A DECIDABILITY RESULT FOR A SECOND
ORDER PROCESS LOGIC

Rohit Parikh

September 1978

MIT/LCS/TM-112
A DECIDABILITY RESULT FOR A SECOND
ORDER PROCESS LOGIC

Rohit Parikh

July 1978
A DECIDABILITY RESULT FOR A SECOND ORDER PROCESS LOGIC

Rohit Parikh
A DECIDABILITY RESULT FOR A SECOND ORDER PROCESS LOGIC.

Rohit Parikh

Laboratory for Computer Science, M.I.T., and Mathematics Department, Boston University

July 31, 1978

Abstract: We prove the decidability of the validity problem for a rather general language for talking about computations. As corollaries of our result, we obtain some decidability results of Pratt, Constable, Fischer-Ladner, and Pnueli and also a new decidability result for deterministic propositional dynamic logic.

Introduction: In recent years various authors have introduced languages to talk in an abstract way about computation processes and programs. Examples include the dynamic logic introduced by Pratt [Pr1] and the monadic program logic introduced by Constable [C]. Constable's logic is decidable as is also the propositional case of dynamic logic [FL]. Both propositional dynamic logic and monadic program logic allow us to talk about what happens if and when the program terminates. But they do not allow us to say what happens during the computation. Recently, Pratt [Pr2] has introduced a logic of processes which allows some process connectives whose semantics is dependent on what happens during computation and not just on what happens at the end. It appears likely that the logic that he gets has the finite model property. I.e. every formula that has a model has a finite model. (It is decidable, as we shall show.) We also have the temporal semantics newly introduced by Pnueli [Pn].

Thus we have various languages introduced by various authors. Why not have a universal language that includes them all? Let's look at the situation in detail.

Consider a computer C and let S be the set of internal states. S need not be finite but we shall assume that it is at most countably infinite. A (computation) path p will be a sequence of states in S and a program will be a set of such paths.

Key Words: dynamic logic, theory of programs, decidability

This research was supported by the National Science Foundation under NSF grant number MCS78-04338.
A statement that we are interested in may be true in some states and false in others, so it will be a unary predicate \( P \) on \( S \).

The language \( L \) for talking about the situation above contains a finite number of unary predicate letters \( P_1, \ldots, P_m \), and a finite number of basic program letters \( a_1, \ldots, a_n \). New programs may be formed by taking (nondeterministic) unions, \( ; \) and \( * \). (It is well known that "if .. then .. else ..", and "while .. do .." can be defined in terms of these. See e.g. [Pr1]). We also have variables for states and paths and atomic formulas will have the form \( P(s) \) (the predicate \( P \) holds at the state \( s \)), \( \alpha(p) \) (the path \( p \) is a computation of the program \( \alpha \)) and \( \Omega(p, s_1, \ldots, s_n) \). (The states \( s_1, \ldots, s_n \) have occurrences on the path \( p \) in the order shown.) New formulas are formed through truth functional combinations and quantifying over states and paths. If \( A \) is a formula with a single state variable, a program \( A? \) can be formed, which will be a test of the truth or falsity of \( A \) [Pr1]. This language can easily be shown to include the monadic program logic of Constable, the propositional dynamic logic of Fischer-Ladner-Pratt, the (recently introduced) process logics of Pratt, and Pnueli's temporal semantics formalised in his system DX.

It turns out that the language of which we have given a rough description above is undecidable since the first order theory of an arbitrary binary relation can be interpreted into it. However, there is a way around this problem, and it doesn't consist of consulting an oracle. Rather imagine our computer to be equipped with a clock. This cannot affect any question we are actually interested in since for the running of a program or its correctness, a clock is neither here nor there. However, a clock prevents a program from ever returning to the same point. The most one can have now is a return to a point that looks very much like an earlier point but is none the less distinct. Now a clock keeps record of time elapsed. We can also keep records of programs we performed starting from \( s_0 \), (some fixed starting state). We can ask that records be kept of the paths chosen (for nondeterministic programs), the number of steps performed along each path, etc. Thus we get a new computer that looks just like the old one, except that these records being part of the state give it a tree-like structure. No computation path belongs to two distinct basic programs, no computation ever returns to the same point and so on. None of this can affect the sorts of questions we are interested in, such as termination or correctness of programs. It turns out, however, that it
can affect questions that can be posed in the language described above. It follows
that the questions we really should concern ourselves with are those that belong to
a subpart of L, the part L- consisting of those formulae that are unaffected by the
addition of auxiliary information. And now comes the payoff! We can show that
the languages of Constable, Pratt, etc., can still be interpreted in our smaller
language L-. Moreover, the theory of L-, which is clearly the same as the theory
of L over tree-like models, can be interpreted in Rabin’s SnS [Ra], and is
therefore decidable. Also, a slight modification of the construction yields the
decidability of propositional dynamic logic where all basic programs are
deterministic.

1. The Language L:

   Alphabet: state variables: \( s_1, s_2, \ldots \)
   path variables: \( p_1, p_2, \ldots \)
   basic programs: \( a_1, a_2, \ldots, a_n \)
   basic predicates: \( P_1, \ldots, P_m \)

   Also: \( u, ;, *, \top, \bot, \exists, \forall, \Omega, (, ), , \) Note that \( = \) is not a symbol of
   \( L(\forall x) \) is an abbreviation for \( \neg(\exists x)\neg \). Similarly \( \wedge, \rightarrow, \) etc.

