MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 118

Reducibility of the Equivalence Problem for Multi-tape Automata to the Equivalence Problem for Independent Location Iteration Schemas

by

Joseph E. Qualitz

This work was supported in part by the National Science Foundation under research grant GJ-34671 and in part by funds from International Business Machines.

March 1975

It has been demonstrated in the literature [2] that the equivalence problem for multi-tape finite automata as defined in [4] reduces to that for monadic, independent location program schemas - schemas which permit operator and test symbols representing functions and predicates of only one variable, and which have the property that the argument variable of an assignment statement is always the same as the assignment variable of the statement. Ashcroft and Manna have shown [1] that for any such schema there exists an equivalent "whilish" schema (i.e. one composed of assignment statements, conditional statements, and iteration statements) and an algorithm is given for constructing this equivalent schema. In general, however, application of this algorithm necessarily results in a whilish schema which is not independent location, and it is therefore reasonable to ask whether or not the equivalence problem for independent location schemas in general can be reduced to that for independent location whilish schemas.

In this memo, we provide an affirmative answer to the question above and, in the process, demonstrate that the equivalence problem for multi-tape finite automata reduces to that for multi-tape automata with a single control state.

I. Multi-tape Finite Automata

In this section, we show that the equivalence problem for the class of multi-tape finite automata [4] reduces to that for the class of multi-tape automata which have a single control state and which move no more than a single tape head during any step in a computation. The definition of multi-tape automaton given below differs slightly from that in [4] since, for reasons that will become apparent, we wish to treat the initial state of an automaton in a special manner. In particular, we allow no transitions back into the initial state of a machine and allow no movement of the tape heads during a transition out of the initial state.

Formally, an *n-tape finite automaton* is a five-tuple

$$M = (K, Q, q_I, f, h)$$

where: K is a finite set of *tape symbols*, including the special *endmarker* \$.

Q is a finite set of *control states*.

$q_I \notin Q$ is the *initial state* of the automaton.

$f: (Q \cup \{q_I\}) \times K^n \rightarrow Q$ is the *state transition function*.

$h: (Q \cup \{q_I\}) \times K^n \rightarrow {}_2\{1,2,\ldots,n\}$ is the *head advancement function*, and satisfies the properties that $h(q_I, \omega) = \emptyset$ and that $i \notin h(q, \omega)$ whenever the ith component of ω is \$, for all $\omega \in K^n$.

A *configuration* of M is a pair (q, Ω) where q is a state (either control or initial) and Ω is an n-tuple of strings in K^+. (For such an Ω, we denote by TAIL(Ω) the string in K^n whose ith symbol is the last symbol of the ith component of Ω, $1 \leq i \leq n$.)

A computation by M is a possibly infinite sequence of distinct configurations $(q_1, \Omega_1), (q_2, \Omega_2), \ldots, (q_k, \Omega_k), (q_{k+1}, \Omega_{k+1}), \ldots$ in which q_1 is the initial state q_I and in which, for all $i > 1$:

(1) $q_i = f(q_{i-1}, \text{TAIL}(\Omega_{i-1}))$

(2) $\Omega_i = \Omega_{i-1} \cdot S$ where S is an n-tuple of strings in $K \cup \{\epsilon\}$, such that for all j, $1 \leq j \leq n$, the jth component of S is ϵ iff $j \notin h(q_{i-1}, \text{TAIL}(\Omega_{i-1}))$.

An n-tuple Ω of strings in K^* is accepted by M if there is a finite computation by M ending with the configuration $(q, \Omega \cdot \$^n)$, for some q in $Q \cup \{q_I\}$. The language recognized by M, written L(M), is the set $\{\Omega \mid \Omega$ is an n-tuple of strings in K^* accepted by M $\}$. If M' is an n-tape finite automaton, then M and M' are equivalent iff $L(M) = L(M')$.

