STANFORD ARTIFICIAL INTELLIGENCE PROJECT .
MEMO AIM-148

COMPUTER SCIENCE DEPARTMENT
REPORT NO. CS-217

DECIDABLE PROPERTIES OFMONADIC FUNCTIONAL SCHEMAS

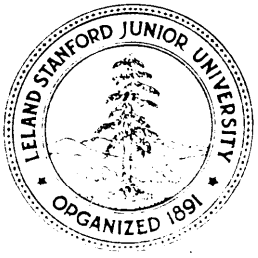
BY

EDWARD ASHCROFT
ZOHAR MANNA
AMIR PNEULI

JULY 1971

COMPUTER SCIENCE DEPARTMENT .

STANFORD UNIVERSITY



DECIDABLEPROPERTIES OFMONADIC FUNCTIONALSCHEMAS

EDWARD ASHCROFT
Computer Science Dept.
University of Waterloo
Waterloo, Canada

ZOHAR MANNA
Computer Science Dept.
Stanford University
Stanford, California

AMIR PNUELI
Applied Mathematics Dept.
Weizmann Institute
Rehovot, Israel

Abstract: We define a class of (monadic) functional schemas which properly includes 'Ianov' flowchart schemas. We show that the termination, divergence and freedom problems for functional schemas are decidable. Although it is possible to translate a large class of non-free functional schemas into equivalent free functional schemas, we show that this cannot be done in general. We show also that the equivalence problem for free functional schemas is decidable. Most of the results are obtained from well-known results in Formal Languages and Automata Theory.

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).
A preliminary version of this paper was presented at the International Symposium on the Theory of Machines and Computation (Haifa, Israel), August 1971.

Reproduced in the USA. Available from the Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia 22151.
Price: full size copy, $3.00; microfiche copy, $.95.

DECIDABLE PROPERTIES OF MONADIC FUNCTIONAL SCHEMAS

EDWARD ASHCROFT
Computer Science Dept.
University of Waterloo
Waterloo, Canada

ZOHAR MANNA
Computer Science Dept.
Stanford University
Stanford, California

AMIR PNUELI
Applied Mathematics Dept.
Weizmann Institute
Rehovot, Israel

I. Monadic Functional Schemas

An alphabet of a (monadic) functional schema S consists of one individual variable x, a finite set of monadic function variables $\{F_i\}$ (with a designated initial function variable F_0), a finite set of monadic function constants $\{f_i\}$, and a finite set of monadic predicate constants $\{p_i\}$. Note that individual constants are not allowed.

A term over Σ_S is any term in the normal sense constructed from the monadic function variables $\{F_i\}$, monadic function constants $\{f_i\}$ and the variable x, e.g., $f_1(F_3(F_0(f_2(x)$ A conditional term over Σ_S is any finite expression of the form

$$\underline{\text{if}}\ p_i(x)\ \underline{\text{then}}\ \tau_1\ \underline{\text{else}}\ \tau_2\ ,$$

where p_i is any predicate constant of Σ_S, and τ_1 and τ_2 are any terms or conditional terms over Σ_S.
A definition of F_i over Σ_S is of the form

$$F_i(x) \Leftarrow \tau\ ,$$

where τ is any term or conditional term over Σ_S. A (monadic) functional schema S (over an alphabet Σ_S) consists of a finite set of definitions over Σ_S, one for each function variable Fi in Σ_S. Whenever the special function variable F_∞ is used, its definition is considered to be $F_\infty(x) \Leftarrow F_\infty(x)$. This definition is usually omitted.

