Theoretical Computer Science 38 (1985) 307-322
North-Holland

PROCESS LOGIC WITH REGULAR FORMULAS

D. HAREL*

Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel, and Department of Mathematics, Bar-Ilan University, Ramat-Gan, Israel

D. PELEG**

Department of Mathematics, Bar-Ilan University, Ramat-Gan, Israel

Communicated by E. Engeler
Received August 1983

Abstract. The paper proposes an extension RPL of the process logic PL of Harel, Kozen and Parikh (1980). The PL formula operators **f** and **suf** are replaced by the operators **chop** and **slice**, corresponding to Kleene's regular operators · and *, thus enabling formulas to express regular sets of paths. The main result is that, in expressive power, PL < RPL, the hard part being in showing that PL ≤ RPL. It is argued that this version of PL comes closer to the desired goal of a natural and powerful (yet decidable) logic for reasoning about the ongoing behavior of programs.

1. Introduction

Ever since the work of Engeler [5], researchers have tried to provide formalized logical tools to enable reasoning about programs. The idea is to incorporate such proof methods as those of Floyd [6] into formal logics which can assert many properties of interest. Following the 'input/output' school of thought cultivated by Floyd [6], Hoare [13], Manna [15] and others, the Warsaw-based system of *algorithmic logic* (see [2, 22]) and the similar USA-based system of *dynamic logic* (see [20, 9]) emerged. In particular, the propositional version of dynamic logic, PDL (see [7]) seems to have become recognized as an appropriate propositional-level tool for the input/output mode of reasoning about sequential programs. Moreover, PDL has a decidable validity problem and a simple complete axiomatization [14]. It was clear, however, that to reason about concurrent or nonterminating programs, or even to reason conveniently about some aspects of the ongoing behaviour of sequential programs, it would be necessary to extend PDL (and with it perhaps the first-order versions too) to make assertions about the activity *during* computations and not only at their start or end.

* Research supported in part by a Bat-Sheva Fellowship.

** The paper is based on part of this author's M.Sc. Thesis written at Bar-Ilan University.

Some initial work in this direction appears in [21], where the connectives 'during' and 'throughout' were added to PDL, in [10], where PDL was shown to be unable to express such connectives, and in [17], where a powerful logic for reasoning about states and paths and their properties was shown decidable. Independently, the *temporal logic* approach to reasoning about concurrent and nonterminating programs was being developed (see [19, 8]). A fundamental difference in the approaches between dynamic and temporal logics is the explicit naming of programs in the former (as in $[\alpha]\langle\beta\rangle Y$; read "after any execution of program α it is possible to execute β making Y true upon termination"), and the presence of a single implicit program to which the entire formula refers in the latter (as in $\Box \Diamond Y$; read "after executing any number of steps of the program it is possible to execute some more and satisfy Y").

In [16] it was first suggested to combine both approaches, and to allow, say, the formula $\langle\alpha\rangle \Box Y$, read "it is possible to execute α such that Y is true after any number of steps, i.e., throughout that execution". It became clear that, for such a suggestion to work, the mode of thought would have to undergo a transition from the input/output approach, captured by binary relations on states, to the 'ongoing' approach captured by *paths*, i.e., sequences of states. Furthermore, to be acceptable, the propositional version of any such proposed logic should be decidable, should be at least as powerful as those proposed as first approximations in [21, 17, 16], and, above all, should be in some sense the '*right*' logic for talking about paths. This last property means that, on the one hand, it should be built up using natural and tractable path connectives but, on the other, should be able to express as large a class of path formulas as is reasonably possible.

In response to this need, a logic, called *process logic* (or PL) following Pratt [21], was proposed in [12]. It was indeed a combination of PDL and the propositional temporal logic TL, was as powerful as any previously suggested version, and was also shown to be decidable and to possess a finite complete axiomatization. In a certain sense, the last requirement above was also satisfied: in [8], TL was shown to be precisely as powerful as the first-order theory of linear order, a fact taken to be evidence that TL was 'right'. In PL, it was this 'expressively complete' version of TL which was combined with the PDL apparatus and thus, in the same sense, PL was 'right' too.

Nevertheless, the path operators of PL are not all that natural. The central path operator, X **suf** Y, deriving from the X **until** Y of TL, is true in a path p if there is a suffix q of p which satisfies Y, and all suffixes of p of which q is a suffix satisfy X. This is clearly a complicated and asymmetric operator. In addition there is the construct $\mathbf{f}X$, true in p if X is true in the first state of p regarded as a path of length 0. Moreover, Wolper [26] showed that there are natural path properties not expressible in TL, and suggested adding to TL operators corresponding to right linear grammars, in effect enabling one to say "X is true in the regular set of paths S".

In this paper we make another step towards defining the 'right' process logic, by extending PL in a way similar to that adopted by Wolper for TL. We define RPL,

for Regular Process Logic, in which the operators **f** and **suf** are replaced by **chop** and **slice**, corresponding essentially to Kleene's regular operations of concatenation and star. Thus, e.g., X **chop** Y is true in path p if p consists of the fusion of q and r (i.e., concatenation with an overlapping common state), X is true in q and Y in r. In this way, the regular operations on programs, $\alpha \cup \beta$, $\alpha\beta$, α^*, have natural counterparts on formulas: $X \vee Y$, X **chop** Y and **slice** X. This has the immediate advantage of enabling one to use any operator which is expressible by regular means, for example the 'shuffle' operator $X \| Y$, which shuffles the paths satisfying X and Y and is of importance in reasoning about concurrently executing processes.

Our main technical result, whose proof takes up most of the paper, is that in the presence of these regular operators on formulas, the old ones, **f** and **suf** are redundant. In other words, PL $\leqslant$ RPL. Since an argument similar to Wolper's [26] can be used to show that regular operators can actually say more, we obtain PL $<$ RPL.

RPL is shown to be decidable (but nonelementary), and it is argued that the regular operators could perhaps yield an easier completeness proof than that of [12]. Furthermore, in the interest of arguing that RPL is closer to being 'right' than previously proposed logics, we observe that a slight extension of RPL results in a highly undecidable validity problem.

The approach is taken one step further in Section 5. There it is shown that since in RPL the operators on both programs and formulas are the regular ones (in addition to $\neg$, which is complementation relative to the set of all paths), and since both programs and formulas are interpreted over paths, one can combine both of these and define a two-sorted logic R, uniformly closed under the PDL diamond operator $\langle X \rangle Y$, regular operations and path complementation. The one sort corresponds to atomic properties of states, and the other to atomic (i.e., binary) transitions between states. We show that while naively doing this results in an undecidable language by [4], if one disallows complementation inside the $\langle\ \rangle$, this unified language is decidable and is strictly stronger than the restriction of RPL to binary atomic programs.