Lemma 1:

Let $M = (K, Q, q_I, f, h)$ be an arbitrary n-tape automaton, for some n. Then we may construct from M an equivalent n-tape automaton M' in which no more than a single tape head moves during any step in a computation by the machine.

Proof: Let q be a state in $Q \cup \{q_I\}$ and ω a string in K^n such that $h(q, \omega) = \{j_1, \ldots, j_m\}$ for some $m > 1$. (If no such state and string exists, then we choose as M' the machine M itself.) We add to Q new states $q', q'', \ldots, q^{m-1}$ and extend f and h to these new states so that $f(q^i, \mu) = q^{i+1}$ and $h(q^i, \mu) = j_{i+1}$, for all $\mu \in K^n$ and all i, $1 \leq i \leq m-1$. For each $\mu \in K^n$, we define $f(q^{m-1}, \mu)$ to be the state $f(q, \omega)$ and we define $h(q^{m-1}, \mu)$ to be $\{j_m\}$. Finally, we redefine

$f(q, \omega)$ to be state q' and $h(q, \omega)$ to be $\{j_1\}$. The procedure is repeated for any additional arguments for which the value of h is a set of cardinality greater than one, and the resultant automaton is the machine M'. The reader may verify that $L(M')$ is precisely the same as $L(M)$.

□

Definition: An n-tape automaton M is in normal form if precisely one tape head is advanced at each but the last step in a computation by the automaton.

Corollary 1.1:

For each multi-tape automaton $M = (K, Q, q_I, f, h)$ we can find an equivalent automaton in normal form.

Proof: We may assume without loss of generality that no more than one tape head is advanced during any step in a computation by M. Then whenever we have $f(q, \omega) = q'$ for some q in Q and ω in K^n (where n is the number of tapes of M) for which $h(q, \omega) = \emptyset$ and $h(q',\omega) \neq \emptyset$, we redefine $f(q, \omega)$ to be $f(q', \omega)$. This procedure is applied recursively until its application is no longer possible. The resultant scheme is in normal form.

□

If M is an n-tape automaton in normal form, its head advancement function can be viewed quite naturally as a (partial) function into $\{1, \ldots, n\}$. We adopt this view for the remainder of this paper.

<u>Definition</u>: Let $\omega = s_{i_1}s_{i_2} \cdots s_{i_k}s_{i_{k+1}} \cdots$ be a string over some alphabet K not containing the special symbol #. Then an <u>expansion</u> of ω is any string of the form:

$$\psi = s_{i_1} \cdot \delta_1 \cdot (\#)^{j_1} \cdot s_{i_2} \cdot \delta_2 \cdot (\#)^{j_2} \cdot \cdots \cdot s_{i_k} \cdot \delta_k \cdot (\#)^{j_k} \cdot s_{i_{k+1}} \cdot \delta_{k+1} \cdot (\#)^{j_{k+1}} \cdots$$

in which for each i, $\delta_i \in K^*$ and j_i is an integer > 0. An <u>expansion</u> of a tuple of strings $(\omega_1, \omega_2, \ldots, \omega_n)$ over K is any tuple of strings $\Psi = (\psi_1, \psi_2, \ldots, \psi_n)$ in which for each i, $1 \le i \le n$, ψ_i is an expansion of ω_i. Finally, if X is a set of tuples of strings over K, then the <u>expansion</u> of X is the set $\mathrm{EXP}(X) = \{\Psi \mid \Psi$ is an expansion of some $\Omega \in X\}$.

We note that if ω is a string over an alphabet K not containing #, the expansion of ω formed by inserting # between each pair of symbols in ω cannot be obtained as an expansion of any string distinct from ω. We thus have:

<u>Lemma 2</u>:

Let M and M' be n-tape finite automata whose tape alphabets do not contain #. Then M and M' are equivalent if and only if EXP(L(M)) = EXP(L(M')).

<u>Proof</u>: Left to the reader.

