REACHABLE AND SEMIFREE
SCHEMAS.

by: Elaine J. Wayuker

DCS-TR-53

Department of Computer Sci.
Rutgers, The State University of N.J.
New Brunswick, New Jersey 08903

Feb. 1977
Abstract

Paterson introduced the notions of freedom and liberality as semantic restrictions on the class of schemas. He felt that such restricted classes of schemas might have solvable decision problems, and also be realistic models of what would generally be considered "good" or "efficient" programs. With the latter viewpoint in mind, we introduce two new classes of schemas: the reachable schemas and the semifree schemas. A schema is reachable if every statement in the schema is executed under some interpretation. A schema is semifree if every test in the schema is necessary in the sense that each exit of the test is taken under some interpretation.

We discuss various decision problems for these classes, and translatability between these classes and other semantically restricted schema classes. We also mention the implications of these results for questions of program optimization and verification.
1. INTRODUCTION

For several years, people have been aware of the usefulness of modeling computer programs by considering abstractions known as program schemas. A great deal of work has been done comparing the relative computational power of classes of schemas with additional features [1], [2], [6]. We are interested in considering classes of schemas whose members fulfill certain semantic requirements. The resulting classes are models of what would generally be considered programs which have desirable properties. With this motivation, we introduce two such classes of schemas, the semifree schemas and the reachable schemas, and consider various decision problems for these classes. We also consider the relative power of these classes. We compare them to the class of all schemas as well as to other well-known semantically restricted classes.

Our definition of schemas is essentially that of Luckham, Park, and Paterson [4]. We consider the set of variables to be composed of three disjoint sets--input variables X, program variables Y, and output variables Z. Input variables may be retrieved from, but never assigned new values, while output variables may be assigned to but never retrieved from. Program variables must be assigned a value before being retrieved from. We also explicitly require that every node of the flow diagram representing a schema lie on a path from the start statement. Thus we do not consider schemas with disconnected graphs.

The semantics of a schema is provided by an interpretation which specifies a domain, assigns actual functions and predicates to the function and predicate symbols of the schema, and also assigns initial values from the domain to each input variable.
The execution sequence for schema \( P \) under interpretation \( I \), denoted \( \sigma(P,I) \), consists of the sequence of instructions of \( P \) executed under interpretation \( I \). For each interpretation \( I \), the computation of the schema \( P \) either terminates (i.e., reaches a halt statement), or diverges. In the former case the value, denoted \( \text{val}(P,I) \), is the \( n \)-tuple of current values of \( P \)'s \( n \) output variables. If \( P \) diverges under \( I \), \( \text{val}(P,I) \) is undefined.

We say two schemas, \( P \) and \( Q \), are strongly equivalent, denoted \( P \equiv Q \), if for every interpretation \( I \), either both \( \text{val}(P,I) \) and \( \text{val}(Q,I) \) are defined and \( \text{val}(P,I) = \text{val}(Q,I) \), or both values are undefined.

A class of schemas \( \mathcal{A}_1 \) is translatable into a class \( \mathcal{A}_2 \), if for every \( P_1 \in \mathcal{A}_1 \), there is a strongly equivalent \( P_2 \in \mathcal{A}_2 \).

2. SEMANTICALLY RESTRICTED CLASSES OF SCHEMAS

Paterson [5] introduced the notions of freedom and liberality as semantic restrictions on the class of schemas. He felt that such restricted classes of schemas might have solvable decision problems, and also eliminate from consideration repetitions in computations.

A schema is \textit{free} if every finite path through its flow diagram from the start statement is an initial segment of some execution sequence. This property has been shown to be equivalent to a restriction that under no Herbrand interpretation is any predicate ever applied to an \( n \)-tuple of elements of the universe more than once.

A schema is \textit{liberal} if for every Herbrand interpretation, no element of the universe is computed more than once.

A \textit{statement} in a schema is \textit{reachable} if there is an interpretation under which that statement is executed. A \textit{schema} \( \hat{P} \) is \textit{reachable} if
every statement in $P$ is reachable.

A schema $P$ is semifree if for every edge in the flow diagram of $P$, there is some interpretation under which that edge is traversed.

We use $\mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{J}$, and $\mathcal{P}$ to represent the classes of free, liberal, reachable, semifree, and all schemas respectively.

