SIMPLE PROGRAMS AND THEIR DECISION PROBLEMS

A. PNUELI
University of Tel-Aviv
Tel-Aviv, Israel

G. SLUTZKI
University of Tel-Aviv
Tel-Aviv, Israel

Introduction

In [1] the authors gave several program-automaton models. To each of these models a suitable description language was defined and it was shown that the problems of Analysis, Synthesis, Equivalence and Optimization (i.e. the Automatic Programming problems, or AP problems) are all algorithmically solvable.

This paper is divided into two parts. The first part, mainly expository, will describe two of the models dealt with in [1]. Main results will be stated without proofs and some motivation for studying these models will be given. The second part will treat some decision problems concerning the possible extensions of the two models described in Part 1. Some rough borderlines will be drawn beyond which algorithmic treatment of the AP problems is no longer possible. Also, few open problems are proposed.

Part 1

a. The FSLP (Finite State Linear Program) Model

A program $P \epsilon FSLP$ is essentially a finite state automaton to which the operations of addition, subtraction, and multiplication by a constant, all manipulating elements of some fixed field R (of characteristic 0), have been added. P is equipped with a finite set of registers of type R, $Z_1, \ldots, Z_n$, each of which can accomodate an arbitrary element of R. At each step of its operation the program reads a record from its input tape. The record consists of an alphabetic field of a single letter out of some finite alphabet Σ, and a numeric field containing an arbitrary element $y \epsilon R$. Thus the input to an FSLP program is a tape of pairs $(\sigma_1, y_1) \ldots (\sigma_k, y_k)$ with $\sigma_i \epsilon \Sigma$, $y_i \epsilon R$. The set of all such tapes is denoted by $(\Sigma \times R)^*$. The program uses the $\sigma \epsilon \Sigma$ to determine its next state (or move) in the program, including a set of arithmetical operations to be performed. These arithmetical operations are of the form of a general linear transformation on the vector of registers, $\overline{Z}$, and the current input value y:

$$\overline{Z} \leftarrow \overline{Z} \cdot M + \overline{\eta} \cdot y \qquad (1)$$

where $\overline{Z}$ is the (row) vector of registers, M and $\overline{\eta}$ are arbitrary R-matrix and R-vector respectively (having suitable dimensions), and y is the current numerical input at

this step.

Thus P∈FSLP is still a finite state device in the sense that its next state is determined by its current state and an input value out of a finite alphabet Σ. Note that there is no test instruction on the value of the registers so that the R-arithmetic performed by P does not influence P's control flow. Fig. 1 gives an example of P∈FSLP which utilizes a single register, q_1 is its initial state and q_3 the final state. The initial value of the register is $Z = A(q_1) = 2$. If P ends processing its input in state q_3 with Z having the value C∈R, then this value is multiplied by $V(q_3)$ to yield the final output. For example, P when fed with the input tape $(\sigma_4,1)(\sigma_5,4)(\sigma_1,30)(\sigma_3,10)$ outputs the value 20.

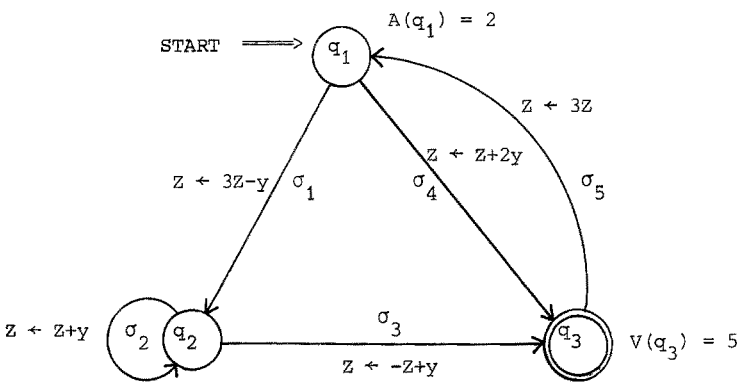

Fig. 1

The motivation for studying the FSLP model and its extensions is (at least) twofold. First we obtain a considerable extension of the concept of K-Σ-Automata and the results concerning it, as given in Eilenberg [2]. Second consideration, not less important, is that FSLP programs may be viewed as (although very primitive) model of data processing device, where the input pair (σ,y) could represent a record of alphanumeric information to be processed. This second aspect, stressed strongly in [1], represents the authors' belief in that parallel to the heuristic approach to the AP problems, it is worthwhile to start at the bottom and study classes of simple programs for which the AP problems are algorithmically solvable.

