FUNCTIONAL SCHEMAS WITH
NESTED PREDICATES

Zvi Galil
TR 73-189

December 1973

Computer Science Department
Cornell University
Ithaca, New York 14850
FUNCTIONAL SCHEMAS WITH
NESTED PREDICATES

Zvi Galil

Abstract:

A class of (monadic) functional schemas with nested predicates is defined. It is shown that termination, divergence and freedom problems for these schemas are decidable. It is proved that when the schemas are more general the freedom problem is undecidable. A procedure is given for deleting the identity function from the schema's definition at the cost of increasing k by 1 when k is the maximum depth of nesting.

Part of our results extend results of [1] about schemas without nesting. Our algorithm for checking freedom is not a natural extension of theirs. Furthermore, using our algorithm for schemas without nesting yields a much more efficient way of deciding freedom than the algorithm suggested in [1].

Keyword and Phrases: 5.22, 5.24

monadic functional schemas, nested predicates, decision problems, equivalence, freedom, polynomial time, DPDA
FUNCTIONAL SCHEMAS WITH
NESTED PREDICATES

Zvi Galil

§1 Introduction

An alphabet $\Sigma_S$ of (monadic) functional schema $S$ consists of one variable $x$ and three finite sets: $V, T, \text{ and } P$. $V = \{F_i\}$ is a set of function variables with a designated initial function $F_0$, $C = \{f_i\}$ is a set of function constants, and $P = \{p_i\}$ is a set of predicates. Note that individual constants are not allowed.

A term over $\Sigma_S$ is defined recursively by:

$$\text{term } \in \{x \mid f(\text{term}) \mid F(\text{term}) \mid f \in C \text{ and } F \in V\}.$$

A conditional term over $\Sigma_S$ is any finite expression of the form $\text{if } p(\tau_1) \text{ then } \tau_2 \text{ else } \tau_3$, where $p \in P$, $\tau_2, \tau_3$ are any terms or conditional terms over $\Sigma_S$ and $\tau_1$ is a constant term over $\Sigma_S$, i.e. all the symbols appearing in it are of function constants. A definition of $F$ over $\Sigma_S$ is of the form $F(x) \triangleq \tau$, where $\tau$ is any term or conditional term over $\Sigma_S$.

A (monadic) functional schema $S$ (over an alphabet $\Sigma_S$) consists of the definitions of each $F \in V$. Whenever the special function $F_\omega \in V$ is used, its definition is considered to be $F_\omega(x) \triangleq F_\omega(x)$ and usually is omitted.

Since we are using a very restricted alphabet and all functions are monadic we omit parentheses and the variable $x$ without causing any confusion. I will stand for the identity function.
For example the definition

\[ F_0(x) = \text{if } p_1(x) \text{ then if } p_2(f(x)) \text{ then } F_\infty(x) \text{ else } F_1(f_1(x)) \text{ else } x \] will be written as

\[ F_0 = \text{if } p_1 \text{ then if } p_2 f \text{ then } F_\infty \text{ else } F_1 f_1 \text{ else } I. \]

Functional schemas without nested predicates were studied in [1] and [2]. They properly include Ianov flowchart schemas (see [3] and [4]).

An interpretation \( \theta \) of a functional schema \( S \) consists of a non-empty set \( D \) (the domain) and the following assignments:

1) \( \xi_0 \in D \) to \( x \); 2) a total monadic function (from \( D \) into \( D \)) \( \nu f \in C; \) and 3) a total monadic predicate (from \( D \) into \( \{T,F\} \)) \( \nu p \in P. \)

For a given interpretation \( \theta \), the pair \( (S, \theta) \), is called a functional program and can be computed by evaluating \( F_0 \) with input \( \xi_0 \) in the usual way (see [1]). The computation either terminates yielding an element of \( D \) denoted by \( \text{Val}(S, \theta) \), or diverges and then \( \text{Val}(S, \theta) \) is said to be undefined.

A functional schema \( S \) is said to terminate (diverge) if for every interpretation \( \theta \) \( \text{Val}(S, \theta) \) is defined (undefined). Two functional schemas \( S_1 \) and \( S_2 \) are equivalent if for every interpretation \( \theta \) \( \text{Val}(S_1, \theta) \) and \( \text{Val}(S_2, \theta) \) are both undefined or both defined and equal.

Consider a class of interpretations of a functional schema \( S \) such that \( D = C^* \) and for each \( f \in C \) we assign the total func-

\[ A^* \text{ means the set of strings consisting of elements of } A. \]
tion mapping every $\tau \in D$ into $f\tau \in D$. These are called Herbrand interpretations and are important because many properties of functional schemas, like termination, divergence and equivalence, can be described and proved just by considering Herbrand interpretations rather than all interpretations. (See [1] and [8]).

From now on the word interpretation will stand for Herbrand interpretation.

A functional schema $S$ is said to be free if for every interpretation $\theta$ of $S$ the computation of $(S, \theta)$ does not test a predicate with the same term more than once.†

We use Greek letters at the beginning of the alphabet for general strings (over $(C \cup V)^*$) and other Greek letters for constant strings (over $C^*$). For a given schema, we use the following notations:

$$\begin{align*}
\theta, k \quad &\text{if } \alpha \text{ is obtained from } F \text{ following } k \text{ steps of the computation under the interpretation } \theta; \\
\theta \quad &\text{if } F \Rightarrow \alpha \text{ for some } k; \\
\theta, k \quad &\text{if } F \Rightarrow \alpha \text{ for some } \theta; \\
\theta \quad &\text{if } \forall \theta \rightarrow (F \Rightarrow \alpha); \\
\theta, k \quad &\text{if } F \Rightarrow \alpha \text{ for some } \theta; \\
\theta \quad &\text{if } F \Rightarrow \alpha; \text{ and} \\
\theta \quad &\text{if } F \Rightarrow \alpha \text{ for some } \theta;
\end{align*}$$

† Don't confuse with free interpretations of [8] which we call here Herbrand interpretations.
In Section 2 we show that some other simple extensions of functional schemas of [1] imply unsolvability of freedom for these schemas. In Section 3 we give a procedure which checks freedom of nested functional schemas. This procedure runs in polynomial time in contrast with the algorithm in [1] for the unnested case. In Section 4 we give an algorithm for deletion of I from free nested schemas. This is done at the cost of increasing the maximum depth of nesting by 1. In Section 5 we show that the equivalence problem for functional schemas with nested predicates is decidable.

§2 More General Schemas

The definition of functional schemas demands that arguments of predicates should be constant terms and does not allow individual constants. It is shown that when we omit one of these restrictions the problem of freedom is undecidable.

