THE INCLUSION PROBLEM FOR MONADIC RECURSION SCHEMES

by

Emily Perlinski Friedman

2-73

Center for Research in Computing Technology

CASE FILE COPY

Harvard University
Cambridge, Massachusetts 02138
THE INCLUSION PROBLEM FOR
MONADIC RECURSION SCHEMES†

by

Emily Perlinski Friedman

2-73

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts 02138

†This research has been supported in part by the National Science Foundation under Grant NSF GJ-30409 and by the National Aeronautics and Space Administration under Grant NGR 22-007-176.
Abstract

The inclusion problem for the class of monadic recursion schemes is shown to be undecidable. The proof illustrates the close relationship between monadic recursion schemes and deterministic pushdown automata. The proof is extended to show that both the weak equivalence problem for the class of monadic recursion schemes and the weak equivalence problem for the class of free schemes without identity are undecidable.
Schemes can be viewed as abstract models for computer programs. They allow us to study aspects of a computation that are independent of the actual functions, variables and predicates involved. In this paper we shall be concerned with one particular class of schemes—monadic recursion schemes.

In a monadic program scheme there is exactly one variable $x$, a set of unary functions $(f_0, \ldots, f_m)$ that assign values to the variable $x$, and a set of unary predicates $(p_0, \ldots, p_n)$ that determine the flow of computation. In addition, there is a set of function variables $(F_0, \ldots, F_k)$ that are defined below.

Let a term be defined in the usual way as constructed from functions and function variables applied to the variable $x$, e.g.,

$$f_1(F_2(F_4(f_0(x))))$$

A conditional term is any expression of the form

$$\text{if } p_i(x) \text{ then } T_1 \text{ else } T_2,$$

where $T_1$, $T_2$ are terms or conditional terms. A function variable definition is:

$$F_i \triangleq T,$$

where $T$ is any term or conditional term. Since $x$ is the only variable, we will abbreviate
\[ F_1(x) \leftarrow \text{if } p_2(x) \text{ then } f_1(F_2(x)) \text{ else } x \]
as
\[ F_1 \leftarrow \text{if } p_2 \text{ then } f_1 F_2 \text{ else } \text{ID}, \]
where \( \text{ID} \) is the identity function.

We can now formally define a monadic recursion scheme \( S \) as a 5-tuple \( S = (V, \mathcal{F}, \mathcal{P}, \mathcal{D}, F_0) \), where

\[ V = \text{a finite set of function variables} \]
\[ \mathcal{F} = \text{a finite set of functions} \]
\[ \mathcal{P} = \text{a finite set of predicates} \]
\[ \mathcal{D} = \text{a finite set of function variable definitions (exactly one for each element in } V) \]
\[ F_0 \in V \text{ the distinguished initial function variable} \]

Unless otherwise noted "scheme" shall mean "monadic recursion scheme".

If we assign a value to the variable \( x \) and associate actual functions and predicates with the scheme (e.g., \( f_1(x) \sim \sqrt{x}, \ p_2(x) \sim x = 0 \) ), then the scheme can be looked upon as an executable program. Such associations are called interpretations. Formally, an interpretation \( I \) has domain \( D \) of possible storage values (for the variable \( x \) ), and a distinguished element \( d_0 \in D \) used as the initial value of \( x \) (or, rather, the input). Since the scheme is monadic, this is the only treatment of variables necessary. For each function \( f \), there is a total function \( I(f): D \to D \), and for each predicate \( p \), a total function \( I(p): D \to \{T, F\} \). So, for any given interpretation \( I \), the scheme \( S \) can be evaluated in the normal sense by applying the initial function variable \( F_0 \) to input \( d_0 \). The computation either terminates, yielding a value called \( \text{val}_I(S) \), or it diverges and \( \text{val}_I(S) \) is undefined.
Given two schemes S and S', we say that S is less defined than S' (S ≤ S') iff for every interpretation I, whenever val_I(S) is defined, then val_I(S') is also defined and val_I(S) = val_I(S'). Because of the obvious difficulty with proving properties over all interpretations, we define a more restrictive type of interpretation. An interpretation will be called free if for variable x, I(x) = ε (i.e., the empty word), and I(f)(a) = fa, where this is just the concatenation of the function symbol f to the string a ∈ Dom(I). Note the resemblance to the Herbrand Universe. In fact, such interpretations are often called Herbrand interpretations in the literature. It is important to realize that we have not restricted the predicates; because of this, we can obtain an infinite number of free interpretations for any scheme. This notion of free interpretation now yields the following useful result:

**Lemma**: Given any two schemes S and S', S ≤ S' iff for all free interpretations I, whenever val_I(S) is defined, then val_I(S') is defined and val_I(S) = val_I(S').

**Proof**. This is similar to the result on equivalence in [5].

To show that a problem is unsolvable, it is often convenient to use the unsolvability of the Post Correspondence Problem (PCP). The Post Correspondence Problem is defined as follows: Let Σ be a finite set containing at least two elements, and let Π be a non-empty sequence of 2-tuples of strings in Σ+. For example,
\[ \mathcal{P} = (x_1, y_1), \ldots, (x_n, y_n) \]

where for \( i = 1, \ldots, n \), \( x_i, y_i \in \Sigma^+ \).

This is an instance of the PCP. The sequence of indices \( i_1, \ldots, i_t \) with \( t > 1 \) is a solution to this instance of the PCP if \( x_{i_1} \cdots x_{i_t} = y_{i_1} \cdots y_{i_t} \).

It is well known that the PCP is undecidable.

In [6], Paterson states that the question of whether or not the inclusion problem (\( S \subseteq S' \)) for monadic recursion schemes is decidable is open. The following theorem shows that this problem is undecidable. First we shall sketch a proof of the known result that the inclusion problem for languages accepted by deterministic pushdown automata (dpda) is undecidable. This construction [6] will give an indication as to how we will later prove the main theorem of this paper.

Let \( \Sigma = \{a, b\} \) and let \( \mathcal{P} = (x_1, y_1), \ldots, (x_n, y_n) \), where for \( i = 1, \ldots, n \), \( x_i, y_i \in \Sigma^+ \). Encode the indices \( 1, \ldots, n \), as symbols \( f_1, \ldots, f_n \) respectively. Define the following two languages \( L_1, L_2 \subseteq \{a, b, c, \$, f_1, f_2, \ldots, f_n\}^* \):

\[
L_1 = \{f_{i_1} \cdots f_{i_t} x_{i_1}^R \cdots x_{i_t}^R \$ | t \geq 1 \text{ and } i_1, \ldots, i_t \text{ are indices from } 1 \text{ to } n\}
\]

\[
L_2 = \{f_{i_1} \cdots f_{i_t} cw | t \geq 1 \text{ and } i_1, \ldots, i_t \text{ are indices from } 1 \text{ to } n, \text{ and } w \not= y_{i_1}^R \cdots y_{i_t}^R\}
\]
Both \( L_1 \) and \( L_2 \) can be accepted by dpda's, and \( L_1 \subseteq L_2 \) iff there does not exist a solution to the PCP for \( \mathcal{F} \). Thus, the inclusion problem for dpda's is undecidable.

**Theorem:** The inclusion problem for monadic recursion schemes is undecidable.

**Proof:** Let \( \Sigma = \{a, b\} \) and let \( \mathcal{F} = (x_1, y_1), \ldots, (x_n, y_n) \) where

\[
\text{for } i = 1, \ldots, n, \quad x_i, y_i \in \Sigma^*.
\]

Let \( \Gamma = \{A, B\} \), and define a homomorphism \( h: \Sigma^* \rightarrow \Gamma^* \) determined by

\[
h(a) = A, \quad h(b) = B.
\]

Let \( \hat{\Gamma} = \{\hat{A}, \hat{B}\} \), and define the function \( g: \Sigma^* \rightarrow \hat{\Gamma}^* \) as follows:

\[
\text{if } zw \in \Sigma^* \text{ where } z \in \{a, b\}, \text{ then}
\]

\[
g(zw) = \begin{cases} 
\hat{A} h(w), & \text{if } z = a \\
\hat{B} h(w), & \text{if } z = b 
\end{cases}
\]