<u>Definition</u>: Let M be an n-tape automaton with tape alphabet K, control states denoted by a set of symbols Q, and the property that no more than a single tape head is moved during any step in a computation by M. Let $\Omega = (\omega_1, \omega_2, \ldots, \omega_n)$ be a tuple of strings accepted by M. Then the <u>trace</u> of Ω with respect to M is the n+4-tuple of strings $\Lambda =$ $(\alpha, \beta, \gamma, \gamma, \omega_1, \omega_2, \ldots, \omega_n)$ over $K \cup Q \cup \{\bar{1}, \bar{2}, \ldots, \bar{n}\}$ such that $\alpha \in K^*$ is the sequence of symbols scanned during the computation by which M accepts Ω, $\beta \in \{\bar{1}, \bar{2}, \ldots, \bar{n}\}^*$ represents the sequence of tape heads moved during the computation, and $\gamma \in Q^*$ is the sequence of states entered during the computation. We denote by TRACE(M) the set $\{\Lambda \mid \Lambda$ is the trace of some Ω accepted by M$\}$.

<u>Lemma 3</u>:

Let M be an n-tape automaton (K, Q, q_I, f, h) in normal form. Then we may construct from M an n+4 tape automaton M' such that M' has a single control state, M' is in normal form, and L(M') = EXP(TRACE(M)).

<u>Proof</u>: A formal definition of M' appears in the Appendix. We describe the behavior of M' informally below:

Let $\hat{Q}$ be the set of symbols $\{\hat{q}_i \mid q_i$ is a state in Q$\}$, and let $\bar{N}$ be the set of symbols $\{\bar{1}, \bar{2}, \ldots, \bar{n}\}$. The alphabet of M' is the set $K' = K \cup Q \cup \bar{N} \cup \{\#\}$.

We may represent a configuration of M' by an n+4-tuple of strings over the alphabet K', and we say that a configuration

Ω is a <u>base</u> configuration of M' if TAIL(Ω) is of the form $s\cdot x\cdot q_i \cdot q_i \cdot \omega$, where x is $\bar{\ell}$ for some $\bar{\ell}$ in $\bar{N}$ such that $\ell = h(q_i, \omega)$ in M, and s is the ℓth symbol in $\omega \in K^n$. Beginning at such a base configuration, the behavior of M' is as follows:

i) M' moves head $\ell + 4$ past any number of other symbols until symbol # is scanned on the $\ell + 4$th tape.

ii) M' then moves head 4 past any number of other symbols until symbol # is scanned on the 4th tape.

iii) M' then moves head 2 past any number of other symbols until symbol # is scanned on the second tape.

iv) M' advances head 4 past any number of copies of the symbol #, stopping when it reaches a symbol $\hat{q}_j$ such that $\hat{q}_j = f(q_i, \omega)$ in M. If the first symbol after the string of #'s is any symbol other than $\hat{q}_j$, M' hangs up, i.e. it scans repeatedly the same n-tuple of symbols without advancing any tape head.

v) M' advances head 1 until # is reached on the first tape, and then head 3 until # is reached on that tape.

vi) M' advances head $\ell + 4$ past any number of #'s until a symbol $s' \in K'$ is scanned on that tape. (M' hangs up if the first symbol after the #'s is not a symbol in K.)

vii) M' advances head 1 past any number of #'s until a symbol s" $\in$ K is scanned on the first tape, such that s" is the symbol under scan on the k+4th tape if $h(q_j, \omega')$ is k, or is the endmarker \$ if $h(q_j, \omega')$ is undefined in M, where ω' is the string formed by concatenating the symbols under scan on the last n tapes of M'. If any symbol other than the appropriate s'' is scanned immediately after the string of #'s, M' hangs up.

viii) If $h(q_j, \omega')$ is defined, where ω' is as in (vii), then M' advances head 2 past any number of #'s until a symbol $\overline{\ell'} \in \overline{N}$ is scanned such that $\ell' = h(q_j, \omega')$. If any symbol other than the appropriate $\overline{\ell'}$ is scanned immediately after the string of #'s, M' hangs up.

ix) Finally, M' advances tape head 3 past any number of #'s until symbol $\hat{q}_j$ is scanned on the third tape. M' hangs up if any symbol other than $\hat{q}_j$ appears immediately after the #'s.