Definition: A general nested (monadic) schema is a functional schema, the predicates of which are of the form pa, a ∈ (C ∪ V)*.

We show that given any PCP (Post Correspondence Problem) we can reduce it to a problem of deciding freedom of a general nested schema, and thus proving the latter is unsolvable.

Let λ₁,…,λₙ, μ₁,…,μₙ be 2n non empty strings of function constants. We define a general (monadic) schema $S_1$:

$F₀ \leftarrow \begin{cases} \text{if} & P₀ \text{ then } F₀ \text{ else } I \\ \text{if} & P₁ \text{ then } F₁ \text{ else } I \\
F₁ \leftarrow \begin{cases} \text{if} & Pᵢ \text{ then } Fᵢ Fᵢ \text{ else } Fᵢ₊₁, i \leq i \leq n \\ \text{if} & Pᵢ \text{ then } Fᵢ Fᵢ \text{ else } Fᵢ₊₁, i \leq i \leq n \\
Fₙ \leftarrow \begin{cases} \text{if} & Pₙ \text{ then } Fₙ Fₙ \text{ else } I \\ \text{if} & Pₙ \text{ then } Fₙ Fₙ \text{ else } I \\
F \leftarrow \begin{cases} \text{if} & P \text{ then } F \text{ else } I \\ \text{if} & P \text{ then } F \text{ else } I. 
\end{cases}
\end{cases}$
\[ G_i \leftarrow \text{if } q_i \text{ then } f_i H_i^\lambda_i \text{ else } G_{i+1}, \ 1 \leq i < n \]

\[ G_n \leftarrow \text{if } q_n \text{ then } f_n H_n^\lambda_n \text{ else } g \]

\[ H_i \leftarrow \text{if } r_i \text{ then } f_i H_i^\lambda_i \text{ else } H_{i+1}, \ 1 \leq i < n \]

\[ H_n \leftarrow \text{if } r_n \text{ then } f_n H_n^\lambda_n \text{ else } I \]

All $3n+1$ predicates $p$, $p_i$, $q_i$, $r_i$ $1 \leq i \leq n$ are distinct.

$f$, $g$ and $f_i$ $1 \leq i \leq n$ are distinct function constants which do not belong to any $\lambda_i$ or $\mu_i$.

**Lemma 1:** Schema $S_1$ is not free if and only if there exists $p > 0$ and $1 \leq i_1, \ldots, i_p \leq n$ satisfying $\lambda_{i_1} \ldots \lambda_{i_p} = \mu_{i_1} \ldots \mu_{i_p}$.

**Proof:** In order to compute $F_0$, one has to compute $G_1$. During $G_1$'s computation, if $q_i$ (or $r_i$) $1 \leq i < n$ is tested with some argument and found to be false, then only $q_k$ (or $r_k$) $k > i$ can be checked with the same argument. After $q_n$ is checked the argument must be changed (the string's length increases). If after $r_n$ is checked, $H_n \rightarrow I$, then the computation of $G_1$ terminates and the $r_i$'s will be tested no more. Therefore no $r_i$ or $q_i$ $1 \leq i \leq n$, can generate any non-freedom by being tested twice with the same term.

Let $A = \{ \lambda \mid \lambda \text{ is a constant term and } G_1 \Rightarrow \lambda \}$. Obviously $A = \{ g \} \cup \{ f_i \ldots f_{i_1}^{\lambda_{i_1} \ldots \lambda_{i_p}} \mid p > 0, 1 \leq i_1, \ldots, i_p \leq n \}$.

For every $\lambda \in A$ there exists an interpretation $\theta$ such that $G_1 \Rightarrow \lambda$ and $p\lambda = T$, since $p\lambda$ is checked for the first time.

Now starts the computation of $F_1$. Similar arguments imply that $p_i$, $1 \leq i \leq n$, cannot be tested twice with the same term and if
B = \{ \mu | \mu \text{ is a constant term and } F_1 \Rightarrow \mu \}, \text{ then}

B = \{ f_{j_1} \ldots f_{j_q} \mu_{j_1} \ldots \mu_{j_q} \mid q \geq 0, 1 \leq j_1, \ldots, j_q \leq n \}.

Since F_1's computation is free, for each \lambda \in A, \mu \in B there exists an interpretation \theta such that F_1 \Rightarrow \mu and G_1 \Rightarrow \lambda. \ p is the only predicate which can generate non-freedom. Therefore, the schema S_1 is not free if and only if A \cap B \neq \emptyset. But A \cap B \neq \emptyset if and only if there exist p > 0 and 

1 \leq i_1, \ldots, i_p \leq n \text{ satisfying } \lambda_{i_1} \ldots \lambda_{i_p} = \mu_{i_1} \ldots \mu_{i_p}. \ This completes the proof of the lemma.

Remark: We need the H_i's definitions, because if we omit their definitions and replace g by I the result of the lemma will not hold, since then I \in A \cap B.

Theorem 1: The freedom of general nested schemas is undecidable.

Proof: If there exists an algorithm which decides whether a given schema is free or not, then for any PCP we construct the corresponding schema S_1 and check if it is free or not. By Lemma 1 we shall be able to solve the given PCP - a contradiction to Post's Correspondence Theorem [6], which proves the theorem.

Let A = \{ a_i \} be a finite set of individual constants. If we change the above definition of term to be:

term + x \mid a \mid f(\text{term}) \mid F(\text{term}) \quad a \in A, f \in C, F \in V;

then we obtain a monadic schema with individual constants.
The definition of interpretation must be extended to include assignments of elements of D to the elements of A. Here too we omit all parentheses and the variable x. Thus Fg means F(g(x)) and Fga (a ∈ A) will stand for f(g(a)).

Similarly to Theorem 1 we prove:

**Theorem 2:** It is undecidable whether a monadic schema with individual constants is free or not.

**Proof:** Given 2n non empty strings of function constants - λ₁, . . . , λₙ, μ₁, . . . , μₙ. We construct the following monadic schema with individual constants S₂:

