
2. If $p, q \in \bar{\Phi}$ then $p \vee q \in \bar{\Phi}$ and $\neg p \in \bar{\Phi}$
   $x \models_p p \vee q$ iff $x \models_p p$ or $x \models_p q$
   $x \models_p \neg p$ iff $x \not\models_p p$
3. If $\alpha \in \bar{\Pi}$, $p \in \bar{\Phi}$ then $\langle \alpha \rangle p \in \bar{\Phi}$
   $x \models_p \langle \alpha \rangle p$ iff $\exists y \in R_\alpha$ such that $xy \models_p p$
   (If $x = (x_0 \cdots x_k)$, $y = (y_0 \cdots y_l)$ and $x_k = y_0$ then $xy = (x_0 \cdots x_k y_1 \cdots y_l)$. If $x_k \neq y_0$ then xy is not defined. All future references to xy implicitly contain the requirement that xy is defined.)

4. If $p, q \in \bar{\Phi}$ then $\mathbf{f}p \in \bar{\Phi}$ and $p\mathbf{suf}q \in \bar{\Phi}$
   $x \models_p \mathbf{f}p$ iff $(x_0) \models_p p$ (x_0 is the first state of x; (x_0) is the path consisting of x_0 alone.)
   $x \models_p p\mathbf{suf}q$ iff there exists a path y such that:
   a. $y < x$ and $y \models_p q$
   b. for every z such that $y < z < x$, $z \models_p p$
5. $\Pi_0 \subseteq \bar{\Pi}$
   $\theta \in \bar{\Pi}$, $R_\theta = \varnothing$
   $u \in \bar{\Pi}$, $R_u = S \times S$
6. If $\alpha, \beta \in \bar{\Pi}$ then $\alpha \cup \beta \in \bar{\Pi}$, $\alpha\beta \in \bar{\Pi}$ and $\alpha^* \in \bar{\Pi}$
   $R_{\alpha \cup \beta} = R_\alpha \cup R_\beta$
   $R_{\alpha\beta} = \{x \mid \exists y \in R_\alpha, \exists z \in R_\beta \text{ s.t. } x = yz\}$
   $R_{\alpha^*} = \bigcup_{i \geqq 0} R_{\alpha^i}$ $(R_{\alpha^0} = \{(s) \mid s \in S\})$
7. If $p \in \bar{\Phi}$ then $p? \in \bar{\Pi}$
   $R_{p?} = \{x \mid x \models_p p\}$

Programs of the form $p?$, in both PDL and BPL, are called *tests.*

3. Results.

DEFINITION.

1. For a BPL program α and a path x, denote $x \in R_\alpha$ simply by $x \in \alpha$.
2. For a PDL program α and a path $x = (x_0, \cdots, x_l)$, $x \in \alpha$ is defined by induction on the structure of α:

If $\alpha \in \Pi_0$ then $x \in \alpha$ iff $l = 1$ and $(x_0, x_1) \in \rho(\alpha)$; $x \notin \theta$ for all x.
$x \in u$ iff $x = (x_0, x_1)$ for some states x_0, x_1;
$x \in \alpha \cup \beta$ iff $x \in \alpha$ or $x \in \beta$;
$x \in \alpha\beta$ iff $\exists j, 0 \leqq j \leqq l$ such that $(x_0, \cdots, x_j) \in \alpha$ and $(x_j, \cdots, x_l) \in \beta$;
$x \in \alpha^*$ iff $\exists i \geqq 1$ such that $x \in \alpha^i$ or $l = 0$ (i.e., $x = (x_0)$);
If $p \in \Phi$ then $x \in p?$ iff $l = 0$ and $(x_0, x_0) \in \rho(p?)$.
Note that for every x, $x \in u^*$.

Denote by $T \subseteq \Pi$ the set of all tests of PDL, i.e., programs of the form $p?$ for $p \in \Phi$. We define Π_u to be the set of all programs over the alphabet $T \cup \{u\}$. Thus, the only atomic program used in Π_u is u. Note, however, that the tests appearing in Π_u may themselves contain programs which are not in Π_u.

THEOREM. *For every* BPL *formula p there exists a* PDL *program* $\zeta(p)$ *in* Π_u *such that, for every path x in every model,* $x \models_p p$ iff $x \in \zeta(p)$.

The proof of the theorem will proceed by induction on the structure of BPL formulas. Accordingly we will present a sequence of propositions and lemmas corresponding to the rules for constructing well formed BPL formulas.

PROPOSITION 1. *To atomic* BPL *formulas and to the special formulas* **true** *and* **false** *there correspond* PDL *programs in the sense of the theorem.*

Proof. For atomic $P \in \bar{\Phi}$ take $\zeta(P) = P?u^*$. Obviously, a path $x = (x_0, \cdots, x_l)$ satisfies P iff $x_0 \models P$ iff $(x_0, \cdots, x_l) \in P?u^*$. Similarly take $\zeta(\mathbf{true}) = u^*$ and $\zeta(\mathbf{false}) = \theta$. □

PROPOSITION 2. *If* $p, q \in \bar{\Phi}$, *two* BPL *formulas, already have corresponding translations* $\zeta(p), \zeta(q) \in \Pi_u$ *in the sense of the theorem, then so does the formula* $p \vee q$.

Proof. Take $\zeta(p \vee q) = \zeta(p) \cup \zeta(q)$. Obviously $x \models_p p \vee q$ iff $x \models_p p$ or $x \models_p q$ iff $x \in \zeta(p)$ or $x \in \zeta(q)$ iff $x \in \zeta(p) \cup \zeta(q)$. □

Proposition 2 can be interpreted as a closure property, namely, that the class of sets of paths definable by Π_u programs is closed under union. For our next step we will need the closure of this class under complementation. Thus we shall show that to every Π_u program α, there corresponds a program $\tilde{\alpha}$ such that $x \notin \alpha \Leftrightarrow x \in \tilde{\alpha}$. Needless

to say, this fact is not (and does not follow directly from) the fact that regular sets are closed under complementation.

For a PDL program α, denote by T_α the set of tests appearing in α. Then α may be regarded as a regular expression over the alphabet $\Pi_0 \cup T_\alpha$.

DEFINITION. Two PDL programs α and β are *equivalent*, denoted by $\alpha \approx \beta$, if for every path x in every model, $x \in \alpha$ iff $x \in \beta$.