We define the notion "program" and "formula" for L by simultaneous recursion.

Programs: 1) Every basic program is a program.
            2) If \( \alpha, \beta \) are programs, so are \( (\alpha \cup \beta), (\alpha; \beta), \alpha^* \).
            3) If \( A \) is a formula with a single free state
               variable, then \( A? \) is a program.

Formulas: 1) If \( P \) is a basic predicate and \( s \) is a state variable, then \( P(s) \) is a
           formula.
           2) If \( s_1, \ldots, s_n \) are state variables, where \( n \geq 1 \), and \( p \) is a path
              variable, then \( O(p, s_1, \ldots, s_n) \) is a formula. (Such formulas are called
              \( O \)-formulas and their intuitive meaning is: "The states \( s_1, \ldots, s_n \) occur, in that
              order, on the computation path \( p \)." The \( s_i \) need not be distinct, but \( O \) stands for
              \( < \) rather than \( \leq \), i.e. the computation path may cross itself but \( O \) records a
              positive passage of time.)
           3) If \( \alpha \) is a program and \( p \) is a path variable, then \( \alpha(p) \) is a formula. (It
means: "The path \( p \) is a computation path of the program \( \alpha \)."
(4) If \( A, B \) are formulas, so are \( \neg A, (A \lor B) \)
(5) If \( A \) is a formula, \( s \) is a state variable and \( p \) is a path variable then
\( (\exists s) A, (\exists p) A \) are formulae.

**Notation:** \( A, B, C, \ldots \) stand for formulae, \( a, b, c \), for basic programs. \( \alpha, \beta, \gamma \) for
arbitrary programs. \( s, t, u \), for state variables, and \( p, q \) for path variables. \( s, t \), stand for actual states (in some model) and \( p, q \) for actual paths.

**Semantics for L:** An Interpretation \( M \) for \( L \) consists of
(1) A nonempty universe \( U \) (This will be the set of states)
(2) For each basic predicate \( P_i \) a subset \( P_i \) of \( U \)
(3) For each basic program \( a_i \), a subset \( R_i \) of \( P \) where \( P = \) the set of all
finite or infinite sequences from \( U \).

**Interpretation of Programs:** For each program \( \alpha \) we define a subset \( R_\alpha \) of \( P \) as follows:
(1) If \( \alpha \) is an \( a_i \) then \( R_\alpha = R_i \).
(2) If \( \alpha \) is \( \beta \gamma \) then \( R_\alpha = R_\beta \cap R_\gamma \)
(3) If \( \alpha \) is \( \beta; \gamma \) then \( R_\alpha \) is the set of all paths \( (s_0, s_1, \ldots, s_n, s_{n+1}, \ldots) \)
where \( (s_0, \ldots, s_n) \) is in \( R_\beta \) and \( (s_n, s_{n+1}, \ldots) \) is in \( R_\gamma \) where the
path in \( R_\beta \) is finite and the path in \( R_\gamma \) may be finite or infinite.
(4) If \( \alpha \) is \( \beta^n \) then \( R_\alpha \) is \( \bigcup_n R_\beta^n \) where \( \beta^n = \beta; \beta; \ldots; \beta \) (\( n \) times, \( n \geq 0 \)).
(5) If \( A \) is a formula with the single free variable \( s \), and \( \alpha \) is \( A \) then \( R_\alpha \) is
the set \( \{ (s, s) \mid s \) satisfies \( A \) in \( M \} \).

**Interpretation of formulas:** This is easiest to explain if we add constants to \( L \) for
each state \( s \in U \) and each path \( p \in P \). We shall consider closed formulas only for the
moment. \( M \models A \) means: \( A \) is true in \( M \).

(1) \( M \models P_i(s) \) iff \( s \in P_i \).
(2) \( M \models O(\rho; s_1, \ldots, s_n) \) iff \( s_1, \ldots, s_n \) have occurrences in that order on
\( \rho \).
(3) \( M \models A(\rho) \) iff \( \rho \) is in \( R_\alpha \)
(4) \( M \models \neg A \) iff \( M \not\models A \)
(5) \( M \models A \lor B \) iff \( M \models A \) or \( M \models B \).
(6) \( M \models (\exists p)A(p) \) iff \( \exists p \) such that \( M \models A(p) \)
(7) \( M \models (\exists s)A(s) \) iff \( \exists s \) such that \( M \models A(s) \)

If \( A \) has a single free variable \( s \) then \( s \) satisfies \( A = A(s) \) iff \( M \models A(s) \).
Similarly if we have several free state and path variables. A formula with free variables is \textit{satisfiable} in \( M \) if there are states and paths that satisfy it. It is \textit{satisfiable} if it is satisfiable in some \( M \). It is \textit{valid} if its negation is not satisfiable.