- \( F_0 \) \( \leftarrow \) \( \text{if } p_0 \rightleftharpoons \text{then } H_{1}G_{1}F \) \( \text{else } I \)
- \( F \) \( \leftarrow \) \( \text{if } p \rightleftharpoons \text{then } a \) \( \text{else } I \)
- \( G \) \( \leftarrow \) \( \text{if } q \rightleftharpoons \text{then } a \) \( \text{else } I \)
- \( H \) \( \leftarrow \) \( \text{if } q \rightleftharpoons \text{then } f \) \( \text{else } I \)
- \( F_i \) \( \leftarrow \) \( \text{if } p_i \rightleftharpoons \text{then } f_iF_i \) \( \text{else } F_{i+1} \), \( 1 \leq i \leq n \)
- \( F_n \) \( \leftarrow \) \( \text{if } p_n \rightleftharpoons \text{then } f_nF_n \) \( \text{else } I \)
- \( G_i \) \( \leftarrow \) \( \text{if } q_i \rightleftharpoons \text{then } f_iH_i \) \( \text{else } G_{i+1} \), \( 1 \leq i \leq n \)
- \( G_n \) \( \leftarrow \) \( \text{if } q_n \rightleftharpoons \text{then } f_nH_n \) \( \text{else } g \)
- \( H_i \) \( \leftarrow \) \( \text{if } r_i \rightleftharpoons \text{then } f_iH_i \) \( \text{else } H_{i+1} \), \( 1 \leq i \leq n \)
- \( H_n \) \( \leftarrow \) \( \text{if } r_n \rightleftharpoons \text{then } f_nH_n \) \( \text{else } I \)

\( F_0 \) can either collapse to identify (the uninteresting case) or generate \( H_{1}G_{1}F \). \( p_0 \) and \( p \) are tested only once and thus cannot cause any non-freedom.
Since $F_i$, $G_i$ and $H_i$, $1 \leq i \leq n$ are exactly those of schema $S_i$, no one of the predicates $p_i$, $q_i$, and $r_i$, $1 \leq i \leq n$ can be tested twice with the same argument. Thus the schema is non-free if and only if $q$ is tested twice with the same term. Such possibilities can occur if $F$ and $G$ collapse to 'a' since otherwise: 1) If both collapse to identity, then $q$ will be checked with strings of different length because $G_i \neq I$. 2) If only one of them collapses to identity, then $q$ will be tested once with a function of $x$ and once with a function of $a$.

Now the proof proceeds exactly as the end of the proof of Lemma 1: Schema $S_2$ is non-free if and only if there exist $p > 0$ and $1 \leq i_1, \ldots, i_p \leq n$ such that $\lambda_{i_1} \ldots \lambda_{i_p} = \mu_{i_1} \ldots \mu_{i_p}$ which implies (like in Theorem 1) that freedom of monadic schema $S$ with individual constants is undecidable.

**Remark:** From now on we shall use the original definition of functional schemas.

§3 Freedom of Functional Schemas

We now develop some tools to construct a polynomial time algorithm for checking freedom of functional schemas with nested predicates.

If we assume that the schema is free then all paths of computations are possible and we can easily construct the following sets:

‡The constructions are similar to corresponding constructions for context free grammars.
\( T = \{ i | F_i \Rightarrow \tau \}^+ \), \( D_1 = \{ i | F_i \Rightarrow F_\infty \} \), \( D_2 = \{ i | \emptyset \) such that the computation of \( F_i \) under \( \emptyset \) does not terminate\}, \( T = \{ i | F_i \Rightarrow \alpha F_j \Rightarrow \alpha' F_j \Rightarrow \tau \} \), \( T = \{ i | F_i \Rightarrow \alpha F_j \Rightarrow \tau \} \).

The constructions of \( T \) and \( D_1 \) are left to the reader.

The construction of \( D_2 \) and \( R_i \) can be accomplished by using a directed graph \( G \) with a set of nodes \( \{ F_i \} \) and a set of arcs \( \{ \langle F_i, F_j \rangle | \text{if } F_i \Rightarrow \alpha F_j \\beta \text{ and } \forall k \text{ such that } F_k \in \beta, k \in T \} \).

Now in \( \hat{G} \), the transitive closure of \( G \), we have an arc \( \langle F_i, F_j \rangle \) iff \( F_j \) is reachable from \( F_i \) by some computation. Thus
\[
\begin{align*}
R_i &= \{ j | \langle F_j, F_i \rangle \text{ is an arc in } \hat{G} \} \quad \text{and} \\
D_2 &= \{ i | \forall j \text{ such that } \langle F_i, F_j \rangle \text{ and } \langle F_j, F_i \rangle \text{ are arcs in } \hat{G} \}.
\end{align*}
\]

Note that all these constructions are polynomial in time.

**Lemma 2:** It is decidable whether or not a free schema with nested predicates terminates or diverges (for every interpretation).

**Proof:** We construct \( T \) and \( D = D_1 \cup D_2 \). Now \( T \) is the set of indices of functions which converge under some interpretation and \( D \) is the set of indices of functions which diverge under some interpretation because we assumed that the schema is free. Thus, the schema terminates if and only if \( F_0 \in T-D \); and the schema diverges if and only if \( F_0 \in D-T \); and Lemma 2 is proved.

**Remark:** Lemma 2 holds also for non-free schemas (see remark b after Theorem 5). But then we cannot guarantee a polynomial-time algorithm for deciding freedom.

\( ^+ \tau \) is a constant term.
Algorithm 1: For a given schema S, construct R_i and delete the definitions of F_i's for which 0 \not\in R_i.

Lemma 3: If S is free, then Algorithm 1 deletes exactly all F_i's such that F_0 \not\Rightarrow \alpha F_i \tau.

The obvious proof is omitted. Note that if S is not free, then Algorithm 1 deletes only definitions of function variables that cannot be reached from F_0 (but perhaps not all of them).

Theorem 3: It is decidable whether a nested schema is free or not.

Proof: Given a schema S we convert it to the form in which there is a single predicate in every definition. (These transformations do not affect freedom).

In this section only we rename the function variables \{F_i\} in a way that no two of them will have the same name in the right handsides of the schema's definition. e.g. the term GFFF in F <= if p then GFFF else I will be replaced by G_3 \cdot F_1 \cdot F_2 and both F_1 and F_2 will have the same definition. This renaming prevents the need for an additional index, showing which occurrence of F is considered. We define recursively two classes of predicates \{0_{i,\lambda}\} and \{I_{i,\lambda}\}. Assuming the schema is free I_{i,\lambda} = (p | F_0 \Rightarrow \alpha F_i \tau \text{ and } p(\lambda \tau) \text{ has been determined during this computation}) and 0_{i,\lambda} = (p | F_0 \Rightarrow \alpha F_i \tau' \Rightarrow \alpha \tau \text{ and } p(\lambda \tau))
has been determined during this computation). In other words
p ∈ I_i,λ if there is a computation in which when we come to
compute F_i we already know what p(λτ) is, and p ∈ O_i,λ if
there is a computation in which when we finish to compute
F_i τ', obtaining τ we already know p(λτ).