We start by defining two special sets of PDL programs. The set of programs whose execution sequences consist of alternations of atomic programs and tests is denoted by Γ:

$\Gamma = \{\alpha \mid \alpha \in \Pi$, for every word w in the language defined by the regular expression α, w is either λ or of the form $p_0?a_1p_1? \cdots a_kp_k?$ for some $k \geqq 0$ and $p_i \in \Phi$, $a_j \in \Pi_0 \cup \{u\}\}$.

Γ_u is defined similarly, except that u is the only atomic program allowed. Thus Γ_u consists of alternations of tests with the universal program u.

The following lemmas and their corollaries establish that Γ and Γ_u are normal forms (in the "path sense") of the set Π of PDL programs, and, respectively, the set Π_u of PDL programs involving no atomic programs but u.

LEMMA 1.

1. *Let* $\alpha, \beta \in \Gamma$; *then* $\alpha \cup \beta \in \Gamma$.
2. *Let* $\alpha, \beta \in \Gamma$; *then there exist programs* $\gamma, \delta \in \Gamma$ *such that* $\gamma \approx \alpha\beta$, *and* $\delta \approx \alpha^*$.

Proof. See Appendix.

COROLLARY 1. *For every* PDL *program* $\alpha \in \Pi$ *there exists a program* $\psi(\alpha) \in \Gamma$ *such that* $\psi(\alpha) \approx \alpha$.

Proof. By induction on the structure of α:

For atomic program a, let $\psi(a) = \textbf{true}?a\textbf{true}?$.

For $\alpha = p?$, let $\psi(\alpha) = \alpha$.

For $\alpha = \beta \cup \gamma$, let $\psi(\alpha) = \psi(\beta) \cup \psi(\gamma)$. By Lemma 1, $\psi(\alpha) \in \Gamma$.

For $\alpha = \beta\gamma$ consider $\psi(\beta)$ and $\psi(\gamma)$, which exists by the inductive hypothesis. Let $\psi(\alpha)$ be the program in Γ equivalent to $\psi(\beta)\psi(\gamma)$ and existing by Lemma 1.

For $\alpha = \beta^*$, let $\psi(\beta)$ be the program existing for β by the inductive hypothesis. Let $\psi(\alpha)$ be the program in Γ equivalent to $\psi(\beta)^*$ and existing by Lemma 1. □

LEMMA 2.

1. *Let* $\alpha, \beta \in \Gamma_u$; *then* $\alpha \cup \beta \in \Gamma_u$.
2. *Let* $\alpha, \beta \in \Gamma_u$; *then there exist programs* $\gamma, \delta \in \Gamma_u$ *such that* $\gamma \approx \alpha\beta$ *and* $\delta \approx \alpha^*$.

Proof. See Appendix.

COROLLARY 2. *For every* PDL *program* $\alpha \in \Pi_u$ *there exists a program* $\psi(\alpha) \in \Gamma_u$ *such that* $\psi(\alpha) \approx \alpha$.

Proof. Similar to the proof of Corollary 1. □

DEFINITION. Given a set of formulas $T = \{p_1, \cdots, p_n\}$, an *atom* of T is a conjunction $\bigwedge_{i=1}^n q_i$ where each q_i is either p_i or $\neg p_i$. T^A is defined to be the set of all atoms of T. Thus, in particular, T_α^A is the set of atoms constructed from the tests appearing in the program α.

LEMMA 3. *Let* α *be a program in* Γ_u *with tests* T_α. *Then there exists a program* $\tilde{\alpha} \in \Gamma_u$ *over the alphabet* $\{u\} \cup T_\alpha^A$ *such that for every path* x *in any model,* $x \in \tilde{\alpha}$ *iff* $x \notin \alpha$.

Proof. See Appendix.

We may thus summarize this sequence of lemmas by the following proposition.

PROPOSITION 3. *If a* BPL *formula* $p \in \bar{\Phi}$ *is translatable into a* PDL *program* $\zeta(p)$ *in* Π_u *in the sense of the theorem, then so is* $\neg p$.

Proof. By Corollary 2 there is a program $\gamma = \psi(\zeta(p))$ in Γ_u, equivalent to $\zeta(p)$. Now let $\zeta(\neg p)$ be $\tilde{\gamma}$, guaranteed to exist and to satisfy the properties by Lemma 3. □

PROPOSITION 4. *If a* BPL *formula p is translatable into a* PDL *program* $\zeta(p)$ *in* Π_u *in the sense of the theorem, then so is* **f***p.*

Proof. Let $\zeta(\mathbf{f}p)$ be Yu^*, where Y denotes the union of the set of all words of the form $q?$ in $\zeta(p)$. Since Y is finite, $\zeta(\mathbf{f}p)$ is regular. □

Even though taking care of union and complementation (cf. Lemmas 2, 3) automatically ensures closure under intersection, even in the "path sense", we do need this property in a more general setting. We therefore consider next closure under intersection.

LEMMA 4.

1. Let $\alpha \in \Pi$ and $\beta \in \Gamma_u$. Then there exists a program $\alpha \cap \beta \in \Gamma$ such that for every path x, $x \in \alpha \cap \beta$ iff $x \in \alpha$ and $x \in \beta$.

2. Let $\alpha, \beta \in \Gamma_u$. Then there exists a program $\alpha \cap \beta \in \Gamma_u$ such that for every path x, $x \in \alpha \cap \beta$ iff $x \in a$ and $x \in \beta$.

Proof. See Appendix.

We now have to deal with formulas in BPL of the form ***psuf****q*.

DEFINITION. Ω_α, the *partition set* for a program $\alpha \in \Gamma_u$, is defined as a subset of $\Pi_u \times \Pi_u$ by induction on α:

1. $\Omega_u = \{(\mathbf{true}?, u), (u, \mathbf{true}?)\}$.
2. $\Omega_{p?} = \{(\mathbf{true}?, p?), (p?, \mathbf{true}?)\}$.
3. $\Omega_{\beta \cup \gamma} = \Omega_\beta \cup \Omega_\gamma$.
4. $\Omega_{\beta\gamma} = \{(\beta_1, \beta_2\gamma) | (\beta_1, \beta_2) \in \Omega_\beta\} \cup \{(\beta\gamma_1, \gamma_2) | (\gamma_1, \gamma_2) \in \Omega_\gamma\}$.
5. $\Omega_{\beta^*} = \{(\mathbf{true}?, \mathbf{true}?)\} \cup \{(\beta^*\beta_1, \beta_2\beta^*) | (\beta_1, \beta_2) \in \Omega_\beta\}$.