Example 1: Let us consider the functional schema S_1:

$$F_0(x) \Leftarrow \underline{\text{if}}\ p_1(x)\ \underline{\text{then}}\ \text{if}\ p_1(x)\ \underline{\text{then}}\ F_1(x)\ \underline{\text{else}}\ F_1(f_1(x))$$
$$\underline{\text{else}}\ x$$
$$F_1(x) \Leftarrow \underline{\text{if}}\ p_3(x)\ \underline{\text{then}}\ F_0(f_2(x))\ \underline{\text{else}}\ f_1(x)\ .$$

Since we are using a very restricted alphabet, parentheses and the individual variable x may be omitted without causing any confusion. Therefore the functional schema S_1 can be rewritten as:

$$F_0 \Leftarrow \underline{\text{if}}\ p_1\ \text{then if}\ p_2\ \underline{\text{then}}\ F\ \underline{\text{else}}\ F_1 f_1$$
$$\underline{\text{else}}\ I$$
$$F_1 \Leftarrow \underline{\text{if}}\ p\ \underline{\text{then}}\ F_0 f_2\ \underline{\text{else}}\ f_1\ ,$$

where I stands for the 'identity function'.*/

An interpretation J of a functional schema S consists of:

1. a non-empty set of elements D (called the domain),
2. an element ξ_0 of D used as the initial value of x, and
3. assignments to the constants of Σ_S:
   (i) a total monadic function (from D into D) for each function constant f_i, and
   (ii) a total monadic predicate (from D into {T,F}) for each predicate constant p_i.

*/ It is worth noting that most of the results in this paper would be trivial if we did not allow the 'identity function'.

For a given interpretation $\mathcal{I}$, the pair $(S,\mathcal{I})$, called a functional program, can be computed by evaluating F_0 with input ξ_0 in the usual way (see McCarthy [1963]). The canputation either (i) terminates yielding an element of D denoted by $\underline{val}(S,\mathcal{I})$, or (ii) diverges (i.e., does not terminate) in which case $\underline{val}(S,\mathcal{I})$ is said to be undefined. The method of canputation is described more fully later for special types of interpretations called 'Herbrand interpretations'.

A functional schema S is said to terminate/diverge if for every interpretation $\mathcal{I}$, $\underline{val}(S,\mathcal{I})$ is defined/undefined. Two functional schemas S_1 and S_2 are said to be equivalent if for every interpretation $\mathcal{I}$ either both $\underline{val}(S_1,\mathcal{I})$ and $\underline{val}(S_2,\mathcal{I})$ are undefined or both are defined and $\underline{val}(S_1,\mathcal{I}) = \underline{val}(S_2,\mathcal{I})$.

The same class of functional schemas has been discussed by DeBakker and Scott [1969].

It is straightforward to show that every functional schema in which any term contains at most one function variable, occurring on the left-hand side of the term (as in Example 1 above), can be translated to an equivalent 'Ianov' flowchart schema (Ianov [1960], see also Rutledge [1964]). However, such simple functional schemas as

$$F_0 \Leftarrow \underline{\text{if}}\ p\ \underline{\text{then}}\ I\ \underline{\text{else}}\ f_1F_0f_2$$

have no equivalent Isnov flowchart schema. Hence, the results in this paper generalize known results about Ianov flowchart schemas.

II. Herbrand Interpretations*/

A Herbrand interpretation $\mathcal{I}^*$ of a functional schema S consists of:

1. The domain D* which is the set of all expressions "τ_c", where τ_c is a constant term constructed from the individual variable x and the function constants f_i of Σ_S, e.g., "x", "$f_1(x)$", "$f_2(f_1(x))$";
2. The expression "x"ϵD* used as the intial value of x;
3. Assignments to the constants of Σ_S:
   (i) For every function constant f_i in Σ_S, we assign the total function mapping every expression "τ_c"ϵD* into the expression "$f_i(\tau_c)$"ϵD*,
   (ii) For every predicate constant p_i in Σ_S, we assign sane total predicate over D*, i.e., for every "τ_c"ϵD* the value of p_i("τ_c") is either T or F.

Note that all Herbrand interpretations of a given functional schema differ only in the assignments to the predicate constants. Henceforth we omit the quotation marks whenever this causes no confusion.

The computation of $(S,\mathcal{I}^*)$ is best described by a (finite or infinite) sequence of terms α_i. The first term α_0 of the sequence is $F_0(x)$. In general, suppose α_n $(n \geq 0)$ contains the sub-term $F_i(\xi)$ where $\xi\epsilon$D*, i.e., F_i is the right-most function variable in α_n. Then α_{n+1} is obtained by substituting a term τ for $F_i(\xi)$ where τ is obtained as follows. First x is replaced by ξ in the definition of F_i, and then the term τ is chosen fran this definition using the values of $p_j(\xi)$ supplied by the interpretation of the predicate constants p_j. The canputation terminates as soon as we reach a constant term τ_c. Then $\underline{val}(S,\mathcal{I}^*) = \tau_c$.