To ensure that a computation by M' begins at a proper base configuration, we define the state transition function of M' in such a way that M' enters its control state from its initial state iff the tail of an initial configuration is of the form $s \cdot \overline{\ell} \cdot q_i \cdot q_i \cdot \omega$, where s is the ℓth symbol in ω, ℓ is $h(q_i, \omega)$ in M, and q_i is $f(q_I, \omega)$ in M.

To ensure that a computation by M' terminates successfully in appropriate cases, we add the following rules to the above:

x) If M' is in a configuration Ω such that TAIL(Ω) is of the form $\$ \cdot \# \cdot q_i \cdot q_i \cdot \$^n$, M' advances head 2 past any number of #'s until the symbol \$ is scanned; M' then advances head 3 past any number of symbols until # is scanned, and then head 4 past any number of symbols until # is scanned.

xi) If M' is in a configuration Ω such that TAIL(Ω) is of the form $\$\$ \cdot \#\# \cdot \$^n$, M' advances head 4 past any number of #'s to scan \$

immediately following the #'s, and then advances head 3 in a similar manner. If a symbol other than \$ appears immediately after the string of #'s in either case, M' hangs up.

The reader may verify that M' accepts the expansion of TRACE(M) and that the conditions governing moves in each of the above cases are unique (and thus that the behavior of M' as defined is consistent with the requirement that M' have a single control state).

□

We are now in a position to prove the main result of this section:

<u>Theorem 1</u>:

Let EMA denote the equivalence problem for multi-tape finite automata, and EMAS the equivalence problem for multi-tape automata with a single control state. Then EMA reduces to EMAS.

<u>Proof</u>: Let M and M' denote arbitrary n-tape automata with the same tape alphabet, for some $n \geq 1$. Let M_s and M_s' be the single-state acceptors for EXP(TRACE(M)) and EXP(TRACE(M)), respectively, constructed as in Lemma 3, and let K and K' be the tape alphabets of M_s and M_s'.

Let M_e be the n+8-tape automaton constructed from M_s as follows: The tape alphabet of M_e is $K \cup K'$. When started on a set of tapes, M_e will behave, with respect to its first n+4 tapes, exactly like M_s. When and if a configuration is reached in which endmarkers are scanned on each of its first n+4 tapes, M_e will advance in sequence the tape heads on each of its last four tapes, regardless of the symbols scanned, until endmarkers are scanned on these tapes also.

Clearly, $L(M_e) = \{(\omega_1, \omega_2, \ldots, \omega_{n+8}) \mid (\omega_1, \omega_2, \ldots, \omega_{n+4}) \in L(M_s)\}$, and M_e need have no more than a single control state.

In a similar manner, we may construct from M_s' a single-state n+8-tape automaton M_e' such that $L(M_e') = \{(\omega_1, \omega_2, \ldots, \omega_{n+8}) \mid (\omega_{n+5}, \omega_{n+6}, \ldots, \omega_{n+8}, \omega_5, \omega_6, \ldots, \omega_{n+4}) \in L(M_s')\}$.

We have that $L(M_e) = L(M_e')$ iff the set of "last n tapes" accepted by M_s and M_s' is the same, i.e. iff EXP(M) = EXP(M'). By Lemma 2, then, $L(M_e) = L(M_e')$ iff $L(M) = L(M')$, and hence EMA reduces to EMAS.

$\square$

II. Iteration Schemas

A (monadic) iteration schema is a seven-tuple

$$S = (L, L_I, L_O, \mathcal{F}, \mathcal{P}, \mathcal{S}, P)$$

where:

L is a finite set of location symbols.