The intuitive meaning is that Ω_α contains the pairs of regular expressions which, when concatenated, yield α.

LEMMA 5. *Let* $\alpha \in \Gamma_u$. *Then for every path x and every* x_1, x_2 *such that* $x = x_1x_2$, $x \in \alpha$ *iff there is a pair* $(\alpha_1, \alpha_2) \in \Omega_\alpha$ *such that* $x_1 \in \alpha_1$ *and* $x_2 \in \alpha_2$.

Proof. Immediate. □

LEMMA 6. *Let* $\alpha \in \Gamma_u$. *Then there exists a program* $\sigma(\alpha) \in \Gamma_u$ *such that* $x \in \sigma(\alpha)$ *iff for every suffix (not necessarily proper)* x' *of* x, $x' \in \alpha$.

Proof. Let $\beta = ((u\mathbf{true}?)^*\tilde{\alpha})$, and take $\sigma(\alpha)$ to be $\tilde{\beta}$. One then observes that $x \in \sigma(\alpha)$ iff $x \notin (u\mathbf{true}?)^*\tilde{\alpha}$, iff for every suffix x' of x, $x' \notin \tilde{\alpha}$, iff for every suffix x' of x, $x' \in \alpha$. □

LEMMA 7. *Let* $\alpha, \beta \in \Gamma_u$. *Then there exists a program* $\mu(\alpha, \beta) \in \Gamma_u$ *such that for every path x,* $x \in \mu(\alpha, \beta)$ *iff there exists* $y \in \beta$, *with* $y < x$, *such that*

(*) *for every z, if* $y < z < x$ *then* $z \in \alpha$.

Proof. Let Ω be a nonempty subset of the partition set Ω_α. Denote:

$$h(\Omega) = \bigcup_{\{\alpha_1 | \exists \alpha_2, (\alpha_1, \alpha_2) \in \Omega\}} \alpha_1,$$

$$t(\Omega) = \bigcap_{\{\alpha_2 | \exists \alpha_1, (\alpha_1, \alpha_2) \in \Omega\}} \alpha_2.$$

Then let

$$\mu_1(\alpha, \beta) = \bigcup_{\Omega \subseteq \Omega_\alpha} (\sigma(h(\Omega))((u\beta) \cap t(\Omega))),$$

$$\mu_0(\alpha, \beta) = (u\beta) \cup (u\mu_1(\alpha, \beta)).$$

We first prove the following claim:

For every path $x = (x_0, \cdots, x_k)$, $x' = (x_1, \cdots, x_k) \in \mu_1(\alpha, \beta)$ iff there exists $y \in \beta$, $y < x'$ and y satisfies (*) above. Indeed, $x' \in \mu_1(\alpha, \beta)$, iff (for some $\Omega \subseteq \Omega_\alpha$, fixed in what follows) $x' \in \sigma(h(\Omega))((u\beta) \cap t(\Omega))$, iff $\exists j$, $1 \leqq j \leqq k$ such that $(x_1, \cdots, x_j) \in$

$\sigma(h(\Omega))$, $(x_j, \cdots, x_k) \in u\beta$ and $(x_j, \cdots, x_k) \in t(\Omega)$, iff $\exists j$, $1 \leqq j \leqq k$ such that $(x_{j+1}, \cdots, x_k) \in \beta$, $\forall l$, $1 \leqq l \leqq j$, $(x_l, \cdots, x_j) \in h(\Omega)$ and $(x_j, \cdots, x_k) \in \alpha_2$, $\forall \alpha_2$ s.t. $\exists \alpha_1$, $(\alpha_1, \alpha_2) \in \Omega$ (by Lemma 4, Part 2), iff $\exists j$, $1 \leqq j \leqq k$ such that $(x_{j+1}, \cdots, x_k) \in \beta$ and $\forall l$, $1 \leqq l \leqq j$, $\exists (\alpha_1, \alpha_2) \in \Omega_\alpha$ such that $(x_l, \cdots, x_j) \in \alpha_1$ and $(x_j, \cdots, x_k) \in \alpha_2$, iff $\exists j$, $1 \leqq j \leqq k$ such that $(x_{j+1}, \cdots, x_k) \in \beta$ and (by Lemma 5) $\forall l$, $1 \leqq l \leqq j$, $\exists (\alpha_1, \alpha_2) \in \Omega_\alpha$ $(x_l, \cdots, x_k) \in \alpha_1\alpha_2 = \alpha$, iff $\exists y \in \beta$, $y \prec x'$ such that y satisfies $(*)$ above. This completes the proof of the claim.

Back to the main proof: Clearly, $\mu_0(\alpha, \beta) \in \Pi_u$ and by Corollary 2 there exists a program $\mu(\alpha, \beta) \in \Gamma_u$ such that $\mu_0(\alpha, \beta) \approx \mu(\alpha, \beta)$. Let x be a path, $x = (x_0, x_k)$. Then $x \in \mu(\alpha, \beta)$ iff $x \in \mu_0(\alpha, \beta)$ iff $(x_1, \cdots, x_k) \in \beta$ or $(x_1, \cdots, x_k) \in \mu_1(\alpha, \beta)$ iff $(x_1, \cdots, x_k) \in \beta$ or (by the claim) there exists $y \in \beta$, $y \prec (x_1, \cdots, x_k)$, such that y satisfies $(*)$ above, iff there exists $y \in \beta$, $y \prec x$, which satisfies $(*)$. □

PROPOSITION 5. *If* BPL *formulas p and q are translatable into* PDL *programs $\zeta(p)$ and $\zeta(q)$ in Π_u in the sense of the theorem then so is* **psuf***q*.

Proof. Let $\zeta(p\mathbf{suf}q) = \mu(\zeta(p), \zeta(q))$, existing by Lemma 7. □

Proof of the theorem. By induction on the structure of p: by Propositions 1-5 all we need to prove is the case $p = \langle\beta\rangle q$. We prove first that for every BPL program β there exists a PDL program $\beta' \in \Gamma$ such that for each path, $z \in \beta$ iff $z \in \beta'$. Define β' by induction on β as follows:

If $\beta \in \Pi_0$ then $\beta' = \mathbf{true}?\beta\mathbf{true}?$.

If $\beta = p?$ then by the main inductive hypothesis and Corollary 2 there exists $\zeta(p) \in \Gamma_u$ such that $z \in \zeta(p)$ iff $z \models_p p$. Thus $(p?)'$ can be taken to be $\zeta(p)$.