Herbrand interpretations are important because many properties of functional schemas can be described and proved just by considering Herbrand interpretations rather than all interpretations. For example,

(i) a functional schema terminates/diverges (under every interpretation) if snd only if it terminates/diverges under every Herbrand interpretation;

(ii) two functional schemas are equivalent (under every interpretation) if and only if they are equivalent under every Herbrand interpretation.

---

*/ Herbrand interpretations are identical to the 'free interpretations' of Luckham, Park and Paterson [1970].

III. Termination and Divergence of Functional Schemas

We show that

THEOREM 1. It is decidable whether or not a functional schema terminates or diverges (for every interpretation).

Proof: The proof depends on the well-known results in Formal Language Theory that it is decidable for any context-free grammar whether or not all (rightmost) derivations are finite and whether or not all (rightmost) derivations are infinite.

For this purpose we associate with every functional schema S a context-free grammar G_S such that: there is a one-to-one correspondence between the set of all computations of S for all Herbrand interpretations, and the set of all (rightmost) derivations of G_S. Furthermore, a cauputation of S diverges if and only if the corresponding derivation of G_S is infinite.

Given a functional schema S with n predicates $p_1,p_2,\ldots,p_n$, function variables $\{F_i\}$ and function constants $\{f_i\}$, we define G_S as follows:

1. The non-terminals are of the form

$$[w_b, F_k, w_a] \qquad F_k \in \{F_i\} \text{ and } w_a, w_b \in W,$$

where W is the set of all strings of length n over $\{T,F\}$. The intuitive meaning of $[w_b,F_k,w_a]$ is that it will generate all possible constant terms τ_c such that τ_c is computed by S starting from $F_k(x)$, under sane Herbrand interpretation for which the values of $p_1,\ldots,p_n$ for x are w_a and for τ_c are w_b.

In addition, we have a special initial non-terminal A.

2. The terminals are the function constants $\{f_i\}$.

3. The productions are obtained from S as follows:

(i) $A \rightarrow [w_b,F_0,w_a]$ for all $w_a,w_b \in W$.

(ii) For every non-terminal $[w_b,F_k,w_a]$ we add productions as follows:

Locate in definition of F_k that term which is selected by w_a. Let the term be

$$\alpha_m \ldots \alpha_2 \alpha_1$$

where each α_i is either a function variable or a function constant. We construct all productions of the form

$$[w_b,F_k,w_a] \rightarrow [w_b,\alpha_m,w_{m-1}][w_{m-1},\alpha_{m-1},w_{m-2}] \ldots \quad \text{where } w_1,w_2,\ldots \in W,$$

after replacing each $[w,\alpha_i,w']$, where α_i is a function constant, by α_i.

In the special case where the term is I, if $w_a \neq w_b$ we generate no production, whereas if $w_a = w_b$ we generate the single production

$$[w_a,F_k,w_a] \rightarrow \Lambda \quad (\Lambda \text{ is the empty word}).$$

Finally, we go through G_S and repeatedly remove all productions containing some non-terminal which is not the left-hand side of any production and all productions in which the non-terminal on the left-hand side cannot be reached from A.

G_S now clearly has the required properties.

Q.E.D.

IV. Functional Schemas in Standard Form

We introduce now an interesting subclass of functional schemas which has a special syntactic form. This form is relevant to our later discussion.

A functional schema S is said to be in standard form if

1. Every conditional term in S is of the form

$$\underline{\text{if}}\ p_j\ \underline{\text{then}}\ \tau_1\ \underline{\text{else}}\ \tau_2$$

where τ_1 and τ_2 are distinct (i.e., there are no 'redundant' tests) and each of τ_1 and τ_2 is of one of the following forms:

(a) a conditional term not containing p_j (i.e., there are no 'redundant' terms),

(b) F_∞ ,

(c) I , or

(d) a term in which F_∞ does not occur and the rightmost symbol is a function constant.