2.1 REACHABLE SCHEMAS

We have already mentioned one motivating force for defining the class $\mathcal{R}$. We stated that reachability is a property of every "good" program. A second reason for our interest in this class is related to our interest in program verification. Generally, a programmer does not intentionally write a program which contains inaccessible code. If we could determine that in fact a program does contain such code, it would frequently signal that the program is not performing as intended for some inputs.

It has been shown [5] that $\mathcal{P}$ is not translatable into $\mathcal{I}$ or $\mathcal{L}$. Since we consider reachability a necessary property of any schema which we would consider "good", it is significant that corresponding to every schema $P$, there is a reachable version $Q$. $Q$ is simply the schema $P$ with any unreachable code removed. Thus we have immediately:

**Proposition 1** $\mathcal{P}$ is translatable into $\mathcal{R}$. □

We shall see, however, that this translation is not effective.

**Lemma 2** It is decidable whether a reachable schema $P$ halts under some interpretation.

**Proof** $P$ halts under some interpretation iff it contains a halt statement. □

**Theorem 1** There is no algorithm which given an arbitrary schema $P$,
constructs a reachable schema Q, such that P ≠ Q.

Proof Assume such and algorithm existed. Then by lemma 2, it would be decidable whether Q, and hence P, halted under some interpretation. But this is a well-known [5] undecidable property for arbitrary schemas. □

Thus we have seen that although every schema has a strongly equivalent reachable schema, we cannot in general effectively obtain it. The next obvious question is, whether we can decide if a schema is reachable. It has been shown [5] that membership in $\mathcal{X}$ is decidable but membership in $\mathcal{Z}$ is not.

Lemma 4 It is not decidable whether an arbitrary assignment statement of a schema is reachable.

Proof Let $P$ be an arbitrary schema. We assume without loss of generality that $P$ contains a single halt statement. We construct a schema $Q$ from $P$ by replacing the halt statement of $P$ by an assignment statement $s$, followed by a halt statement. Then if it were decidable whether $s$ is reachable in $Q$, it would be decidable whether $P$ halted under some interpretation. □

Theorem 5 It is not decidable whether a schema is reachable.

Proof We prove this theorem by showing that there is an algorithm which, given an arbitrary schema $P$ and assignment statement $s_k$ of $P$, constructs a schema $Q$ such that $Q$ is reachable iff $s_k$ is reachable in $P$. Let $P$ be an arbitrary schema with statements $s_1, \ldots, s_n$. Let $s_n$ be an assignment statement of $P$. Let $\mathcal{P}$ be an n-exit predicate symbol (or alternately a series of $n-1$ two exit predicate symbols) which does not appear in $P$. We construct schema $Q$ from $P$ by inserting an initializing assignment statement, $\forall x$, immediately following the start statement.
The notation indicates that we are assigning the value of some input variable $x$ to every program variable $y$. This is done simply to guarantee that every program variable is assigned a value before it is retrieved from. We also insert the test $p(x)$ after instruction $s_k$. The branch from the $m$-th exit of $p$ enters statement $s_m$. The construction of $Q$ is outlined in figure 1.

![Diagram of schema Q]

We now show that $s_k$ is reachable in $P$, iff $Q$ is a reachable schema. If $s_k$ is reachable in $P$, then clearly it is reachable in $Q$, and hence the predicate $p$ is reachable in $Q$. Thus $Q$ is a reachable schema. If $s_k$ is unreachable in $P$, then it is unreachable in $Q$ and hence $Q$ is not a reachable schema. □

Before considering similar questions for $\mathcal{R}$, we shall consider two other decision problems for $\mathcal{R}$. Since the complete proof of the first of these problems requires considerable detail, we merely outline it here. The complete construction and proof may be found in [8].

**Theorem 6** It is not decidable whether a reachable schema halts under every interpretation.
Proof (In outline) We prove this theorem by showing that there is an algorithm which given an arbitrary schema P, constructs a reachable schema Q such that P halts under every interpretation iff Q halts under every interpretation. Thus, since Paterson [5] showed that it is undecidable whether an arbitrary schema halts under every interpretation, there can be no algorithm for deciding whether a reachable schema halts under every interpretation. Intuitively, Q is constructed so that it can be in one of two modes. One mode is used to simulate P, and the other is used to guarantee that Q is reachable. Q halts under every interpretation which causes it to enter the latter mode, and thus we have the required reduction. We note that P is not in general strongly equivalent to Q, for when Q is in reachability mode, we are indifferent to the value computed. 