We come now to the formal definitions.

Definition 1: An FSLP program P (over R) is a system
$P = \langle K,\Sigma,\overline{Z},\delta,T,U,B,A,F,V\rangle$ where $\langle K,\Sigma,\delta,B,F\rangle$ is a finite state (nondeterministic) automaton with:

1) $K = \{q_1,\ldots,q_m\}$, $\Sigma = \{\sigma_1,\ldots,\sigma_t\}$, the states and input alphabet respectively.

2) $B \subseteq K$ subset of initial states; $F \subseteq K$ subset of final states.

3) $\delta : K \times \Sigma \to 2^K$ a (non-deterministic) transfer function; δ is extended to Σ^* and B in the usual manner.

$\langle K,\Sigma,\delta,B,F\rangle$ is the underlying control structure of P. P's computation structure consists of:

4) $\overline{Z} = (Z_1,\ldots,Z_n)$ (row) vector of registers.

5) $T : K \times \Sigma \times K \to R_{n,n}$ is a mapping from possible moves in the program to n × n matrices over R. $T(q,\sigma,q')$ is the matrix M from (1) corrsponding to the edge (q,σ,q').

6) $U : K \times \Sigma \times K \to R^n$ is a mapping from possible moves in the program to (row) n-vectors over R. $U(q,\sigma,q')$ is the vector $\overline{\eta}$ from (1) corresponding to the edge (q,σ,q').

7) $A : B \to R^n$ assigns an initial value to the register vector at state $q \in B$. For $q \notin B$ we agree to have $A(q) = 0$ (all zero arrays will be denoted by 0).

8) $V : F \to R^n$ assigns a (columm) vector in R^n to each accepting state. For $q \notin F$, $V(q) = \omega$ (ω is the undefined value; all undefined arrays will be denoted by ω).

The computation function $Val_P : (\Sigma \times R)^* \times K \to R^n$, for $P \in FSLP$, is a vector valued function such that $Val_P(w,q)$ expresses the current value of the register vector $\overline{Z}$ at state q, after reading the input $w \in (\Sigma \times R)^*$. The recursive definition of Val_P is:

$$\left.\begin{aligned} Val_P(\Lambda,q_j) &= A(q_j) \\ Val_P(w\cdot(\sigma,y),\ q_j) &= \sum_{q_k \in \delta(B,\lambda(w))} [Val_P(w,q_k)\cdot T(q_k,\sigma,q_j) + U(q_k,\sigma,q_j)\cdot y] \end{aligned}\right\} \quad (2)$$

where for $w = (\sigma_1,y_1)\ldots(\sigma_s,y_s)$, $\lambda(w) = \sigma_1\ldots\sigma_s$ is the Σ-component of w. The (partial) output function $Out_P : (\Sigma \times R)^* \to R$, for $P \in FSLP$, is defined by:

$$Out_P(w) \equiv \sum_{q \in F \cap \delta(B,\lambda(w))} Val_P(w,q)\cdot V(q) \quad (3)$$

where the relation '$\equiv$' means that either both sides are defined and equal or both sides are undefined. It is stipulated that if the sum in (3) is empty then $Out_P(w)$ is undefined.

Special cases of the general FSLP model are of interest.

(1) $P \in FSLP$ is <u>deterministic</u> if $|B| = 1$ (i.e. P has a single initial state), and for every $q \in K$, $\sigma \in \Sigma$, $|\delta(q,\sigma)| \leq 1$.

(2) If for every edge $e = (q,\sigma,q') \in K \times \Sigma \times K$, $U(e) = 0$, we obtain a <u>Multiplicative Automaton</u> (MA). In MA the y-inputs can be ignored.