We will now define two schemes \( S, S' \) such that \( S \subseteq S' \) iff there does not exist a solution to the PCP for \( \mathcal{F} \). We will see that \( S \subseteq S' \Rightarrow \)

\[
\{\text{val}_I(S) \mid I \text{ is a free interpretation for } S\} \subseteq \{\text{val}_I(S') \mid I \text{ is a free interpretation for } S'\}.
\]

Let \( S = (V, \mathcal{F}, \mathcal{D}, F_0) \), where

\[
V = \{F_0, F_1, \ldots, F_n\} \cup \{X, U\},
\]

\[
\mathcal{F} = \{a, b, c, s\} \cup \{f_1, \ldots, f_n\},
\]

\[
\mathcal{D} = \{q_1, \ldots, q_n\} \cup \{p_a, p_b, p_s\}, \quad \text{and } \mathcal{D} \text{ is defined as follows:}
\]

(The comments that follow point out the similarities to acceptance by a dpda):
\[ F_0 \leftarrow XF_1 \]  
Mark the end of a computation. This is similar to placing a marker on the bottom of the pushdown store in a DPDA.

\[ F_1 \leftarrow \text{if } q_1 \text{ then } h(x_1)F_1f \text{ else } F_2 \]  
h\(x_i\) is a string that encodes \(x_i\).

\[ F_2 \leftarrow \text{if } q_2 \text{ then } h(x_2)F_1f_2 \text{ else } F_3 \]  
Note how this is like pushing the string \(x_i\) onto the pushdown store when an index \(f_i\) is read from the input string. The operation \(c\) indicates the end of "reading" indices.

\[ F_n \leftarrow \text{if } q_n \text{ then } h(x_n)F_1f \text{ else } c \]  
Recall that \(h(x_i)\) is a string in \((A,B)^+\).

\[ A \leftarrow \text{if } p_a \text{ then } a \text{ else } U \]  
We can view predicates \(p_a, p_b,\) and \(p_\$\) as "testing if the symbol read from the input string is an \(a, b,\) or \(\$,\) respectively."

\[ B \leftarrow \text{if } p_a \text{ then } U \text{ else if } p_b \text{ then } b \text{ else } U \]  
Thus, \(A\) is undefined ("rejects") unless \(p_a\) is true; \(B\) is undefined unless \(p_b\) is true.

\[ X \leftarrow \text{if } p_a \text{ then } U \text{ else if } p_b \text{ then } U \text{ else if } p_\$ \text{ then } \$ \text{ else } U \]  
\(X\) is the leftmost function variable, and it is undefined unless \(p_a, p_b\) are false and \(p_\$\) is true. That is, we have reached "the end of the string."

\[ U \leftarrow U \]  
This is a loop, so that whenever \(U\) is encountered in a computation, the value of the scheme is undefined for that interpretation.

It is clear from the definition of \(S\) that

\[
\{\text{val}_I(S) \mid I \text{ is a free interpretation for } S\} = \\
\{Sx_{i_1}...x_{i_t}c_{f_1}...f_{t_1} \mid \text{for } j=1,...,t, \ 1 \leq i_j \leq n\} \cup \{$c$\} 
\]
We could have written a "simpler" scheme that would have also produced the same set of values by using the following definitions of $\mathcal{D}$:

\[
\begin{align*}
F_1 &\leftarrow \text{if } q_1 \text{ then } x_1 F_1 f_1 \text{ else } F_2 \\
&\hspace{1cm} \vdots \\
F_n &\leftarrow \text{if } q_n \text{ then } x_n F_1 f_n \text{ else } c
\end{align*}
\]

It will become clear later why we have chosen not to proceed in this manner.