2. Every definition in S (except for F_∞) contains a conditional term.

3. No function variable in S always diverges (i.e., no matter which function variable in Σ_S is taken as the initial function, the schenas do not diverge), except for F_∞ and F_0 if $F_0 <= F_\infty$.

Note that our definition of standard form has the flavor of Greibach Normal Form in Formal Languages Theory.

'The interest in this form arises from the fact that every functional schema can be effectively transformed to an equivalent functional schema in standard form. This is done-easily by first recognizing all 'divergent' function variables (using the technique described in the previous section) and replacing all terms in which they occur by F_∞ . Then we repeatedly replace every function variable occurring as the rightmost symbol of a term by its definition, applying straightforward simplifications, as illustrated in Example 3 below.

Example 2: The functional schema S_2 e

$$F_0 <= \underline{\text{if}}\ p\ \underline{\text{then}}\ F_0F_1f_1\ \underline{\text{else}}\ f_2$$

$$F_1 <= \underline{\text{if}}\ p\ \underline{\text{then}}\ F_1f_3\ \underline{\text{else}}\ I\ ,$$

is clearly in standard form.

Example 3: The functional schema S_3 where

$$F_0 <= \underline{\text{if}}\ p\ \underline{\text{then}}\ f_1F_1\ \underline{\text{else}}\ f_2$$

$$F_1 <= \underline{\text{if}}\ p\ \underline{\text{then}}\ F_1f_3\ \underline{\text{else}}\ f_4$$

is not in standard form because of the term f_1F_1 . This term can be removed by first replacing F_1 in f_1F_1 by its definition to obtain:

$$F_0 <= \underline{\text{if}}\ p\ \underline{\text{then}}\ \underline{\text{if}}\ p\ \underline{\text{then}}\ f_1F_1f_3\ \underline{\text{else}}\ f_1f_4 * f_2\ ,$$

and then simplifying it to

$$F_0 <= \underline{\text{if}}\ p\ \underline{\text{then}}\ f_1F_1f_3\ \underline{\text{else}}\ f_2$$

Thus, we obtain the functional schema S_3' , which is clearly in standard form and equivalent to S_3 :

$$F_0' <= \underline{\text{if}}\ p\ \underline{\text{then}}\ f_1F_1'f_3\ \underline{\text{else}}\ f_2$$

$$F_1' <= \underline{\text{if}}\ p\ \underline{\text{then}}\ F_1'f_3\ \underline{\text{else}}\ f_4$$

v. **Freedom or Functional Schemas**

A **functional schema** S is said to be free if for every Herbrand interpretation $\mathcal{J}^*$ of s the computation of $(S,\mathcal{J}^*)$ does not test 8 predicate with the same term (from D^*) more than once.

Examples :

1. The functional schema Sl (Example 1 above) is clearly free.
2. The functional schema S_2 (Example 2 above) is not free, since there is 8 Herbrand interpretation $\mathcal{J}^*$ for which the computation of $(S_2,\mathcal{J}^*)$ is of the form

$$F_0(x) \rightarrow F_0(F_1(f_1(x))) \overset{(a)}{\rightarrow} F_0(f_1(x)) \overset{(b)}{\rightarrow} f_2(f_1(x)) \ \cdots \ .$$

Predicate p tests term $f_1(x)$ at both steps (8) and (b).

3. The functional schema S_3 (Example 3 above) is not free, since there is 8 Herbrand interpretation $\mathcal{J}^*$ for which the computation of $(S_3,\mathcal{J}^*)$ is of the form

$$F_0(x) \overset{(a)}{\rightarrow} f_1(F_1(x)) \overset{(b)}{\rightarrow} f_1(F_1(f_3(x))) \rightarrow \cdots \ .$$

Predicate p tests term x at both step6 (a) and (b).

There is 8 crucial difference between the non-freedom demonstrated in the functional schema S_2 and that of S_3 . In Example 3 the non-freedom results from the substitutions at step (b) for the function variable F_1 which resulted from the substitution at step (a). In Example 2 the non-freedau results from the substitution at step (b) for the function variable F_0 which was already present before step (8). Note that in this case the function variable F_1 substituted for at step (8) produce6 the identity function. This will always be the case when non-freedom of the second type occurs.