\textbf{Theorem 1:} The validity problem for \( L \) is undecidable. \textbf{Proof:} We show that the validity problem for the first order theory of a binary relation can be coded into the validity problem for \( L \). Given a binary relation \( R \) over a set \( X \), we can take an interpretation \( M \) where \( U = X \), there is a single program letter \( a \) where \( R_a \) is intended to be \( \{(s,t) | (s,t) \in R\} \) and now \( R_{xy} \) can be written \( (\exists p)(a(p) \land O(p,x,y)) \).
We extend this map to the predicate calculus in the obvious way. If \( A \) is a formula of the predicate calculus and \( A^+ \) its translation in \( L \), then it is easily seen that \( A \) is valid in the predicate calculus iff \( A^+ \) is a valid formula of \( L \).
\textbf{Q.E.D.}

2. The Language SOAPL

We saw in the last section that the language \( L \) is undecidable. We now define a sublanguage \( L^- \) of \( L \) obtained by \textit{limiting the class of formulas}. \( L^- \) has the property that whenever a formula has a model it also has a tree-like model. This property can be used to code \( L^- \) into Rabin's SnS and hence the validity problem for \( L^- \) is decidable. Because satisfiable formulae of \( L^- \) can always have cycle-free models, we shall refer to \( L^- \) as second order acyclic process language or SOAPL. Note that we are not \textit{requiring} SOAPL only to have acyclic models, but merely asserting that whenever there is a model, there is one that is acyclic.

\textbf{Alphabet:} SOAPL has the same alphabet as \( L \) plus the symbol \( Bg \). \( Bg(w,p) \) stands for: "\( w \) is the first state of \( p \)" and can be written \( O(p,w) \land \neg(\exists s)O(p,s,w) \), but the restrictions on formulae that we shall impose make it necessary to have this as a separate symbol.
Programs: Same as L except, having fewer formulae, we shall have fewer programs of the A? type.

Formulas: Basically our formulae enable us to define unary predicates of states and paths. We define first three auxiliary notions.

\( \mathcal{O} \)-formulas: \( \mathcal{O}(p,s_1...s_k) \) is an \( \mathcal{O} \)-formula where \( k \geq 1 \).

Now we define path formulas and state formulas by simultaneous recursion.

State formulas: 1) \( P(w) \) is a state formula  
2) If A, B are states with the same free state variable, so are \( \neg A \), \( (A \lor B) \).  
3) If \( A(p) \) is a path formula then \( (\exists p)(Eg(w,p) \land A(p)) \) is a state formula, where \( \alpha \) is any program.

Path formulas: 1) let \( C \) be a truth functional combination of \( \mathcal{O} \)-formulas with the same path variable and state formulas such that no state variable occurs in two different \( \mathcal{O} \)-formulas nor twice in the same one. If \( s_1...s_k \) are all the free state variables of \( C \) and \( Q_1...Q_k \) are quantifiers, then \( (Q_1s_1)...(Q_ks_k)C \) is a path formula. (The quantifiers are thought of as relativised to the path in question. E.g. if the path is \( p \) then \( (\exists w)(\ldots w_{\ldots}) \) in SOAPL means: \( (\exists w)(w\in p \land \ldots w_{\ldots}) \).)
   2) If A, B are path formulas with the same path variable then so are \( \neg A \), \( (A \lor B) \).

And now we can define arbitrary formulas of SOAPL.

Formulas: (of SOAPL) are obtained from the state and path formulas by taking truth functional combinations and/or quantifying. Note that state and path formulas define unary predicates of states and paths.

Basically we have set up a situation where we can say about a path what sorts of states it has, and in what order. We can say of a state that it satisfies some \( P_i \) or doesn't satisfy it and we can say what sorts of paths begin at it. Also these properties can feed off each other by recursion back and forth. However, we can only define unary predicates this way, and while we do have access to \( \mathcal{O} \), we have
put limitations on its use, so that it cannot be used to create genuine binary predicates. Ultimately, since the = sign is absent, and we do not allow dual occurrences of state variables in O-formulas, even states and paths do not have real existence but only serve as a means of saying things about programs. E.g. we can say about programs α, β that they both have paths with some complicated property, but we can't say in SOAPL that this is the same path. This fact is the clue to the decidability of (the validity-satisfiability problem of) SOAPL. Since SOAPL is closed under truth functional combinations, this implies also the decidability of the consequence problem for sentences.

**Theorem 2:** (a) SOAPL is decidable
(b) SOAPL is decidable if we restrict ourselves to deterministic programs.

**Proof:** Section 3. Note that (a) does not imply (b), since there is no way in SOAPL to say that all basic programs are to be deterministic.

**Theorem 3:** Propositional dynamic logic, the process logic of Pratt, the logical system DX of Pnueli, and the monadic programming logic of Constable are all interpretable in SOAPL.

