IS THE INTERESTING PART OF PROCESS LOGIC UNINTERESTING?:

A TRANSLATION FROM PL TO PDL

R. Sherman, A. Pnueli, D. Harel
Department of Applied Mathematics
The Weizmann Institute of Science
76100 Rehovot, Israel

Abstract

With the (necessary) condition that atomic programs in PL be binary, we present an algorithm for the translation of a PL formula X into a PDL program $\tau(X)$ such that a finite path satisfies X iff it belongs to $\tau(X)$. This reduction has two immediate corollaries: 1) validity in this PL can be tested by testing validity of formulas in PDL ; 2) all finite-path program properties expressible in this PL are expressible in PDL.

The translation, however, seems to be of non-elementary time complexity. The significance of the result to the search for natural and powerful logics of programs is discussed.

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 logn as we wish to study the input-output relations computed by a program, 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 in their in-out behavior, then dynamic logic seems inadequate. The need for reasoning about such behavior arises naturally in the study of non-terminating programs such as operating systems, and in the investigation of concurrent systems. Consequently, alternative logics have been proposed to formalize the continous behavior of programs. One such logic, temporal logic, has been used successfully in the analysis of concurrent systems [BP]. The problem with some of these alternatives, temporal logic included, is their lack of compositionality, a property which is present in such formal systems as Hoare's logic, predicate transformers, denotational semantics, dynamic logic, etc. 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 single fixed program as a fixed context and requires the analysis of the program as a whole without the possibility of studying its subparts.

In view of this apparent dichotomy - a compositional system which cannot deal with mid-execution properties, and a non-compositional system which can - 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

The research reported here was supported in part by a grant of the Israeli Academy of Science, The Basic Research Foundation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1982 ACM 0-89791-065-6/82/001/0347 $00.75

dynamic logic, and the temporal connectives 'f' and 'suf' from temporal logic and combines them into a single system.

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 $[\alpha]$SomeX, for example, states that in every execution of the program α there must be at least one state satisfying X . It can be shown [H] that this property cannot be expressed in PDL, and of course not in TL (which has no way of explicitly mentioning programs such as α).

Having demonstrated this significantly 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 non-elementary 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]. So much for background.

The investigation reported upon here was prompted by the following simple observation. Let us reconsider the PL statement $[\alpha]$SomeX . As mentioned, it is inexpressible in PDL for an abstract α . But suppose we knew the internal structure of α : for example, let $\alpha = (a;b)^*;c$ where a,b,c are <u>atomic</u> instructions. By assuming that they are atomic, we imply that there are no observable states <u>during</u> the execution of any of them. Thus if X , 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 X holds before and after every atomic instruction of α it may be considered as holding continuously throughout the execution of α . Thus the property $[(a;b)^*;c]$SomeX , stating, that X must hold somewhere in every execution of $\alpha = (a;b)^*;c$ is equivalent to the PDL statement:

$$[((\sim X)?;a;(\sim X)?;b)^*;(\sim X)?c;(\sim X)?]\text{false} .$$

This PDL formula states that there can be no computation such that X 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 , X is realized; in the equivalent PDL formalization we have to explicitly state that it is not true that X 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 compatibity between models of PL and PDL. To this end one must adopt the <u>locality</u> 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], the decidability of PL is lost without it. Moreover, one must adopt the <u>atomicity</u> 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 <u>binary process logic</u>. With the above restrictions, 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 X of BPL a PDL program $\tau(X)$ such that for all finite paths p in any model, $p \models X$ in BPL iff p is a possible computation path of $\tau(X)$ in PDL , i.e., $p \in \tau(X)$. 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 was 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, atomicy 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

$\tau(X)$ 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.

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 X is satifiable in BPL iff $<\tau(X)>true$ is satisfiable in PDL.

As another application, consider using BPL for the description of program properties. One soon realizes that the properties of interest are <u>state</u> properties referring to the initial state of the computation. Thus statements of the form: "all computations of program α eventually realize X", or: "there exists an α computation such that all its β extensions perform some task" etc., are to be true of the initial state of the computation. These properties are uniformly expressible in BPL by formulas of the form fX where f is the PL connective first. We will show that our translation is such that for a path $p = (p_0, p_1, \ldots)$ $p \models fX$ in BPL, iff $p_0 \models <\tau(fX)>true$, in PDL provided that any test appearing in X also describes a program property. Thus we have a direct translation of formulas of BPL to formulas of PDL for the frequently used formulas of the form fX.

Returning to the title of the paper, we believe that in spite of the restrictions imposed in BPL, it retains all the important features of PL, rendering it an advanced system for reasoning compositionally about the continous 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 $[\alpha]$<u>Some</u>X for the natural utterance: "in all executions of α, X is somewhere true". To state the same in PDL we must have full information about the structure of α and X, and in expressing the statement we must exhaust all possible ways of partitioning α and X. This need for detailed information about the structure of α and X, violates the principles of encapsulation and information hiding, and implies that the PDL style of expressing this property is by necessity a low level one in comparison with the PL style.

Section 2 contains the definitions of PDL and BPL, Section 3, the main one of the paper contains our technical results, and in Section 4 we discuss the issue of complexity.

2. <u>PDL and BPL.</u>

Notation: Φ_0 - the set of atomic formulas.

Σ_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

$\rho : \Sigma_0 \to 2^{S \times S}$ is an interpretation for atomic programs

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

1. $\Phi_0 \subseteq \Phi$

   $true \in \Phi$ and $\forall\, s \in S$, $s \models true$

   $false \in \Phi$ and $\forall\, s \in S$, $s \not\models false$

2. If $X, Y \in \Phi$ then $X \vee Y \in \Phi$

   and $s \models X \vee Y$ iff $s \models X$ or $s \models Y$

3. If $X \in \Phi$ then $\sim X \in \Phi$ and $s \models \sim X$ iff $s \not\models X$

4. If $\beta \in \Sigma$, $X \in \Phi$ then $<\beta>X \in \Phi$

   and $s \models <\beta> X$ iff $\exists t$, $(s,t) \in \rho(\beta)$ and $t \models X$

5. $\Sigma_0 \subset \Sigma$

$\theta \in \Sigma, \rho(\theta)$ is the empty set; $u \in \Sigma$,

$\rho(u) = S\times S$ i.e. the universal program.

6. If $\alpha, \beta \in \Sigma$ then $\alpha \cup \beta \in \Sigma$

and $\rho(\alpha\cup\beta) = \rho(\alpha) \cup \rho(\beta)$

7. If $\alpha,\beta \in \Sigma$ then $\alpha;\beta \in \Sigma$ and $\rho(\alpha;\beta) =$

$= \{(s,t) \mid \exists t' , (s,t') \in \rho(\alpha), (t',t) \in \rho(\beta)\}$

8. If $\alpha \in \Sigma$ then $\alpha^* \in \Sigma$ and $\rho(\alpha^*) = \bigcup_{i\geq 0} \rho(\alpha^i)$

$(\rho(\alpha^0) = \{(s,s) \mid s \in S\})$

9. If $X \in \Sigma$ then $X? \in \Sigma$ and $\rho(X?) = \{(s,s)\ s \models X\}$

<u>Syntax and Semantics of BPL</u> :

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 <u>a priori</u> 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. 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 <u>states</u>, BPL formulas are interpreted over <u>paths</u>.