Functional schemas in standard form cannot have non-freedom of the first type but may have non-freedan of the second type. Thus, by bringing a functional schema to standard form we always eliminate any non-freedan of the first type (but unfortunately preserving any non-freedom of the second type). Since the second type of non-freedan involves the identity function, it follows that any functional schema not containing I can be made free by putting it into standard form.

Although 8 large class of functional schemas containing I can still be translated to equivalent free functional schemas, this cannot be done in general as follows from

THEOREM 2: The non-free schema S_4 where

$$F_0 \Leftarrow \text{if } p \text{ then } F_1F_0f \text{ else } I$$

$$F_1 \Leftarrow \text{if } q \text{ then } f \text{ else } I \ ,$$

has no equivalent free functional schema.

Proof: Consider the following two families $\{\mathcal{J}_n^*\}$ and $\{\mathcal{K}_n^*\}$, $n \geq 1$, of Herbrand interpretations, where

$$\mathcal{J}_n^* = \begin{cases} p(f^i(x)) = T & \text{iff } i<n \\ q(f^i(x)) = T & \text{iff } i \neq n \end{cases}$$

$$\mathcal{K}_n^* = \begin{cases} p(f^i(x)) = T & \text{iff } i <n \\ q(f^i(x)) = T & \text{for all } i \ . \end{cases}$$

Note that two corresponding interpretations $\mathcal{J}_n^*$ and $\mathcal{K}_n^*$ differ by Only one value, that of $q(f^n(x))$. It is clear that $\underline{val}(S_4,\mathcal{J}_n^*) = P(x)$ and $\underline{val}(S_4,\mathcal{K}_n^*) = f^{2n}(x)$ for all $n \geq 1$.

Suppose there exists 8 free functional schema G in standard form that is equivalent to S_4 . We shall derive 8 contradiction by showing that there exists 8 positive integer n_o for which $\underline{val}(G,\mathfrak{A}^*_{n_0}) \neq f^{2n_o}(x)$.

Let n be an arbitrary positive integer. If we apply interpretations $\mathfrak{I}^*_n$ and $\mathfrak{A}^*_n$ to G we must reach in both cases a term of the form $\alpha_n(f^n(x))$, since for $f^i(x)$, i <n , G cannot distinguish between $\mathfrak{I}^*_n$ and $\mathfrak{A}^*_n$.

Since for $\mathfrak{I}^*_n$ this term must produce f"(x) , it follows that every symbol in α_n is a function variable that 'collapses' to identity for predicate values determined by $\mathfrak{I}^*_n$ for the term $f^n(x)$. Thus, since G is free and no predicate in G can test under interpretation $\mathfrak{I}^*_n$ the term $f^n(x)$ twice, the number of symbols in α_n can be no more than the number of distinct predicate constants $\{p_i\}$ in G .

Therefore, there must exist two distinct positive integers nl and n_2 , $n_1 \neq n_2$, such that α_{n_1} is identical to $a_{"2}$. By definition of $\{\mathfrak{A}^*_n\}$, the continuation of the computation of $\alpha_{n_1}(f^{n_1}(x))$ under $\mathfrak{A}^*_{"1}$ is identical to that of $\alpha_{n_2}(f^{"2}(x))$ under $\mathfrak{A}^*_{n_2}$. Therefore, since $n_1 \neq n_2$, either $\underline{val}(G,\mathfrak{A}^*_{n_1}) \neq f^{2n_1}(x)$ or $\underline{val}(G,\mathfrak{A}^*_{n_2}) \neq f^{2n_2}(x)$. Contradiction.

Q.E.D.

Despite the above result, we still have

THEOREM 5: It is decidable whether a functional schema is free or not.

Proof (sketch): The proof consists of showing that a functional schema S is non-free if and only if non-freedan occurs in 8 canputation under sane Herbrand interpretation within a number K of steps, where K depends only on S The decision method is then to explore all the different ways in which canputation can proceed for K steps under any Herbrand interpretation. Since there are 8 finite number of such ways it is possible to decide whether or not non-freedom will ever occur.