**Cor:** All the above mentioned logics are decidable.

We shall not give a detailed proof of Theorem 3. Rather we shall give actual translations of certain crucial notions from the various logics into SOAPL. From that it will be evident that the whole logic can be translated. We also show that the notions of partial and total correctness can be translated into SOAPL.

a. A{α}B. The partial correctness of α with respect to A, B can be expressed by the formula:

\[(\forall w)(\forall p)(Bg(w,p) \land \alpha(p) \land A(w) \rightarrow (\forall s)(\exists t)(O(p,s,t) \lor B(s))) \]

b. A<α>B. The total correctness of α with respect to A, B can be expressed:

\[(\forall w)(\forall p)(Bg(w,p) \land \alpha(p) \land A(w) \rightarrow ((\exists s)(\forall t)(\neg O(p,s,t) \land B(s)))) \text{ Note: the notation above is ad hoc. There seems to be no standard notation.} \]
c. $[\alpha]A$. This is a notion from dynamic logic [Pr1]. $[\alpha]A$ holds at $w$ if every state reachable from $w$ by $\alpha$ satisfies $A$. Here we assume that $A$ has already been translated as $A^\ast$. Then $[\alpha]A$ translates as

$$(\forall p)(B_g(w,p) \land \alpha(p) \rightarrow (\forall s)(\exists t)(\neg O(p,s,t) \rightarrow A^\ast(s)))$$

d. $\alpha[J]A$. "$\alpha$ preserves $A" [Pr2], means that if, during an $\alpha$ computation, $A$ holds at any state, it continues to hold. It can be translated as:

$$(\forall w)(\forall p)(B_g(w,p) \land \alpha(p) \rightarrow (\forall s)(\forall t)\{(O(p,s,t) \land A(s)) \rightarrow A(t)\})$$

e. In Constable's monadic programming logic, the basic programs are treated as functions. It would seem that they are not only deterministic but are also everywhere defined. However, we can translate without such assumptions. Thus, e.g., $P(f_1(w))$ can be translated as:

$$(\forall p)(B_g(w,p) \land f_1(p) \rightarrow (\exists s)(\forall t)(\neg O(p,s,t) \land P(s)))$$ (Here $f_1$ is some $a_i$ in our notation.)

f. We cannot say in SOAPL that a program is deterministic. This is because we cannot say in L that two paths or states are equal or unequal. However, the logic of deterministic programs can be handled by using a new translation of SOAPL into SnS which allows programs only one path (at most) per state. Then the decidability of the validity problem for SnS implies the decidability of the validity problem for SOAPL interpreted over deterministic programs.

g. We can say in SOAPL that a particular program terminates. E.g., "$\alpha$ always terminates at $w$" is

$$(\forall p)(B_g(w,p) \land \alpha(p) \rightarrow (\exists s)(\forall t)(\neg O(p,s,t)))$$.

h. Pnueli [Pn], in his system DX, introduces two temporal connectives which are unary functions from statements to statements. The context in which he works is that of a single deterministic program which we shall call $\alpha$. If $A(s)$ (our notation) is a state formula then $GA(s)$ means that $A$ holds during all future moments of the program, whereas $XA(s)$ means that $A$ holds at the next moment of the program. If his program is $\alpha$, then let $\beta$ be the program which executes exactly one step of $\alpha$ when $\alpha$ has not ended. Then we can write $GA$ as $[\beta^\ast]A$, and $XA$ as $[\beta]A$, and we have already shown, in c. above, how to express these. Note
that we need not confine ourselves to one program as Pnueli does, but can take
arbitrary regular combinations of arbitrarily many programs.

**Concluding Remark:** It is often said that the decidability or undecidability of a
particular problem in general does not give us adequate guidance for actual
practice. The L, SOAPL example illustrates this since L is undecidable, but SOAPL
is adequate to express most notions that interest us which are expressible in L.

### 3. The Decidability of SOAPL

Since this result is proved by reducing SOAPL to SnS, we shall briefly review
SnS. SnS, the second order theory of n successors, can be thought of as the theory
of \( \Sigma^k \) where \( \Sigma = \{ \sigma_1, ..., \sigma_n \} \). The language of SnS includes the n successor
functions defined by \( S_i(x) = x \sigma_i \), \( i = 1, ..., n \) where \( x \in \Sigma^k \), \( \sigma_i \in \Sigma \). We also
have equality in this language, the usual truth functional connectives and two
kinds of quantifiers. We have first order quantifiers which range over \( \Sigma^k \) (these
will be \( x, y, z, ... \)) and second order quantifiers \( (X, Y, Z, ...) \) which range over
subsets of \( \Sigma^k \). Since we have in mind a unique model, the one described above, we
have the set of true sentences defined uniquely. This set is recursive by Rabin
[8], and non-elementary by Meyer [3].

The reduction of SOAPL to SnS is accomplished in the following way: we give a
translation of L into SnS which is not meaning preserving for the whole of L.
However it has the property that if the formula A is in SOAPL then A is valid iff
its translation is true. This shows that the validity problem for SOAPL is
decidable. We begin by showing that certain notions are definable in SnS. Note
that concatenation cannot be one of them, since it known from [Q] that even the
first order theory of concatenation is undecidable.
Notation: "x is in X" is written X(x).

(1) x ≤ y. x is an initial segment of y, is defined by

\[(\forall X)(X(x) \land (\forall z)(X(z) \rightarrow \exists y X(S(y(z)) \rightarrow X(y))).\]

More generally we can also define \(x \leq_{\Delta} y\) where \(\Delta\) is a sub-alphabet of \(\Sigma\), by restricting the conjunction on \(i=1,\ldots,n\), above to \(\Delta\) only. \(x \leq_{\Delta} y\)
means: there is a string \(z\) in \(\Delta^*\) such that \(y = xz\).