Now define:

$(\beta_1 \cup \beta_2)' = \beta_1' \cup \beta_2'$.

$(\beta_1\beta_2)' = \beta_1'\beta_2'$.

$(\beta_1^*)' = (\beta_1')^*$.

To continue the proof, note that $x \models_p p$ iff there is $y \in \beta$ such that $xy \models_p q$, iff (by the inductive hypothesis and the construction of β') $\exists y \in \beta'$ such that $xy \in \zeta(q)$, iff $\exists y \in \beta'$, $\exists (\alpha_1, \alpha_2) \in \Omega_{\zeta(q)}$ such that $x \in \alpha_1$, $y \in \alpha_2$ and xy is defined, iff $\exists (\alpha_1, \alpha_2) \in \Omega_{\zeta(q)}$ such that $x \in \alpha_1$, y_0 is the last state of x and $y_0 \models \langle\beta' \cap \alpha_2\rangle\mathbf{true}$ (the existence of $\beta' \cap \alpha_2$ is guaranteed by Lemma 4, Part 1), iff $x \in \zeta'(p)$ where $\zeta'(p) \in \Pi_u$ is defined as $\bigcup_{(\alpha_1, \alpha_2) \in \Omega_{\zeta(q)}} \alpha_1(\langle\beta' \cap \alpha_2\rangle\mathbf{true})?$.

By Corollary 2 there exists $\zeta(p) \in \Gamma_u$ with $\zeta(p) \approx \zeta'(p)$, and the proof is complete. □

Examples. Let P be an atomic formula, and a an atomic program. The following are simplified forms of $\zeta(q)$ for some sample formulas q.

1. A simplified form of $\zeta(\langle a\rangle P)$: $P?u^*(\langle a\rangle\mathbf{true})?$.
2. A simplified form of $\zeta([a^*]\mathbf{all}P)$: $P?(uP?)^*([a^*(\neg P)?a^*]\mathbf{false})?$.
3. A simplified form of $\zeta(\langle a^*\rangle\mathbf{some}P)$: $u^*P?u^* \cup u^*(\langle a^*\rangle P)?$.

COROLLARY 3. *For every* BPL *formula p there exists a* PDL *formula p' such that p is valid, i.e., true of all finite paths in all models, iff p' is valid, i.e., true of all states in all models.*

Proof. We will show that a BPL formula q is satisfiable iff $\langle\zeta(q)\rangle\mathbf{true}$ is satisfiable in PDL. Accordingly, p' is taken to be $[\zeta(\neg p)]\mathbf{false}$. Indeed, assume $x \models_p q$ in some model M. Let M' be the extension of M in which u is interpreted as the universal program connecting any two states. By the theorem $x \in \zeta(q)$ in M', and hence $x_0 \models \langle\zeta(q)\rangle\mathbf{true}$ in M'. Conversely, if $s \models \langle\zeta(q)\rangle\mathbf{true}$ in some model M, then there is some path x in M, with $x_0 = s$, such that $x \in \zeta(q)$. By the theorem $x \models_p q$ in M. □

DEFINITION. A *program model* over a set $\bar{a} = \{a_1, \cdots, a_n\}$ of atomic programs, is a model in which any two states are connected by some sequence of elements of $\bar{a}$.

COROLLARY 4. *For every* BPL *formula p there exists a* PDL *formula p' such that for every state s in any program model M,* $M, (s) \models_p \mathbf{f}p$ *iff* $M, s \models p'$.

Proof. Let s be a state in a program model M over $\bar{a} = \{a_1, \cdots, a_n\}$. By the theorem $(s) \models_p \mathbf{f}p$ in M iff $(s) \in \zeta(\mathbf{f}p)$ in the model extending M by interpreting u as the universal program. Define $\zeta'(\mathbf{f}p)$ to be the program $\zeta(\mathbf{f}p)$ where every appearance of the universal program u is replaced by $(\cup_{i=1}^{n} a_i)^*$, then by the definition of a program model $x \in \zeta(\mathbf{f}_p)$ in the extended model iff $x \in \zeta'(\mathbf{f}p)$ in M for any path x. Since (s) is of length zero, it follows that $M, (s) \models_p p$ iff $M, (s) \models_p \mathbf{f}p$ iff $M, s \models \langle \zeta'(\mathbf{f}p) \rangle \mathbf{true}$. □

As pointed out by one of the referees, the converse of Corollary 4 is also true. Namely, PDL formulas are translatable in BPL formulas such that satisfaction in a state is transferred into satisfaction in the appropriate zero-length path. To see this, the formula $\neg(\mathbf{false\ suf\ true})$ (true only in paths of length zero) is added to each test, and the PDL formula is otherwise left untouched. Thus, in the sense of Corollary 4, BPL and PDL are actually equivalent in expressive power.

4. Complexity. The operator "**chop**" was defined in [HKP] by $x \models_p p\mathbf{chop}q$ iff $\exists y, z$ such that $x = yz$ and $y \models_p p$, $z \models_p q$. A formula containing **chop** can be easily translated to a PDL program by $\zeta(p\mathbf{chop}q) = \zeta(p)\zeta(q)$. Thus, in particular our result gives a decision procedure for validity in BPL + **chop**. A simple analysis of the translation algorithm presented herein shows a nonelementary complexity. This is actually not accidental since (as observed in [HP]) PL + **chop** is nonelementary even for program-free formulas. Thus BPL + **chop** is also nonelementary.

Appendix. The closure properties of Γ and Γ_u claimed in Lemmas 1-4 are proven here by considering special forms of finite automata accepting the programs in these classes. Although regular sets are closed under composition, Kleene-star, complementation and intersection, we are interested in these operations in the "path sense".

For example, the required complement $\tilde{\alpha}$ of α is not a program which defines a set of words over atomic programs and tests which is the complement of that defined by α, but rather is a program whose set of paths in the model is the complement of that of α. The existence of such a complement $\tilde{\alpha}$ is indeed established below by considering a certain deterministic automaton for α and appropriately complementing the set of final states, but this construction has to be done with care, and does not follow from the classical one.

First, for $\alpha \in \Gamma$, let $\mathcal{M}_\alpha$ be a nondeterministic finite automaton defining the language of α, which alternates in a predetermined way between two disjoint subsets of its set of states when encountering tests and atomic programs.

$$\mathcal{M}_\alpha = \langle K_1 \cup K_2 \cup \{d\}, \Pi_0 \cup T_\alpha, k_0, \eta, F \rangle$$

where:

$K_1 \cap K_2 = \emptyset$, $k_0 \in K_1$, $F \subseteq K_2$.