We begin with empty sets and follow the 4 rules:
If in F_i's definition we find the predicate pξ

<table>
<thead>
<tr>
<th>and the term</th>
<th>then the rule is</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) αF_j u</td>
<td>I [ I_{j, \nu} + I_{j, \nu} U I_{i, \lambda} ] for every ( \lambda = \nu \mu )</td>
</tr>
<tr>
<td></td>
<td>II [ I_{j, \nu} + I_{j, \nu} U {p} ] if ( \xi = \nu \mu )</td>
</tr>
<tr>
<td>(2) μF_j a</td>
<td>[ 0_{i, \nu} + 0_{i, \nu} U 0_{j, \lambda} ] for every ( \lambda = \nu \mu )</td>
</tr>
<tr>
<td>(3) μ</td>
<td>I [ 0_{i, \nu} + 0_{i, \nu} U I_{i, \lambda} ] for every ( \lambda = \nu \mu )</td>
</tr>
<tr>
<td></td>
<td>II [ 0_{i, \nu} + 0_{i, \nu} U {p} ] if ( \xi = \nu \mu )</td>
</tr>
<tr>
<td>(4) αF_j μF_kβ</td>
<td>I [ I_{j, \nu} + I_{j, \nu} U 0_{k, \lambda} ] for every ( \lambda = \nu \mu )</td>
</tr>
</tbody>
</table>

We denote the predicate which appears in F_i's definition by
q_i(q_iξ_i). It is easy to see that non-empty \( \text{O}_{i, \lambda} \) or \( \text{I}_{i, \lambda} \) are
possible only for \( \lambda \) which are prefix of some \( \xi_i \) \(|\lambda| \leq \max|\xi_i| = \kappa\),
therefore the algorithm of reapplying these rules until the sets stop
increasing must terminate.

In Lemma 7 we shall prove that S is free if and only if
q_i \( \notin I_{i, \xi_i} \) for every i. Thus we obtain an algorithm which decides
whether a schema is free or not, proving the theorem.
Now we will try to give an intuitive explanation for these rules, assuming $S$ is free.

If we have $F_1 \leftarrow \text{if } p \text{ then } F_2 f \text{ else } I$, then if in our construction we get $q \in I_{1,hf}$ we should put $q$ in $I_{2,h}$ since if we come to compute $F_1 \tau$ and know $q(h\tau)$ we will come to compute $F_2(f\tau)$ and know $q(h(f\tau))$. Also we will have to put $p$ in $I_{2,g}$ since we come to compute $F_2 f \tau$ and know $p(g(f\tau))$. These are exactly both parts of rule 1.

If we have $F_1 \leftarrow \text{if } p \text{ then } F_2 F_3 f$, then if $q \in I_{0,3,hg}$ we should put $q$ in $I_{2,h}$ since $F_3 f \tau$ can compute $\tau$ with checking $g(hg\tau)$ which implies that we may come to compute $F_2 \sigma \tau$ knowing $p(h(g\tau))$. This was rule 4. Here we see why we do need to rename before we start applying the rules: We are interested in those predicates that will be determined when we finish the computation of this occurrence of $F_3$.

Rules 2 and 3 are explained similarly.

Example 1: Consider the schema $S$:

$$F_0 \leftarrow \text{if } p g f h \text{ then } F_2 f F_1 \text{ else } I$$
$$F_1 \leftarrow \text{if } q \text{ then } q F_1 h \text{ else } f$$
$$F_2 \leftarrow \text{if } p \text{ then } I \text{ else } F_\infty .$$

$S$ is not free since if we take interpretation with

$p g f h = q = \text{TRUE}$ and $q h = \text{FALSE}$ we have

$$F_0 \rightarrow F_2 f F_1 \rightarrow F_2 f g F_1 h \rightarrow F_2 f g f h \ldots$$

and $p g f h$ is checked twice.

Using the above procedure, omitting irrelevant steps (and without renaming since here it is unnecessary) we get the following:
Definition of

<table>
<thead>
<tr>
<th>Rule number</th>
<th>( u )</th>
<th>( v )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_0 )</td>
<td>1 ( \text{II} )</td>
<td>( \epsilon^+ )</td>
<td>( fgfh )</td>
</tr>
<tr>
<td>( F_1 )</td>
<td>1 ( I )</td>
<td>( h )</td>
<td>( fgf )</td>
</tr>
<tr>
<td>( F_1 )</td>
<td>3 ( I )</td>
<td>( f )</td>
<td>( fg )</td>
</tr>
<tr>
<td>( F_1 )</td>
<td>2</td>
<td>( g )</td>
<td>( f )</td>
</tr>
<tr>
<td>( F_0 )</td>
<td>4</td>
<td>( f )</td>
<td>( \epsilon )</td>
</tr>
</tbody>
</table>

\( p \in I_2, \epsilon, \xi_2 = \epsilon \) and \( p = q_2 \Rightarrow S \) is not free

**Lemma 3:** If \( S \) is free then, \( q \in I_1, \lambda \) implies that

\( F_0 \Rightarrow \alpha F_1 \tau \) during which \( q \lambda \tau \) is determined; and \( q \in 0_1, \lambda \)

implies that \( F_0 \Rightarrow \alpha F_1 \tau' \Rightarrow \alpha \tau \) during which \( q \lambda \tau \) is determined.

**Proof:** The selection made by algorithm (1) and the freedom of \( S \) implies that every \( F_1 \) can be reached from \( F_0 \). During the construction of the sets \( \{0_1, \lambda\} \{I_1, \lambda\} \) we use another index \( \tau \) with initial value 0. \( \tau \) is increased by 1 every time one of the sets increases because one of the rules is applied. Our proof is an induction on \( \tau \) which is the stage in the construction of the sets. At the beginning all sets are empty and \( \tau \) becomes 1 because the second part of rule No. 1 or rule No. 3 is used.

It is obvious that in both cases the lemma holds. We assume the lemma is true for predicates which joined some set in one of the first \( \tau \) stages. The proof that it is true also for \( \tau + 1 \) is immediate. We actually have to check what happens when each rule is applied.

\( \epsilon \) is the empty string
If \( q \) joins \( I_{j, \lambda} \) in the \((r+1)\)th stage due to rule No. 1 it happens because of one of two possibilities:

i) For some \( \lambda = \nu \mu \), \( q \in I_{i, \lambda} \) before this rule is applied. Then by induction hypothesis \( F_0 \Rightarrow \beta F_i \tau \), during which \( q\lambda\tau = q\nu\mu\tau \) is determined. Thus (since \( F_i \Rightarrow \alpha F_j \mu \)) \( F_0 \Rightarrow \beta a F_j (\mu\tau) \) during which \( q\nu(\mu\tau) \) is determined, which is exactly the lemma's claim. (It becomes more obvious if we substitute: \( \tau' = \mu\tau \) and \( a' = \beta a \), i.e. \( F_0 \Rightarrow a' F_j \tau' \) during which \( q\nu\tau' \) is determined).

ii) \( \xi = \nu \mu \) and \( q_i \xi_i = q\xi \). We take any computation which satisfies \( F_0 \Rightarrow \beta F_i \tau \), when we use \( F_i \)'s definition \( q\xi\tau = q\nu(\mu\tau) \) is determined. But \( F_0 \Rightarrow \beta F_j (\mu\tau) \) and the lemma holds. (Again the same substitution makes it compatible with the lemma's notations).