L_I L is a finite set of input location symbols.

L_O L is a finite set of output location symbols.

$\mathcal{F}$ is a finite set of function letters.

$\mathcal{P}$ is a finite set of predicate letters.

$\mathcal{S}$ is a finite set of iteration schemas, the subschemas of S.

P is the program of S, a finite sequence of statements of the following types:

i) Assignment statement of the form

$L_i := f(L_j)$

or of the form

$L_i := L_j$

where L_i and L_j are location symbols and f is a function letter.

ii) Conditional statement of the form

IF $p(L_i)$ THEN S_t ELSE S_f

where p is a predicate letter, L_i is a location symbol, and S_t and S_f are subschemas.

iii) Iteration statement of the form

WHILE $p(L_j)$ DO S_i

or of the form

UNTIL $p(L_j)$ DO S_i

where p is a predicate letter, L_j is a location symbol, and S_i is a subschema.

If $S' = (L', L_I', L_O', \mathcal{F}', \mathcal{P}', \mathcal{S}', P')$ is a subschema of S, then we require that $L' \subset L$, $\mathcal{F}' \subset \mathcal{F}$, $\mathcal{P}' \subset \mathcal{P}$, and $\mathcal{S}' \subset \mathcal{S}$. Also, an iteration schema is not allowed to be recursive, i.e. it may not be a subschema of itself, nor may it have a subschema which is a subschema of itself.

Interpretations, Computations, and Equivalence

Let S be an iteration schema with input location symbols $L_I = \{L_{i_1}, \ldots, L_{i_k}\}$, predicate letters in a set of symbols $\mathcal{P}$, and function letters in a set of symbols $\mathcal{F}$. Then a (free) interpretation for S consists of:

i) The domain D of all strings in $\mathcal{F}^* \cdot K$, where K is a set of symbols $\{\Delta_{i_1}, \ldots, \Delta_{i_k}\}$, such that $K \cap \mathcal{F}$ is empty.

ii) For each j, $1 \le j \le k$, the association of the string Δ_{i_j} with the location denoted by L_{i_j}, and the association of ϵ with all other locations.

iii) The association, for each symbol f in $\mathcal{F}$, of the total function $\varphi_f : D \to D$ defined by $\varphi_f(\omega) = f \cdot \omega$.

iv) An association, for each symbol p in $\mathcal{P}$, of a total predicate $\Pi_p : D \to \{$true, false$\}$.

Each interpretation for a schema defines a computation by that schema in a straightforward way: we simply execute the statements in the program of the schema in the order in which they are encountered, associating appropriate domain individuals with locations as required by assignment statements, and executing the programs of subschemas in accordance with the values of predicates applied during the execution of conditional or iteration statements. When an assignment instruction of the form $L_i := L_j$ is executed, the element of the domain currently associated with L_j is associated with L_i; when an assignment statement of the form $L_i := f(L_j)$ is executed, the value $\varphi_f(\omega)$ is associated with L_i, where ω is the element of the domain associated with L_j at the time of execution. When a conditional statement of the form IF $p(L_i)$ THEN S_t ELSE S_f is executed, the predicate Π_p is evaluated at the element of the domain currently associated with L_i, and the program of S_t or of S_f is executed, according as the outcome of the evaluation is true or false; when the execution of the appropriate program is completed, execution of the main program resumes with the statement immediately following the conditional. Finally, when an iteration statement of the form WHILE(UNTIL) $p(L_i)$ DO S_i is executed, the predicate Π_p is evaluated at the current value of L_i and the program of S_i is executed if the outcome of the evaluation is <u>true</u> (<u>false</u>); upon completion of the program's execution, execution of the main program resumes with the iteration statement. This cycle is repeated until an evaluation of Π_p has outcome <u>false</u> (<u>true</u>), at which point execution proceeds to the statement following the iteration statement. The computation by the schema is complete when and if all statements in the program have been executed.