For $k \in K_1$, if $p \in T_\alpha$ then $\eta(k, p) \subseteq K_2$, and if $a \in \Pi_0$ then $\eta(k, a) = \{d\}$,

For $k \in K_2$, if $p \in T_\alpha$ then $\eta(k, p) = \{d\}$, and if $a \in \Pi_0$ then $\eta(k, a) \subseteq K_1$,

If $p \in T_\alpha$ then $\eta(d, p) = \{d\}$, if $a \in \Pi_0$ then $\eta(d, a) = \{d\}$.

Let $\bar{K}_1$ be the set of states leading to a final state:

$\bar{K}_1 = \{k | k \in K_1, \exists p \in T_\alpha$ such that $\eta(k, p) \cap F \neq \emptyset\}$,

For any $k \in \bar{K}_1$ define: $T_\alpha(k) = \{p | p \in T_\alpha, \eta(k, p) \cap F \neq \emptyset\}$.

Note. For a program $\alpha \in \Gamma$, and a path $x = (x_0, \cdots, x_l)$, $x \in \alpha$ iff there is a word w defined by α, $w = p_0?a_0 \cdots a_{l-1}p_l?$ such that $x \in w$; that is, $x_i \in p_i?$ i.e. $x_i \models p_i$, $0 \leqq i \leqq l$, and $(x_i, x_{i+1}) \in \rho(a_i)$, $0 \leqq i < l$.

Proof of Lemma 1.

1. Obvious.

2. Let $\mathcal{M}_\alpha$, $\mathcal{M}_\beta$ be the automata defining α, β respectively:
$\mathcal{M}_\alpha = \langle K_1 \cup K_2 \cup \{d\}, \Pi_0 \cup T_\alpha, k_0, \eta, F\rangle$,
$\mathcal{M}_\beta = \langle K_1' \cup K_2' \cup \{d'\}, \Pi_0 \cup T_\beta, k_0', \eta', F'\rangle$,
Define: $T_\alpha \wedge T_\beta = \{p \wedge q \mid p \in T_\alpha, q \in T_\beta, p \neq q\} \cup \{T_\alpha \cap T_\beta\}$.
Define the automaton $\mathcal{M}_\gamma$ by:

$$\mathcal{M}_\gamma = \langle \hat{K}_1 \cup \hat{K}_2 \cup \{\hat{d}\}, \Pi_0 \cup T_\alpha \cup T_\beta \cup \{T_\alpha \wedge T_\beta\}, k_0, \hat{\eta}, \hat{F}\rangle$$

where

$$\hat{F} = \begin{cases} F \cup F' & \text{if } \lambda \in \beta \text{ (for example, if } \beta \text{ is of the form } \beta_1^* \text{ or } \beta_1^* \beta_2^*), \\ F' & \text{otherwise,} \end{cases}$$

$$\hat{K}_1 = K_1 \cup K_1' \cup \{e_1\}, \quad \hat{K}_2 = K_2 \cup K_2' \cup \{e_2\}.$$

The transition function $\hat{\eta}$ is given by the following cases:

A) For $k \in \hat{K}_1$

if $a \in \Pi_0$ then $\hat{\eta}(k, a) = \{\hat{d}\}$

if $p \in T_\alpha - T_\beta$ then

$$\hat{\eta}(k, p) = \begin{cases} \eta(k, p), & k \in K_1, \\ \{e_2\}, & k \in K_1' \cup \{e_1\} \end{cases}$$

if $p \in T_\beta - T_\alpha$ then

$$\hat{\eta}(k, p) = \begin{cases} \eta'(k, p), & k \in K_1', \\ \{e_2\}, & k \in K_1 \cup \{e_1\} \end{cases}$$

if $p \in T_\alpha \cap T_\beta$ then

$$\hat{\eta}(k, p) = \begin{cases} \eta'(k, p), & k \in K_1', \\ \eta(k, p), & k \in K_1 - \bar{K}_1, \\ \{e_2\}, & k = e_1, \\ \eta(k, p) \cup \eta'(k_0', p), & p \in T_\alpha(k) \cap T_\beta, k \in \bar{K}_1, \\ \eta(k, p), & p \notin T_\alpha(k) \cap T_\beta, k \in \bar{K}_1 \end{cases}$$

if $p \in T_\alpha \wedge T_\beta - \{T_\alpha \cap T_\beta\}$ then

$$\hat{\eta}(k, p) = \begin{cases} \eta'(k_0', q) \cup \eta(k, r), & p = q \wedge r, r \in T_\alpha(k), q \in T_\beta, k \in \bar{K}_1, \\ \{e_2\}, & p \notin T_\alpha(k) \wedge T_\beta, k \in K_1 \text{ or } k \notin \bar{K}_1 \end{cases}$$

B) For $k \in \hat{K}_2$

if $p \in T_\alpha \cup T_\beta \cup \{T_\alpha \wedge T_\beta\}$ then $\hat{\eta}(k, p) = \{\hat{d}\}$

if $a \in \Pi_0$ then

$$\hat{\eta}(k, a) = \begin{cases} \eta(k, a) & k \in K_2, \\ \eta'(k, a) & k \in K_2', \\ \{e_1\} & k = e_2 \end{cases}$$

C) For $\hat{d}$

if $p \in T_\alpha \cup T_\beta \cup \{T_\alpha \wedge T_\beta\} \cup \Pi_0$ then $\hat{\eta}(\hat{d}, p) = \{\hat{d}\}$

The program γ is then taken to be some regular expression corresponding to $\mathcal{M}_\gamma$.