(3) If P is an MA and $\dim \overline{Z} = 1$ (i.e. P utilizes a single register) then we have a <u>Scalar MA</u>. Scalar MA is an R - Σ - Automaton in the terminology of [2].

b. The FRLP (Finite Relational Linear Program) Model

An FRLP program recognizes a single numerical data type, namely the set R (elements of which were second components of the input pairs (σ,y), to FSLP programs). Thus an input to an FRLP program is a string from R^*. Programs in FRLP manipulate a finite set of registers of type R exactly as in FSLP. The crucial generalization is that the control flow of the program depends on the numerical input and thus the issues of computation and control are no longer separated as in the FSLP model. The way in which this dependence is effected is by testing, at each state, the input value $y \in R$. Each program has a finite number of tests (effective, non-vacuous, unary predicates) which label the edges of the program in much the same way as the elements of Σ label the edges of an FSLP program (see Fig. 1). Passage through an edge is permitted only if the current input y satisfies the test labeling that edge. In the sequel we assume that if a test σ is such that there exists a unique $c_o \in R$ which satisfies it (i.e. $\sigma = \{c_o\}$) then σ is the equality test "$y = c_o$". In Fig. 2 is an example of an FRLP program. Note that a program in FRLP could be intuitively viewed as an "interpreted FSLP" program, where the elements of Σ (in FSLP) have been interpreted as tests (compare the programs of Figs. 1 and 2).

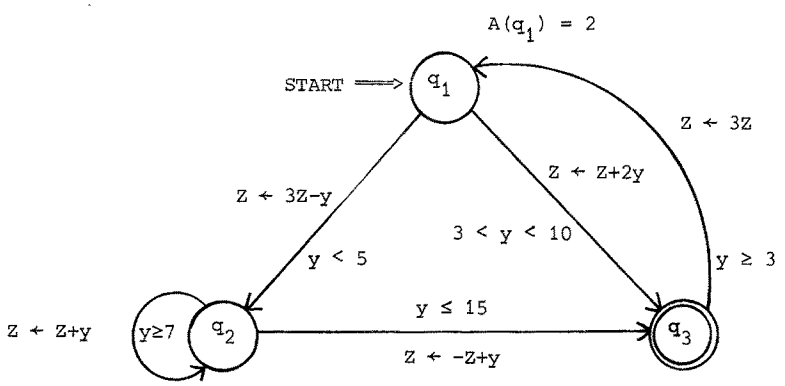

Fig. 2

We can now define formally:

Definition 2: An FRLP program P (over R) is a system $P = \langle K,\Sigma,\overline{Z},\nu,T,U,B,A,F,V\rangle$ where $\Sigma = \{\sigma_1,\ldots,\sigma_t\}$ is the set of tests labeling the edges of P, $\nu : K \times \Sigma \to 2^K$ is a (non-deterministic) function such that $\nu(q,\sigma)$ is the set of states reachable from q by edges labeled by the test σ, and the other components of P are as in Definition 1.

Definition 3: A function $g : R \to 2^\Sigma$ is defined by $g(y) = \{\sigma \in \Sigma \mid y \text{ satisfies } \sigma\}$.

g is extended to R^* by $g(y_1 \ldots y_k) = \{\rho_1 \ldots \rho_k \mid \rho_i \in \Sigma \ , \ \rho_i \in g(y_i)\}$.

Definition 4: Let P∈FRLP be given as in Definition 2. The transfer function for P, $\delta : K \times R \to 2^K$ is defined by $\delta(q,y) = \bigcup_{\sigma \in g(y)} \nu(q,\sigma)$. δ is extended to R^* and B in the usual way.

Definition 5: P∈FRLP is deterministic if $|B| = 1$ and for each $q \in K$, $y \in R$, $|\delta(q,y)| \leq 1$.

The computation function $Val_P : R^* \times K \to R^n$ is defined recursively:

$$\left.\begin{aligned} Val_P(\Lambda,q_j) &= A(q_j) \\ Val_P(w{\cdot}y,q_j) &= \sum_{q_k \in \delta(B,w)} \ \sum_{\sigma \in g(y)} [Val_P(w,q_k){\cdot}T(q_k,\sigma,q_j) + U(q_k,\sigma,q_j){\cdot}y] \end{aligned}\right\} \quad (4)$$

and the (partial) output function $Out_P : R^* \to R$ is:

$$Out_P(w) \equiv \sum_{q \in F \cap \delta(B,w)} Val_P(w,q){\cdot}V(q) \quad (5)$$

For technical reasons it is desirable that for edges on which equality is tested the U-vectors should be null. This can be achieved easily by adding an auxiliary register Z_c whose value is permanently 1, and noting that the value of the y-input on such an edge is actually known. Let the programs which satisfy this property be called Constant-free (C-free for short). In the sequel we assume that all programs are C-free unless stated to the contrary.
Let $\Sigma = \{\sigma_1, \ldots, \sigma_t\}$ be the set of tests of P∈FRLP.