The second case is exactly one of two possibilities when \( r = 1 \).

We used here (in both cases) the freedom assumption when we chose the expression in the \( F_i \)'s definition.

Similar proof when we use one of the other rules is omitted.

**Lemma 5**: If \( F_0 \Rightarrow \alpha F_k \nu F_m \beta \) and \( \lambda = \nu \mu \) then \( 0_{m, \lambda} \subseteq I_{k, \nu} \).

**Proof**: By induction on \( \tau \) - the computation's length. For \( \tau = 1 \) it is obvious as rule No. 4 is applied. Now suppose that the argument holds for \( \tau \), \( F_0 \Rightarrow a F_k \nu F_m \beta \) and
\( \lambda = \nu \mu \). If the sub-expression \( F_\lambda \mu F_m \) appeared before, the argument holds because of induction's assumption. On the other hand if it is generated completely in the last step, it holds because rule No. 4 is applied.

The only possibility left is \( F_0 \vdash a F_\lambda \mu_1 F_\mu \eta \) and \( F_\mu = \mu_2 F_m \xi \). Thus \( \mu = \mu_1 \mu_2 \) and \( \beta = \xi \eta \) (constant). Since \( \lambda = \nu \mu_1 \mu_2 \) rule No. 2 implies \( 0_m, \lambda \subseteq 0_p, \nu \mu_1 \) and induction's hypothesis implies \( 0_p, \nu \mu_1 \subseteq I_\mu, \nu \) thus \( 0_m, \lambda \subseteq I_\mu, \nu \), completing the proof.

Lemma 6: If \( F_0 \Rightarrow a F_\lambda \tau \) during which \( q \lambda \tau \) is determined, then \( q \in I_\mu, \lambda \).

Proof: Again we use an induction on \( t \) - the computation's length. If \( t = 1 \), then \( q \lambda \tau \) can be determined only if \( F_0 \Rightarrow a F_\mu \nu \) and \( \xi_0 = \lambda \mu \tau \), thus the second part of rule No. 1 implies \( q \in I_\mu, \lambda \).

We assume that the argument holds for \( t \) and \( F_0 \Rightarrow a F_\mu \tau \) during which \( q \lambda \tau \) is determined. If \( q \lambda \tau \) was determined because of the last step of computation, the argument holds exactly like in the case of \( t = 1 \). Thus we may assume that \( q \lambda \tau \) was determined before the last step. Therefore there are two possibilities for the \((t+1)\)th step:

i) \( F_0 \vdash a' F_j \tau_1, F_j \vdash a'' F_i \tau_2 \) and \( q \lambda \tau = q \lambda \tau_2 \tau_1 \). The induction hypothesis implies \( q \in I_j, \lambda \tau_2 \) and rule No. 1 implies \( I_j, \lambda \tau_2 \subseteq I_\mu, \lambda \) thus \( q \in I_\mu, \lambda \).
ii) \( F_0 \vdash aF_i \tau_j F_j \tau_1 \), \( F_j \vdash \mu \) and \( \tau = \tau_2 \mu_1 \). By induction's assumption \( q \in I_j, \lambda \tau_2 \mu \), rule No. 3 implies \( I_j, \lambda \tau_2 \mu \subseteq 0_j, \lambda \tau_2 \) and Lemma 5 implies \( 0_j, \lambda \tau_2 \subseteq I_i, \lambda \), thus \( q \in I_i, \lambda \).

Q.E.D.

Lemma 7: \( S \) is free if and only if \( q_i \not\in I_i, \xi_i \) for every \( i \).

Proof: If the schema is free and \( q_i \not\in I_i, \xi_i \) for some \( i \), Lemma 4 implies that \( F_0 \vdash aF_i \tau \) and \( q_i \xi_i \tau \) is determined during this computation — a contradiction to freedom assumption. On the other hand if the schema is not free, then there exists \( i \) such that \( F_0 \vdash aF_i \tau \) and \( q_i \xi_i \tau \) is determined during the computation. But Lemma 6 implies that \( q_i \in I_i, \xi_i \), completing the proof.

Remark: When the predicates are not nested (or when it is possible to replace every nested predicate by an unnested one) the algorithm becomes much simpler. In this case all the \( \xi_i \)'s are empty and all the \( \lambda, \mu, \nu \) must be empty as well. We may omit the second index and obtain the following rules: If in the definition of \( F_i \) appears the predicate \( p \) and

<table>
<thead>
<tr>
<th>Rule</th>
<th>Term</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( aF_j )</td>
<td>( I_j \uplus I_j \uplus I_i \uplus \nu { p } )</td>
</tr>
<tr>
<td>(2)</td>
<td>( F_j \alpha )</td>
<td>( 0_i \uplus 0_i \uplus 0_j )</td>
</tr>
<tr>
<td>(3)</td>
<td>( I )</td>
<td>( 0_i \uplus I_i \uplus 0_i \uplus \nu { p } )</td>
</tr>
<tr>
<td>(4)</td>
<td>( aF_k F_j \beta )</td>
<td>( I_k \uplus I_k \uplus 0_j )</td>
</tr>
</tbody>
</table>
Note that our algorithm for deciding freedom is polynomial in time since we have non-empty \( O_i, \) or \( I_i, \) only for \( \lambda \) which is a prefix of some \( \xi_i \) (\( \xi_i \equiv \nu \nu, \nu \) can be empty).

Kurt Mehlhorn has suggested another method which is in the spirit of the algorithm in [1] for unnested schemas: by finding an integer \( K \) such that if the schema is not free there is a partial interpretation of length less than \( K \) under which non-freedom occurs. Then the algorithm is to compute \( K \) and check all partial interpretation of length up to \( K \). This algorithm turns out to be exponential in time. Its advantage is that the proof that it works is much simpler.

§4 Deletion of Identity from Free Nested Schemas

Given a free nested schema, \( S \), we show how to construct an equivalent free nested schema \( S' \) without \( I \). \( S' \) will have a level of nesting bigger by one than that of \( S \).