2. Definitions

Process Logic, PL, is interpreted over path models, in which one may talk about (finite or infinite) paths of states. All formulas of PL are path formulas, i.e., a formula X is either true or false in path p. We assume familiarity with the basic notions from [12] but provide a brief description for self-containment.

The following are basic notions regarding paths. The first and last state of a path p are denoted first(p) and last(p), respectively. If p and q are two paths such that last(p) = first(q), then $p \odot q$ denotes the fusion of p and q. If last(p) $\neq$ first(q), then $p \odot q$ is not defined. $p \cdot q$ denotes the concatenation of p and q, which is always defined. For example, if $p = s_1 s_2$ and $q = s_2 s_3$, then $p \odot q = s_1 s_2 s_3$, $q \cdot p = s_2 s_3 s_1 s_2$, and $q \odot p$ is not defined. For sets of paths G, H, define $G \odot H$ and $G \cdot H$ in the

usual way, and also define $G^i = G \cdot \ldots \cdot G$, i times; $G^{\textcircled{1}}$ is the same with $\odot$, $G^+ = \bigcup_{i=1}^{\infty} G^i$, and $G^{\oplus} = \bigcup_{i=1}^{\infty} G^{\textcircled{1}}$. The same notation will be used for words and languages.

Syntax

The basic elements of PL are a set AF of atomic formulas (denoted by $P, Q, \ldots$) and a set AP of atomic programs (denoted by $a, b, \ldots$). For programs α, β and formulas X, Y, $\alpha \cup \beta$, $\alpha \cdot \beta$, and α^* are also programs, and $X \vee Y$, $\neg X$, $\langle \alpha \rangle X$, $\mathbf{f}X$ and X **suf** Y are formulas.

Semantics

A *path model* is a triple $M = (S, \models, \rho)$ where S is a set of *states*, $\models$ is a *satisfiability relation* for primitive propositions, and ρ is an assignment of sets of paths to primitive programs. A path satisfies primitive proposition P iff its first state does. We write $p \models P$ if path p satisfies primitive proposition P, and $p \in \rho_a$ if p is a member of the set of paths assigned to primitive program a.

The relations $\models$ and ρ are extended to compound propositions and programs according to the following rules:

$\rho_{\alpha\beta} = \rho_\alpha \odot \rho_\beta$,

$\rho_{\alpha \cup \beta} = \rho_\alpha \cup \rho_\beta$,

$\rho_{\alpha^*} = \bigcup_i \rho_\alpha^{\textcircled{1}}$,

$p \models X \vee Y$ iff $p \models X$ or $p \models Y$,

$p \models \neg X$ iff not $p \models X$,

$p \models \langle \alpha \rangle X$ iff there is a path $q \in \not{p}_\alpha$ such that $p \odot q \models X$,

$p \models \mathbf{f}X$ iff $\text{first}(p) \models X$,

$p \models X$ **suf** Y iff there is q such that

(i) q is a proper suffix of p and $q \models Y$, and

(ii) for each proper suffix r of p, if q is a proper suffix of r, then $r \models X$.

Define the operator $\mathbf{L}_0$ by: $p \models \mathbf{L}_0$ iff p is of length 0. $\mathbf{L}_0$ is definable in PL by $\mathbf{L}_0 \equiv \neg(0$ **suf** $1)$. Here 1 and 0 stand for *true* and *false*. Now, define the two additional connectives **chop** and **slice** as follows:

$p \models X$ **chop** Y iff there are paths q, r such that $p = q \odot r$, $q \models X$, and $r \models Y$.

$p \models$ **slice** X iff there are $q_1, \ldots, q_n$ for some $n \geq 1$ such that $p = q_1 \odot \cdots \odot q_n$, and, for all $1 \leq i \leq n$, $q_i \models X$.

Let PL^+ be defined just like PL but with these additional operators added, and let RPL be defined just like PL^+ but without **f** or **suf**, though with $\mathbf{L}_0$.

We need some notions regarding languages and regular sets. First, we define an analog of **suf** for languages. Given two languages L_1, L_2, L_1 **suf** L_2 will denote $\{x \mid$ there is some proper suffix y of x such that $y \in L_2$, and for each proper suffix z of x, if y is a proper suffix of z, then $z \in L_1\}$.

Next, we define some classes of regular sets of finite and infinite words. The class of *λ-free regular expressions* over Σ, $R_\lambda(\Sigma)$, is defined exactly as are standard regular expressions, except that + is used instead of *. The set $L(R_\lambda(\Sigma))$ of languages defined by expressions in $R_\lambda(\Sigma)$ is clearly closed under union, intersection, concatenation, +, complementation with respect to Σ^+, **suf**, fusion, and $\oplus$.

We now turn to deal with infinite words. Σ^ω will denote the set of infinite words of order type ω over Σ. For a set $L \subseteq \Sigma^*$, L^ω denotes the set of all words $x \in \Sigma^\omega$ which are an infinite concatenation of nonempty words in L. A set L is *ω-regular* if it is a finite union of sets of the form $U \cdot V^\omega$, where U and V are regular sets. The collection of all ω-regular sets is denoted by $LR_\omega(\Sigma)$. It is known (cf. [3]) that $LR_\omega(\Sigma)$ is equivalent to the class of all sets definable by McNaughton automata.

We shall need yet another system of regular sets of infinite words, more suitable for our purposes. Define the collection of *$\sim$-regular* expressions (denoted $R_\sim(\Sigma)$) as follows: (1) $\emptyset \in R_\sim(\Sigma)$; (2) if $U \in R_\lambda(\Sigma)$ and $A, B \in R_\sim(\Sigma)$, then $A \cup B$, $U \cdot A$, $\sim A \in R_\sim(\Sigma)$ (where $\sim A$ denotes $\Sigma^\omega - A$). It is known [25] that $L(R_\sim(\Sigma)) = LR_\omega(\Sigma)$, and therefore that $L(R_\sim(\Sigma))$ is closed under union, intersection, complementation with respect to Σ^ω, **suf**, and fusion on the left with λ-free regular sets.

3. Results

Theorem 3.1. $PL < PL^+$.