For the α^* case define $\mathcal{M}_\delta$ by:

$$\mathcal{M}_\delta = \langle \hat{K}_1 \cup \hat{K}_2 \cup \{\hat{d}\}, \Pi_0 \cup T_\alpha \cup \{T_\alpha \wedge T_\alpha\} \cup \{\textbf{true}\}, k_0, \hat{\eta}, \hat{F}\rangle$$

where: $\hat{K}_1 = K_1 \cup \{e_1\}$, $\hat{K}_2 = K_2 \cup \{k_t, e_2\}$, $\hat{F} = F \cup \{k_t\}$. The transition function $\hat{\eta}$ is given by the following cases:

A) For $k \in \hat{K}_1$

if $a \in \Pi_0$ then $\hat{\eta}(k, a) = \{\hat{d}\}$

if $p \in T_\alpha$ then

$$\hat{\eta}(k, p) = \begin{cases} \eta(k, p), & k \in K_1, \\ \{e_2\}, & k = e_1 \end{cases}$$

if $p \in T_\alpha \wedge T_\alpha$ then

$$\hat{\eta}(k, p) = \begin{cases} \eta(k_0, q) \cup \eta(k, r), & p = q \wedge r, r \in T_\alpha(k), q \neq r, q \in T_\alpha, k \in \bar{K}_1, \\ \eta(k_0, p) \cup \eta(k, p), & p \in T_\alpha(k), k \in \bar{K}_1, \\ \{e_2\}, & k \notin \bar{K}_1 \text{ or } p \notin T_\alpha(k) \wedge T_\alpha \end{cases}$$

$$\hat{\eta}(k, \textbf{true}) = \begin{cases} \{k_t\}, & k = k_0, \\ \{e_2\}, & \text{otherwise} \end{cases}$$

B) For $k \in \hat{K}_2$

if $p \in T_\alpha \cup \{T_\alpha \wedge T_\alpha\}$, $\hat{\eta}(k, p) = \{\hat{d}\}$

if $a \in \Pi_0$

$$\hat{\eta}(k, a) = \begin{cases} \eta(k, a), & k \in K_2, \\ \{e_1\}, & k \in \{k_t, e_2\} \end{cases}$$

C) For $\hat{d}$

if $p \in T_\alpha \cup \{T_\alpha \wedge T_\alpha\} \cup \Pi_0$ then $\hat{\eta}(\hat{d}, p) = \{\hat{d}\}$

The program δ is then taken to be some regular expression corresponding to $\mathcal{M}_\delta$. □

Proof of Lemma 2. Similar to the proof of Lemma 1. □

Proof of Lemma 3. To prove the lemma we first define a *path deterministic* automaton (and later the complement automaton) in the sense that a *path* is "accepted" by at most a single sequence of states of the automaton.

Consider, for example, the automaton given in Fig. 1. While this automaton is deterministic in the usual sense, it is not path deterministic, because, for example, the path $x = (x_0, x_1)$ where $x_0 \models P \wedge Q$, and $x_1 \models R \wedge U$, is "accepted" by both sequences (s_0, s_1, s_2, s_3) and (s_0, s_4, s_5, s_6) of the automaton. Thus, while the labels on the edges originating in s_0 are disjoint, there exist paths satisfying $P \wedge Q$ which are compatible with both.

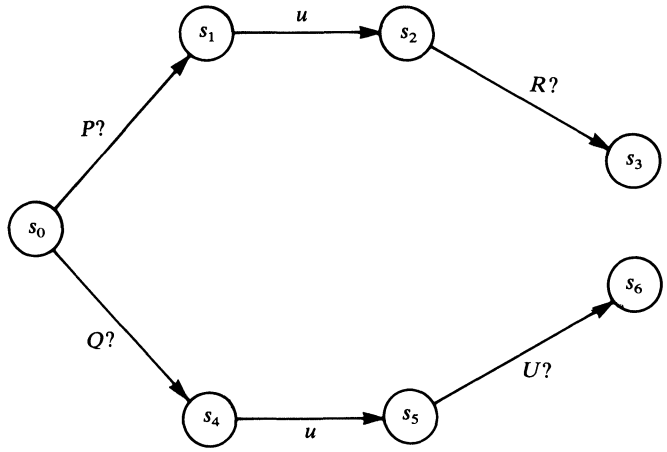

FIG. 1

We will be using the set of atoms T_α^A.

DEFINITION. Let $\alpha \in \Gamma_u$ and let $\mathcal{M}_\alpha = \langle K_1 \cup K_2 \cup \{d\}, \{u\} \cup T_\alpha, k_0, \eta, F\rangle$ be the automaton defining α. Then $\mathcal{M}_\alpha^E$, the (*nondeterministic*) *extended automaton* for α, is defined by:

$$\mathcal{M}_\alpha^E = \langle \hat{K}_1 \cup \hat{K}_2 \cup \{\hat{d}\}, \{u\} \cup T_\alpha^A, k_0, \eta^E, F\rangle$$

where: $\hat{K}_1 = K_1 \cup \{e_1\}$, $\hat{K}_2 = K_2 \cup \{e_2\}$, and T_α^A is the set of atoms for α. The transition function η^E is given by the following cases:

A) For $k \in \hat{K}_1$

$$\eta^E(k, u) = \{\hat{d}\}$$

for $k \in K_1$

$$\eta^E(k, \textstyle\bigwedge_{i=1}^n q_i) = \begin{cases} \{e_2\}, & \{q_i\}_{i=1}^n \cap T_\alpha = \varnothing, \\ \bigcup_j \{\eta(k, q_j) | q_j \in T_\alpha\}, & \text{otherwise} \end{cases}$$

$$\eta^E(e_1, \textstyle\bigwedge_{i=1}^n q_i) = \{e_2\}$$

B) For $k \in \hat{K}_2$

$$\eta^E(k, \textstyle\bigwedge_{i=1}^n q_i) = \{\hat{d}\}$$

$$\eta^E(k, u) = \begin{cases} \eta(k, u), & k \in K_2, \\ \{e_1\}, & k = e_2 \end{cases}$$

C) If $p \in T_\alpha^A \cup \{u\}$, $\eta^E(\hat{d}, p) = \{\hat{d}\}$

Claim 1. Let $\alpha \in \Gamma_u$, and let α^E be any program defining the set of words over $\{u\} \cup T_\alpha^A$ accepted by $\mathcal{M}_\alpha^E$. Then:

1. $\alpha^E \in \Gamma_u$.
2. $\alpha \approx \alpha^E$.

Proof.

1. Obvious from the form of $\mathcal{M}_\alpha^E$.

2. Let $x = (x_0, \cdots, x_k)$ be a path, $x \in \alpha$. Then there is a word $w \in \alpha$, $w = r_0?u \cdots ur_k?$, and $(x_i) \in r_i?$, $0 \leqq i \leqq k$. For every $p \in T_\alpha$ either $(x_i) \in p?$ or $(x_i) \in (\neg p)?$. It follows that for every i, $0 \leqq i \leqq k$ there exist $q_i^1, \cdots, q_i^n \in T_\alpha \cup \neg T_\alpha$ such that $q_i^i = r_i$ and $(x_i) \in (\bigwedge_{j=1}^n q_i^j)?$ for $0 \leqq i \leqq K$. By the definition of $\mathcal{M}_\alpha^E$ the word w^E given by: $w^E = (\bigwedge_{j=1}^n q_0^j)?u \cdots u(\bigwedge_{j=1}^n q_k^j)?$ is accepted by $\mathcal{M}_\alpha^E$ by the sequence of states in $\mathcal{M}_\alpha$ which accepts w, and $(x_i) \in (\bigwedge_{j=1}^n q_i^j)?$ for every $0 \leqq i \leqq k$; that is, $x \in \alpha^E$.