Definition 6: The set of basic tests w.r.t. Σ is the set of tests $\Delta = \{\rho_1, \ldots, \rho_s\}$ satisfying 1) for all $1 \leq i,j \leq s$ $\rho_i \cap \rho_j = \phi$, i.e. there exists no $y \in R$ which satisfies both ρ_i and ρ_j; 2) each $\sigma \in \Sigma$ is a union of some elements of Δ.
The actual construction of Δ from Σ is very easy. It should also be obvious how to rewrite P in terms of Δ. The resulting program, if not C-free should be "C-freed".

Definition 7: P∈FRLP is basic FRLP (BFRLP) if 1) its set of tests is basic 2) it is C-free.
Note that for P∈BFRLP the second equation in (4) can be simplified to:

$$Val_P(wy,q_j) = \sum_{q_k \in \delta(B,w)} [Val_P(w,q_k){\cdot}T(q_k,g(y),q_j) + U(q_k,g(y),q_j){\cdot}y] \quad (6)$$

It is then easily proved that every P∈FRLP can be transformed into an equivalent basic version.

Let $x = \sigma_1 \ldots \sigma_k \in \Sigma^*$ and $w = y_1 \ldots y_k \in R^*$. For such x and w we define an element $x * w = (\sigma_1,y_1) \ldots (\sigma_k,y_k) \in (\Sigma \times R)^*$.

Definition 8: Let P∈FRLP. We define $\hat{P}$∈FSLP, called the companion of P. Formally $\hat{P}$

has the same components as P. $\hat{P}$'s edges are labeled by elements of Σ and its input tapes are from $(\Sigma \times R)^*$ as usual. $\hat{P}$'s transfer function is ν (see Definition 2). Thus $\hat{P}$ has been obtained from P by just viewing P's tests as formal symbols forming the first components of the input pairs to FSLP programs.

For example the program of Fig. 1 becomes the companion of the program of Fig. 2 if instead of $\sigma_1,\ldots,\sigma_5$ we put the formal symbols: "$y < 5$", "$y \geq 7$", "$y \leq 15$", "$3 < y < 10$" and "$y \geq 3$" respectively. Note that in an input pair (σ,y) to $\hat{P}$ y need not satisfy the test σ. Nevertheless we have the following theorem which is fundamental in the treatment of FRLP programs.

Theorem 1: Let $P \epsilon BFRLP$ and let $\hat{P} \epsilon FSLP$ be its companion. Then for every $w \in R^*$:

$$Out_P(w) \equiv Out_{\hat{P}}(g(w) \ * \ w) \tag{7}$$

If $g(w) = \phi$ then both sides of (7) are undefined.

c. The Description Languages

The Description Language for FSLP programs is GAE (Generalized Amplification Expressions). Very roughly, GAE's involve regular expressions over $\Sigma \cup R$ subject to some syntactical restrictions.

The Description Language for FRLP programs is GRAE (Generalized Relational AE). GRAE is defined in terms of GAE.

Because the description languages are involved only in the issues of Analysis and Synthesis (whose formulations and solutions so resemble the analogous problems and solutions in the standard finite state automata theory) we purposedly refrain from giving any details and refer the interested reader to [1].

d. Main Results (for proofs see [1]).

The results for the two models will be stated in as much as possible parallel fashion.

Theorem 2: For every $P \epsilon FSLP$ ($P \epsilon FRLP$) there exists an equivalent and deterministic $P' \epsilon FSLP$ ($P' \epsilon FRLP$) [equivalence means that for every input w $Out_P(w) \equiv Out_{P'}(w)$; the equivalence relation between programs is denoted by "$\simeq$"]. P' may have more registers and states than P.

Theorem 3: For every $P \epsilon FSLP$ ($P \epsilon FRLP$) there exists an equivalent $P' \epsilon FSLP$ ($P' \epsilon$ FRLP) which utilizes a single register. P' is possibly non-deterministic.