**Definition:** A schema \( S \) is said to be in **standard form** if

1. Every conditional term in \( S \) is of the form

   \[
   \text{if } \varphi \text{ then } \tau_2 \text{ else } \tau_3 \]

   where
   \( \tau_2 \) and \( \tau_3 \) are of one of the following forms:
   
   (a) a conditional term not containing \( \varphi \),
   (b) \( F_\infty \),
   (c) \( I \), or
   (d) \( a f, f \in C \text{ and } F_\infty \notin a \).

2. Every definition in \( S \) (except for \( F_\infty \)) contains a conditional term.
3. Every function variable in $S$ does not diverge always (except $F_\infty$).

**Lemma 8:** Every free schema can be effectively transformed to an equivalent free schema in standard form.

**Proof:** It is done exactly as it was done to schemes without nesting in [1], since all diverging non terminal functions can be identified (by Lemma 2).

**Theorem 4:** Given a free schema $S$ in standard form in $F_0 \not\vdash I$. Then there exists another free schema $S'$ equivalent to $S$ without $I$.

**Proof:** Let $A = \{F \mid F \in V, F \vdash I\}$ and $B = (V - A) \cup C$. We construct a schema $S'$ with the same function constants as in $S$. The function variables of $S'$ are elements of the set $Q$,

$$Q = \{[\alpha G] \mid G \in B, \alpha \in A^*\} \cup \{F_\infty\}.$$ 

For every string of functions in $S$, $\alpha$, which terminates with a function of $B$, we define a string $\tilde{\alpha}$ in $S'$ in the following way: if $\alpha$ contains $F_\infty$ then $\tilde{\alpha}$ is $F_\infty$ otherwise if $\alpha = G_1 \ldots G_n$ and $G_{i_1}, \ldots, G_{i_t}, \ 0 = i_0 < i_1 \ldots < i_t = n, t \geq 1$ are all the functions which belong to $B$ and appear in $\alpha$; then let

$$H'_p = [G_{i_p-1} \ldots G_{i_p}] \ 1 \leq p \leq t$$

and

$$H'_p = \begin{cases} f & \text{if } H'_p = [f], \ f \in C \\ H'_p & \text{otherwise.} \end{cases}$$
Then $\bar{a} = H_1 \ldots H_t$.

An immediate result of these definitions is: For $\bar{a}_1, \ldots, \bar{a}_r$ which are strings of functions in $S$ that terminate with a function of $B$ and $\tau \in C^*$ we have $\bar{a}_1 \ldots \bar{a}_r = \bar{a}_1 \ldots \bar{a}_r$. (We shall use this result in Lemmas 9 and 10 without mentioning it).

If $[a] \in Q$, then there are two possibilities:

1) $a = \alpha' F_1$ and $F_1 \in V-A$. The definition of $[a]$ is obtained by replacing in $F_1$'s definition any string $\beta$ with $\overline{\alpha' \beta}$.
Since $\beta \neq I$ ($F_1 \in B$) then $\beta$ terminates with a function constant ($S$ is in standard form) which belongs to $B$. Thus $\overline{\alpha' \beta}$ is defined.

2) $a = \alpha' F_1 f$, $F_1 \in A$ and $f \in C$. In this case the definition of $[a]$ is obtained by replacing any $\beta$ in $F_1$'s definition by $\overline{\alpha' \beta f}$ (which is defined since $f \in B$) and instead of any predicate $pr$ we put $prf$. This completes the construction of $S'$.

Note that the level of nesting in $S'$ is at most bigger by one than that of $S$. We agree that $\bar{F}_0 = F_0$ and prove

Lemma 9: If $F_0 \Rightarrow \alpha$, then $[F_0] \Rightarrow \bar{a}$; and

Lemma 10: If $[F_0] \Rightarrow \alpha^*$, then there exists $\alpha$ such that $\alpha^* = \bar{a}$ and $F_0 \Rightarrow \bar{a}$.

Lemmas 9 and 10 imply that $S$ and $S'$ are equivalent. Since $S$ is free, if $F_0 \Rightarrow \alpha$, then $\alpha$ does not contain more than $n$ successive elements of $A$, where $n$ is the number of function variables of $S$. (Otherwise a successive collapse to $I$
causes that the same function variable is computed twice with the same argument). By Lemma 10 the variables which can be reached from \([F_0]\) are only \([\alpha G]\) for \(|\alpha| \leq n\). Therefore, the number of function variables in \(S'\) is finite. The proof of Lemmas 9 and 10 will show that the same predicates are checked with the same arguments during corresponding computations. Thus \(S'\) is free, and the proof is completed.

**Proof of Lemma 9:** We use an induction on \(t\) (the computation's length). If \(t = 1\), then \(\alpha \neq I\) and the result follows the definition of \([F_0]\). If \(F_0 \overset{\theta,t}{\Rightarrow} \beta = \delta' \cdot f' = \delta \cdot a'\cdot f'\) (\(\tau\) and \(\gamma\) are non-empty). According to induction hypothesis \([F_0] \overset{\theta,t}{\Rightarrow} \beta' \cdot f'\). We distinguish between two possibilities:

1) If \(F_1 \in B\), then \(\beta' \cdot f' = \gamma (\delta F_1) \cdot \tau\) and \([\delta F_1] \cdot \tau = \delta \cdot a_1\cdot f_1\).

   Thus \([F_0] \overset{\theta,t+1}{\Rightarrow} \\gamma \delta a_1 \cdot \overline{\tau} = \overline{\gamma \delta a_1 \cdot \tau} = \alpha\) (\(\gamma \delta = \alpha'\))

2) If \(F_1 \in A\), then \(\beta' \cdot f' = \gamma (\delta F_1) \cdot \tau\) and \([\delta F_1] \cdot \tau = \delta \cdot a_1 \cdot f_1\).

   Thus \([F_0] \overset{\theta,t+1}{\Rightarrow} \\gamma \delta a_1 \cdot f_1' = \gamma \delta a_1 \cdot f_1' = \alpha\), completing the proof.

**Remark:** The change in the predicate is necessary in order that it will be checked with the same argument in both computations, and thus to get the same behavior under \(\alpha\).

**Proof of Lemma 10:** Again we use the same induction. When \(t = 1\) the result follows \([F_0]\)'s definition and the fact that \(P_0 \overset{\tau}{\Rightarrow} I\). By induction hypothesis \([F_0] \overset{\theta,t}{\Rightarrow} \beta^*\) implies that \(\beta^* = \beta = \beta' \cdot f_1' = \alpha'\) and \([F_0] \overset{\theta,t}{\Rightarrow} \alpha' = \beta = \beta' \cdot f_1'\).
Here too we have two possibilities:

1) If $F_i \epsilon B$, then $\bar{\delta}'F_i^T = \gamma[\delta F_i]T$; and $[\delta F_i]T \theta \delta a_i^T$ only if $F_i T \theta a_i^T$. Thus, $F_0 \theta^{T+1} \beta' a_i^T = a$ and $a^* = \gamma \delta a_i^T = \bar{a}$.