Define scheme $S'$ as follows: $S' = (V', \mathcal{F}, \mathcal{P}, \mathcal{D'}, F_1')$, where $V' = \{F_1', \ldots, F_n'\} \cup \mathcal{T} \cup \{T, T, E, U\} \cup \{F_1, \ldots, F_n\}$, and $\mathcal{D'}$ is defined as follows:

\[
\begin{align*}
F_1' &\leftarrow \text{if } q_1 \text{ then } g(y_1) F_1 f_1 \text{ else } F_2' \\
F_2' &\leftarrow \text{if } q_2 \text{ then } g(y_2) F_2 f_2 \text{ else } F_3' \\
&\hspace{1cm} \vdots \\
F_n' &\leftarrow \text{if } q_n \text{ then } g(y_n) F_1 f_n \text{ else } c
\end{align*}
\]

The same comments apply as for scheme $S$, except that the leftmost function variable is now marked with a $\hat{\text{ }}$. Also, $F'$ has $c$ as the else clause instead of $c$, since we do not have a leftmost indicator $x$.

\[
\begin{align*}
\hat{F}_1 &\leftarrow \text{if } q_1 \text{ then } h(y_1) \hat{F}_1 f_1 \text{ else } \hat{F}_2 \\
\hat{F}_2 &\leftarrow \text{if } q_2 \text{ then } h(y_2) \hat{F}_1 f_2 \text{ else } \hat{F}_3 \\
&\hspace{1cm} \vdots \\
\hat{F}_n &\leftarrow \text{if } q_n \text{ then } h(y_n) \hat{F}_1 f_n \text{ else } c
\end{align*}
\]

same comments as for scheme $S$
A + if \( p_a \) then a else if
\[ \begin{align*}
& p_b \text{ then } \text{Th else if} \\
& p_S \text{ then } \text{ID else U}
\end{align*} \]
If \( p_a \) is true ("a is read from the input string"), then continue computing for the remaining function variables. Otherwise, if \( p_b \) is true, then we "accept the string by reading until $ is reached" (via function variable T). However, if \( p_S \) is true, then the "string is shorter than \( y_1 \cdots y_L \), so accept" by reducing to ID. All remaining function variables also reduce to ID until the leftmost is encountered (\( \hat{A} \) or \( \hat{B} \)), which then computes $.

B + if \( p_a \) then Ta else if
\[ \begin{align*}
& p_b \text{ then } b \text{ else if} \\
& p_S \text{ then } \text{ID else U}
\end{align*} \]
dual of A above

\( \hat{A} + if \) \( p_a \) then Ea else if
\[ \begin{align*}
& p_b \text{ then } \hat{\text{Th else if}} \\
& p_S \text{ then } \$ \text{ else U}
\end{align*} \]
This is the leftmost function variable so if \( p_a \) is true, then we must check via function variable E whether the "next symbol in the input string is the endmarker $, indicating the end of the string." Otherwise, "accept the remaining string that ends in $,"

\( \hat{B} + if \) \( p_a \) then Ta else if
\[ \begin{align*}
& p_b \text{ then } \hat{\text{Eb else if}} \\
& p_S \text{ then } \$ \text{ else U}
\end{align*} \]
dual of A above

T + if \( p_a \) then Ta else if
\[ \begin{align*}
& p_b \text{ then } \text{Th else if} \\
& p_S \text{ then } \text{ID else U}
\end{align*} \]
This acts like a state in a dpda that reads until the endmarker $; when \( p_S \) is finally true, the ID function causes all remaining function variables to also reduce to ID until the leftmost is encountered (\( \hat{A} \) or \( \hat{B} \)), which then computes $.
\[ \hat{T} \rightarrow \text{if } p_a \text{ then } \hat{T}a \text{ else if } \]
\[ p_b \text{ then } \hat{T}b \text{ else if } \]
\[ p_S \text{ then } \$ \text{ else } U \]

Similar to \( T \) above, but \( \hat{T} \) is encountered only when no other function variable remains to be computed. Hence, when \( p_S \) is true, the function \( \$ \) is applied immediately.

\[ \hat{E} \rightarrow \text{if } p_a \text{ then } \hat{T}a \text{ else if } \]
\[ p_b \text{ then } \hat{T}b \text{ else } U \]

This basically checks for the end-marker \( \$ \). \( p_S \) true indicates that we have "read a string \( $y_1 \ldots y_t \), so reject." Otherwise, "accept" via \( \hat{T} \).

\[ U \rightarrow U \]

Loop.