The <u>value</u> of a computation by a schema S defined by an interpretation I for S (written VAL(S,I)) is the sequence of values associated with the output locations of S when the computation defined by I is complete, and is undefined if the computation fails to terminate.

Let S and S' be iteration schemas with the same set of input location symbols and the same set of output location symbols. Then S and S' are equivalent iff for each interpretation I for S and S', VAL(S, I) = VAL(S', I).

Independent and Restricted Location Schemas

An iteration schema S is an independent location schema if each assignment statement in the program of S is of the form $L_i := f(L_i)$ for some location symbol L_i and some function letter f. A schema S is a restricted location schema if each of its subschemas is an independent location schema (though S itself need not be).

A Note on Notation

For notational convenience, we shall permit an arbitrary boolean expression to appear in conditional statements of an iteration schema, where the set of boolean expressions for a schema with predicate letters $\mathcal{P}$ and location symbols L is defined recursively as follows:

i) For each letter p in $\mathcal{P}$ and location symbol L_i in L, $p(L_i)$ is a boolean expression.

ii) If b is a boolean expression, then $(\neg b)$, "not b", is a boolean expression.

iii) If b and b' are boolean expressions, then $(b \wedge b')$, "b and b'", and $(b \vee b')$, "b or b'", are boolean expressions.

We note that IF $(\neg b)$ THEN S_t ELSE S_f is equivalent to IF b THEN S_f ELSE S_t, and that IF $(b \wedge b')$ THEN S_t ELSE S_f is equivalent to IF b THEN S_t' ELSE S_f where S_t' has program IF b' THEN S_t ELSE S_f. We are assured, therefore, that permitting boolean expressions in conditionals is, indeed, merely a notational convenience and does not alter at all the class of programs representable by iteration schemas.

Also for notational convenience, we will in general define schemas in terms of their programs and the programs of their subschemas only, the function and predicate letters and location symbols of the schema being defined implicitly as those appearing in the programs. In many instances, in fact, we will not bother to distinguish between a schema and its program, referring to either as "schema".

Equivalence Problems.

The equivalence problem for iteration schemas is easily shown to be unsolvable, as is the equivalence problem for restricted location schemas [3]. The equivalence problem for independent location iteration schemas (which we shall refer to as EIL) is, however, open. In the next section, we show that a positive solution to this problem implies a positive solution to the equivalence problem for multi-tape finite automata.

III. A Second Reducibility

In this section we demonstrate that EMAS reduces to EIL, a result which does not follow directly from any of the results in the literature. In particular, while Luckham, Park, and Paterson [2] have demonstrated that any multi-tape finite automaton may be simulated by an independent location program schema, and Ashcroft and Manna [1] have demonstrated that any such schema is equivalent to some iteration schema, the equivalent iteration schema will not, in general, be independent location. (In fact, it is easy to show that there exist very simple single location program schemas which are not equivalent to any independent location iteration schema.)

Let $M = (\{s_1,s_2,\ldots,s_k,\$\}, \{q\}, q_I, f, h)$ be an arbitrary n-tape single state automaton in normal form, for some $n > 0$. We show how to construct an independent location iteration schema S_M which converges for an interpretation if and only if that interpretation defines, in a reasonable manner, a set of tapes accepted by M:

Let $\omega_1, \omega_2, \ldots, \omega_{(k+1)^n}$ be an enumeration of the strings of length n over the alphabet of M, and let $\omega_{i_1}, \ldots, \omega_{i_m}$ be an enumeration of those strings for which $f(q_I, \omega_{i_j}) = q$, $1 \le j \le m$.