2) If $F_i \epsilon A$, then $\bar{\delta}'F_i^T = \gamma[\delta F_i]T$; and $[\delta F_i]T \theta \delta a_i^T$ only if $F_i T \theta a_i^T$. Thus $F_0 \theta^{T+1} \beta' a_i^T = a$ and $a^* = \bar{a}$, and the lemma is proved.

Remark: If $F_0 \rightarrow I$, then by introducing a new initial function variable, $F_0$, (with the same definition as $F_0$) and applying the technique above we get an equivalent schema $S'$ with $I$ only in $F_0$'s definition and $F_0$ doesn't appear elsewhere in $S'$ i.e. the collapse to $I$ is possible only at the first step of the computation.

§5 Equivalence of Free Nested Schemas

In this section we give a procedure which decides if two given free nested schemas are equivalent.

**Lemma 11:** Given a free schema, $S$, with an initial function variable $F_0$, we can construct a free schema $S'$ with an initial function variable $F_0'$ such that 1) $S'$ is in standard form 2) $S'$ without $I$ and 3) $F_0 \theta \tau$ if and only if $F_0' \theta \tau$.

**Proof:** By Lemma 8 we construct a free schema $S_1$ with the same initial function variable which is in a standard form and equivalent to $S$. Let $S_2$ be the free schema obtained from $S_1$ by adding to it a new initial function variable $F_0$. The definition of $F_0$ is that of $F_0$ if we replace any term $a$, which
is not an argument of a predicate by \( f_a \). \( (fF_\infty = F_\infty) \);

\[ \text{e.g. if } F_0 \Leftarrow \text{if } p f_1 \text{ then } f_1 f_2 \text{ else } F_\infty, \text{ then} \]

\[ F_0 \Leftarrow \text{if } p f_1 \text{ then } f f_1 f_2 \text{ else } F_\infty. \]

Note that \( F_0 \)'s definition is not deleted. Obviously \( S_2 \)
is in standard form and \( F_0 \not\Rightarrow \alpha \) if and only if \( F_0 \not\Rightarrow \alpha' \).

Since \( F_0 \not\Rightarrow I \) we construct (by Theorem 4) a free schema

\( S_3 \) which is equivalent to \( S_2 \) and does not contain \( I \).

Denote the initial variable of \( S_3 \) by \( F_0'' \). Now transform

\( S_3 \) to standard form to obtain \( S' \) with an initial function

variable \( F_0' \). \( S' \) satisfies all the properties stated above,

since the transformation to standard form does not introduce

any new non-freedom and does not generate \( I \).

**Corollary:** The problem of equivalence of free schemas reduces
to that of free schemas which are in standard form and do not
contain \( I \).

Suppose we are given two free functional schemas \( S \) and

\( S' \) both in standard form without \( I \). We construct a Deterministic Push-Down Automaton (DPDA) which in some sense

simulates the joint action of \( S \) and \( S' \) (since in case of

conflict it stops). A interprets each input symbol as an

assignment of truth values to all the predicates of both

schemas, part of which might be needed to continue the simula-

tion: If \( A \) reaches the \( r \)-th input symbol we must have \( F_0 \Rightarrow \alpha' \) and
$F_0' \Rightarrow \alpha \tau$, $|\tau| = r-1$ and $\tau$ is a constant term. Then the $r$-th input symbol gives truth values to all the predicates in $S$ and $S'$ with argument $\tau$. The set of input tapes includes also 'inconsistent tapes'. A tape is said to be inconsistent if at two different points it assigns different values to a predicate with the same argument. This is possible since we have nested predicates. Inconsistency can be revealed only during the action of $A$. $A$ accepts an input tape if and only if it is consistent, and $S$ and $S'$ are inequivalent under the corresponding interpretation. Since it is decidable whether or not the language accepted by a DPDA is empty ([7]), it follows that the equivalence problem for free schemas is decidable.

Suppose $S$ and $S'$ use $n$ distinct forms of predicates $q_t = p_{i_t}^\xi_j^\tau_t$, $t = 1, \ldots, n$. The input alphabet of $A$ is $\{(T,F)^n\}$. When $A$ reads such a word (input symbol), the $t$-th letter denotes the current value of $q_t$. Before the appearance of first inconsistency the $r$-th input symbol represents the values of $p_{i_t}^\xi_j^\tau_t$ $t = 1, \ldots, n$, where $\tau$ is a string of function constants, $|\tau| = r-1$, which was obtained in a partial computation of $F_0$ and $F_0'$ (the initial function variables of $S$ and $S'$ respectively) using the first $r-1$ input symbols. (The following construction will make sure that $A$ will reach the $r$-th symbol only when the same $\tau$ is obtained in both schemas). Inconsistency can be generated only in the following circumstances: Suppose that $q_{t_1} = p_{\xi_\lambda}^\tau$ ($|\lambda| = k$), $q_{t_2} = p\xi$, and there exists an interpretation $\theta$ satisfying $F_0 \theta \Rightarrow \alpha \lambda \tau$ and $F_0' \theta \Rightarrow \alpha \lambda \tau$ ($|\tau| = m$). Now, take a tape with
first \( l + m \) input symbols of the representation, including the 
\((m+1)\)th input symbol \( a_1, \ldots, a_n \), and with \((l + m + 1)\)th input 
symbol \( b_1, \ldots, b_n \) such that \( a_{t_1} \neq b_{t_2} \). But \( a_{t_1} \) stands for 
\( p e \lambda (\tau) \) and \( b_{t_2} \) stands for \( p e (\lambda \tau) \).

Let \( k \) be the maximal depth of nesting in \( S \) and \( S' \) 
(i.e. \( k = \max |\xi| \) \( p e \) appears in \( S \) or \( S' \)). In order to check 
inconsistency \( A \) must remember only the last \( k \) input symbols 
he has read. Obviously, it is a feasible action of a multi state 
DPDA.

To simulate the joint action of \( S \) and \( S' \) for a given 
tape, we let \( A \) have a two-track push-down stack. Each track 
will hold a modified version of the current term in the computa-
tion sequence of the corresponding schema under the corresponding 
Herbrand interpretation. (We suppose that any inconsistency has 
not been reached yet).