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 $\bar{\Sigma}$ and their extended interpretation $\models_p, R$ are defined by:

A. $\Phi_0 \subseteq \bar{\Phi}$. For a path p and atomic formula $X \in \Phi_0$, $p \models_p X$ iff $p_0 \models X$, p_0 the first state of p.

true $\in \bar{\Phi}$ $\forall p$, $p \models_p$ *true*

false $\in \bar{\Phi}$ $\forall p$, $p \not\models_p$ *false*

B. If $X, Y \in \bar{\Phi}$ then $X\vee Y \in \bar{\Phi}$ and $p \models_p X\vee Y$ iff $p \models_p X$ or $p \models_p Y$

C. If $X \in \bar{\Phi}$ then $\sim X \in \bar{\Phi}$ and $p \models_p \sim X$ iff $p \not\models_p X$

D. If $X \in \bar{\Phi}$, $\beta \in \bar{\Sigma}$ then $<\beta>X \in \bar{\Phi}$ and $p \models_p <\beta>X$ iff $\exists q \in R_\beta$ such that $pq \models_p X$.
if $p = (p_0 \ldots p_k)$, $q = (q_0 \ldots q_\ell)$ and $p_k = q_0$ then $pq = (p_0 \ldots p_k q_1 \ldots q_\ell)$

E. If $X \in \bar{\Phi}$ then $fX \in \bar{\Phi}$ and $p \models_p fX$ iff $(p_0) \models_p X$ (p_0 - the first state of p ; (p_0) the path consisting of p_0 alone.)

F. If $X,Y \in \bar{\Phi}$ then X suf Y $\in \bar{\Phi}$ and $p \models_p$ X suf Y iff there exists a path q such that:

a. q is a proper suffix of p (i.e. if $p = (p_0,\ldots,p_k)$ then $q = (p_i,\ldots,p_k)$ for some $i \geq 1$) and $q \models_p Y$

b. for every r such that r is a proper suffix of p and q is a proper suffix of r (r lying strictly between p and q), $r \models_p X$.

G. $\Sigma_0 \subset \bar{\Phi}, \theta \in \bar{\Sigma}$ $R_\theta = \emptyset$ (the empty set),
$u \in \bar{\Phi}$ $R_u = \{(s,t) \mid s,t \in S\}$
for $a \in \Sigma_0$ $R_a = \{(s,t)\ (s,t) \in \rho(a)\}$.

H. If $\alpha,\beta \in \bar{\Sigma}$ then $\alpha \cup \beta \in \bar{\Sigma}$ and $R_{\alpha\cup\beta} = R_\alpha \cup R_\beta$.

I. If $\alpha,\beta \in \bar{\Sigma}$ then $\alpha; \beta \in \bar{\Sigma}$ and $p \in R_{\alpha;\beta}$ iff $\exists q \in R_\alpha$, $\exists r \in R_\beta$ such that $p = qr$.

J. If $\alpha \in \bar{\Sigma}$ then $\alpha^* \in \bar{\Sigma}$ and $R_{\alpha^*} = \bigcup_{i\geq 0} R_{\alpha^i}$
$(R_{\alpha^0} = \{(s) \mid s \in S\}$, $\alpha^{i+1} = \alpha;\alpha^i)$

K. $X \in \bar{\Phi}$ then $X? \in \bar{\Sigma}$ and $R_{X?} = \{p \mid p \models X\}$

3. Results

Definition

1. For $\alpha \in \bar{\Sigma}$ a BPL program and a path p, $p \in \alpha$ iff $p \in R_\alpha$.

2. For $\alpha \in \Sigma$ a PDL program and a path $p = (p_0, \ldots, p_\ell)$ $p \in \alpha$ is defined by induction on the structure of α:

   a. If $\alpha \in \Sigma_0$ then $p \in \alpha$ iff $\ell = 1$ and $(p_0, p_1) \in \rho(\alpha)$. For every $p_0, p_1 (p_0, p_1) \in u$.

   b. If $\alpha, \beta \in \Sigma$ then $p \in \alpha \cup \beta$ iff $p \in \alpha$ or $p \in \beta$.

   c. If $\alpha, \beta \in \Sigma$ then $p \in \alpha;\beta$ iff $\exists j$, $0 \le j \le \ell$ such that $(p_0, \ldots, p_j) \in \alpha$, $(p_j, \ldots, p_\ell) \in \beta$,

   d. If $\alpha \in \Sigma$ then $p \in \alpha^*$ iff $\exists i \ge 1$ such that $p \in \alpha^i$ or $\ell = 0$ (i.e., $p = (p_0)$).

   e. If $X \in \Sigma$ then $p \in X?$ iff $\ell = 0$ and $(p_0, p_0) \in \rho(X?)$.

Note that for every path p, $p \in u^*$.

If α is a program with no tests it may be considered both as a PDL program or a BPL program. For such a program the different notions of $p \in \alpha$ separately defined for BPL and PDL programs coincide.

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

Main theorem.

For every BPL formula $X \in \Phi$ there exists a PDL program $\tau(X) \in \Sigma_u$ such that $p \models_p X$ iff $p \in \tau(X)$, for every path p in every model.

To prove this result we will proceed by induction on the structure of BPL formulas. Thus we will present a sequence of lemmas corresponding to the rules for constructing **well** formed BPL **formulas**.

Definition

Let $\alpha, \beta \in \Sigma$, then α and β (defined over Φ_0 and Σ_0) are equivalent, denoted by $\alpha \approx \beta$, if: for every path p in every model over Σ_0, Φ_0, $p \in \alpha$ iff $p \in \beta$.

Lemma A.

To each atomic BPL formula and the special formulas *true*, *false* there corresponds a PDL program in the sense of the theorem.

Proof: To an atomic BPL formula $P \in \bar{\Phi}$ there corresponds the PDL program $\tau(P) = P?;u^*$. Obviously, a path $p = (p_0, p_1, \ldots, p_\ell)$ satisfies P iff $p_0 \models P$ iff $(p_0, \ldots, p_\ell) \in P?;u^*$.

Similarly $\tau(\mathit{true}) = u^*$ and $\tau(\mathit{false}) = \theta$.

Lemma B.

If $X, Y \in \bar{\Phi}$, two BPL formulas, already have corresponding translations $\tau(X), \tau(Y) \in \Sigma$ in the sense of the theorem then so does the formula $X \vee Y$.

Proof: We define $\tau(X \vee Y) = \tau(X) \cup \tau(Y)$.

Obviously $p \models_p X \vee Y$ iff $p \models_p X$ or $p \models_p X$ iff $p \in \tau(X)$ or $p \in \tau(Y)$ iff $p \in \tau(X) \cup \tau(Y)$.