It is implicit in the comments above that

\[ \{ \text{val}_I(S') \mid I \text{ is a free interpretation for } S' \} = \]

\[ \{ \$wcf_{i_t} \ldots f_i \mid \text{for } j=1,\ldots,t, \ 1 \leq i_j \leq n, \text{ and } w \in \Sigma^*, \]
\[ w \neq y_{i_1} \ldots y_{i_t} \cup \{ \$ \} \cup \{ c \} \]

The theorem follows from the undecidability of the correspondence problem and the following claim.

**Claim.** \( S \not\equiv S' \) iff there does not exist a solution to the PCP for \( \mathcal{P} \).

**Proof.** \( \Leftarrow \) Suppose that \( S \) is not less defined than \( S' \). Clearly, for any free interpretation \( I \), \( S \) and \( S' \) apply functions in \( \{ f_1, \ldots, f_n \} \) in the same order, so we only need to check what happens in a computation after the application of function \( c \). The only predicates involved there are \( p_a, p_b \) and \( p_S \). These can be viewed as testing if the function to be applied is an \( a, b, \) or \( \$, \) respectively. Both
$S$ and $S'$ are constructed so that each such function variable first tests $p_a$, then $p_b$, and then $p'_S$. Except where the loop function variable $U$ is encountered, both schemes $S$ and $S'$ behave such that if $p_a$ is true then function $a$ is applied; if $p_a$ is false and $p_b$ is true then function $b$ is applied; if $p_a, p_b$ are both false and $p'_S$ is true then function $*$ is applied and the computation terminates. Thus, we can see that there can be no free interpretation $I$ such that $\text{val}_I(S)$ and $\text{val}_I(S')$ are both defined but $\text{val}_I(S) \neq \text{val}_I(S')$. So, since it is not the case that $S \subseteq S'$, it is sufficient to consider free interpretations $I$ such that $\text{val}_I(S)$ is defined and $\text{val}_I(S')$ is undefined. By the reasoning above, we can see that this can only occur when the computation reaches a point where function variable $E$ (in scheme $S'$) must be replaced by its definition, where $p_a, p_b$ are false but $p'_S$ is true. Since $S'$ is not less defined than $S'$, $S$ is defined here, and $S'$ is undefined. But this is the case where $\text{val}_I(S) = \$wcf_{i_1} \ldots f_{i_t}$, where $w = x_{i_1} \ldots x_{i_t} = y_{i_1} \ldots y_{i_t}$. Thus, $i_1, \ldots, i_t$ is a solution to the PCP for $\mathcal{S}$.

$\implies$ Suppose $S \subseteq S'$. Let $I$ be any free interpretation such that $\text{val}_I(S)$ is defined—hence, $\text{val}_I(S) = \text{val}_I(S') = \$x_{i_1} \ldots x_{i_t} cf_{i_1} \ldots f_{i_t}$. In scheme $S'$, immediately after application of the function $c$, we have $cf_{i_1} \ldots f_{i_t}$ as the value of the variable, with the string of function variables $a = g(y_{i_1})h(y_{i_2}) \ldots h(y_{i_t})$ remaining to be computed according to the interpretation $I$. We have four possible cases to consider:
1) The interpretation $I$ is such that all function variables in $a$ are computed with $p_a$ true whenever function variables $A$ or $\hat{A}$ are considered (so that function $a$ is applied), and $p_a$ is false but $p_b$ is true whenever function variables $B$ or $\hat{B}$ are considered (so that function $b$ is applied). Hence, the variable will eventually have value \( wcf_{i_1} \ldots f_{i_t} \), where $w = y_{i_1} \ldots y_{i_t}$, and function variable $E$ remains to be computed for interpretation $I$. But for $\text{val}_I(S')$ to be defined, we must have either $p_a$ true (where function $a$ is applied) or $p_a$ false and $p_b$ true (where function $b$ is applied). Hence, $\text{val}_I(S') = \$x_{i_1} \ldots x_{i_t} cf_{i_1} \ldots f_{i_t}$, where $y_{i_1} \ldots y_{i_t}$ is a proper suffix of $x_{i_1} \ldots x_{i_t}$. Thus, $i_1, \ldots, i_t$ is not a solution to the PCP for $\mathcal{P}$.