Proof. The proof is similar to that of Wolper [26] for TL. Let **even** = **slice**($\mathbf{L}_2$) $\in PL^+$. Here $\mathbf{L}_2$ (definable in PL^+ as 0 **suf** (0 **suf** $\mathbf{L}_0$) is true precisely in paths of length 2 (i.e., consisting of three states). The formula **even** says that a path is of even length. We show that this property cannot be expressed in PL, by constructing a model M with only one state s, and empty assignments of ρ_a and $\not\models P$ for all $a \in$ AP, $P \in$ AF. For a formula $X \in$ PL, define nest(X) as the maximal depth of nesting of the **suf** operator in X. The following two claims are easy to prove and yield the result.

Claim 3.1.1. $p \models$ **even** *iff* $p = (s^{2n+1})$ *for some* $n \geq 1$.

Claim 3.1.2. *Let* $X \in$ PL. *For every* $i >$ nest(X), $(s^i) \models X$ *iff* $(s^{i+1}) \models X$. □

The remedy, proposed by Wolper for the similar lack of expressiveness in TL, was defining a family of extended operators, each corresponding to some right linear

grammar. His *extended* TL, ETL, is thus equivalent in expressive power to program-free RPL.

Our main theorem, whose proof is sketched in the next section, is the following.

Theorem 3.2. $\mathrm{PL}^+ \equiv \mathrm{RPL}$.

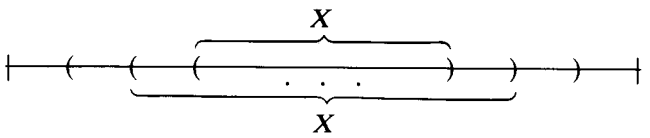
Corollary 3.2.1. $\mathrm{PL} < \mathrm{RPL}$.

We note the following theorem, to be contrasted with the following facts: (i) for the weaker but more succint PL the result is not known to hold, and (ii) for TL the problem is PSPACE-complete [23].

Theorem 3.3. *The validity problem for* RPL *is decidable but is non-elementry even for program-free formulas.*

Proof. Decidability is obtained by an easy extension of the reduction of PL to SnS in [12] (see [18]). That there is no elementary-time decision procedure even for program-free formulas follows from a straightforward linear time reduction from the emptiness problem for regular expressions with complementation (see [1]). □

Slight extensions of RPL are highly undecidable. For example, let the operator ζX be defined by $p \models \zeta X$ iff all subpaths q of p obtained by deleting from p an equal number of states from either end, satisfy X. This is depicted by the following diagram.



Theorem 3.4. *The validity problem for* RPL *with the additional operator* ζ *on formulas is* Π_1^1*-complete.*

Proof. Let X_0 be defined as

$$\mathbf{L}_0 \vee (\mathbf{slice}((\mathbf{L}_0 \wedge P)\mathbf{chop}\,\mathbf{L}_1))\mathbf{chop}(\mathbf{slice}\,(\mathbf{L}_1\,\mathbf{chop}(\mathbf{L}_0 \wedge \neg P))),$$

for some $P \in \mathrm{AF}$, and where $\mathbf{L}_1$ (definable in RPL) is true precisely in paths of length 1. The formula ζX_0 can be seen to hold in all paths of even length for which P is true along the first half, and false along the last half (its value in the middle state is immaterial). This set $\{P^i\mathbf{L}_0(\neg P)^i \mid i \geq 0\}$ and its dual obtained by switching the roles of P and $\neg P$ in X_0, can serve to reduce to this extension of RPL a recurring-dominoes problem, as is done in [11, Theorem 4.10]. □

4. Proof of Theorem 3.2

We show in this section that $\mathrm{PL}^+ \leq \mathrm{RPL}$, and hence establish $\mathrm{PL}^+ = \mathrm{RPL}$, and $\mathrm{PL} < \mathrm{RPL}$.

Define the abbreviation $\mathbf{s}X = \mathbf{L}_0 \wedge X$. We regard $\mathbf{s}X$ as a 'local appearance' of X. The following immediate lemma shows that the connective **f** is redundant in PL$^+$.

Lemma 4.1. *For any* $X \in \mathrm{PL}^+$, $\mathbf{f}X \equiv (\mathbf{s}X)$ **chop** 1.

Unfortunately, no such uniform translation is known for **suf**. The proof that **suf** is nevertheless eliminable goes along the following lines. First, a minor change is made in both PL$^+$ and RPL, with no effect on expressiveness. The semantic rule for atomic formula P is rephrased to read: $p \models P$ iff $\exists s(p = (s)$ and $s \models P)$. Clearly, the 'old' P can still be expressed (as P **chop** 1), while the 'new' P is expressed in the original version by $\mathbf{s}P$.

Next, a sublanguage of RPL, named **loc** PL, is defined. A formula of the form $\langle\alpha\rangle X$ may appear in **loc** PL only in local form, i.e., in subformulas of the form $\mathbf{s}(\langle\alpha\rangle X)$. The operators **chop** and **slice** do not appear in **loc** PL at all. Instead, two new connectives are used, namely, **rchop** and **rslice**. These are definable using the former, and are related to them (by the semantic rules) precisely as concatenation $(\cdot)$ relates to fusion $(\odot)$. All this enables us to use conjunctions of literals (i.e., atomic formulas or their negations) called *atoms* (see [14, 24]) as the letters of an alphabet Σ, and to establish a natural correspondence between $R_\lambda(\Sigma)$, $R_\sim(\Sigma)$ and sublanguages of program-free-**loc** PL. This will yield the closure of program-free-**loc** PL under **chop**, **slice**, and **suf**, and hence will imply program-free-PL$^+ \leqslant$ program-free-**loc** PL.

The presence of programs complicates the situation, since PL$^+$ may in general use programs non-locally (i.e., outside the **s**-connective). Nevertheless, we exhibit, for each α and X, a formula of **loc** PL equivalent to $\langle\alpha\rangle X$. The heart of the proof is a rather complex case analysis showing that any formula in **loc** PL can be decomposed into a collection of pairs of formulas which exhaust all possible ways of satisfying X by compound paths.

Definition of **loc** PL

First, define the following abbreviations:

X **rchop** Y abbreviates $(X$ **chop** $\mathbf{L}_1)$ **chop** Y,

rslice X abbreviates $X \vee ($**slice** $(X$ **chop** $\mathbf{L}_1))$ **chop** X,

The language **loc** PL is taken to be the sublanguage of RPL, defined by the following rules:

(1) AF $\subseteq$ **loc** PL.