This Lemma can be interpreted as a closure property, namely, that the class of sets of paths definable by Σ_u programs is closed under union. For our next step we will need another closure of this class, namely, closure under complementation and intersection. It will state that to every Σ_u program α, there corresponds a complementary program $\tilde{\alpha}$ such that $p \notin \alpha \Leftrightarrow p \in \tilde{\alpha}$. The closure under complementation and intersection is established in Lemmas C1-C7.

For a program $\alpha \in \Sigma$, denote by T_α the set of tests in α . Then α may be regarded as a regular expression over the alphabet $(\Sigma_0 \cup T_\alpha)$.

We start by defining a certain normal form for PDL programs. The set of programs in Σ formed by alternations of atomic programs and tests is denoted by Qr : $Qr = \{\alpha | \alpha \in \Sigma$, for every word w in the language defined by the regular expression α, w is of the form $W_0?a_0W_1?;\ldots;W_{k-1}?a_{k-1}W_k?$ for some $k \geq 0$ and $W_i \in \Phi$, $a_i \in \Sigma_0 \}$.

We denote by M_α any nondeterministic automation defining α .

For $\alpha \in Qr$, M_α must be of the general form:

$M = \langle K_1 \cup K_2 \cup \{d\}\ , \Sigma_0 \cup T_\alpha, q_0, \delta, F\rangle$

where: $q_0 \in K_1$, $F \subseteq K_2$.

If $q \in K_1$, $X \in T_\alpha$ then $\delta(q,X) \subseteq K_2$

$a \in \Sigma_0$ then $\delta(q,a) = \{d\}$

If $q \in K_2$, $X \in T_\alpha$ then $\delta(q,X) = \{d\}$

$a \in \Sigma_0$ then $\delta(q,a) \subseteq K_1$

If $X \in T_\alpha$ then $\delta(d,X) = \{d\}$, if $a \in \Sigma_0$ then $\delta(d,a) = \{d\}$.

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

$\bar{K}_1 = \{q | q \in K_1 , \exists X \in T_\alpha$ such that $\delta(q,X) \cap F \neq \emptyset\}$

For any $q \in \bar{K}_1$ define:

$T_\alpha(q) = \{X | X \in T_\alpha , \delta(q,X) \cap F \neq \emptyset\}$

<u>Note:</u> For a program $\alpha \in Qr$, path $p = (p_0,\ldots,p_k)$, $p \in \alpha$ iff $\exists w$ a word defined by α,

$w = W_0?\ a_0 \ldots a_{k-1}W_k?$ such that $p \in w$; that is,

$(p_i) \in W_i?$ i.e. $p_i \models W_i$, $0 \leq i \leq k$, and

$(p_i,p_{i+1}) \in \rho(a_i)$, $0 \leq i < k$.

<u>Lemma C1</u>

1. Let $\alpha,\beta \in Qr$ then $\alpha \cup \beta \in Qr$
2. Let $\alpha,\beta \in Qr$ then there exists a program $\bar{\gamma} \in Qr$ such that $\bar{\gamma} \approx \alpha;\beta$
3. Let $\alpha \in Qr$ then there exists a program $\bar{\gamma} \in Qr$ such that $\bar{\gamma} \approx \alpha^*$

<u>Proof</u>:

1. Obvious
2. Let M_α, M_β be the automata defining α,β respectively:

$M_\alpha = \langle K_1^\alpha \cup K_2^\alpha \cup \{d^\alpha\},\ \Sigma_0 \cup T_\alpha,\ q_0^\alpha, \delta_\alpha, F_\alpha\rangle$

$M_\beta = \langle K_1^\beta \cup K_2^\beta \cup \{d^\beta\},\ \Sigma_0 \cup T_\beta,\ q_0^\beta, \delta_\beta, F_\beta\rangle$

define: $T_\alpha \wedge T_\beta = \{X \wedge Y | X \in T_\alpha,\ Y \in T_\beta,\ X \neq Y\} \cup \{T_\alpha \cap T_\beta\}$

We define an automaton M_γ by:

$M_\gamma = \langle \hat{K}_1 \cup \hat{K}_2 \cup \{\hat{d}\}, \Sigma_0 \cup T_\alpha \cup T_\beta \cup \{T_\alpha \wedge T_\beta\}, q_0^\alpha, \hat{\delta}, \hat{F}\rangle$

where

$$\hat{F} = \begin{cases} F_\alpha \cup F_\beta & \text{if } \lambda \in \alpha \text{ (for example, if } \alpha \text{ is of the form } \alpha_1^* \text{ or } \alpha_1^*\alpha_2^*) \\ F_\beta & \text{otherwise} \end{cases}$$

$\hat{K}_1 = K_1^\alpha \cup K_1^\beta \cup \{e_1\}$, $\hat{K}_2 = K_2^\alpha \cup K_2^\beta \cup \{e_2\}$.

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

A) For $q \in \hat{K}_1$,

if $a \in \Sigma_0$ then $\hat{\delta}(q,a) = \{\hat{d}\}$

$$\text{if } X \in T_\alpha - T_\beta \text{ then } \hat{\delta}(q,X) = \begin{cases} \delta_\alpha(q,X) & q \in K_1^\alpha \\ \{e_2\} & q \in K_1^\beta \cup \{e_1\} \end{cases}$$

$$\text{for } X \in T_\beta - T_\alpha, \hat{\delta}(q,X) = \begin{cases} \delta_\beta(q,X) & q \in K_1^\beta \\ \{e_2\} & q \in K_1^\alpha \cup \{e_1\} \end{cases}$$

$$\text{for } X \in T_\alpha \cap T_\beta, \hat\delta(q,X) = \begin{cases} \delta_\beta(q,X) & q \in K_1^\beta \\ \delta_\alpha(q,X) & q \in K_1^\alpha - \bar K_1^\alpha \\ \delta_\alpha(q,X) \cup \delta_\alpha(q_0^\beta,X) & \\ \text{if } X \in T_\alpha(q) \cap T_\beta,\ q \in \bar K_1^\alpha & \\ \delta_\alpha(q,X) & \\ \text{if } X \notin T_\alpha(q) \cap T_\beta, q \in \bar K_1^\alpha & \end{cases}$$

for $Z \in T_\alpha \wedge T_\beta - \{T_\alpha \cap T_\beta\}$

$$\hat\delta(q,Z) = \begin{cases} \delta_\beta(q_0^\beta,Y), & Z = X\wedge Y,\ X \in T_\alpha(q),\ q \in \bar K_1^\alpha \\ \{e_2\} & Z \notin T_\alpha(q)\wedge T_\beta,\ q \in K_1^\alpha \text{ or } q \notin \bar K_1^\alpha \end{cases}$$