2) The interpretation $I$ is such that some function variable $A$ or $\hat{A}$ in $a$ is computed with $p_a$ false and $p_b$ true, so $b$ is applied to the value of the variable. Since $\text{val}_I(S') = \$x_{i_1} \ldots x_{i_t} cf_{i_1} \ldots f_{i_t}$, there exist $z_1, z_2, z_3 \in \Sigma^*$ such that

$$x_{i_1} \ldots x_{i_t} = z_1 b z_2 \quad \text{and} \quad y_{i_1} \ldots y_{i_t} = z_3 a z_2.$$ 

Hence, $i_1, \ldots, i_t$ is not a solution to the PCP for $\mathcal{P}$.

3) The interpretation $I$ is such that some function variable $B$ or $\hat{B}$ in $a$ is computed with $p_a$ true, so $a$ is applied to the value of the variable. This is just the dual of 2), so again $i_1, \ldots, i_t$ is not a solution to the PCP for $\mathcal{P}$.
4) The interpretation $I$ is such that some function variable $A$, $B$, $\hat{A}$, or $\hat{B}$ in $a$ is computed with $p_a$, $p_b$ false and $p_\$ true. All function variables ($A$, $B$) in $a$ remaining to be computed are replaced by ID until only one function variable ($A$ or $B$) remains to be computed. [Note that $S'$ is constructed so that there is exactly one occurrence of either $\hat{A}$ or $\hat{B}$ in any computation.] Since we still have $p_a$, $p_b$ false and $p_\$ true, $\hat{A}$ and $\hat{B}$ cause function $\$ to be applied, thus ending the computation. Hence $\text{val}_I(S') = \$x_{i_1} \ldots x_{i_t} c_{f_1} \ldots f_{i_1}$, where $x_{i_1} \ldots x_{i_t}$ is a proper suffix of $y_{i_1} \ldots y_{i_t}$. Thus, $i_1, \ldots, i_t$ is not a solution to the PCP for $\_$. \[\square\]

Two other relations between schemes are the following:

**Strong Equivalence.** $S \equiv S'$ iff for every (free) interpretation $I$, either both $\text{val}_I(S)$ and $\text{val}_I(S')$ are undefined, or both are defined with $\text{val}_I(S) = \text{val}_I(S')$.

**Weak Equivalence.** $S = S'$ iff for every (free) interpretation $I$, either $\text{val}_I(S)$ or $\text{val}_I(S')$ is undefined, or both are defined with $\text{val}_I(S) = \text{val}_I(S')$.

The three relations described in this paper ($\equiv$, $\subseteq$, $\preceq$) are all reasonable relations in the terminology of [5]. That is, let $S$ and $S'$ be any two schemes and $\_\preceq$ be any relation between $S$ and $S'$.
Then \( \prec \) is reasonable on the class of monadic recursion schemes if for any two schemes \( S \) and \( S' \),

1) \( S \equiv S' \implies S \prec S' \)

2) \( S \prec S' \implies S = S' \)

We have just shown the undecidability of the inclusion problem for monadic recursion schemes. The decidability of the strong equivalence problem for schemes remains open, however the construction of the above proof gives us another result.

**Corollary.** The weak equivalence problem for monadic recursion schemes is undecidable.

**Proof.** The construction is similar to the one above, except now define the function variable \( E \) in scheme \( S' \) as

\[
E \equiv \begin{cases} 
  p_a & \text{if } \hat{T}_a \\
  p_b & \text{if } \hat{T}_b \\
  p_\emptyset & \text{if } a \\
  U & \text{else}
\end{cases}
\]

A scheme is **free** iff for every free interpretation the computation of the scheme has no predicate that ever tests the variable \( x \) with the same value more than once. Ashcroft, Manna, and Pnueli [1] prove that it is decidable whether or not a scheme is free. In the proof of the main theorem above, scheme \( S \) is free, but scheme \( S' \) is not free. The non-freedom of \( S' \) is introduced by the use of the ID function in the definitions of function variables \( A, B \) and \( T \). This non-freedom is
essential to the proof. The strong equivalence problem for free schemes is known to be decidable [1], whereas the inclusion problem for free schemes is open.