(2) If α is a program and $X \in$ **loc** PL, then $\mathbf{s}(\langle\alpha\rangle X), \mathbf{s}([\alpha]X) \in$ **loc** PL.

(3) If $X, Y \in$ **loc** PL, then $\neg X$, $X \vee Y$, X **rchop** Y, **rslice** $X \in$ **loc** PL.

Let $\Gamma = \{P_1, \ldots, P_k\} \subseteq$ AF. An *atom* for Γ is a formula $\sigma = \mathbf{s}(A_1 \wedge \cdots \wedge A_k)$ where $A_i \in \{P_i, \neg P_i\}$. Denote by Σ_Γ the set of atoms for Γ.

For any model $M=(S,\models,\rho)$, let $h:(\Sigma_\Gamma^+\cup\Sigma_\Gamma^\omega)\to 2^{S^+\cup S^\omega}$ be defined by

$$h(\sigma)=\{P\mid P\models\sigma\} \qquad \text{for } \sigma\in\Sigma_\Gamma,$$

$$h(\sigma_1\sigma_2\ldots)=h(\sigma_1)h(\sigma_2)\ldots \quad \text{for } \sigma_1,\sigma_2,\ldots\in\Sigma_\Gamma.$$

Note that, for each $p\in h(\sigma)$, $|p|=0$. The function h induces a partition on the set $2^{S^+\cup S^\omega}$ of paths in M, since if $w\neq w'$, then $h(w)\cap h(w')=\emptyset$.

Over Σ_Γ, we have the set of λ-free regular expressions, $R_\lambda(\Sigma_\Gamma)$, and the set of infinite regular expressions, $R_\sim(\Sigma_\Gamma)$. Define functions I, F as follows:

$I: R_\sim(\Sigma_\Gamma)\to$ **loc** PL

$I(\emptyset)=0$,

For $A_1, A_2\in R_\sim(\Sigma_\Gamma)$ and $U\in R_\lambda(\Sigma_\Gamma)$:

$I(A_1\cup A_2)=I(A_1)\vee I(A_2)$,

$I(U\cdot A_1)=F(U)$ **rchop** $I(A_1)$,

$I(\sim A_1)=(\neg I(A_1))\vee$ **inf**;

$F: R_\lambda(\Sigma_\Gamma)\to$ **loc** PL

$F(\emptyset)=0$,

$F(\sigma)=\sigma$ for every $\sigma\in\Sigma_\Gamma$,

for $U_1, U_2\in R_\lambda(\Sigma_\Gamma)$:

$F(U_1\cup U_2)=F(U_1)\vee F(U_2)$,

$F(U_1\cdot U_2)=F(U_1)$ **rchop** $F(U_2)$,

$F(U_1^+)=$ **rslice** $F(U_1)$.

Here, **fin** abbreviates $\mathbf{L}_0\vee(1$ **rchop** $\mathbf{L}_0)$ and it is true precisely in all finite paths, while **inf** is its negation, holding precisely in infinite paths.

Now define the following sublanguages of **loc** PL:

$$\textbf{inf PL}=\{X\mid\exists\Gamma=\{P_1,\ldots,P_k\}\subseteq \text{AF}, \exists A\in R_\sim(\Sigma_\Gamma), (X=I(A))\},$$

$$\textbf{fin PL}=\{X\mid\exists\Gamma=\{P_1,\ldots,P_k\}\subseteq \text{AF}, \exists U\in R_\lambda(\Sigma_\Gamma), (X=F(U))\}.$$

Note that I and F are $1-1$ and onto **inf** PL and **fin** PL respectively. The idea is that $p\models X$ for $X\in$ **inf** PL (respectively **fin** PL) only if p is infinite (respectively finite).

The following lemma can easily be proved by induction on X.

Lemma 4.2. *For every path p, and every* $X\in$ **loc** PL,

(1) *if* $X\in$ **fin** PL, *then* $p\models X\Leftrightarrow h^{-1}(p)\in L(F^{-1}(X))$, *and*

(2) *if* $X\in$ **inf** PL, *then* $p\models X\Leftrightarrow h^{-1}(p)\in L(I^{-1}(X))$.

Lemma 4.3. *For every program-free formula* $X \in$ **loc** PL *there are formulas* $X_F \in$ **fin** PL *and* $X_I \in$ **inf** PL *such that* $X \equiv X_F \vee X_I$.

Proof. The claim will be shown by induction on the structure of X, relative to the set $\Gamma = \{P_i \mid P_i \in \mathrm{AF}, P_i \text{ appears in } X\}$.

Case ($X = P_i \in \mathrm{AF}$): Take $X_F = \bigvee_{\{\sigma \mid P_i \text{ appears positively in } \sigma\}} \sigma$, and $X_I = 0$.

In all composite cases assume (by the inductive hypothesis) the existence of appropriate Z_F, $W_F \in$ **fin** PL and Z_I, $W_I \in$ **inf** PL for subformulas Z, W of X.

Case ($X = Z \vee W$): Take $X_F = Z_F \vee W_F$ and $X_I = Z_I \vee W_I$.

Case ($X = \neg Z$): $R_\lambda(\Sigma_\Gamma)$ is closed under λ-free complementation, and therefore there exists a λ-free regular expression U such that $L(U) = \Sigma_\Gamma^+ - L(F^{-1}(Z_F))$. Take $X_F = F(U)$ and $X_I = \neg Z_I \wedge$ **inf**.

Case ($X = Z$ **rchop** W): Take $X_F = Z_F$ **rchop** W_F and $X_I = Z_F$ **rchop** W_I.

Case ($X =$ **rslice** Z): Take $X_F =$ **rslice** Z_F and $X_I = Z_I \vee ($**rslice** $Z_F)$ **rchop** Z_I.

In all cases, the proof that $X_F \in$ **fin** PL, $X_I \in$ **inf** PL, and $X \equiv X_F \vee X_I$ is straightforward. □

Lemma 4.4. *Program-free-***loc** PL *is closed under* **suf**, **chop**, *and* **slice**.