B) For $q \in \hat K_2$

for $X \in T_\alpha \cup T_\beta \cup \{T_\alpha \wedge T_\beta\}$ $\quad \hat\delta(q,X) = \{\hat d\}$

for $a \in \Sigma_0$

$$\hat\delta(q,a) = \begin{cases} \delta_\alpha(q,a) & q \in K_2^\alpha \\ \delta_\beta(q,a) & q \in K_2^\beta \\ e_1 & q = e_2 \end{cases}$$

C) For $X \in T_\alpha \cup T_\beta \cup \{T_\alpha \wedge T_\beta\} \cup \Sigma_0$ $\quad \hat\delta(\hat d,X) = \hat d$

A program $\bar\gamma$ can be defined such that the set of words accepted by M_γ is the set of words defined by the regular expression $\bar\gamma$. Then, by the form of M_γ , $\bar\gamma \in Qr$ and obviously $p \in \bar\gamma$ iff $p \in \alpha;\beta$

3. Let M_α be the automaton defining α

$M_\alpha = \langle K_1 \cup K_2 \cup \{d\},\ \Sigma_0 \cup T_\alpha,\ q_0, \delta, F\rangle$

As before we denote $\bar K_1 = \{q \mid q \in K_1, \exists X \in T_\alpha ,$

$\delta(q,X) \cap F \neq \emptyset\}$

Define M_γ by:

$M_\gamma = \langle \hat K_1 \cup \hat K_2 \cup \{d\},\ \Sigma_0 \cup T_\alpha \cup \{T_\alpha \wedge T_\alpha\} \cup \{\text{true}\},$
$q_0, \hat\delta, \hat F\rangle$

where $\hat K_1 = K_1 \cup \{e_1\}$, $\hat K_2 = K_2 \cup \{q_t, e_2\}$

$\hat F = F \cup \{q_t\}$

For $q \in \hat K_1$: if $a \in \Sigma_0$ then $\hat\delta(q,a) = d$

$$\text{for } X \in T_\alpha, \hat\delta(q,X) = \begin{cases} \delta(q,X) & q \in K_1 \\ e_2 & q = e_1 \end{cases}$$

for $Z \in T_\alpha \wedge T_\alpha$,

$$\hat\delta(q,Z) = \begin{cases} \delta(q_0,Y) & Z = X \wedge Y, X \in T_\alpha(q),\ X \neq Y \quad q \in \bar K_1 \\ \delta(q_0,Z) & Z \in T_\alpha(q) \quad q \in \bar K_1 \\ e_2 & q \notin \bar K_1 \text{ or } Z \notin T_\alpha(q) \wedge T_\alpha \text{ and } q \in \bar K_1 \end{cases}$$

$$\hat\delta(q,\text{true}) = \begin{cases} q_t & q = q_0 \\ e_2 & \text{otherwise} \end{cases}$$

For $q \in \hat K_2$

if $X \in T_\alpha \cup \{T_\alpha \wedge T_\alpha\}$ then $\hat\delta(q,X) = d$

for $a \in \Sigma_0$

$$\hat\delta(q,a) = \begin{cases} \delta(q,a) & q \in K_2 \\ e_1 & q \in \{q_t, e_2\} \end{cases}$$

For $X \in T_\alpha \cup \{T_\alpha \wedge T_\alpha\} \cup \Sigma_0$ $\quad \hat\delta(d,X) = d$.

Let $\bar\gamma$ be a program defining the same set of words as M_γ . Then $\bar\gamma \in Qr$ and obviously for every path p , $p \in \bar\gamma$ iff $p \in \alpha^*$. □

<u>Lemma C2</u>

For every PDL program $\alpha \in \Sigma$ there exists a program $\gamma(\alpha) \in Qr$ such that $\gamma(\alpha) \approx \alpha$.

<u>Proof:</u> By induction on the strucure of α :

1. For $\alpha \in \Sigma_0$ we let $\gamma(\alpha)$ = true?; α; true?
2. For $\alpha = X?$ where $X \in \Phi$ we let $\gamma(\alpha) = \alpha$
3. For $\alpha = \beta \cup \delta$ we let $\gamma(\alpha) = \gamma(\beta) \cup \gamma(\delta)$

   by Lemma C1 $\gamma(\alpha) \in Qr$ and $\gamma(\alpha) \approx \beta \cup \delta = \alpha$
4. For $\alpha = \beta;\delta$ let $\gamma(\beta), \gamma(\delta)$ be the programs corresponding to β, δ by the induction hypothesis. Let $\bar\gamma$ be the program defined by Lemma C1 for $\gamma(\beta);\gamma(\delta)$, then we let $\gamma(\alpha) = \bar\gamma$. By Lemma C1 $\gamma(\alpha) \in Qr$ and $\gamma(\alpha) \approx \gamma(\beta);\gamma(\delta) \approx \beta;\delta = \alpha$
5. For $\alpha = \beta^*$: Let $\gamma(\beta)$ be the program for β by the induction hypothesis: $\gamma(\beta) \in Qr$ and $\gamma(\beta) \approx \beta$. Let $\bar\gamma$ be the program corresponding to $(\gamma(\beta))^*$ by

Lemma C1, then we let $\gamma(\alpha) = \bar{\gamma}$. Then by Lemma C1 $\gamma(\alpha) \in Qr$ and $\gamma(\alpha) \approx (\gamma(\beta))^* \approx \beta^* = \alpha$. □

<u>Definition</u>

$P_r = \{\alpha | \alpha \in \Sigma_u$ and for every word w, $w \in \alpha$

$w = W_0?uW_1? \ldots, W_{k-1}?uW_k? \quad k \geq 0 \ , \ W_i \in \Phi\}$

A program α in Σ_u is a regular exprssion over the alphabet $(\{u\} \cup T_\alpha)$. For α in Pr let M_α be the automaton defining α similar to the automata defining programs in Qr , where the only atomic program allowed is u .

<u>Lemma C3</u>

1. Let $\alpha,\beta \in Pr$ then $\alpha \cup \beta \in Pr$
2. Let $\alpha,\beta \in Pr$ then there exists a program $\bar{\gamma} \in Pr$ such that $\bar{\gamma} \approx \alpha;\beta$.
3. Let $\alpha \in Pr$ then there exists a program $\bar{\gamma} \approx Pr$ such that $\bar{\gamma} \approx \alpha^*$.

<u>Proof:</u> Similar to the proof of Lemma C1. □

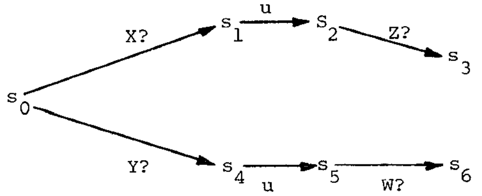
<u>Lemma C4</u>

For every program $\alpha \in \Sigma_u$ there exists a program $\gamma(\alpha) \in Pr$ such that $\gamma(\alpha) \approx \alpha$.

<u>Proof:</u> Similar to the proof of Lemma C2. □

We would like to define a <u>path deterministic</u> automaton (and later the complement automaton) in the sense that a <u>path</u> is "accepted" by at most a single sequence of states of the automaton.