(2) LO(X): X is linearly ordered.
\[(\forall x)(\forall y)(X(x) \land X(y) \rightarrow x \leq y \lor y \leq x)\]

(3) X ⊆ Y: \((\forall x)(X(x) \rightarrow Y(x)).\) Similarly X = Y ∩ Z,
\(X = Y \cap Z,\) etc.

(4) \(x = \text{Init}(X).\) x is the first point of \(X\). \(\text{LO}(X) \land X(x) \land (\forall y)(X(y) \rightarrow x \leq y).\)

(5) \(x = \text{Fin}(X).\) x is the last point of \(X\). Just like (4) above.

(6) \(X = \text{Seg}(x,y).\) X is the segment between \(x\) and \(y\).
\(x = \text{Init}(X) \land y = \text{Fin}(X) \land (\forall z)(x \leq z \land z \leq y \rightarrow X(z))\)

(7) Let \(Y\) be a fixed regular set of strings. Let \(R_{xy}\) mean:
\[(\exists z)(Y(z) \land xz = y),\]
then \(R_{xy}\) is definable in SnS.

We sketch the proof which is by induction on the complexity of the regular expression defining \(Y\). If \(Y = \{\sigma_i\}\), then the formula \(A(x,y)\) defining \(R_{xy}\) is:
\(y = S_i(x).\)

If \(Y = Y_1 \cup Y_2\), then \(A(x,y)\) is
\(A_1(x,y) \lor A_2(x,y),\) where \(A_1\) and \(A_2\) are the formulae corresponding to \(Y_1\) and \(Y_2\) respectively.

If \(Y = Y_1 \cdot Y_2\), where . denotes the product operation on sets of strings, then \(A(x,y)\) is
\[(\exists z)(A_1(x,z) \land A_2(z,x)).\]
And finally if \( Y = (Z)^* \) and \( B(x,y) \) corresponds to \( Z \), then the formula for \( Y \) is:
\[
(\exists w)(w(x) \land (\forall z)(w(z) \land z \leq y \rightarrow (\exists w)(z \leq w \land w \leq y \land w(z) \land B(z,w)) \land x \leq y).
\]
A set that we are particularly interested in is the regular set \( K = ((a_1^{+} \ldots a_n^+ \cdot c(b^+))^*) \). The binary relation \( K \) stands for: \( (\exists z)(K(z) \land y = xz) \).

Note that we can use Init, Seg, etc. as if they were functions. E.g. \( A(Seg(x,y)) \) is an abbreviation for \( (\exists x)(A(x) \land X = Seg(x,y)) \) where \( X \) is a variable free in \( A(X) \).

Let \( A \) be a formula of \( L \) and \( M \) a model of it. We shall confine ourselves to countable models \( M \). We can also assume without loss of generality that \( M \) is connected in the sense that there is an origin state \( s_0 \) from which each other state is reachable by executing programs. For the universe \( U \) of states be \( \{s_0, s_1, \ldots \} \) and introduce a new program a which takes \( s_i \) to \( s_{i+1} \) (\( s_p \) to itself if \( U \) is a finite set and there is a last state \( s_p \)). We have to show how \( M \) can be coded into a model for \( S(n+2)S \) with certain auxiliary sets \( V, \ldots, X_n, Y_1, \ldots, Y_m \) of strings, which code information about \( M \). In particular, the \( X_i \) code the programs \( a_i \), and the \( Y_j \) code the predicates \( P_1 \). The set \( V \subseteq \Sigma^* \) codes the universe \( U \) of \( M \).

Suppose \( a_1 \ldots a_n \) are the basic program letters and suppose for simplicity that the programs \( R_1, \ldots, R_n \) in \( M \) are deterministic. (Later we shall show the necessary modifications if these are not deterministic. The translation for deterministic programs is not a special case of that for nondeterministic programs, but they are similar.) We let \( \Sigma = \{a_1, \ldots, a_n, b, c\} \) and consider the map \( \Phi \) from \( \Sigma^* \), into \( U \) and the map \( \Psi \) defined on certain subsets of \( \Sigma^* \) as arguments. The domain of \( \Phi \) is a proper subset of the \( \Sigma^* \), and is in fact a subset of the regular set \( K \) that we defined before. The domain of \( \Psi \) consists of certain linearly ordered subsets of \( \Sigma^* \).

\[ \text{a) } \Phi(\lambda) = s_0, \text{ where } \lambda \text{ is the empty string.} \]

\[ \text{b) Suppose that } \Phi(x) \text{ is defined and equals } s \text{ and that } x \text{ does not end in some } a_i \text{ or in } c. \text{ Suppose there is a path } p \text{ in } R_i \text{ (the path will be unique by the assumption above) which starts at } s, \text{ then } \Phi(x) = \Phi(xa_i) = \Phi(xa_i c) = s. \]
Moreover, for every \( k < l(p) \) (\( l(p) \) is length of \( p \) = number of states on \( p \)), the strings
$x_a^i c b^j$ are put in $X_i$ which is the correlate of $R_i$ in $\Sigma^*$ and if $s_j$ is the jth state on $\rho$, then $\Phi(x_a^i c b^j) = s_j$. If $X = \{x_a^i c b^j | 0 \leq k < l(\rho)\}$ then we let $\Psi(X) = \rho$.