We can assume that no function variable in S is simply defined by the term I , since such function variables cm be removed without affecting freedom in any way.

If non-freedom occurs it will be of the first or the second types. It can easily be shown that if non-freedan of the second type is ever to occur in S , then for some Herbrand interpretation the functional schema must have a subcanputation of the form

$$\cdots \rightarrow \tau F_{j_m} F_{j_{m-1}} \cdots F_{j_1} \tau_c \overset{(a)}{\rightarrow} \tau F_{j_m} \cdots F_{j_2} \tau_c \rightarrow \cdots \rightarrow \tau F_{j_m} \overset{(b)}{\tau_c} \rightarrow \cdots ,$$

where τ_c is a'constant term and $m \leq n+1$ (n is the number of distinct predicate constants in Σ_S .) Non-freedom occurs between steps (a) and (b), i.e., at (b) sane predicate constant is testing τ_c which also tested τ_c at step (a). Note that $F_{j_1},\ldots,F_{j_{m-1}}$ all must collapse to identity for τ_c under this interpretation.

Let us assume that the earliest occurrence of non-freedom, in any computation, is of the second type, and results from a computation of $\tau F_{j_m} \ldots F_{j_1} \tau_c$ as above. If it takes J steps to get $\tau F_{j_m} \ldots F_{j_1} \tau_c$, then all computations of S are free for at least their first J steps. These initial computations therefore result from all possible substitutions, analogously to the rightmost derivations of 8 grammar. We can use the following easily proved result of Formal Language Theory to show that J is bounded.

In any context free grammar, let $S \overset{*}{\Rightarrow} \alpha\beta\gamma$ be the shortest rightmost derivation of a sentential form containing a given substring β (of terminals and non-terminals) of length m , with some terminal string on its right (i.e., is a terminal string). Then this derivation has no more than m LMN steps where:

L is the number of non-terminals in G ,

M is the maximal number of non-terminals on the righthand side of any production of G , and

N is the upper bound on the minimum number of steps needed for each non-terminal of G to produce a terminal string, if any; (i.e., every non-terminal of G can generate 8 terminal string, if at all, within N Steps).

Similarly defining L, M, and N for schema S gives us 8 bound on J depending only on S and M . Since m $\leq$ n+1 we have a bound K on the first occurrence of non-freedan if such an occurrence is of the second type. It can be easily shown that any non-freedan of the first type must also be discovered within K steps, and hence the method of exploring the initial segments of all possible cauputations can decide whether non-freedom (of any type) is present.

Q.E.D.

VI. Equivalence of Free Functional Schemas

THEOREM 4: It is decidable whether or not two free functional schemas are equivalent (for every interpretation).

Proof: Suppose we are given two free functional schemas S and S' . Since the translation of functional schema6 into standard form never introduces new non-freedom, we can assume without loss of generality that S and S' are in standard form.

We construct 8 Deterministic Push-Down Automaton (DPDA) A , whose input tape is a representation of some Herbrand interpretation and which simulates the canputation of S and S' under this interpretation. A accepts an input tape if and only if S and S' are inequivalent under the corresponding interpretation. Since it is decidable whether or not the language accepted by a DPDA is empty (see, for example, Hopcroft and Ullman [1969]), it follows that the equivalence problem for free functional schemas is decidable.

The construction of A makes use of sane ideas developed by Rosenkrantz and Stearns [1970] (see also Korenjak and Hopcroft [1966]).

Suppose S and S' are n predicate constants $p_1, p_2, \ldots, p_n$. The input alphabet consists of words of length n over {T,F} where we intend that when A reads such 8 word (input symbol), the i-th letter denotes the current truth value of p_i . This requires some more explanation.