Consider, for example, the automaton given by:



While this automaton is deterministic in the usual sense, it is not path deterministic, because, for example, the path $p = (p_0,p_1)$ where $p_0 \models X \wedge Y$, $p_1 \models Z \wedge W$ is "accepted" by both sequences (s_0,s_1,s_2,s_3) , (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 $X \wedge Y$ which are compatible with both.

As a first step in the definition of a path deterministic automaton, we define an extended alphabet of tests T^E_α such that for every state s in a model with α-paths there is one and only one formula $X \in T^E_\alpha$ such that $s \models X$.

<u>Definition</u>

Given $T_\alpha = \{X_1,\ldots,X_n\}$ an <u>atom</u> of T_α is a conjunction $\bigwedge_{i=1}^{n} Y_i$ where each Y_i is either X_i or a negation of X_i .

T^E_α - The extended alphabet for α is the set of all atoms of T_α .

<u>Definition</u>

Let $\alpha \in Pr$

$M_\alpha = \langle K_1 \cup K_2 \cup \{d\} \ , \ \{u\} \cup T_\alpha \ , \ q_0,\delta,F\rangle$ the automation defining α , then

M^E_α - <u>the (nondeterministic) extended automaton</u> for α , is defined by:

$M^E_\alpha = \langle \hat{K}_1 \cup \hat{K}_2 \cup \{d\} \ , \ \{u\} \cup T^E_\alpha, q_0, \delta^E, F\rangle$

where: $\hat{K}_1 = K_1 \cup \{e_1\}$, $\hat{K}_2 = K_2 \cup \{e_2\}$, and T^E is the extended alphabet for α , such that:

For $q \in \hat{K}_1$

$\delta^E(q,u) = d$

for $q \in K_1$,

$$\delta^E(q, \bigwedge_{i=1}^{n} Y_i) = \begin{cases} e_2 & \{Y_i\}_{i=1}^{n} \cap T_\alpha = \emptyset \\ \cup \ \{\delta(q,Y_j) \mid Y_j = X_j \in T_\alpha\} & \text{otherwise} \end{cases}$$

$$\delta^E(e_1, \bigwedge_{i=1}^{n} Y_i) = e_2$$

For $q \in \hat{K}_2$,

$$\delta^E(q, \bigwedge_{i=1}^{n} Y_i) = d$$

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

For $X \in T_\alpha^E \cup \{u\}$, $\delta^E(d,X) = d$

Lemma C5

Let $\alpha \in Pr$. Let α^E be a program defining the same set of words over $(\{u\} \cup T_\alpha)$ as M_α^E .

Then:

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

Proof:

1. Obvious from the form of M_α^E
2. a. Let p be a path, $p = (p_0,\ldots,p_k)$, $p \in \alpha$ then $\exists w \in \alpha$, $w = W_0?u \ldots .uW_k?$, and $(p_i) \in W_i?$, $0 \leq i \leq k$

For every $X \in T_\alpha$ either $(p_i) \in X?$ or $(p_i) \in (\sim X)?$. It follows that for every i , $0 \leq i \leq k$ there exist $Y_i^1 \ldots Y_i^n \in T_\alpha \cup \sim T_\alpha$ such that $Y_i^i = W_i$ and

$$(p_i) \in \bigwedge_{j=1}^{n} Y_i^j \qquad 0 \leq i \leq k$$

By the definition of M_α^E the word w^E given by

$$w^E = (\bigwedge_{j=1}^{n} Y_0^j)?u \ldots u(\bigwedge_{j=1}^{n} Y_k^j)?$$

is accepted by M_α^E and $(p_i) \in (\bigwedge_{j=1}^{n} Y_i^j)?$ for every $0 \leq i \leq k$; that is, $p \in \alpha^E$.

b. Let $p = (p_0,\ldots,p_k)$ $p \in \alpha^E$, then: $\exists w^E \in \alpha^E$, $w^E = W_0^E?u \ldots uW_k^E?$, $p \in w^E$, $W_i^E \in T_\alpha^E$; that is,

$$W_i^E = \bigwedge_{j=1}^{n} Y_i^j , \; Y_i^j \in T_\alpha \cup \sim T_\alpha .$$

By the fact that w^E is accepted by M_α^E it follows that for every i there exists a $j = j_i$ such that $Y_i^{j_i} \in T_\alpha$, otherwise $\delta^E(q, \bigwedge_{j=1}^{n} Y_j) = e_2$. Since $(p_i) \in W_i^E?$ then $(p_i) \in Y_i^{j_i}$, $0 \leq i \leq k$, and the word $w = Y_0^{j_0}?u \ldots uY_k^{k_j}?$ is accepted by M_α .

Thus we have $p \in \alpha$. □

Definition

Let $\alpha \in Pr$ and let M_α^E be the extended automaton for α .

$$M_\alpha^E = \langle K_1 \cup K_2 \cup \{d\},\{u\} \cup T^E,q_0,\delta^E,F\rangle .$$

The deterministic automaton for α, M_α^D , is defined from the nondeterministic one in the usual way by:

$$M_\alpha^D = \langle K_1^D \cup K_2^D \cup \{d\},\{u\} \cup T_\alpha^E,\{q_0\},\delta^D,F^D\rangle$$

where: $K_1^D = 2^{K_1}$, $K_2^D = 2^{K_2}$

$$F^D = \{\bar{q} | \bar{q} \in K_2^D \text{ and } \bar{q} \cap F \neq \emptyset\} .$$

For $\bar{q} \in K_1^D$ $\delta^D(\bar{q},u) = d$

and if $X \in T_\alpha^E$ $\delta^D(q,X) = \cup \{\delta^E(q,X) | q \in \bar{q}\}$

For $\bar{q} \in K_2^D$

$\delta^D(\bar{q},u) = \cup\{\delta^E(q,u) | q \in \bar{q}\}$

if $X \in T^E$ then $\delta^D(\bar{q},X) = d$

and for every $X \in T_\alpha^E \cup \{u\}$ $\delta^D(d,X) = d$.

Lemma C6

Let $\alpha \in Pr$. Consider α^D , a program defining the words accepted by M_α^D , the deterministic automaton for α . Then:

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

Proof:

1. Obvious by the form of M_α^D .
2. If $w \in \alpha^D$ then obvious $w \in \alpha^E$.

Suppose $w \in \alpha^E$, $w = W_0?u,\ldots,uW_k?$. By the form of M_α^E

$\{\delta^E(q,u)\} \subseteq K_1$, $\forall q \in K_2$

$\{\delta^E(q,X)\} \subseteq K_2$, $\forall q \in K_1$, $\forall X \in T_\alpha^E$,

from which it follows that w is accepted by M_α^D and we get $w \in \alpha^D$. □

Lemma C7

Let $\alpha \in Pr$ be a program over (Σ_0,Φ_0) with tests T_α. Then there exists a program $\tilde{\alpha} \in Pr$ over the aphabet $(\{u\} \cup T_\alpha^E)$ such that: For every path p in a model over (Σ_0,Φ_0) : $p \in \tilde{\alpha}$ iff $p \notin \alpha$.