Corollary 7 It is not decidable whether two reachable schemas are strongly equivalent.

Proof Let P be a reachable schema with input variable x, and output variables z₁,...,zₙ. Let Q be the reachable schema shown in figure 2. Note that Q halts under every interpretation.

\[\text{START} \overset{z_i}{\rightarrow} \vdots \overset{z_n}{\rightarrow} \text{HALT}\]

**FIGURE 2-Schema Q**
We construct schema $R$ by replacing each halt statement of $P$ by the sequence of instructions:

$$\xrightarrow{Z_1 \leftarrow X} \ldots \xrightarrow{Z_n \leftarrow X} \text{HALT}$$

Clearly $R$ is reachable iff $P$ is reachable.

Also, $Q \equiv_R$ iff $R$ halts under every interpretation

iff $P$ halts under every interpretation.

Thus if strong equivalence were decidable for reachable schemas, we could decide whether a reachable schema halts under every interpretation, contradicting theorem 6. □

2.2 SEMIFREE SCHEMAS

The semifree schemas represent an additional restriction on the class of reachable schemas. Not only must we be able to reach every statement in the schema, but we must also be able to leave via any exit.

In this section we shall extend several of the results for reachable schemas to include semifree schemas. If the proof is similar to the corresponding proof for $R_L$, it is omitted.

Paterson showed that a schema is free iff it does not contain any repeated tests. Furthermore, there are schemas which are inherently
nonsense in the sense that they are not strongly equivalent to any free schema, and thus must repeat some tests in order to do the desired computation. Our motivation for defining semifreedom is to somehow distinguish between schemes which have redundant (i.e. unnecessarily repeated) tests, and those which contain a repetition of a test only when it is actually required for the calculation. Of course, it is difficult to say exactly what it means for a test to be required. Using the current model which contains 2-way test statements, we cannot simply consider a test in a schema $P$ useless provided the schema formed by removing the test from $P$ is strongly equivalent to $P$. In general the removal of such a test provides us with no obvious means of deciding which of the two exits from the test should become the successor statement of the test's predecessor statements.

We shall say that a test $t$ is necessary if there are interpretations $I_0$ and $I_1$ such that the 0-exit of $t$ is taken under $I_0$ and the 1-exit is taken under $I_1$.

Intuitively, this says that for some interpretations one exit of the test is taken, and for other interpretations, the other exit is taken. Thus, one feels that the test is really testing something, and in that sense is necessary for the computation.

**Proposition 8** A schema $P$ is semifree iff every test in $P$ is necessary.$\Box$

**Proposition 9** Every semifree schema is reachable.$\Box$

We see that corresponding to every schema $P$, there is a semifree version $Q$. In this case, unnecessary tests must be removed in addition to any unreachable code. To remove an unnecessary test $t$, we simply delete the test and connect its direct predecessors to the statement at the exit which is always taken. We shall see that as in the case of
\[ \Lambda, \text{ this translation is not effective.} \]

**Proposition 10** \( \Lambda \) is translatable into \( \mathcal{J} \). \( \square \)

**Theorem 11** There is no algorithm which given an arbitrary schema \( \Lambda \), constructs a semifree schema \( Q \), such that \( \Lambda \equiv Q \).

**Proof** This is a direct consequence of proposition 9 and theorem 3. \( \square \)

The next theorem tells us that as in the cases of \( \Lambda \) and \( \mathcal{J} \), membership in \( \mathcal{J} \) is not decidable.

**Theorem 12** It is not decidable whether an arbitrary schema is semifree.

**Proof** (In outline) the proof of this theorem is similar in spirit to that of theorem 5, but requires a more complicated construction. In this case, there is an algorithm which given an arbitrary schema \( P \) and assignment statement \( s_K \) of \( P \), constructs a schema \( Q \), such that \( Q \) is semifree iff \( s_K \) is reachable in \( P \). The complete construction can be found in [8]. \( \square \)

The final question which we shall consider in this section is the equivalence problem for semifree schemas. We first consider the question of halting under every interpretation for this class.