Each step in the cctnputation sequence of 8 schema for sane interpretation J consists of taking the current term, $\tau F_j \tau_c$ say, and substituting for F_j according to the truth values of the predicates applied to τ_c . Supposc this substitutes some term $\tau' F_j' f_k$ for F_j . Then, at the next step, to substitute for F_j' , we need only know the truth values of the predicates applied to $f_k \tau_c$, i.e., the current truth values. At each step we need only know the current truth values, for example in this case the truth values of the predicates for $f_i \tau_c$, $f_i \neq f_k$, will not affect the canputation in any way. Each interpretation, by specifying 8 cunputation, has a corresponding sequence of n-tuples over {T,F} ; and vice-versa, each sequence of n-tuples over {T,F} , by specifying a computation, gives the truth values of the predicates for certain constant terms and therefore indicate a set of interpretations, all giving this computation. Hence we can represent interpretations by sequences of n-tuples over {T,F} , which is what We use as input tapes to our automaton A .

To simulate the joint action of S and S' for a given Herbrand interpretation, we let A have a two-track push-down stack. Each track will hold 8 modified version of the current term in the computation sequence of the corresponding schema under the given interpretation.

The modification of the computation terms is such that if S and S' are equivalent, both tracks are of the same length during corresponding cauputations of S and S' . This enables us to put both tracks in a single push-down stack. To understand this modification we introduce the notion of the *thickness' $T(\tau)$ of a term τ (that does not contain F_∞), namely the length of the shortest possible constant term computable fran τ for some Herbrand interpretation. For free schemas, $T(\tau_1\tau_2) = T(\tau_1) + T(\tau_2)$, since no decision made while computing τ_1 may affect the freedom of choice in computing τ_2 . The required modification of a term to give its stack representation is to make $T(F_i)$ copies of each function variable F_i . Thus the length of stack representing term τ is $T(\tau)$. Note that if the corresponding terms in the canputation of two equivalent

functional schemas are τ_1 and τ_2, then $T(\tau_1) = T(\tau_2)$; so the modified terms have the required property. To erase a function variable F_i from such a stack, the automaton A will in fact erase $T(F_i)$ copies of F_i, which is 8 feasible action of a multi-state DPDA.

The only problem that may arise is that it is possible that $T(F_i) = 0$, which is the case when the definition of F_i contains I. In this case we would like to add F_i to the preceding stack position and hence not increase the length of the stack. Note that if, for the next function variable F_j, we have again $T(F_j) = 0$ F_i will also be added to the same stack position; and so on. However, for free functional schemas the number of function variables we have to add to the same stack position will never exceed the number n of predicate symbols. This is because there can never be more than n successive function variables collapsing to I (otherwise some predicate would be tested more than once on the same term). We therefore can resolve the above difficulty by making each position in a track wide enough to hold n+l ordered symbols. Then symbols of thickness zero are written on the preceding position, after the other symbols already residing there. In order to make no exception for the bottom position of the stack, we may agree to test the equivalence of fF_0 and fF'_0 instead of F_0 and FE, .

The behavior of the DPDA A is as follows.

For each input symbol, A simulates all the possible actions of S and S'. If the topmost letter of the corresponding track is a function constant, no change is made. Otherwise, it must be a function variable F_i and we modify the top of that track according to the term in the definition of F_i selected by the current input symbol. These actions will terminate either with some new stack-trackwith a function constant at the top or F_∞ will be encountered. The crucial point is that for free functional schemas in standard form, the variation that occurs during these actions in the stack is bounded and can be temporarily remembered in 8 finite control until both operations for S and S' are completed.

Before moving to the next input letter, A now proceeds 8s follows:

(i) If F_∞ is encountered for both tracks, we pass to a rejecting state, (i.e., S and S' are equivalent under the given interpretation).

(ii) If we encounter F_∞ for one track and not in the other, we pass to an accepting state (i.e., S and S' are inequivalent for some Herbrand interpretation under which the other track goes to a constant term).

(iii) II the two tracks are not of the same length, we pass to an accepting state (i.e., S and S' are inequivalent for some Herbrand interpretation under which the shorter track produces its shortest constant term).

(iv) If the two function constants at the top of the two tracks are different, we pass to an accepting state.

(v) Otherwise, (i.e., ssme length tracks with the same topmost function constant) we remove the topmost letters. If both stacks are still non-empty, we move to the next input symbol, otherwise both tracks are empty and we pass to 8 rejecting state.

Thus the two given free functional schenas S and S' are equivalent if and only if the DPDA A accepts no input tapes, which is a known decidable problem.