Proof:

Let M_α^D be the deterministic automaton for α

$M_\alpha^D = <K_1 \cup K_2 \cup \{d\},\{u\} \cup T_\alpha^E,q_0,\delta^D,F>$

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

$\tilde{M}_\alpha = <K_1 \cup K_2 \cup \{d\},\{u\} \cup T_\alpha^E,q_0,\delta^D,K_2-F>$

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

Obviously $\tilde{\alpha} \in Pr$.

Consider a path $p = (p_0,\ldots,p_k)$ such that $p \notin \alpha$. Let $T_i \in T_\alpha^E$ $(0 \leq i \leq k)$ be the T_α atom which is true in p_i. (Obviously there exists one and only one element of T_α^E which is true in p_i.) Then the word $T_0uT_1u \ldots uT_k$ is not accepted by M_α^D $(p \notin \alpha)$. Neither does it lead M_α^E to the error state d. Hence it leads M_α^E to a state in K_2-F and is acceptable by $\tilde{M}_\alpha$. Therefore $p \in \alpha^D$.

Suppose $p \in \tilde{\alpha}$ and $p \in \alpha$ then $p \in \alpha^D$.

Let $p = (p_0,\ldots,p_k)$

Then: $\exists \tilde{w} \in \tilde{\alpha}$ $\quad \tilde{w} = \tilde{W}_0?u \ldots u\tilde{W}_k?$ $\quad p \in \tilde{w}$

$\exists w \in \alpha^D$ $\quad w = W_0?u\ldots W_k?$ $\quad p \in w$

$W_i \in T_\alpha^E \Rightarrow W_i = \bigwedge_{j=1}^{n} Y_i^j$, $\quad Y_i^j \in \{X_j,\sim X_j\}$ for every $X_j \in T_\alpha$

$\tilde{W}_i \in T_\alpha^E \Rightarrow \tilde{W}_i = \bigwedge_{j=1}^{n} \tilde{Y}_i^j$, $\quad Y_i^j \in \{X_j,\sim X_j\}$ for every $X_j \in T_\alpha$

We know that for every state p_i and every formula $X \in T_\alpha$, either $(p_i) \in X?$ or $(p_i) \in (\sim X)?$ but not both. It follows that $Y_i^j = \tilde{Y}_i^j$, $1 \leq j \leq n$, $0 \leq i \leq k$. But then $w = \tilde{w}$ is accepted by both M_α^D and $\tilde{M}_\alpha$, in contradiction with the definition of $\tilde{M}_\alpha$. □

We may thus summarize this sequence of C-Lemmas by:

Lemma C

If a BPL formula X is translatable into a PDL program in Σ_u in the sense of the Theorem, then so is $\sim X$.

Proof:

Let $\tau(X)$ be the PDL program translation of X. Then we let $\tau(\sim X) = \tilde{\tau}(X)$ as defined above.

Even though taking care of union and complementation automatically ensures closure under intersection, we do need this property in a more general setting. We therefore consider next closure under intersection.

Lemma D1

A. Let $\alpha \in \Sigma$, $\beta \in Pr$. Then there exists a program $\alpha \cap \beta \in Qr$ such that for every path p

$p \in \alpha \cap \beta$ iff $p \in \alpha$ and $p \in \beta$.

B. Let $\alpha, \beta \in Pr$. Then there exists a program $\alpha \cap \beta \in Pr$ such that for every path p

$p \in \alpha \cap \beta$ iff $p \in \alpha$ and $p \in \beta$

Proof:

A. Without loss of generality we can assume $\alpha \in Qr$ by Lemma C2. Let M_α, M_β be the automata defining α,β respectively.

$M_\alpha = <K_1^\alpha \cup K_2^\alpha \cup \{d^\alpha\},\Sigma_0 \cup T_\alpha,q_0^\alpha,\delta^\alpha,F^\alpha>$

$M_\alpha = <K_1^\beta \cup K_2^\beta \cup \{d^\beta\},\Sigma_0 \cup T_\beta,q_0^\beta,\delta^\beta,F^\beta>$

The automaton $M_{\alpha\cap\beta}$ is defined by:

$$M_{\alpha\cap\beta} = <\hat{K}_1 \cup \hat{K}_2 \cup \{d\}, \Sigma_0 \cup \{T_\alpha \wedge T_\beta\}, q_0, \hat{\delta}, F>$$

where:

$$\hat{K}_1 = K_1^\alpha \times K_1^\beta \qquad \hat{K}_2 = K_2^\alpha \times K_2^\beta$$

$$\hat{F} = F^\alpha \times F^\beta \qquad \hat{q}_0 = (q_0^\alpha, q_0^\beta)$$

Let $(q_\alpha, q_\beta) \in \hat{K}_1$

for $Z \in \{T_\alpha \wedge T_\beta\}$, $\hat{\delta}((q_\alpha,q_\beta),Z) =$

$$\begin{cases} \delta^\alpha(q_\alpha,X) \times \delta^\beta(q_\beta,Y) & Z = X\wedge Y, \quad X \neq Y \\ \delta^\alpha(q_\alpha,Z) \times \delta^\beta(q_\beta,Z) & Z \in T_\alpha \cap T_\beta \end{cases}$$

for $a \in \Sigma_0$, $\hat{\delta}((q_\alpha,q_\beta),a) = d$.

Let $(q_\alpha,q_\beta) \in \hat{K}_2$.

For $Z \in \{T_\alpha \wedge T_\beta\}$, $\hat{\delta}((q_\alpha,q_\beta),Z) = d$

for $a \in \Sigma_0$, $\hat{\delta}((q_\alpha,q_\beta),a) = \delta^\alpha(q_\alpha,a) \times \delta^\beta(q_\beta,u)$

for $X \in \{T_\alpha \wedge T_\beta\} \cup \Sigma_0$, $\hat{\delta}(d,X) = d$

Let $\alpha \cap \beta$ be a program defined by $M_{\alpha\cap\beta}$, then $\alpha \cap \beta \in Qr$. Let $p = (p_0,\dots,p_k)$ be a path. Then: $p \in \alpha\cap\beta \Leftrightarrow$ there exists a word w in the language defined by $\alpha \cap \beta$ such that $w = W_0?\ a_0 \dots a_{k-1}W_k?$, where for each $i = 0,\dots,k$ $(p_i) \in W_i?, (p_i,p_{i+1}) \in a_i$ and $W_i \in T_\alpha \wedge T_\beta$; that is, $\exists X_i \in T_\alpha$, $Y_i \in T_\beta$ such that, $W_i = X_i \wedge Y_i \Leftrightarrow w_\alpha = X_0?a_0\dots a_{k-1}X_k?$ is accepted by M_α , $w_\alpha = Y_0?u \dots uY_k?$ is accepted by M_β and $p \in w_\alpha$, $p \in w_\beta \Leftrightarrow p \in \alpha$ and $p \in \beta$.