**Theorem 13** It is not decidable whether a semifree schema halts under every interpretation.

**Proof** (In outline) This theorem is proved by producing an algorithm which given an arbitrary reachable schema \( P \), constructs a semifree schema \( Q \) such that \( P \) halts under every interpretation iff \( Q \) halts under every interpretation. Thus it follows from theorem 8 that it is undecidable whether \( Q \) halts under every interpretation. \( Q \) is constructed so that it can be in either "simulation of \( P \) mode" or "guaranteeing semifreedom mode". The mode is determined by an interpretation, and may not be changed during the computation. We
outline the construction of Q here.
Let $t_1, \ldots, t_n$ denote the test statements of $P$, and $k_i$ denote the $i$-successor of $t_k$, $k=1, \ldots, n; i=0, 1$.
Let $p$ be a predicate symbol not appearing in $P$, and let $q$ be an $n$-exit predicate symbol not appearing in $P$.
We construct $P'$ from $P$ by inserting a copy of the test statement $px$ as the 0-successor of each test $t_k$, and also as the 1-successor of each $t_k$ (thus $2n$ copies of the test $px$ have been added to $P$). The 0-exit of the test $px$ which is the 1-successor of $t_k$ enters statement $k_0$. The 1-exit enters a halt statement. Intuitively, $(Ip)(x)=1$ will be semifreedom mode and this portion of the construction will guarantee that $Q$ will always halt when in this mode. All edges which enter $k_i$ in $P$, enter the $px$ test preceding $k_i$ in $P'$. Thus, if in $P$ we have:

![Diagram](attachment:image.png)

where $\rightarrow$ denotes all edges entering $k_i$ in $P$. Then in $P'$ we have:

![Diagram](attachment:image.png)
We denote $P$, so modified, by $P'$. We next construct $Q$ from $P'$. Let $s_1$ designate the successor to the start statement in $P'$. When constructing $Q$ from $P'$, we insert an initializing statement $\overline{y} \leftarrow x$ immediately following START. As in the construction in theorem 5, this simply assures compliance with the restriction that every program variable be assigned a value before it is tested or retrieved from. Following this initializing statement, we insert a copy of the test $p_x$. The 0-successor of this test is $s_2$, the 1-successor will be the $n$-exit test $q_x$. The $k$-th exit of $q$ has a branch to statement $t_k$. Thus $Q$ is as follows.

A straightforward argument can assure us that $Q$ has the required properties, and is omitted here. □

**Corollary 14** It is not decidable whether two semifree schemas are strongly equivalent.

**Proof** The proof of this corollary follows immediately from theorem 13, the proof of corollary 7, and the observation that the schema $Q$ of that proof is semifree. □
3. **RELATIONSHIPS BETWEEN \( \mathcal{R} \) AND \( \mathcal{A} \) AND OTHER SEMANTICALLY RESTRICTED CLASSES OF SCHEMAS**

It follows easily from our definitions that \( \mathcal{F} \subseteq \mathcal{A} \subseteq \mathcal{R} \subseteq \mathcal{P} \), and it can be demonstrated that these inclusions are strict. We have seen in the previous section that many properties are not decidable for schemas in \( \mathcal{R} \) and \( \mathcal{A} \), and in fact even membership in these two classes is undecidable. In this section we investigate under what circumstances knowing that a schema possesses certain desirable properties, is sufficient to allow us to decide whether it has other desirable properties.

We begin by considering whether knowing that a schema is reachable is sufficient to allow us to decide whether it is semifree. Since reachability is partially decidable, such an algorithm would be particularly helpful for automatic code improvement. We omit the proof of the next theorem as the construction is quite complicated and somewhat similar to the one used in the proof of Theorem 12. A complete proof can be found in [8].

**Theorem 15** It is not decidable whether a reachable schema is semifree.

Furthermore, it follows immediately from corollary 14 and proposition 9 that we cannot decide whether a reachable schema and a semifree schema are strongly equivalent.

Similarly, we had hoped that since every free schema is semifree, semifreedom might be sufficient to allow us to decide freedom. The next theorem tells us that this is not the case.