S_M will have location symbols $L_1,\ldots,L_{n+1}$, predicate symbols $p_1,\ldots,p_k$, $p_\$$, and p_M, and a single function letter f. S_M will have, among others, the subschemas S_e whose program contains no statements, and S_d whose program is

```
WHILE $p_M(L_1)$ DO $S_e$
UNTIL $p_M(L_1)$ DO $S_e$
```

and thus diverges for any interpretation.

For notational convenience, we provide simple representation for certain frequently used boolean expressions:

i) For each i, $1 \le i \le k$, and each j, $1 \le j \le n$, we represent by $P_i(L_j)$ the expression $p_i(L_j) \wedge (\neg(p_1(L_j) \vee p_2(L_j) \vee \ldots \vee p_{i-1}(L_j) \vee p_{i+1}(L_j) \vee \ldots \vee p_k(L_j) \vee p_\$(L_j))$.

ii) Similarly, for each j, $1 \le j \le n$, we represent by $P_\$(L_j)$ the expression $p_\$(L_j) \wedge (\neg(p_1(L_j) \vee p_2(L_j) \vee \ldots \vee p_k(L_j))$.

iii) For each j, $1 \le j \le n$, we represent by $\underline{\text{REAS}}(L_j)$ the expression $P_1(L_j) \vee P_2(L_j) \vee \ldots \vee P_k(L_j) \vee P_\$(L_j)$.

iv) We represent by $\underline{\text{REASONABLE}}$ the expression $\underline{\text{REAS}}(L_1) \wedge \ldots \wedge \underline{\text{REAS}}(L_n)$.

v) Finally, for each j, $1 \le j \le (k+1)^n$, we represent by $P\omega_j$ the expression $P_{j_1}(L_1) \wedge P_{j_2}(L_2) \wedge \ldots \wedge P_{j_n}(L_n)$, where $s_{j_1} \cdot s_{j_2} \cdot \ldots \cdot s_{j_n} = \omega_j$.

We now define the remaining subschemas of S_M:

i) For each i, $1 \le i \le (k+1)^n$, the subschema S_i is

$L_{h(q,\ \omega_i)} := f(L_{h(q,\ \omega_i)})$

IF $\underline{\text{REAS}}(L_{h(q,\ \omega_i)})$ THEN S_e ELSE S_d

if $h(q, \omega_i)$ is defined, and is S_d otherwise.

ii) For each i, $1 \le i < (k+1)^n$, the subschema $\overline{S}_i$ is

IF $P_{\omega_{i+1}}$ THEN S_{i+1} ELSE $\overline{S}_{i+1}$

iii) The subschema $\overline{S}_{(k+1)^n}$ is the schema S_d.

iv) The subschema S_0 is

```
    IF $p_M(L_{n+1})$ THEN $S_1$ ELSE $S_d$
    $L_{n+1} := f(L_{n+1})$
```

The schema S_M is then defined as follows:

```
    IF REASONABLE THEN $S_e$ ELSE $S_d$
    IF $(P\,\omega_{i_1} \vee P\,\omega_{i_2} \vee \ldots \vee P\,\omega_{i_m})$ THEN $S_e$ ELSE $S_d$
    WHILE $p_M(L_{n+1})$ DO $S_0$
    IF$(P_\$(L_1) \wedge P_\$(L_2) \wedge \ldots \wedge P_\$(L_n))$ THEN $S_e$ ELSE $S_d$
```

Let I be a free interpretation for S_M and for each integer j, $1 \le j \le n$, let I_j be the least integer ≥ 0 such that $\Pi_{P_\$}(f^{I_j} \cdot \Delta_j)$ is true. The interpretation I is a reasonable interpretation if REASONABLE is a tautology when restricted to the domain $\{f^m \cdot \Delta_i \mid 1 \le i \le n,\ 0 \le m \le I_i\}$, and $\Pi_{P_M}(f^m \cdot \Delta_{n+1})$ is true for all $m < \sum_{i=1}^{n} (I_i - 1)$ and is false for $m = \sum_{i=1}^{n} (I_i - 1)$.