Let $x = (x_0, \cdots, x_k)$, $x \in \alpha^E$. Then there is a word $w^E \in \alpha^E$, $w^E = r_0^E?u \cdots ur_k^E?$, $x \in w^E$, $r_i^E \in T_\alpha^A$; that is, $r_i^E = \bigwedge_{j=1}^n q_i^j$, $q_i^j \in T_\alpha \cup \neg T_\alpha$. Since w^E is accepted by $\mathcal{M}_\alpha^E$, it follows that for every i there is a j_i such that $q_i^{j_i} \in T_\alpha$, otherwise $\eta^E(k, \bigwedge_{k=1}^n q_j) = \{e_2\}$. Since $(x_i) \in r_i^E?$, $(x_i) \in q_i^{j_i}?$, $0 \leqq i \leqq k$, and the word $w = q_0^{j_0}?u \cdots uq_k^{j_k}?$ is accepted by $\mathcal{M}_\alpha$. Thus we have $x \in \alpha$. □

DEFINITION. Let $\alpha \in \Gamma_u$ and let $\mathcal{M}_\alpha^E$ be the extended automaton for α.

$$\mathcal{M}_\alpha^E = \langle K_1 \cup K_2 \cup \{d\}, \{u\} \cup T_\alpha^A, k_0, \eta^E, F\rangle.$$

The *deterministic automaton for* α, $\mathcal{M}_\alpha^D$, is defined from the nondeterministic one in the usual way by:

$$\mathcal{M}_\alpha^D = \langle K_1^D \cup K_2^D \cup \{d\}, \{u\} \cup T_\alpha^A, \{k_0\}, \eta^D, F^D\rangle$$

where: $K_1^D = 2^{K_1}$, $K_2^D = 2^{K_2}$, $F^D = \{\bar{k} | \bar{k} \in K_2^D \text{ and } \bar{k} \cap F \neq \varnothing\}$.

For $\bar{k} \in K_1^D$
$\eta^D(\bar{k}, u) = \{d\}$
if $p \in T_\alpha^A$ then $\eta^D(\bar{k}, p) = \cup\{\eta^E(k, p) | k \in \bar{k}\}$
For $\bar{k} \in K_2^D$
$\eta^D(\bar{k}, u) = \cup\{\eta^E(k, u) | k \in \bar{k}\}$
if $p \in T_\alpha^A$ then $\eta^D(\bar{k}, p) = \{d\}$
For every $p \in T_\alpha^A \cup \{u\}$, $\eta^D(d, p) = \{d\}$.

Claim 2. Let $\alpha \in \Gamma_u$ and let α^D be any program defining the set of words accepted by $\mathcal{M}_\alpha^D$, the deterministic automaton for α. Then:

1. $\alpha^D \in \Gamma_u$.
2. For every word w over $\{u\} \cup T_\alpha^A$, $w \in \alpha^E$ iff $w \in \alpha^D$.

Proof.

1. Obvious by the form of $\mathcal{M}_\alpha^D$.
2. If $w \in \alpha^D$ then obviously $w \in \alpha^E$.

Suppose $w \in \alpha^E$, $w = r_0?u \cdots ur_k?$. By the form of $\mathcal{M}_\alpha^E$
$\{\eta^E(k, u)\} \subseteq K_1$, for each $k \in K_2$
$\{\eta^E(k, p)\} \subseteq K_2$, for each $k \in K_1$, and $p \in T_\alpha^A$
from which it follows that w is accepted by $\mathcal{M}_\alpha^D$, and we have $w \in \alpha^D$. □

To complete the proof of the lemma let $\mathcal{M}_\alpha^D$ be the deterministic automaton for α

$$\mathcal{M}_\alpha^D = \langle K_1 \cup K_2 \cup \{d\}, \{u\} \cup T_\alpha^A, k_0, \eta^D, F\rangle.$$

Define $\tilde{\mathcal{M}}_\alpha$ by

$$\tilde{\mathcal{M}}_\alpha = \langle K_1 \cup K_2 \cup \{d\}, \{u\} \cup T_\alpha^A, k_0, \eta^D, K_2 - F\rangle$$

and let $\tilde{\alpha}$ be a program defined by $\tilde{\mathcal{M}}_\alpha$. Obviously $\tilde{\alpha} \in \Gamma_u$.

Consider a path $x = (x_0, \cdots, x_k)$ such that $x \notin \alpha$. Let $T_i \in T_\alpha^A$ $(0 \leqq i \leqq k)$ be the T_α atom which is true in x_i (obviously there exists a unique element of T_α^A which is true in x_i). Then the word $T_0?u \cdots uT_k?$ is not accepted by $\mathcal{M}_\alpha^D$ $(x \notin \alpha)$. Neither does it lead $\mathcal{M}_\alpha^D$ to the error state d. Hence it leads $\mathcal{M}_\alpha^D$ to a state in $K_2 - F$ and is acceptable by $\tilde{\mathcal{M}}_\alpha$. Therefore $x \in \tilde{\alpha}$.

Suppose $x \in \tilde{\alpha}$ and $x \in \alpha$, then also $x \in \alpha^D$. Let $x = (x_0, \cdots, x_k)$; then:

there is a word $\tilde{w} \in \tilde{\alpha}$,	$\tilde{w} = \tilde{r}_0?u \cdots u\tilde{r}_k?$ and $x \in \tilde{w}$,
there is a word $w \in \alpha^D$,	$w = r_0?u \cdots ur_k?$ and $x \in w$,
$r_i \in T_\alpha^A \Rightarrow r_i = \bigwedge_{j=1}^n q_i^j$,	$q_i^j \in \{p_j, \neg p_j\}$ for every $p_j \in T_\alpha$,
$\tilde{r}_i \in T_\alpha^A \Rightarrow \tilde{r}_i = \bigwedge_{j=1}^n \tilde{q}_i^j$,	$\tilde{q}_i^j \in \{p_j, \neg p_j\}$ for every $p_j \in T_\alpha$.