Proof. Let X and Y be program-free-**loc** PL formulas, with their I and F portions as in Lemma 4.3, and let

$$\Gamma = \{P_i \mid P_i \in \mathrm{AF}, P_i \text{ appears in } X_F, X_I, Y_F, \text{ or } Y_I\}.$$

By the closure properties of $R_\sim$ and R_λ, there are regular expressions U_1, U_2, $U_3 \in R_\lambda(\Sigma_\Gamma)$ and A_1, A_2, $A_3 \in R_\sim(\Sigma_\Gamma)$ such that

$$L(A_1) = L(I^{-1}(X_I) \textbf{ suf } I^{-1}(Y_I)),$$

$$L(A_2) = L(F^{-1}(X_F) \odot I^{-1}(Y_I)),$$

$$L(A_3) = L(I^{-1}(X_I) \cup (((F^{-1}(X_F))^{\oplus}) \odot I^{-1}(X_I))),$$

$$L(U_1) = L(F^{-1}(X_F) \textbf{ suf } F^{-1}(Y_F)),$$

$$L(U_2) = L(F^{-1}(X_F) \odot F^{-1}(Y_F)),$$

$$L(U_3) = L((F^{-1}(X_F))^{\oplus}).$$

Now take the final formulas to be $F(U_i) \vee I(A_i)$ for $i = 1, 2, 3$, and the result follows from Lemma 4.2. □

Lemma 4.5. *Program-free-*$\mathrm{PL}^+ \leq$ *program-free-***loc** PL.

Proof. The proof immediately follows from Lemma 4.4. □

We now proceed to deal with the presence of programs. First define some new operators:

$$\bar{\mathbf{L}}_0 = \neg\mathbf{L}_0,$$

$$X \wedge Y = \neg(\neg X \vee \neg Y),$$

$$X \textbf{ drchop } Y = \neg(\neg X \textbf{ rchop } \neg Y),$$

$$\textbf{drslice } X = \neg(\textbf{rslice } \neg X).$$

In the sequel we refer to these as the *duals* of $\mathbf{L}_0$, $\vee$, **rchop**, and **rslice**, respectively, and vice versa. Clearly,

$$p \models \bar{\mathbf{L}}_0 \quad \Leftrightarrow |p| > 0,$$

$$p \models X \wedge Y \quad \Leftrightarrow p \models X \text{ and } p \models Y,$$

$$p \models X \textbf{ drchop } Y \Leftrightarrow \forall q, r(p = q \cdot r \Rightarrow (q \models X \text{ or } r \models Y)),$$

$$p \models \textbf{drslice } X \quad \Leftrightarrow \forall n \geq 1, q_1, \ldots, q_n\ (p = q_1 \cdot \ldots \cdot q_n \Rightarrow \exists i,$$
$$(1 \leq i \leq n \text{ and } q_i \models X)).$$

A formula will be called *local* if it is either in AF or is of the form $\mathbf{s}X$. In this sense, every appearance of $\langle\alpha\rangle X$ (or $[\alpha]X$) in **loc** PL is in local form.

For formulas in **loc** PL we now eliminate negations occurring non-locally (except in $\mathbf{L}_0$ and $\bar{\mathbf{L}}_0$) according to Table 1, in which $\hat{X}$ and $\tilde{X}$ are, respectively, equivalent to X and $\neg X$.

Table 1. Elimination of negations.

For $X =$	let $\hat{X} =$	and $\tilde{X} =$	
P	P	$\mathbf{s}(\neg P) \vee \bar{\mathbf{L}}_0$	
Y **op** Z	$\hat{Y}$ **op** $\hat{Z}$	$\tilde{Y}$ **dual** $\tilde{Z}$	for **op** $\in \{\vee, \wedge,$ **rchop, drchop**$\}$
op Z	**op** $\hat{Z}$	**dual** $\tilde{Z}$	for **op** $\in$ **{rslice, drslice}**
$\mathbf{s}(\langle\alpha\rangle Y)$	$\mathbf{s}(\langle\alpha\rangle \hat{Y})$	$\bar{\mathbf{L}}_0 \vee \mathbf{s}([\alpha]\tilde{Y})$	
$\mathbf{s}([\alpha]Y)$	$\mathbf{s}([\alpha]\hat{Y})$	$\bar{\mathbf{L}}_0 \vee \mathbf{s}(\langle\alpha\rangle\tilde{Y})$	
$\neg Y$	$\tilde{Y}$	$\hat{Y}$	

Let $X \in$ **loc** PL. A finite set of pairs of formulas

$$D_X = \{\langle X_i^1, X_i^2\rangle \mid 1 \leq i \leq m_X\} \subseteq \textbf{loc PL}$$

is called a *proper decomposition* of X if for all paths q and r,

$$q \cdot r \models X \Leftrightarrow \text{for some } i, 1 \leq i \leq m_X, q \models X_i^1 \text{ and } r \models X_i^2.$$

The following lemma contains the main technical fact needed in the sequel. Its proof is rather tedious and appears in Appendix A.

Lemma 4.6. *Every formula X of* **loc** PL *admits a proper decomposition.*

Using Lemma 4.6 it is now not too hard to finish off the proof of Theorem 3.2. As mentioned we need a **loc** PL equivalent of $\langle\alpha\rangle X$.

Lemma 4.7. *Given a formula $X \in$* **loc** PL, *let $D_X = \{\langle X_i^1, X_i^2\rangle \mid 1 \leq i \leq m_X\}$ be a proper decomposition of X, and let $X^\alpha = \mathbf{s}(\langle\alpha\rangle X) \vee \bigvee_{1\leq i\leq m_X} (X_i^1 \textbf{ rchop } (\mathbf{s}(\langle\alpha\rangle X_i^2)))$. Then $X^\alpha \equiv \langle\alpha\rangle X$.*

Proof. ($\Leftarrow$): For a path p, let p_i, for $i \leq |p|$, denote the ith state on p, and let ${}_ip_j$, for $i \leq j \leq |p|$, denote $(p_i, \ldots, p_j)$. Assume $p \models \langle\alpha\rangle X$, i.e., there is some q in ρ_α such that $p \odot q \models X$. If $|p| = 0$, then $p \models \mathbf{s}(\langle\alpha\rangle X)$, and therefore $p \models X^\alpha$. Otherwise, we have ${}_1p_{|p|-1} \models X_i^1$ and $p_{|p|} \odot q \models X_i^2$, hence $p_{|p|} \models \mathbf{s}(\langle\alpha\rangle X_i^2)$. Therefore, $p = {}_1p_{|p|-1} \cdot p_{|p|} \models X_i^1$ **rchop** $(\mathbf{s}(\langle\alpha\rangle X_i^2))$, and again we have $p \models X^\alpha$.