The modification of the computation terms is such that if 
\( S \) and \( S' \) are equivalent, both tracks are of the same length 
during corresponding computations 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' \( T(\alpha) \) 
of a term \( \alpha \) (that does not contain \( F_\lambda \)): \( T(\alpha) = \min( |\tau|, \tau \) is a 
string of function constants and \( \alpha \Rightarrow \tau \)). For free schemas we have 
\( T(\alpha_1 \alpha_2) = T(\alpha_1) + T(\alpha_2) \), i.e. the shortest string computed from 
\( \tau_2 \), followed by that computed from \( \tau_1 \). This follows from the 
fact that for free schemas if \( \alpha_1 \Rightarrow_{\theta_1} \tau_1 \) and \( \alpha_2 \Rightarrow_{\theta_2} \tau_2 \), then 
there is \( \theta \) such that \( \alpha_1 \alpha_2 \Rightarrow \tau_1 \tau_2 \). The required modification
of a term to give its stack representation is to make \( T(F_i) \) copies of each \( F_i \in V \). Thus the length of stack representation of term \( \alpha \) is \( T(\alpha) \). Thus if \( S \) and \( S' \) are equivalent and the corresponding terms in their computation are \( \alpha \) and \( \alpha' \); then \( T(\alpha) = T(\alpha') \), so that the modified form has the required property stated above. To erase a function variable \( F_i \) from the stack, the automaton \( A \) will actually erase \( T(F_i) \) copies of \( F_i \) which is a feasible action of a multi-state DPDA.

The behavior of the DPDA is as follows:

For each input symbol \( A \) first checks the consistency of the tape. If any inconsistency is revealed \( A \) passes to a rejecting state, otherwise \( A \) simulates the actions of \( S \) and \( S' \) under the corresponding interpretation. If the topmost letter of the corresponding track is a function constant no change is made. Otherwise, it must be a function variable and we modify the top of that track according to the term in the definition of \( F_i \) selected by the current symbol. These actions will terminate either with some new stack-track with a function constant at the top or \( F_\infty \) will be encountered. The crucial point is that for free schemas without \( I \) in standard form only a bounded amount of information must be remembered for checking consistency and for changing the tracks' contents concurrently.

Before moving to the next input symbol, \( A \) proceeds as follows:

a) If \( F_\infty \) is encountered for both tracks it passes to a rejecting state, (i.e. \( S \) and \( S' \) are equivalent under the corresponding interpretation).
b) If \( F_\infty \) is encountered only for one track \( A \) passes 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).

c) If the two tracks are not of the same length \( A \) passes 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).

d) If two different constant functions appear at the top of both tracks \( A \) passes to an accepting state.

e) Otherwise (i.e. same length tracks with the same topmost function constant) \( A \) removes the topmost letters. If both stacks are still non-empty \( A \) moves to the next input symbol, otherwise both tracks are empty and \( A \) passes to a rejecting state.

Thus \( S \) and \( S' \) are equivalent if and only if the DPDA accepts no input tapes, which is a known decidable problem. Thus we have

**Theorem 5:** It is decidable whether or not two free schemas are equivalent (for every interpretation).

**Remarks:**

(a) In the proofs of Theorem 4 and Theorem 5 we used some ideas developed in [4]. The deletion of I from free schemas is somewhat similar to deletion of \( \epsilon \)-rule from LL(k) grammars. Also the automaton \( A \) which checks the equiva-
lence of two free schemas, and the DPDA which checks equivalence of two LL(k) grammars are similarly constructed. Perhaps there exists a deeper relation between free nested schemas and LL(k) grammars.

(b) Given a schema $S$ we can construct a one track DPDA to simulate $S$ (as $A$ simulates $S$ and $S'$ in the proof of Theorem 5). Here we don't have to modify terms in order to get their stack representation. (Since we have only one schema). Thus we can easily decide termination (or divergence) for $S$ by considering the DPDA.

Example 2:
Consider the schemas

$S: F \leftarrow \text{if } p \text{ then } F F f g \text{ else } h$

$S': F' \leftarrow \text{if } p \text{ then } F' G' g \text{ else } h$

$G' \leftarrow \text{if } p f \text{ then } F' G' g f \text{ else } h f$

A short glance will convince the reader that $S$ and $S'$ are free since in both cases whenever $p f$ is checked before $p f_2$

$\tau_1$ must be shorter than $\tau_2$, and $S$ is equivalent to $S'$ since $S'$ is obtained by substituting in $S G'$ for $F f$ to get $F'$ and then obtaining the definition of $G'$ from the definition of $F$.

Now, $\tau(F) = \tau(F') = 1$ and $\tau(G') = 2$. We have $p$ and $p f$ hence the input symbols are binary words of length 2: $ab$ is the value of $p$ and $b$ is the value of $p f$. ($a, b \in \{T, F\}$)
We describe below the behavior of the DPDA given the input

\[ \text{input: } \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & F \\
F & F & F & F \\
\end{array} \ldots \]

1 \quad 2

\[ \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & F & f & g \\
\end{array} \quad \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & G' & G' \\
F & G' & G' & g \\
\end{array} \]

3 \quad 4

\[ \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & F & f & g \\
\end{array} \quad \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \]

5 \quad 6

\[ \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \quad \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \]

7 \quad 8

\[ \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \quad \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \]

9

\[ \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \quad \begin{array}{cccc}
T & F & F & F \\
\uparrow & F & F & f \\
F & h & f & g \\
\end{array} \]

Input is rejected.
Note that in the lower track we pushed in and later popped out 2 copies of \( G' \) that because \( T(G') = 2 \). This enabled us to have tracks of equal length.

The input \[ \text{TF FF TF FF } \ldots \] will be rejected because of different reasons: Since its first input symbols are the same as before we have

\[
\begin{array}{c|c}
\text{TF FF TF FF } \ldots \\
\hline
5 & \text{F F} \\
\hline
\text{F' h} \\
\end{array}
\]

The previous input symbol was \[ \text{FF} \] and the second meant that \( p_{\tau} \) is false \( \tau \) was then the argument \( (\tau = q_x) \). Now the input symbol is \[ \text{TF} \] and \( T \) means that \( p_{\tau'} \) is true and \( \tau' \) is the current argument. But since \( f \) has been erased \( \tau' = f \tau \) and \( p_{\tau} \) can't be true, i.e. this tape is inconsistent and is therefore rejected.

The input \[ \text{TF TF FF FF ...} \] is consistent and will cause the same behavior as the first input. It is consistent because in the first input character the \( F \) means \( p_{fx} \) is false and in the second the \( T \) means \( p_{gx} \) is true since the argument at stage 3 is \( g_x \).

Acknowledgements:

I am indebted to K. Mehlhorn and R. L. Constable for advice and many suggestions and to A. Pnueli for introducing me to the topic of functional schemas and for many discussions.
References