Korenjak and Hopcroft [4] define a type of pushdown automaton called an s-machine. This is a real-time (no e-moves) deterministic pushdown automaton with only one state that accepts by empty store. The inclusion problem for languages accepted by s-machines is still open. By appropriate encoding, it can be shown that this problem is equivalent to the inclusion problem for free schemes with no ID function.

**Theorem.** The weak equivalence problem for free schemes without identity is undecidable.

**Proof.** The proof technique is similar to that used in the theorem above. Let $\Sigma$, $\mathcal{P}$, $\mathcal{h}$ and $S$ be defined as in the proof of the previous theorem. We will define a new scheme $S''$ such that $S \equiv S''$ iff there does not exist a solution to the PGP for $S''$.

$$S'' = (V'', \mathcal{F}'', \mathcal{P}'', \mathcal{D}'', F_0)$$

where

- $V'' = \{F_1', \ldots, F_n', F_0', C_1, \ldots, C_n\} \cup \{A, B, X', U\}$
- $\mathcal{F}'' = \{f_1, \ldots, f_n\} \cup \{a, b, c, \$\}$
- $\mathcal{P}'' = \{q_1, \ldots, q_n\} \cup \{p_a, p_b, p_c\}$

and $\mathcal{D}''$ is defined as follows:
\[ F''_n \leftarrow X''F''_1 \]

\[ F''_n \leftarrow \text{if } q_n \text{ then } h(y_n)G_{f_n} \text{ else } F''_n \]

Same comments as for scheme S, except the else clause of \( F''_n \) is now a loop.

\[ G_n \leftarrow \text{if } q_n \text{ then } h(y_n)G_{f_n} \text{ else } G \]

Same comments as \( F_n \) for scheme S.

\[ A \leftarrow \text{if } p_a \text{ then } a \text{ else } U \]

\[ B \leftarrow \text{if } p_a \text{ then } U \text{ else if } \]

\[ \text{if } p_b \text{ then } b \text{ else } U \]

\[ X'' \leftarrow \text{if } p_a \text{ then } U \text{ else if } \]

\[ \text{if } p_b \text{ then } U \text{ else if } \]

\[ \text{if } p_S \text{ then } a \text{ else } U \]

\[ U \leftarrow U \]

\text{Loop.}

We can see that for any free interpretation \( I \) with both \( \text{val}_I(S) \) and \( \text{val}_I(S'') \) defined, we have \( \text{val}_I(S) = wcf_{i_1} \ldots f_{i_t} \) and \( \text{val}_I(S'') = \)

\[ awf_{i_1} \ldots f_{i_t} \]

where \( w = x_{i_1} \ldots x_{i_t} = y_{i_1} \ldots y_{i_t} \). Also, if \( i_1, \ldots, i_t \)

is a solution to the PCP for \( \mathcal{P} \), then there exists a free interpretation

\( I \) such that \( \text{val}_I(S) = \$x_{i_1} \ldots x_{i_t} c_{i_1} \ldots f_{i_t} \) and \( \text{val}_I(S'') = \)
Hence, $S = S''$ iff there does not exist a solution to the PCP for $S$. Thus, the undecidability of the weak equivalence problem for free schemes without ID follows from the undecidability of the PCP. □

An immediate consequence is:

**Corollary.** The weak equivalence problem for free schemes is undecidable.

There are subclasses of monadic recursion schemes that have a known decidable inclusion problem. For example, we can subclassify monadic recursion schemes as linear [3] if each term in a function variable definition contains at most one function variable. The inclusion and weak equivalence problems for linear schemes (not necessarily free) are shown to be decidable in [2].