($\Rightarrow$): Assume $p \models X^\alpha$. If $p \models \mathbf{s}(\langle\alpha\rangle X)$, then clearly $p \models \langle\alpha\rangle X$. Otherwise, there is some $1 \leq i \leq m_X$ for which $p \models X_i^1$ **rchop** $(\mathbf{s}(\langle\alpha\rangle X_i^2))$. Therefore, $|p| > 0$, ${}_1p_{|p|-1} \models X_i^1$ and $p_{|p|} \models \langle\alpha\rangle X_i^2$. The latter means that there is some t in ρ_α such that first(t) = last(p) and $t \models X_i^2$. Since D_X is a proper decomposition, from ${}_1p_{|p|-1} \models X_i^1$ and $t \models X_i^2$ we obtain $p \odot t = {}_1p_{|p|-1} \cdot t \models X$, and hence $p \models \langle\alpha\rangle X$. □

For the following lemma, denote by $X|_Z^Y$ the formula obtained from X by replacing every occurrence of Y with Z. The lemma completes the proof that $\mathrm{PL}^+ \leq$ **loc** PL.

Lemma 4.8. *For every formula $X \in \mathrm{PL}^+$ there is a formula $X' \in$* **loc** PL *so that $X' \equiv X$.*

Proof. The proof follows by induction on the structure of X.

Case (atomic P): Take $X' = P$.

Case ($Y \vee Z, \neg Y$): Set $(Y \vee Z)' = Y' \vee Z'$ and $(\neg Y)' = \neg Y'$.

Case ($\langle\alpha\rangle Y$): By Lemmas 4.6 and 4.7, and since $Y' = Y$ from the inductive hypothesis, there is a formula $(Y')^\alpha$ in **loc** PL equivalent to $\langle\alpha\rangle Y'$, and hence to $\langle\alpha\rangle Y$ too. X' is taken to be $(Y')^\alpha$.

Case (**slice** Y): Assume Y' from the inductive hypothesis contains k appearances of $\langle\ \rangle$ on the highest level (i.e., not nested within other $\langle\alpha\rangle Z$). Let these be $\mathbf{s}(\langle\alpha_1\rangle Z_1), \ldots, \mathbf{s}(\langle\alpha_k\rangle Z_k)$. Let $Q_1, \ldots, Q_k$ be symbols in AF appearing nowhere in X, and denote

$$Y_Q = (\textbf{slice } Y')|_{Q_1}^{\mathbf{s}(\langle\alpha_1\rangle Z_1)}| \ldots |_{Q_k}^{\mathbf{s}(\langle\alpha_k\rangle Z_k)}.$$

Y_Q is program-free, and so by Lemma 4.5 there is an equivalent $X_Q \in$ **loc** PL, with $X_Q \equiv Y_Q$. Now take X' to be

$$X_Q|_{\mathbf{s}(\langle\alpha_1\rangle Z_1)}^{Q_1}| \ldots |_{\mathbf{s}(\langle\alpha_k\rangle Z_k)}^{Q_k}.$$

It is easy to see that

$$X' \equiv Y_Q|^{Q_1}_{\mathbf{s}(\langle\alpha_1\rangle Z_1)}| \ldots |^{Q_k}_{\mathbf{s}(\langle\alpha_k\rangle Z_k)} \equiv \textbf{slice}\ Y' \equiv \textbf{slice}\ Y \equiv X.$$

Case (Y **suf** Z, Y **chop** Z): These cases are similar to **slice**. □

Finally, from Lemma 4.8 we obtain RPL $\leqslant$ PL$^+$ $\leqslant$ **loc** PL $\leqslant$ RPL thus completing the proof of Theorem 3.2.

5. The unified language *R*

R has two sets of atomic letters: state formulas, ASF, and transition formulas, ATF. It has a single set of operators, which yield the formulas $\neg X$, $X \vee Y$, $X \odot Y$, $X^{\oplus}$, and $\langle X\rangle Y$. The semantic rules of R are the usual. In particular,

$$p \models X \text{ only if } p \in S \qquad \text{for } X \in \text{ASF},$$

$$p \models X \text{ only if } p \in S \times S \quad \text{for } X \in \text{ATF},$$

$$p \models \langle X\rangle Y \text{ iff } \exists q(q \models X \text{ and } p \odot q \models Y).$$

Here, too, we have to include the formula $\mathbf{L}_0$ as in RPL.

This definition of R causes, again, undecidability of the validity problem, as is shown in [4]. In order to retain decidability, we require that, in formulas of the form $\langle X\rangle Y$, X contains no occurrences of $\neg$.

Let binary-RPL be RPL with the interpretations ρ_a for atomic a restricted so that $\rho_a \subseteq S \times S$.

Theorem 5.1. *binary-*RPL $<$ *R.*

Proof. Clearly, binary-RPL is contained in R. On the other hand, the simple formula $a \in$ ATF cannot be expressed in RPL, as can be easily demonstrated (see [18]). □

Theorem 5.2. *The validity problem for R is decidable but nonelementary.*

Proof. The proof is a modification of the decidability proof of [12] (see [18]). The nonelementariness is shown just as in Theorem 3.3 above. □

Appendix A. Proof of Lemma 4.6

By the construction of Table 1 we need only consider formulas of **loc** PL including the four operators, their duals, and with negation appearing only locally. We use induction on the structure of X, with $\bar{\mathbf{L}}_0$ and local formulas taken as the basis of the induction.

For local X, take $m_X = 0$ and $D_X = \emptyset$. Since $q \cdot r \not\models X$ for all q and r, the claim holds.

(1) $p_1 = q$ (or $p_2 = r$)

(2) $\exists t, p_2 = t \cdot r$ (or $q = p_1 \cdot t$)

(3) $\exists t, p_1 = q \cdot t$ (or $r = t \cdot p_2$)

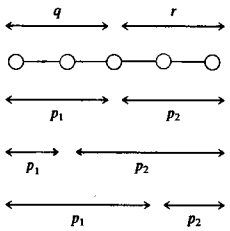


Fig. A.1. Dividing $q \cdot r$ into p_1 and p_2.

For $X = \bar{\mathbf{L}}_0$, take $D_X = \{\langle 1, 1\rangle\}$. Clearly, for all paths q, r, $q \cdot r \vDash \bar{\mathbf{L}}_0$ and $q \vDash 1$, $r \vDash 1$ hold.