These two theorems express the tradeoff relationship between determinism, number of states and number of registers.

Theorem 4: (Analysis Theorem): For each P∈FSLP (P∈FRLP) there exists an algorithmically constructible GAE(GRAE) t such that:

$$Out_P(w) \equiv \{t,w\} \tag{8}$$

for all $w \in (\Sigma \times R)^*$ $(w \in R^*)$. $\{t,w\}$ denotes the GAE (GRAE) valuation function.

Theorem 5: (Synthesis Theorem): For every GAE (GRAE) t there exists an algorithmically constructible P∈FSLP (P∈FRLP) such that (8) holds.

Definition 9: The Zero Computation Problem (ZCP) is to determine, for a given P∈FSLP (P∈FRLP) whether it outputs zero for each input for which it is defined.

Lemma 1: Let P∈BFRLP and let $\hat{P}$∈FSLP be its companion. Then P is ZC (Zero Computing) iff $\hat{P}$ is ZC.

Lemma 1 obviously reduces the ZCP for FRLP to that for FSLP. Since the latter is solvable (see [1]) we have:

Theorem 6: The ZCP is algorithmically solvable for FSLP (FRLP).

Also, since the Equivalence Problem (EP) is easily reducible to ZCP we have:

Theorem 7: The EP is algorithmically solvable for FSLP (FRLP). Moreover, if $P,Q \in$ BFRLP are deterministic over the same set of basic tests, then $P \simeq Q$ iff $\hat{P} \simeq \hat{Q}$.

Passing now to optimization problems we remark that only deterministic programs are treated. The reason is that state optimization, even in the case of non-deterministic finite state automata is not feasibly solvable, while register optimization received a trivial solution in Theorem 3.

Definition 10: For a given P∈FSLP (P∈FRLP) and a state q of P let P_q be P modified by taking q to be the only initial state of P_q with initial value A(q) = 0. Two states q_1, q_2 of P are defined to be equivalent, $q_1 \tilde{\simeq} q_2$, if $P_{q_1} \simeq P_{q_2}$.

The algorithmic solution of EP enables us to effectively partition the set of states of a program into equivalence classes.

Theorem 8: Let N be the number of different equivalence classes of states of P∈FSLP (P∈FRLP). Then there exists P'∈FSLP (P'∈FRLP) which is equivalent to P, and P' has N states. P' is minimal state version of P.

Almost all program constructions mentioned so far considerably increase the

the number of registers of the resulting programs. To alleviate this penalty we have:

Theorem 9: For any PϵFSLP (PϵFRLP) there exists an equivalent P'ϵFSLP (P'ϵFRLP) such that P and P' have identical control structure and P' utilizes a minimal number of registers. In other words, among all FSLP (FRLP) programs equivalent to P and having the same underlying control structure as P, P' has the minimal number of registers.

This completes the survey of the PSLP and FRLP and the relevant results as given in [1].

Part 2

It would be most satisfactory if we could deal algorithmically with "FRLP-like" programs which allow to test the values of the registers. Unfortunately, it is quite easy to see that unconstrainted use of tests on the registers makes our programs equipotent with Turing Machines. A way which is often taken in such cases is to try to clarify the question of freedom. To be more specific let us assume that we are dealing with Generalized FRLP (GFRLP) which extend the FRLP model by allowing tests on registers.

Definition 10: PϵGFRLP is free, if for every path in P, which starts at some initial state, there exists an input $w \in R^*$ such that P's computation on w can proceed along this path.

As the basic, established unsolvability result we use the Post Correspondence Problem (PCP). Let $\Delta = \{\sigma_1,\ldots,\sigma_m\}$, $m \geq 2$. An instance of PCP is a pair $D^n_{\alpha\beta} = \{<\alpha_1,\ldots,\alpha_n>, <\beta_1,\ldots,\beta_n>\}$, $n \geq 1$, of n-tuples of non-empty words over Δ. We say that $D^n_{\alpha\beta}$ has a solution if there exists a sequence of indices $i_1,\ldots,i_k$ such that $\alpha_{i_1}\ldots\alpha_{i_k} = \beta_{i_1}\ldots\beta_{i_k}$.