Each reasonable interpretation I for S_M defines an n-tuple Ω_I of strings over the alphabet of M in a straightforward manner - the ith symbol in the jth string, $1 \le j \le n$, $1 \le i \le I_j$, is s_ℓ if and only if $\Pi_{p_\ell}(f^{i-1} \cdot \Delta_j)$ is true.

The reader may verify that S_M is guaranteed to diverge for all unreasonable interpretations, and converges for the reasonable interpretation I if and only if Ω_I is accepted by M. We thus have:

<u>Theorem 2</u>:

EMAS reduces to EIL.

<u>Proof</u>: Let M and M' be arbitrary n-tape single-state, normal form automata. We may extend, if necessary, their tape alphabets to ensure that the alphabets are the same in each case. Let S_M and $S_M{}'$ be the independent location iteration schemas constructed from M and M', respectively, as above. From the last remark we have that M and M' are equivalent if and only if S_M is equivalent to $S_M{}'$.

□

<u>Corollary 2.1</u>:

EMA reduces to EIL.

<u>Proof</u>: Theorems 1 and 2.

□

IV. Discussion.

It should be noted that the construction outlined in the previous section is possible only because the automata to be simulated have a single control state. In particular, if one attempts to apply the method of Theorem 2 to a multi-state machine, adddditional predicates, corresponding to the states of the machine, would be required. Given a pair of such machines, the simulating schemas would, in general, be able to diverge independently for interpretations which were "unreasonable" with respect to the predicates corresponding to states, rather than those corresponding to tape symbols. Thus, unless the original machines were essentially identical, the simulating schemas would not be equivalent. It does not seem, therefore, that Corollary 2.1 can be obtained independent of Theorem 1.

It is easy to show that free[†] independent location program schemas are capable of simulating normal form multi-tape automata, and that multi-tape automata in general are capable of simulating non-free independent location program schemas. From Corollary 1.1, therefore, we have (as noted in [2]) that the equivalence problem for independent location program schemas reduces to that for free independent location program schemas. It is open whether or not a similar reducibility result can be shown for independent location iteration schemas. In particular, can single state automata be simulated by normal form multi-state automata which are "block structured" in the sense of [1]? If so, it would then be possible to show that the equivalence problem for independent location iteration schemas reduces to that for free independent location iteration schemas.

[†] A schema is free if no predicate is ever applied twice to the same value during the computation defined by any free interpretation for the schema.

It should be noted, in any case, that the schema S_M described in Section III fails to be free for reasons other than the inclusion of the schema S_d in the set of subschemas for S_M - in particular, it should be kept in mind that the symbols of the form P_{ω_i} do not represent distinct predicates for each i, but merely distinct boolean expressions over a common set of predicates.

It can be shown that if boolean expressions are permitted in iteration statements as well as conditional statements, an independent location iteration schema can be found which is equivalent to schema S_M and is <u>conditional-free</u>, i.e. free of conditional statements in its program and the programs of its subschemas. It is not known whether such a schema can be found if general boolean expressions are not permitted in iteration statements, but the question is of some interest since equivalence problems for such schemas are likely to be somewhat easier to solve than those for independent location schemas in general. (In particular, equivalence can be shown decidable for such schemas which are free.)

REFERENCES

1. Ashcroft, E., and Manna, Z.
The Translation of 'GoTo' Programs to 'While' Programs.
Information Processing 71, North-Holland Publishing Co., 1972.

2. Luckham, D., Park, D., and Paterson, M.S.
On Formalised Computer Programs.
J. of Computer and Systems Sciences, Vol.4., No.3, 1970.

3. Leung, C., and Qualitz, J.E.
Undecidability of Equivalence for Restricted Location Schemas.
Computation Structures Group Note 21, MIT, Feb. 1975.

4. Rabin, M., and Scott, D.
Finite Automata and Their Decision Problems.
IBM Journal of Research and Development, 3,2 1959.