In all composite cases we assume (by inductive hypothesis) the existence of decompositions $D_Y = \{\langle Y_i^1, Y_i^2\rangle \mid 1 \leqslant i \leqslant m_Y\}$ and $D_Z = \{\langle Z_i^1, Z_i^2\rangle \mid 1 \leqslant i \leqslant m_Z\}$ for Y and Z.

Case ($X = Y \vee Z$): Take D_X to be $D_Y \cup D_Z$. That D_X behaves right is trivial.

Case ($X = Y$ **rchop** Z): Take D_X to be

$$\{\langle Y, Z\rangle\} \cup \{\langle Y_i^1, Y_i^2 \textbf{ rchop } Z\rangle \mid 1 \leqslant i \leqslant m_Y\} \cup \{\langle Y \textbf{ rchop } Z_i^1, Z_i^2\rangle \mid 1 \leqslant i \leqslant m_Z\}.$$

To see that D_X is a proper decomposition, consider paths q and r.

(⇐): Assume $q \cdot r \vDash Y$ **rchop** Z. Then there are paths p_1, p_2 such that $q \cdot r = p_1 \cdot p_2$ and $p_1 \vDash Y$, $p_2 \vDash Z$. There are three basic possibilities, illustrated in Fig. A.1.

(1) $p_1 = q$. In this case, the pair $\langle Y, Z\rangle$ meets the condition, since $q \vDash Y$ and $r \vDash Z$.

(2) There is a path t such that $p_2 = t \cdot r$ (or $q = p_1 \cdot t$). Hence, $t \cdot r \vDash Z$, and by the inductive hypothesis there exists some $1 \leqslant i \leqslant m_Z$ such that $t \vDash Z_i^1$ and $r \vDash Z_i^2$. Since $p_1 \vDash Y$, we get $q = p_1 \cdot t \vDash Y$ **rchop** Z_i^1; hence, the pair $\langle Y$ **rchop** $Z_i^1, Z_i^2\rangle$ satisfies the requirement.

(3) There is a path t such that $p_1 = q \cdot t$ (or $r = t \cdot p_2$). The analysis is similar to possibility (2).

(⇒): Assume $q \vDash X_i^1$ and $r \vDash X_i^2$ for some $1 \leqslant i \leqslant m_X$. There are three cases, corresponding to the definition of D_X.

(1) The decomposition is $\langle Y, Z\rangle$, in which case $q \cdot r \vDash X$ is immediate.

(2) For some $1 \leqslant j \leqslant m_Z$, $\langle X_i^1, X_i^2\rangle = \langle Y$ **rchop** $Z_j^1, Z_j^2\rangle$. In this case, q can be divided into $q = t_1 \cdot t_2$ so that $t_1 \vDash Y$ and $t_2 \vDash Z_j^1$. By the inductive hypothesis on Z, $t_2 \cdot r \vDash Z$, and, therefore, $q \cdot r = t_1 \cdot (t_2 \cdot r) \vDash Y$ **rchop** Z.

(3) For some $1 \leqslant j \leqslant m_Y$, $\langle X_i^1, X_i^2\rangle = \langle Y_j^1, Y_j^2$ **rchop** $Z\rangle$. Similar to case (2).

Case ($X =$ **rslice** Y): Take D_X to be

$$\{\langle \textbf{rslice } Y, \textbf{rslice } Y\rangle\}$$
$$\cup \{\langle Y_i^1, Y_i^2\rangle \mid 1 \leqslant i \leqslant m_Y\}$$
$$\cup \{\langle (\textbf{rslice } Y) \textbf{ rchop } Y_i^1, Y_i^2\rangle \mid 1 \leqslant i \leqslant m_Y\}$$
$$\cup \{\langle Y_i^1, Y_i^2 \textbf{ rchop } (\textbf{rslice } Y)\rangle \mid 1 \leqslant i \leqslant m_Y\}$$
$$\cup \{\langle (\textbf{rslice } Y) \textbf{ rchop } Y_i^1, Y_i^2 \textbf{ rchop } (\textbf{rslice } Y)\rangle \mid 1 \leqslant i \leqslant m_Y\}.$$

The five sets in this union cover the five ways of dividing a path $q \cdot r$ into n sections $p_1, \ldots, p_n$, each of which satisfies Y (see Fig. A.2). That this is a proper decomposition is proved similarly to the previous case, and is left to the reader.

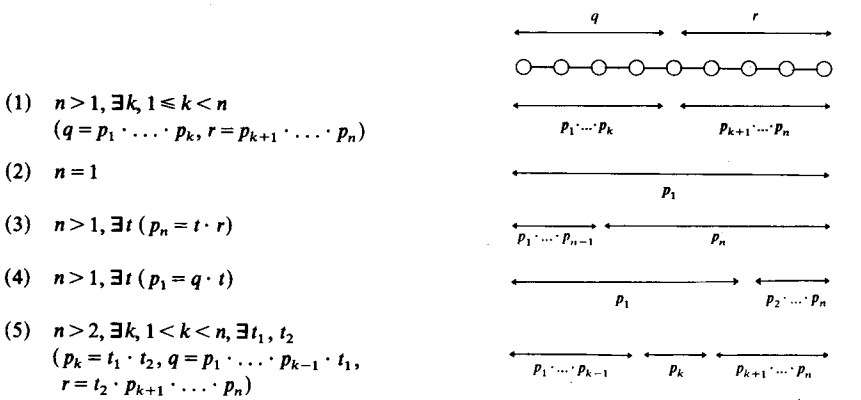


Fig. A.2. Dividing the path $q \cdot r$ into n sections $p_1, \ldots, p_n$.