**Theorem 16** It is not partially decidable whether a semifree schema is free.
Proof Assume such a partial decision procedure existed. It can be demonstrated that semifreedom is partially decidable. Thus we can apply the hypothesized partial decision procedure and select the semifree, free schemas. But \( \mathcal{F} \subseteq \mathcal{S} \) and hence we would have a partial decision procedure for freedom, which Paterson [5] showed was not partially decidable. □

Using an argument similar to that of corollary 7, the next related question follows.

Theorem 17 It is not decidable whether a semifree schema, and a free schema are strongly equivalent. □

We next consider the relationship between \( \mathcal{L} \) and \( \mathcal{R} \). Paterson [5] showed that \( \mathcal{L} \) is effectively translatable into \( \mathcal{L} \cap \mathcal{J} \). Thus is effectively translatable into \( \mathcal{R} \).

Lemma 18 It is decidable for a liberal schema, whether it halts under some interpretation.

Proof Let \( P \) be a liberal schema. Let \( Q \) be the liberal, free schema obtained by applying Paterson's algorithm to \( P \). \( P \equiv Q \). Clearly \( P \) halts under some interpretation iff \( Q \) halts under some interpretation. But \( Q \) halts under some interpretation iff it contains a halt statement. □

Theorem 19 It is decidable for a liberal schema, whether an arbitrary statement is reachable.

Proof Let \( P \) be a liberal schema with statements \( s_1, \ldots, s_n \).

For each \( k, 1 \leq k \leq n \), we shall construct a liberal schema \( P_k \) from \( P \), such that \( P_k \) halts under some interpretation, iff \( s_k \) is reachable in \( P \). To construct \( P_k \), we replace each halt statement of \( P \) by a \text{LOOP} statement, and statement \( s_k \) by a halt statement. If statement \( s_k \) was anything other than a halt statement, we remove all edges leaving \( s_k \),
and delete any portions of the flow diagram which are disconnected as a result of this replacement. Clearly \( P_k \), so constructed, has the required properties. Therefore by lemma 18 we can decide whether \( P_k \) halts under some interpretation and hence whether \( s_k \) is reachable in \( P \).

**Corollary 20** It is decidable whether a liberal schema is reachable.

The last question we consider in this section, is whether liberality is sufficient to allow us to decide semifreedom. We first show that for liberal schemas we can decide whether a test is necessary.

**Theorem 21** It is decidable whether an arbitrary test statement of a liberal schema is necessary.

**Proof** Let \( P \) be a liberal schema containing a test statement \( t_k \). We shall construct a liberal schema \( P_k \) from \( P \) such that \( t_k \) is a necessary test in \( P \) iff specific statements in \( P_k \) are reachable. \( P_k \) is constructed as follows.

Let \( y \) be a program variable which does not appear in \( P \), \( f \) a function symbol not appearing in \( P \), and \( x \) an input variable of \( P \). We insert the assignment statement \( y \leftarrow fx \) between the start statement and its successor.

If we have the following construction in schema \( P \),

![Flow diagram](image-url)
we replace it with the following construction in $P_K$

\[ \begin{array}{c}
\downarrow \\
\downarrow \\
\downarrow
\end{array} \]

\[ t_K \]

\[ \begin{array}{c}
\downarrow \\
\downarrow \\
\downarrow
\end{array} \]

\[ y \leftarrow f y \]

\[ S_L \]

\[ \begin{array}{c}
\downarrow \\
\downarrow \\
\downarrow
\end{array} \]

\[ y \leftarrow f y \]

\[ S_R \]

$t_K$ is a necessary test in $P$ iff both copies of the instruction $y \leftarrow f y$ are reachable in $P_K$. But by theorem 19, since $P_K$ is liberal, it is decidable whether these statements are reachable.

Corollary 22 It is decidable whether a liberal schema is semifree.

We note that the preceding four results are particularly interesting since liberality is a decidable property.

4. REMARKS

Classes analogous to the ones discussed here for schemas have also been introduced for programs [8]. In most cases similar results were obtained when we considered programs directly instead of looking at their abstractions. We feel that the results described here, and others included in [8] are of both theoretical interest and also provide particular insight into some difficulties inherent in algorithmic program optimization and improvement.
ACKNOWLEDGEMENTS

The author is grateful to Tom Ostrand and Ann Yasuhara for all their encouragement, helpful discussions and suggestions.
References