We know that for every state x_i and every formula $p \in T_\alpha$, either $(x_i) \in p?$ or $(x_i) \in (\neg p)?$ but not both. It follows that $q_i^j = \tilde{q}_i^j$, $1 \leqq j \leqq n$, $0 \leqq i \leqq k$. But then $w = \tilde{w}$ is accepted by both $\mathcal{M}_\alpha^D$ and $\tilde{\mathcal{M}}_\alpha^D$, contradicting the definition of $\tilde{\mathcal{M}}_\alpha^D$. This completes the proof of Lemma 3. □

Proof of Lemma 4.

1. Without loss of generality we can assume $\alpha \in \Gamma$ by Corollary 1. Let $\mathcal{M}_\alpha$ and $\mathcal{M}_\beta$ be the automata defining α, β respectively.

$$\mathcal{M}_\alpha = \langle K_1 \cup K_2 \cup \{d\}, \Pi_0 \cup T_\alpha, k_0, \eta, F\rangle,$$

$$\mathcal{M}_\beta = \langle K_1' \cup K_2' \cup \{d'\}, \{u\} \cup T_\beta, k_0', \eta', F'\rangle.$$

The automaton $\mathcal{M}_{\alpha \cap \beta}$ is defined by:

$$\mathcal{M}_{\alpha \cap \beta} = \langle \hat{K}_1 \cup \hat{K}_2 \cup \{\hat{d}\}, \Pi_0 \cup \{T_\alpha \wedge T_\beta\}, \hat{k}, \hat{\eta}_0, \hat{F}\rangle$$

where

$$\hat{K}_1 = K_1 \times K_1', \qquad \hat{K}_2 = K_2 \times K_2',$$
$$\hat{F} = F \times F', \qquad \hat{k}_0 = (k_0, k_0').$$

A) Let $(k, k') \in \hat{K}_1$

for $p \in \{T_\alpha \wedge T_\beta\}$

$$\hat{\eta}((k, k'), p) = \begin{cases} \eta(k, q) \times \eta'(k', r), & p = q \wedge r,\ q \neq r, \\ \eta(k, p) \times \eta'(k', p) & p \in T_\alpha \cap T_\beta \end{cases}$$

for $a \in \Pi_0$,

$$\hat{\eta}((k, k'), a) = \{\hat{d}\}$$

B) Let $(k, k') \in \hat{K}_2$

for $p \in \{T_\alpha \wedge T_\beta\}$, $\hat{\eta}((k, k'), p) = \{\hat{d}\}$
for $a \in \Pi_0$, $\hat{\eta}((k, k'), a) = \eta(k, a) \times \eta'(k', u)$
for $p \in \{T_\alpha \wedge T_\beta\} \cup \Pi_0$, $\hat{\eta}(\hat{d}, p) = \{\hat{d}\}$

Let $\alpha \cap \beta$ be a program defined by $\mathcal{M}_{\alpha \cap \beta}$, then $\alpha \cap \beta \in \Gamma$. Let $x = (x_0, \cdots, x_l)$ be a path. Then: $x \in \alpha \cap \beta$ iff there exists a word w in the language defined by $\alpha \cap \beta$ such that $w = r_0?a_0 \cdots a_{l-1}r_l?$, where for each i, $0 \leqq i \leqq l$, $(x_i) \in r_i?$, $(x_i, x_{i+1}) \in a_i$ and $w_i \in T_\alpha \wedge T_\beta$; that is, $\exists p_i \in T_\alpha$, $q_i \in T_\beta$ such that $r_i = p_i \wedge q_i$ iff $w_\alpha = p_0?a_0 \cdots a_{l-1}p_l?$ is accepted by $\mathcal{M}_\alpha$, $w_\beta = q_0?u \cdots uq_l?$ is accepted by $\mathcal{M}_\beta$ and $x \in w_\alpha$, $x \in w_\beta \Leftrightarrow x \in \alpha$ and $x \in \beta$.

2. The proof for $\alpha, \beta \in \Gamma_u$ is similar to the proof of part (1) except that the alphabet for $\mathcal{M}_{\alpha \cap \beta}$ is $\{u\} \cup \{T_\alpha \wedge T_\beta\}$ and consequently Π_0 is replaced by $\{u\}$ in the proof. □

Acknowledgment. Two detailed and thoughtful reports by anonymous referees prompted us to prepare a considerably improved revision replacing a rather unreadable first draft.

REFERENCES

[CHMP] A. CHANDRA, J. HALPERN, A. MEYER AND R. PARIKH, *Equations between regular terms and an application to process logic*, 13th ACM Symposium on Theory of Computing, 1981, pp. 384–390.

[D] E. W. DIJKSTRA, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[FL] M. J. FISCHER AND R. E. LADNER, *Propositional dynamic logic of regular programs*, J. Comp. Syst. Sci., 18 (1979), pp. 194–211.

[H] D. HAREL, *Two results on process logic*, Inform. Proc. Letters, 8 (1979), pp. 195–198.

[HKP] D. HAREL, D. KOZEN AND R. PARIKH, *Process logic: expressiveness, decidability, completeness*, J. Comp. Syst. Sci., 25 (1982), pp. 144–170.

[HP] D. HAREL AND D. PELEG, *Process logic with regular formulas*, submitted.

[Ho] C. A. R. HOARE, *An axiomatic basis for computer programming*, Comm. Assoc. Comput. Mach., 12 (1969), pp. 576–580.

[MP] Z. MANNA AND A. PNUELI, *The modal logic of programs*, 6th International Colloquium on Automata, Languages and Programming, Graz, Austria, 1979, Lecture Notes in Computer Science 71, Springer-Verlag, New York, pp. 385–409.

[N] H. NISHIMURA, *Descriptively complete process logic*, Acta Inform., 14 (1980), pp. 359–369.

[Pa] R. PARIKH, *A decidability result for second order process logic*, 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 177–183.

[Pr1] V. R. PRATT, *Semantical considerations on Floyd-Hoare logic*, 17th IEEE Symposium on Foundations of Computer Science, 1976.

[Pr2] V. R. PRATT, *Process logic*, 6th ACM Symposium on Principles of Programming Languages, 1979, pp. 93–100.

[R] M. O. RABIN, *Decidability of second order theories and automata on infinite trees*, Trans. Amer. Math. Soc., 141 (1969), pp. 1–35.

[S] R. S. STREETT, *Global process logic is undecidable*, Proc. 2nd Conference on Foundations of Software Technology and Theoretical Computer Science, 1982, Bangalore, India.