Case $(X = Y \wedge Z)$: Take

$$D_X = \{\langle Y_i^1 \wedge Z_j^1,\ Y_i^2 \wedge Z_i^2\rangle \mid 1 \leq i \leq m_Y, 1 \leq j \leq m_Z\}.$$

For the two remaining cases, we use some additional notation. Let $D_X = \{\langle X_i^1, X_i^2\rangle \mid 1 \leq i \leq m_X\}$. Let $\{g_i^X\}$, $1 \leq i \leq 2^{m_X}$, be an enumeration of the subsets of $\{1, \ldots, m_X\}$. We now define notations for conjuncts and disjuncts of subsets of formulas X^1, X^2. Let CX_i^1 and DX_i^1 denote $\bigwedge_{k \in g_i^X} X_k^1$ and $\bigvee_{k \in g_i^X} X_k^1$, respectively, and let SX_i^1 denote $CX_i^1 \wedge \bigwedge_{k \notin g_i^X} \tilde{X}_k^1$. Here $\tilde{X}$ is as defined in Table 1, and is equivalent to $\neg X$. CX_i^2, DX_i^2, and SX_i^2 are defined analogously. Now let $\{f_j^X\}$, $1 \leq j \leq 2^{(2^{m_X})}$, enumerate the subsets of $\{1, \ldots, 2^{m_X}\}$. We define the formulas

$$\textbf{all}\ SX_i^1 = (\textbf{drslice}\ X)\ \textbf{drchop}\left(\bigvee_{k \in f_i^X} SX_k^1\right) \wedge \bigwedge_{k \in f_i^X} ((\textbf{rslice}\ \tilde{X})\ \textbf{rchop}\ SX_k^1)$$

and

$$\textbf{all}\ DX_i^1 = \bigwedge_{k \in f_i^X} DX_k^1.$$

Now, for the two remaining cases:

Case $(X = Y\ \textbf{drchop}\ Z)$: Take D_X to be

$$\{\langle SY_i^1 \wedge Y \wedge (Y\ \textbf{drchop}\ DZ_j^1),\ (DY_i^2\ \textbf{drchop}\ Z) \wedge SZ_j^2\rangle \mid 1 \leq i \leq 2^{m_Y}, 1 \leq j \leq 2^{m_Z}\}$$
$$\cup \{\langle SY_i^1 \wedge (Y\ \textbf{drchop}\ DZ_j^1),\ (DY_i^2\ \textbf{drchop}\ Z) \wedge Z \wedge SZ_j^2\rangle \mid 1 \leq i \leq 2^{m_Y}, 1 \leq j \leq 2^{m_Z}\}.$$

Case ($X =$ **drslice** Y): Take D_X to be

$$\{\langle \textbf{drslice } Y \wedge SY_i^1 \wedge \textbf{all } SY_j^1, DY_i^2 \wedge (DY_i^2 \textbf{ drchop } (\textbf{drslice } Y)) \wedge \textbf{all } DY_j^2 \wedge (\textbf{all } DY_j^2 \textbf{ drchop } (\textbf{drslice } Y))\rangle \mid 1 \leq i \leq 2^{m_Y}, 1 \leq j \leq 2^{(2^{m_Y})}\}$$
$$\cup \{\langle SY_i^1 \wedge \textbf{all } SY_j^1, \textbf{drslice } Y \wedge DY_i^2 \wedge (DY_i^2 \textbf{ drchop } (\textbf{drslice } Y)) \wedge \textbf{all } DY_j^2 \wedge (\textbf{all } DY_j^2 \textbf{ drchop } (\textbf{drslice } Y))\rangle \mid 1 \leq i \leq 2^{m_Y}, 1 \leq j \leq 2^{(2^{m_Y})}\}.$$

Showing that these are proper decompositions is a tedious but straightforward application of the definitions above and the inductive hypothesis. □

Acknowledgment

We are grateful to M. Vardi for his helpful comments.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading, MA, 1975).
[2] L. Banachowski, Modular properties of programs, *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* **23** (3) (1975).
[3] Y. Choueka, Theories of automata on ω-tapes: A simplified approach, *J. Comput. System Sci.* **8** (1974) 117-141.
[4] A. Chandra, J. Halpern, A. Meyer and R. Parikh, Equations between regular terms and an application to process logic, *13th ACM Symp. on Theory of Computing* (1981) 384-390.
[5] E. Engeler, Algorithmic properties of structures, *Math. System Theory* **1** (1967) 183-195.
[6] R.W. Floyd, Assigning meaning to programs, in: J.T. Schwartz, ed., *Mathematical Aspects of Computer Science, Proc. Symp. in Applied Math.* **19** (Amer. Math. Soc., Providence, RI, 1967) 19-32.
[7] M.J. Fischer and R.E. Ladner, Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18** (1979) 194-211.
[8] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, On the temporal analysis of fairness, *Proc. 7th ACM Symp. on Principles of Programming Languages* (1980) 163-173.
[9] D. Harel, Dynamic logic, in: *Handbook of Philosophical Logic II* (Reidel, Dordrecht, 1984) 497-604.
[10] D. Harel, Two results on process logic, *Inform. Process. Lett.* **8** (4) (1979) 195-198.
[11] D. Harel, Recurring dominoes: Making the highly undecidable highly understandable, *Ann. Discrete Math.* **24** (1985) 51-72.
[12] D. Harel, D. Kozen and R. Parikh, Process logic: Expressiveness, decidability, completeness, *J. Comput. System Sci.* **25** (1982) 144-170.
[13] C.A.R. Hoare, An axiomatic basis for computer programming, *Comm. ACM* **12** (1969) 576-580.
[14] D. Kozen and R. Parikh, An elementary proof of the completeness of PDL, *Theoret. Comput. Sci.* **14** (1981) 113-118.
[15] Z. Manna, *Mathematical Theory of Computation* (McGraw-Hill, New York, 1974).
[16] H. Nishimura, Descriptively complete process logic, *Acta Informatica* **14** (1980) 359-369.
[17] R. Parikh, A decidability result for second order process logic, *Proc. 19th IEEE Symp. on Foundations of Computer Science* (1978) 178-183.
[18] D. Peleg, Regular process logics, M.Sc. Thesis, Bar-Ilan University, Ramat Gan, Israel, 1982 (in Hebrew).

[19] A. Pnueli, The temporal logic of programs, *Proc. 18th IEEE Symp. on Foundations of Computer Science* (1977) 46–57.
[20] V.R. Pratt, Semantical considerations on Floyd–Hoare logic, *Proc. 17th IEEE Symp. on Foundations of Computer Science* (1976) 109–121.
[21] V.R. Pratt, Process logic, *Proc. 6th ACM Symp. on Principles of Programming Languages* (1979) 93–100.
[22] A. Salwicki, Formalized algorithmic languages, *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* **18** (5) (1970) 227–232.
[23] A.P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logics, *14th ACM Symp. on Theory of Computing* (1982) 159–167.
[24] R. Sherman, A. Pnueli and D. Harel, Is the interesting part of PL uninteresting? A translation from PL to PDL, *SIAM J. Comput.* **13** (1984).
[25] W. Thomas, Star free regular sets of ω-sequences, *Inform. Control* **42** (1979) 148–156.
[26] P. Wolper, Temporal logic can be more expressive, *Inform. and Control* **56** (1983) 72–99.