Let now Δ and $D^n_{\alpha\beta}$ be as above. Let $M = m + 1$. Coding σ_i by i, every word over Δ can be uniquely coded as a number in R. For example if $\alpha_i = \sigma_{i_1}\ldots\sigma_{i_k}$ then its code number is $i_1M^{k-1} + \ldots + i_{k-1}M + i_k$. Let $c^{\alpha}_i(c^{\beta}_i)$ be the code number of $\alpha_i(\beta_i)$, and by $\ell^{\alpha}_i(\ell^{\beta}_i)$ denote the length of $\alpha_i(\beta_i)$, $1 \leq i \leq n$. Partition R into n disjoint, non-empty ranges $R = R_1 \cup \ldots \cup R_n$.

The programs which we shall construct will utilize three registers. Z_α will accumulate the code of the α_i's. Z_β does the same for the β_i's. Z_o is an auxiliary register whose value is permanently 1. For brevity let us denote by A_i the assignment:

$$\left.\begin{array}{l} Z_\alpha \leftarrow M_i^{\ell_i^\alpha} \cdot Z_\alpha + C_i \cdot Z_o \\ Z_\beta \leftarrow M_i^{\ell_i^\beta} \cdot Z_\beta + C_i \cdot Z_o \\ Z_o \leftarrow Z_o \end{array}\right\} \qquad 1 \le i \le n$$

Theorem 10: The Freedom Problem for GFRLP is undecidable.
Proof: The program $P^n_{\alpha\beta}$ of Fig. 3 is free iff $D^n_{\alpha\beta}$ has no solution.

$P^n_{\alpha\beta}$:

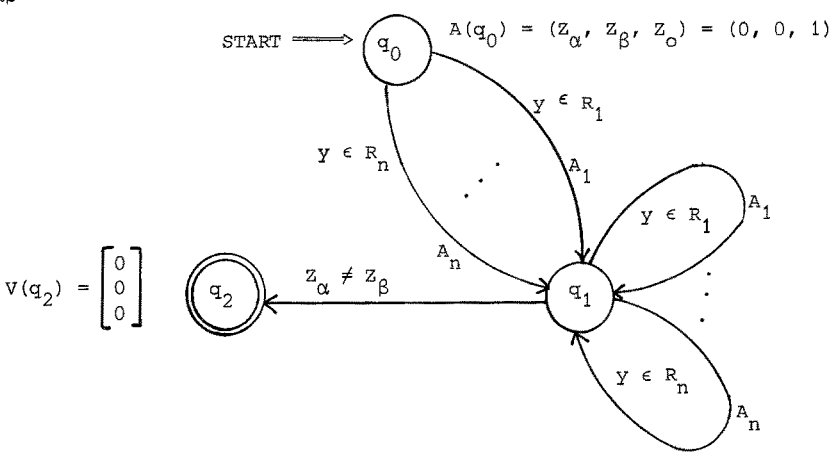

Fig. 3

By adding to $P^n_{\alpha\beta}$ an additional final state q_3, with $V(q_3) = (0, 0, 1)^T$, and an additional edge from q_1 to q_3 labeled by the test $Z_\alpha = Z_\beta$ we obtain a program $Q^n_{\alpha\beta} \in$ GFRLP.

Corollary 1: The ZCP (see Definition 9 and extend it to any class of programs) for GFRLP is undecidable.
Proof: $D^n_{\alpha\beta}$ has a solution iff $Q^n_{\alpha\beta}$ is not ZC.

Definition 11: A program (of any class) is said to satisfy the Sometimes Zero Property (SZP) if there exists an input for which the program outputs zero.

Theorem 11: The SZP is undecidable for FSLP.
Proof: The program $\overline{P}^n_{\alpha\beta}$ of Fig. 4 has SZP iff $D^n_{\alpha\beta}$ has a solution.