Finally we let $Y = \{x | x \in \Sigma^* \text{ and } \Psi(\Phi(x)) \text{ holds}\}$
and $\nu = \{x | \Phi(x) \text{ is defined}\}$.

Example: Suppose for example that the model $M$ consists of the states
$\{s_i | 0 \leq i \leq 2\}$, a predicate $P$ that holds at $s_i$ for $i = 1, 2$, and two programs $R_1, R_2$, where $R_1 = \{(s_0, s_1), (s_1, s_2)\}$, and $R_2 = \{(s_1, s_2)\}$. Then $M'$, its translation into $S4S$, will have
$\nu = \{\lambda, a_1, a_1c, a_1cb, a_1cba1, a_1cba1c, a_1cba1cb, a_1cba2, a_1cba2c, a_1cba2cb\}$.

Of these, $\lambda$, $a_1$, $a_1c$, will all represent $s_0$, and $a_1cb$, $a_1cba1$, $a_1cba1c$, $a_1cba2$, $a_1cba2c$, will represent $s_1$. Similarly $a_1cba1cb$ will represent $s_2$ and $a_1cba2cb$ will also represent $s_2$. The set $Y$ will contain all the strings that represent either $s_1$ or $s_2$. The picture below shows both $M$ and $M'$. In $M'$, the strings that belong to $Y$ are enclosed in little squares. Note that the path $W$ starting at $a_1c$ and ending at $a_1cba1cb$ satisfies $\Phi(a_1; a_1)(W)$, but not $\Phi(a_1)(W)$, because the piece $a_1cb$, $a_1cba1$, $a_1cba1c$, which is all within $s_1$ as it were, is missing. If we think of the paths $a_i$ as railway lines, then we can think of pieces like $a_1cb$, $a_1cba1$, $a_1cba1c$ as taxi rides from one station to another.

These taxi rides perform the important function of allowing us to represent both paths and programs as subsets of $\Sigma^*$, rather than programs as sets of subsets of $\Sigma^*$.

\[ M: \quad s_0 \longrightarrow s_1 \longrightarrow s_2 \]
Now we define some further notions in $\text{SnS}$. We write (writing $xa_1$ for $S_1(x)$ etc.)

(8) $\text{Suc}(x, y) : y = xa_1c \lor \ldots \lor y = xa_nc$. $\Phi(x)$ and $\Phi(y)$ are the same, but $y$ is the beginning point of a program while $x$ is not. If $s = \Phi(x) = \Phi(y)$, we need these two copies of $s$ to prevent the following problem. Really $\alpha(p)$ should mean: $p$ is an element of $\alpha$, but since we don't have sets of sets, we coded this as: $p$ is a subset of $\alpha$. This creates the possibility that if the end state of a program is also a start state, then the program would run twice without our intending this. We avoid this problem by making $x$ and $y$ distinct, and to start up the program again, we would need the connecting link from $x$ to $y$. We shall also need, $\text{Sam}(x, y) : (\exists z)(\exists w) (\text{Suc}(z, w) \land z \leq x \land z \leq y \land x \leq w \land y \leq w)$ $\text{Sam}(x, y)$ says that $x$ and $y$ represent the same state in $\mathcal{M}$.

(9) $\text{Uni}(V) : (\forall x)(\forall y)\{V(x) \land y \leq x \rightarrow V(y)\}$
$\land (\forall x)(\forall y)\{V(x) \land V(y) \land x < y \rightarrow xb \leq y \lor$
$\exists z)(\text{Suc}(x, z) \land (z \leq y \lor cy = z))\} \land V \subseteq K$
$\text{Uni}(V)$ means that $V$ is subset of $\Sigma^*$ which could correspond to the $U$ for some model $\mathcal{M}$. 
(10) \( Y = \text{Conc}(X, Z) : (\exists x_1)(\exists x_2)(\text{Suc}(x_1, x_2) \land Y(x_1) \land Y(x_2) \land (\forall z)((X(z) \leftrightarrow Y(z) \land z \leq x_1) \land (Z(z) \leftrightarrow Y(z) \land z_2 \leq z)) \land \text{LO}(Y) \land Y = X \cup Z \cup \text{Seg}(x_1, x_2) \)

This says that the path \( Y \) has been obtained from the paths \( X, Z \) by taking their union and putting in the connecting piece we mentioned in (8) above. If \( \Psi(X) = p \) and \( \Psi(Z) = q \), then \( \Psi(Y) \) is defined and equals \( pq \).

(11) \( \alpha(W) \):
   (a) If \( \alpha \) is \( a_i \) then \( \alpha(W) \) is \( W \subseteq X_i \land \text{LO}(W) \)
   (b) If \( \alpha = \beta \cup \gamma \) then \( \alpha(W) \) is \( \beta(W) \lor \gamma(W) \)
   (c) If \( \alpha \) is \( \beta; \gamma \) then \( \alpha(W) \) is \( (\exists W_1)(\exists W_2)(W = \text{Conc}(W_1, W_2) \land \beta(W_1) \land \gamma(W_2)) \)
   (d) If \( \alpha \) is \( \beta^* \) then \( \alpha(W) \) is \( \text{LO}(W) \land (\exists Z)(Z(\text{Init}(W)) \land Z \subseteq W \land (\forall y)((Z(y) \land \lnot (y = \text{Fin}(W)) \rightarrow (\exists z)(\exists x)(\text{Suc}(y, z) \land z < x \land Z(x) \land \text{Seg}(z, x) \land \text{Seg}(z, x) \subseteq W)) \).