B. The proof for $\alpha,\beta \in Pr$ is similar to the proof of A except that the alphabet for $M_{\alpha\cap\beta}$ is $\{u\} \cup \{T_\alpha \wedge T_\beta\}$ and, consequently, we replace Σ_0 by $\{u\}$ in the proof. □

Lemma D

If PL formulas, X and Y are translatable into PDL program in Σ_u in the sense of the theorem, then so is $X\wedge Y$.

Proof:

Let $\tau(X)$, $\tau(Y) \in \Sigma_u$ be the PDL translations of X,Y respectively. By Lemma C4, they are representable as Pr programs. Hence $\tau(X) \cap \tau(Y) \in Pr$ exists and yields $\tau(X\wedge Y) = \tau(X) \cap \tau(Y)$. □

Lemma E

If a BPL formula X is translatable into a PDL program in Σ_u in the sense of the theorem, then so is $f(X)$.

Proof:

Let $\tau(X)$ be the program corresponding to X , then let $\tau(fX) = (\tau(X) \cap true?);u^*$

Let $p = (p_0,\dots,p_\ell)$ be a path

$p \models fX \Rightarrow (p_0) \models X \Rightarrow (p_0) \in \tau(X)$. Then certainly $(p_0) \in true? \Rightarrow (p_0) \in \tau(X) \cap true? \Rightarrow p\in(\tau(X)\cap true?);u^*$

For the other direction, suppose $p \in (\tau(X) \cap true?);u^*$ then $(p_0) \in \tau(X) \cap true?$ $(p_0) \in \tau(X) \Rightarrow p_0 \models X \Rightarrow$ $p \models fX$ □

Definition

B_α , the partition set for a program α Pr , is defined as a subset of $\Sigma_u \times \Sigma_u$ by induction on α :

1. $B_u = \{(true?,u) , (u,true?)\}$
2. $B_{X?} = \{(true?,X?) , (X?,true?)\}$
3. $B_{\beta\cup\gamma} = B_\beta \cup B_\gamma$
4. $B_{\beta;\gamma} = \{(\beta_1,\beta_2;\gamma) \mid (\beta_1,\beta_2) \in B_\beta\}$
   $\cup \{(\beta;\gamma_1,\gamma_2) \mid (\gamma_1,\gamma_2) \in B_\gamma\}$
5. $B_{\beta*} = \{(ture?,ture?)\} \cup \{(\beta^*\beta_1,\beta_2\beta^*) \mid (\beta_1,\beta_2) \in B_\beta\}$

The intuitive meaning is that B contains all pairs of regular expressions which when concatenated yield α .

Lemma F1

Let $\alpha \in Pr$ then: for every path p and every p_1, p_2 such that $p = p_1 p_2$, $p \in \alpha$ iff $\exists(\alpha_1, \alpha_2) \in B_\alpha$ such that $p_1 \in \alpha_1$ and $p_2 \in \alpha_2$.

Proof:

Immediate. □

Lemma F2

Let $\alpha \in Pr$, then there exists a program $\alpha^c \in Pr$ (continuously α) such that: $p \in \alpha^c$ iff for every suffix (not necessarily proper) p' of p, $p' \in \alpha$.

Proof:

Let $\alpha^c = \sim((u;true?)^*;\tilde{\alpha})$. Then by Lemmas C7, C3 $\alpha^c \in Pr$. Let p be any path: $p \in \alpha^c$ iff $p \notin (u;true?)^*;\tilde{\alpha}$ iff for every suffix p' of p, $p' \notin \tilde{\alpha}$ iff for every suffix p' of p, $p' \in \alpha$. □

Lemma F3

Let $\alpha, \beta \in Pr$ then there exists a program $\mu(\alpha,\beta) \in Pr$ (the merge of α and β) such that: for every path p: $p \in \mu(\alpha,\beta)$ iff there exists q, a proper suffix of p such that $q \in \beta$, and for every r, such that r is a proper suffix of p and q is a proper suffix of r, $r \in \alpha$.

Proof:

Let B be a nonempty subset of the partition set B_α denote:

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

$t(B) = \{\alpha_2 | \exists\alpha_1, (\alpha_1,\alpha_2) \in B\}$

Then let

$$\mu_0(\alpha,\beta) = (u;\beta) \cup (u; \bigcup_{B \subseteq B_\alpha} \Big[(\bigcup_{h(B)} \alpha_1)^c; ((u;\beta) \wedge (\bigwedge_{t(B)} \alpha_2)) \Big]).$$

Clearly, $\mu_0(\alpha,\beta) \in \Sigma_u$ and by C4 there exists a program $\mu(\alpha,\beta) \in Pr$ such that $\mu_0(\alpha,\beta) \approx \mu(\alpha,\beta)$. Let p be a path, $p = (p_0,\ldots,p_k)$. Then $p \in \mu_0(\alpha,\beta)$ iff $p \in \mu_0(\alpha,\beta)$ iff $(p_1,\ldots,p_k) \in \beta$ or there exists some $B \subseteq B_\alpha$ such that

$$(p_1,\ldots,p_k) \in \bigcup_{B \subseteq B} ((\bigcup_{h(B)} \alpha_1)^c; ((u;\beta) \wedge (\bigwedge_{t(B)} \alpha_2)))$$

iff $(p_1,\ldots,p_k) \in \beta$ or $\exists B \subseteq B_\alpha$, $\exists j$, $1 \leq j \leq k$ such that $(p_1,\ldots,p_j) \in (\bigcup_{h(B)} \alpha_1)^c$, $(p_j,\ldots,p_k) \in (u;\beta)$, and $(p_j,\ldots,p_k) \in \alpha_2$, $\forall\alpha_2 \in t(B)$ (by Lemma D1, Part B) iff $(p_1,\ldots,p_k) \in \beta$ or $\exists B \in B_\alpha$, $\exists j$, $1 \leq j \leq k$ such that $(p_{j+1},\ldots,p_k) \in \beta$, and $\forall \ell$, $1 \leq \ell \leq j$, $(p_\ell,\ldots,p_j) \in \bigcup_{h(B)} \alpha_1$ and $(p_j,\ldots,p_k) \in \alpha_2$, $\forall\alpha_2 \in t(B)$ iff $(p_1,\ldots,p_k) \in \beta$ or $\exists B \subseteq B_\alpha$, $\exists j$, $1 \leq j \leq k$, such that $(p_{j+1},\ldots,p_k) \in \beta$ and $\forall \ell$, $1 \leq \ell \leq j$, $\exists \alpha_1 \in h(B)$ such that $(p_\ell,\ldots,p_k) \in \alpha_1;\alpha_2 = \alpha$ iff $\exists q$ proper suffix of p such that $q \in \beta$ and $\forall r$, r proper suffix of p, q proper suffix of r, $r \in \alpha$. □

Lemma F:

If BPL formulas X and Y are translatable into PDL programs, in Σ_u in the sense of the theorem then so is X suf$Y$.