$\bar{P}^n_{\alpha\beta}$:

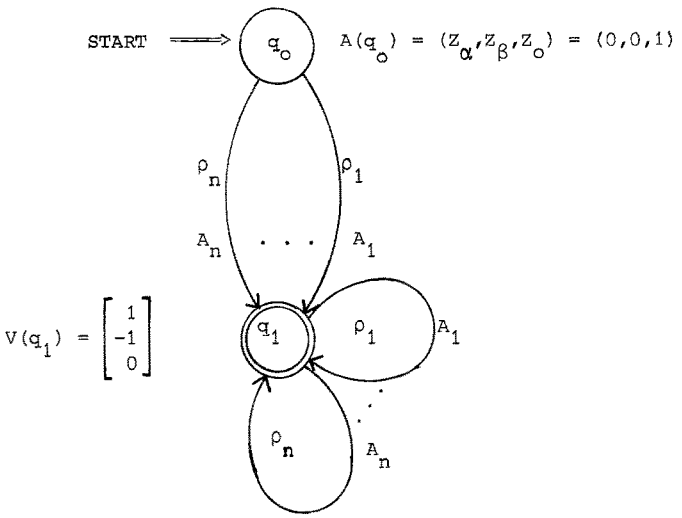

Fig. 4

Definition 12: The Intersection Problem (IP), for some class of programs is to decide, given P_1, P_2 in that class, whether or not there exists an input for which P_1 and P_2 output the same value.

Corollary 2: The IP for FSLP is undecidable.
Proof: Easily follows from Theorem 11.

Since the class GFRLP is so hopeless let us consider a subclass of GFRLP, called ξFRLP, whose programs are guaranteed to be free. ξFRLP extends the FRLP model in that the values of the registers are allowed to be tested in a "roundabout" way. In fact we require that the tests have to be applied to arguments of the form $\bar{Z} \cdot \bar{\xi} + y$, where $\bar{Z}$ is the current value of the register vector, $\bar{\xi}$ is some column vector, and y is the current input value. Each edge of a ξFRLP program may have a different $\bar{\xi}$-vector associated with it. Clearly, the presence of the y's in the arguments makes our programs free.

Definition 13: The Determinism Problem (DP), for some class of programs, is to decide whether a given P in the class is determistic, i.e. for every input there exists at most one computation path induced by that input.

Theorem 12: The DP for ξFRLP is undecidable.
Proof: Notation remains the same except that R is partitioned into n + 1 ranges, $R = R_1 \cup \ldots \cup R_n \cup R_{n+1}$. The program $\bar{Q}^n_{\alpha\beta}$ of Fig. 5 is nondeterministic iff $D^n_{\alpha\beta}$

has a solution.

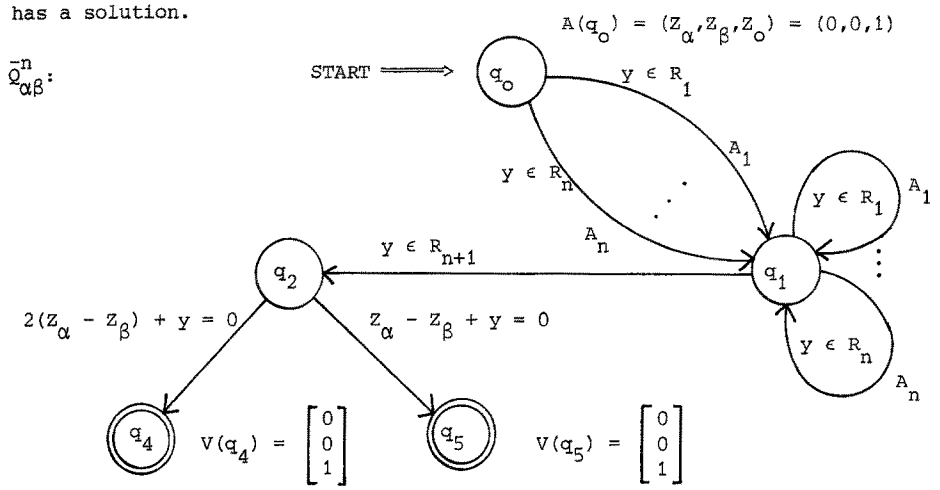

Fig. 5

Assume that we are given the class of Deterministic ξFRLP (DξFRLP). The following holds (see [1]):

Theorem 13: The ZCP for DξFRLP is decidable.

We conclude by proposing two, so far, unsettled problems.

Open Problem 1: Is the Equivalence Problem for DξFRLP decidable?

Open Problem 2: Is the ZCP for ξFRLP decidable?

REFERENCES.

[1] Slutzki, G. - Logical Analysis of Simple Programs, Doctoral Dissertation, Tel-Aviv University (1976).

[2] Eilenberg, S. - Automata, Languages and Machines, Vol. A, Academic Press (1974).