Q.E.D.

Example 4: Consider the following two free functional schemas:

S_5: $F_0 \Leftarrow f_3F_3F_1f_2$

$F_1 \Leftarrow$ if p_1 then F_1f_2 else I

$F_2 \Leftarrow$ if p_2 then F_2f_2 else I

$F_3 \Leftarrow$ if p_2 then $F_3F_1f_2$ else I

$F_4 \Leftarrow$ if p_1 then $F_4F_2f_2$ else f_3

"6' $F_0' = F_4F_2f_2$

and F_1 , F_2 , F_3 and F_4 are as in S_5 .

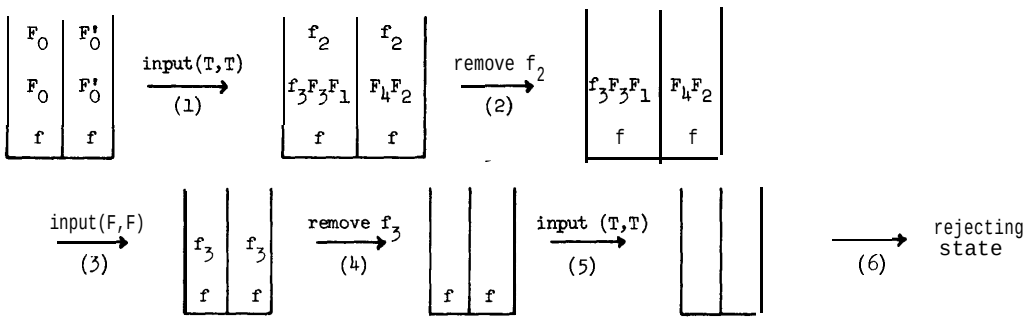
Here,

$$T(F_1) = T(F_2) = T(F_3) = 0 \ , \ T(F_4) = 1 \ , \text{ and } T(F_0) = T(F_0') = 2 \ ,$$

This implies, for example, that the terms $F_0F_4f_2$, $F_4F_2f_2$ and $f_3F_3F_1f_2$ would be stacked, respectively, as follows:

f_2	f_2	f_2
F_4	F_4F_2	$f_3F_3F_1$
F_0		
F_0		

We illustrate now the behavior of the DPDA A simulating the joint operation of S_5 and S_6 for any input tape starting with (T,T) , (F,F) , (T,T) , ...



Since the input element indicates that both p_1 and p_2 are false at step (3), we have by definition that $F_1 = F_2 = F_3 = I$ and $F_4 = f_3$. Therefore both terms $f_3F_3F_1$ and F_4F_2 reduce to f_3. Note that we shall get the same sequence of stacks for any input tape for which the second element is (F,F) .

Acknowledgment: We are indebted to Mike Paterson for his critical reading of the manuscript and subsequent helpful suggestions. Mike proved independently that the equivalence problem is decidable for the class of all schemas without I .

References

J. W. De-Bakker and D. Scott [1969]. A Theory of Programs. Unpublished memo.

J. E. Hopcroft and J. D. Ullman [1969]. Formal Languages and Their Relation to Automata. Addison-Wesley.

Y. I. Ianov [1960]. "The Logical Schemes of Algorithms". In Problems in Cybernetics, Vol. 1, Pergamon Press, New York, pp. 82-140.

A. J. Korenjak and J. E. Hopcroft[1966]. Simple Deterministic Languages, IEEE 7th Annual Symposium on Switching and Automata Theory. pp. 36-46.

D. C. Luckham, D. M. R. Park and M. S. Paterson [1970]. "On Formalized Computer Programs". Journal of Computer and System Sciences, Vol. 4, pp. 220-249.

J. McCarthy [1963]. "A Basis for a Mathematical Theory of Computation". In Computer Programming and Formal Systems, North-Holland, Amsterdam, pp. 33-70.

D. J. Rosenkrantz and R. E. Stearns [1970]. "Properties of Deterministic Top-Down Grammars". Information and Control 17, pp. 226-256.

J. D. Rutledge [1964]. "On Ianov's Program Schemata". J. ACM 11, 1, pp. 1-g.