Proof:

Let $\tau(X\text{suf}Y) = \mu(\tau(X),\mu(Y))$ of Lemma F3. □

Proof of the Main Theorem:

By induction on the structure of X :

By Lemmas A-F all we need to prove is the case $X = \langle\beta\rangle Y$. We prove first that for every BPL programs β there exists a PDL program $\gamma_\beta \in Qr$ such that $\forall r$, $r \in \beta$ iff $r \in \gamma_\beta$.

Define γ_β by induction on β as follows:

If $\beta \in \Sigma_0$ then $\gamma_\beta = \beta$

If $\beta = Z?$ then by the main induction and Lemma C4 there exists $\tau(Z) \in Pr$ such that $r \in \tau(Z)$ iff $r \models_p Z$. Thus $\gamma_{Z?} = \tau(Z)$

$\gamma_{\beta_1 \cup \beta_2} = \gamma_{\beta_1} \cup \gamma_{\beta_2}$

$\gamma_{\beta_1 ; \beta_2} = \gamma_{\beta_1} ; \gamma_{\beta_2}$

$\gamma_{\beta_1^*} = (\gamma_{\beta_1})^*$

To continue the proof, note that

$p \models X$ iff $\exists q \in \beta$ such that $pq \models Y$,

iff (induction hypothesis and construction of γ_β) $\exists q \in \gamma_\beta$ such that $pq \in \tau(Y)$,

iff $\exists q \in \gamma_\beta$, $\exists(\alpha_1,\alpha_2) \in B_{\tau(Y)}$ such that $p \in \alpha_1$, $q \in \alpha_2$ and q_0 = last state of p,

iff $\exists(\alpha_1,\alpha_2) \in B_{\tau(Y)}$ such that $p \in \alpha_1$, q_0 = last state of p and $q_0 \models \langle\gamma_\beta \cap \alpha_2\rangle true$ (the existence of $\gamma_\beta \cap \alpha_2$ is guaranteed by Lemma D1, part A),

iff $p \in \tau'(X)$, where $\tau'(X) \in \Sigma_u$ is defined as $\bigcup_{(\alpha_1,\alpha_2)\in B_{\tau(Y)}} \alpha_1 ; (\langle\gamma_\beta \cap \alpha_2\rangle true)?$

By Lemma C4 there exists $\tau(X) \in Pr$ with $\tau(X) \approx \tau'(X)$ and the proof is complete. □

Examples:

Let X be an atomic formula, and a an atomic program. The following are simplified forms of $\tau(Y)$ for some sample formulas Y .

1. $\tau(\langle a\rangle X) = X?;u^*;\ (\langle a\rangle true)?$
2. $\tau([a^*]all\ X) = X?;(u;X?)^*;[a^*;(\sim X)?;a^*]false)?$
3. $\tau(\langle a^*\rangle some\ X) = u^*;X?;u^* \cup u^*;(\langle a^*\rangle X)?$

Corollary 1: For every BPL formula X there exists a PDL formula $Y(X)$ such that X is valid, i.e., true of all finite paths in all models, iff $Y(X)$ is valid, i.e., true of all states in all models.

Proof:

We will show that $Z \in$ BPL is satisfiable iff $\langle\tau(Z)\rangle true$ is satisfiable in PDL. Accordingly, $Y(X)$ is taken to be $[\tau(\sim X)]false$. Indeed, assume $p \models_p Z$ 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 main theorem $p \in \tau(Z)$ in M' , and hence $p_0 \models \langle\tau(Z)\rangle true$ in M' . Conversely, if $s \models \langle\tau(Z)\rangle true$ in some model M then there is some path p in M , with $p_0 = s$, such that $p \in \tau(Z)$. By the main theorem $p \models_p Z$ in M .

□

Definition

A program model is a model in which any two states are connected by some atomic program.

Corollary 2.

For every BPL formula X there exists a PDL formula $Z(X)$ such that for every state s in any program model M: $M,(s) \models_p fX$ iff $M,s \models Z(X)$.

Proof:

Let s be a state in a program model M. By the main theorem $(s) \models_p fX$ in M iff $(s) \in \tau(fX)$ in the model extending M by interpreting u as the universal program. Let $a_1, \ldots, a_n$ be all the atomic programs in X. Define $\tau'(fX)$ to be the program $\tau(fX)$ where every appearance of the universal program u is replaced by $\bigcup_{i=1}^{n} a_i$, then by the definition of a program model $p \in \tau(fX)$ in the extended model iff $p \in \tau'(fX)$ in M for any path p. Since (s) is of length zero, it follows that $M,(s) \models_p fX$ iff $M,s \models \langle\tau'(fX)\rangle true$.

4. Complexity

The operator "chop" was defined in [HKP] by $p \models X \text{ chop } Y$ iff $\exists q,r$ such that $p = qr$ and $q \models X$, $r \models Y$. A formula containing chop can be easily translated to a PDL program by $\tau(X \text{ chop } Y)=\tau(X);\tau(Y)$. Thus, in particular our result gives a decision procedure for validity in BPL + chop + tests. A simple analysis of the translation algorithm presented herein shows a nonelementary complexity. It is unknown yet if validity in BPL + chop is elementary. By [CHMP] PL + chop is nonelementary. If BPL + chop can be proved to be nonelementary too, then any translation algorithm must indeed be of non-elementary time complexity.

REFERENCES

[BP] M. Ben Ari and A. Pnueli: "Temporal Logic Proofs of Concurrent Programs", TR 80-44 Tel Aviv University (1981)

[CHMP] A. Chandra, J. Halpern, A. Meyer, R. Parikh: "Equations between Regular Terms and an Application to Process Logic" STOC 81, 384-390.

[D] E.W. Dijkstra "A Discipline of Programming", Prentice-Hall 1976.

[FL] M.J. Fischer and R.E. Ladner "Propositional Dynamic Logic of Regular Programs", J. Comput. Syst. 18:2.

[H] D. Harel, "Two results on Process Logic", Info. Proc. Letters, 8:4 (1979), 195-198.

[HKP] D. Harel, D. Kozen and R. Parikh: "Process Logic: expressiveness, decidability, completeness", Proc. 21st FOCS, (1981).

[Ho] C.A.R. Hoare: "An Axiomatic basis for computer programming", CACM, 12 (1969), 576-580.

[N] H. Nishimura: "Descriptively Complete Process Logic", Acta Informatica, 14, 359-369 (1980).

[Pa] R. Parikh: "A Decidability result for second order process logic", Proc. 19th FOCS (1978), 177-183.

[Pr1] V.R. Pratt: "Semantical considerations of Floyd-Hoare Logic", Proc. 17th FOCS, (1976) 109-121.

[Pr2] V.R. Pratt: "Process Logic" Proc. 6th POPL, (1979), 93-100.

[R] M.O. Rabin: "Decidability of second order theories and automata on infinite trees" Trans. AMS 141 (1969), 1-35.