\( \alpha(W) \) says that \( W \) is a path in the program \( \alpha \).

Lemma: \( \alpha(W) \) holds in \( M' \) iff there is a \( p \) in \( R_\alpha \) such that \( \Psi(W) = p \).

Proof: By induction on the complexity of \( \alpha \). The result holds by definition if \( \alpha \) is some program \( a_i \). Otherwise \( \alpha \) is of the form \( \beta \cup \gamma \), \( \beta; \gamma \), or \( \beta^* \).

   a) \( \alpha = \beta \cup \chi \). This case is easy. \( \alpha(W) \) holds iff \( \beta(W) \) or \( \chi(W) \) holds, by (11b) above, iff \( \Psi(W) \) is in \( R_\beta \) or in \( R_\chi \), iff \( \Psi(W) \) is in \( R_\alpha \).

   b) \( \alpha = \beta; \gamma \). Then by (11c) above, if \( \alpha(W) \) holds then \( W \) is obtained by 'conc'ing two pieces \( U \) and \( V \) and \( \beta(U) \) and \( \gamma(V) \) hold. Hence \( \Psi \) maps \( U \), \( V \) respectively into paths \( q \) and \( r \) which are respectively in \( R_\beta \) and \( R_\gamma \), and by the last part of (10) above \( \Psi(W) \) will be the path \( p = qr \). Conversely, if \( \Psi(W) \) is in \( R_\alpha \), then since \( \alpha = \beta; \gamma \), \( p \) must fall into pieces \( q \), \( r \), which are in \( R_\beta \), \( R_\gamma \), respectively. There must be pieces \( U \), and \( V \) (plus a connecting little piece) of \( W \), such that \( \Psi(U) = q \), and \( \Psi(V) = r \).

   c) \( \alpha \) is of the form \( \beta^* \). This case is just like b) above.

(12) \( \text{Prog}_1(X) \), \( X \) codes \( R_1 \) for some model \( M \), is:

\( (\forall Y)((\text{LO}(Y) \land Y \subseteq X \rightarrow (\exists x)(\exists y)(y = x_0 \land Y(y) \land (\forall z)(Y(z) \rightarrow y \in \{b\} z))) \land X \subseteq V \)
(13) \text{Pred}(Y_i) : Y_i \text{ represents a predicate for } V \\
\quad (\forall x)(\forall y)(\forall z)(\text{Suc}(x,z) \land y = z \land V(x) \land V(y) \\
\quad \land V(z) \rightarrow (((Y_i(x) \leftrightarrow Y_i(y)) \land (Y_i(x) \leftrightarrow Y_i(z)))))

Now for each formula A of L involving a_{1,...,n}, P_1,...,P_m we define a formula 
A' of SnS involving X_1,...,X_n, Y_1,...,Y_m, V. We make the convention that the 
string variables, x, x_i correspond to the state variables, s, s_i and the set 
variables, W, W_i correspond to the path variables, p, p_i.

\text{Def:} \ (1) A \text{ is } P_i(s_j) \text{ then } A' \text{ is } Y_i(x_j) \\
(2) A \text{ is } O(p,s_1,...,s_n). \text{ Then } A' \text{ is} \\
\quad \text{LO}(W) \land W(x_1) \land ... \land W(x_n) \land \\
\quad x_1 < x_2 \land ... \land x_{n-1} < x_n \land \neg(\text{Sam}(x_1,x_2) \lor ... \lor \text{Sam}(x_{n-1},x_n)) \\
(3) A \text{ is } \alpha(p). \text{ Then } A' \text{ is } \alpha(W) \text{ defined under condition (11) above.} \\
(4) A \text{ is } Bg(s,p). \text{ Then } A' \text{ is} \\
\quad (\exists y)(\text{Sam}(x,y) \land y = \text{Init}(W)). \\
(5) a) A \text{ is } \neg B. \text{ Then } A' \text{ is } \neg B' \\
\quad b) A \text{ is } (BvC). \text{ Then } A' \text{ is } B'vC'. \\
(6) a) A \text{ is } (\exists s)B(s). \text{ Then } A' \text{ is } (\exists x)((V(x) \land B'(x)) \\
\quad b) A \text{ is } (\exists p)B(p). \text{ Then } A' \text{ is } (\exists W)(\text{LO}(W) \land W \subseteq V \land B(W)).

\text{Lemma:} \ (A) \text{ If } \Phi(x) = s \text{ and } M \models A(s) \text{ then } M' \models A'(x) \text{, where } A \text{ is a state} \\
\quad \text{formula.} \\
\quad (B) \text{ If } \psi(W) = p \text{ and } M \models A(p) \text{ then } M' \models A'(W) \text{ where } A \text{ is a path formula}

\text{Proof:} \text{ By induction on the complexity of } A.

(A) \ (1) A(s) \text{ is } P_i(s). \text{ Then } M \models A(s) \text{ iff } s \in P_i \text{ iff } \Phi(x) \in P_i \text{ iff } x \in Y_i \\
(2) A \text{ is } BvC, \text{ or } \neg B. \text{ This case is trivial by induction hypothesis.} \\
(3) A(s) \text{ is } (\exists p)(Bg(s,p) \land \alpha(p) \land B(p)) \text{ where } B(p) \text{ is a path formula.} \\
\text{If } M \models A(s) \text{ then there is such a path, call it } p. \text{ There is a corresponding } W \text{ in} \\
M', \text{ which is an appropriate linearly ordered subset satisfying } \alpha(W), \text{ and} \\
\text{beginning at } x, \text{ and such that } W(p) = p. \text{ Since } p \text{ satisfies } B(p), \text{ by induction} \\
hypothesis } W \text{ satisfies } B'(W). \text{ Hence } M' \models A'(x). \text{ Conversely if there is a } W \text{ in} \\
M' \text{ then there must have been a } p \text{ in } M.
(B) (1) Note that $\psi$ is the image of $W$ in $\mathbf{M}'$ and clearly the strings in $W$
 corresponding to the states in $\psi$ satisfy the same state formulas. Hence $\mathbf{M}\models A(\psi)$
 iff $\mathbf{M}'\models A'(W)$.

(2) $A$ is BVC or $\neg B$. This case is immediate by induction hypothesis.

Q.E.D.

Cor: Let $A$ involving $a_1, \ldots, a_n$, $P_1, \ldots, P_m$ be a closed formula in SOAPL. Let
$A'(X_1, \ldots, X_n, Y_1, \ldots, Y_m, V)$ be its image. Then $A$ is satisfiable, i.e. true in some
$\mathbf{M}$ iff

$Tr(A) = (\exists X_1, \ldots, X_n, Y_1, \ldots, Y_m, V)(\exists \psi(V_w \wedge$
$\text{Prog}_1(X_1) \land \ldots \land \text{Prog}_n(X_n) \land \text{Pred}(Y_1) \land \ldots \land \text{Pred}(Y_m) \land \psi A')$ is true in $\text{SnS}$.

Proof: Formulae of SOAPL are obtained by quantification and truth functional
operations from state formulae and path formulae. Each of the latter contains only
one free variable. So we can separate variables. I.e. each formula of SOAPL is
equivalent to a truth functional combination of formulae of the form $(\exists p) B(p)$,
$(\exists s) C(s)$, where $B(p)$ is a path formula and $C(s)$ is a state formula. Hence w.l.o.g.
we can assume that $A$ itself is $(\exists p) B(p)$, or $(\exists s) C(s)$. So suppose for instance that
$(\exists p) B(p)$ is satisfiable. Then there is an $\mathbf{M}$, $\psi$ such that $B(\psi)$ is true in $\mathbf{M}$.
then there is a $W$ in a corresponding $\mathbf{M}'$ such that $B'(W)$ is true. Let
$X_1, \ldots, X_n, Y_1, \ldots, Y_m, V$ be the subsets of $2^\times$ which code $\mathbf{M}'$. Then
$\exists \psi V_w \land \text{Pred}(Y_m) \land \psi A'$ is true in $\text{SnS}$. Hence $Tr(A)$ is true. Conversely suppose
$(\exists X_1, \ldots, X_n, Y_1, \ldots, Y_m, V)(\exists \psi V_w \land \text{Pred}(Y_m) \land A')$ is true, then there are sets $X_1, \ldots, V_w$
which satisfy $\exists \psi V_w \land \text{Pred}(Y_m) \land A'$. Hence $X_1, \ldots, X_n, Y_m, V_w$ describe a model $\mathbf{M}'$.
But it is easily seen that $\mathbf{M}'$ codes a model $\mathbf{M}$ obtained by identifying points $x_1, x_2, x_3$ as a single state.

Q.E.D.

We still have to discuss the changes necessary if we are allowing the basic
programs $R_1$ to be nondeterministic. Note that if $x$ represents the state $s$, then
in certain circumstances so will $y = x \cup c$, but the latter will represent $s$ as the
starting state of the program $a_1$. Now if $a_1$ can have several paths starting at
$s$, then the corresponding starting strings will have to be distinct. We can
accomplish this with another letter $d$ in $\Sigma$ and let $x_1 \cup d^k c$ represent $s$ as the
starting point of the kth path in R1 starting at s. The rest of the construction will now go through as before.

Open question 1: Does SOAPL have the finite model property? The particular acyclic models coded into Sns can be forced to be infinite. However there may be other finite models.

Open question 2: Does SOAPL have an elementary decision procedure? A positive answer to question 1 above would probably also yield a positive answer to this one.

Open question 3: How strong is SOAPL compared to the various special languages due to Pratt etc. that we have mentioned? D. Harel [Ha] has shown that there are process connectives expressible in SOAPL that are not expressible in Pratt's process logic, but the question still remains what happens if Harel's example and a few others like it are added. Harel's particular connective $\eta(\alpha, A, B)$ says: If A holds during any computation of $\alpha$, then B holds at the same time or sometime later.

Open question 4: What is the degree of undecidability of the full language L? We know it is undecidable, but probably, in the presence of path quantifiers, the degree of undecidability is quite high.

Bibliography


