Decidability of Equivalence for a Class of Data Flow Schemas

Joseph E. Qualitz

March 1975

This work was supported by the National Science Foundation
under research grant GJ-34671.

This document was originally published as
Computation Structures Group Memo 100-1,
November 1974.

In this paper we examine a class of computation schemas and consider the problem of deciding when pairs of elements in this class represent equivalent programs. We are able to show that equivalence is decidable for a non-trivial class of unary operator data flow schemas, and consider the applicability of this result to the problem of deciding equivalence in related models of computation.

The model described below is a restricted version of the data flow schema described by Dennis and Fosseen in [1]. The reader is referred to that source for a more complete discussion of the properties of data flow schemas.

I. Unary Operator Schemas

A _unary operator data flow schema_ (UDFS) is a bipartite directed graph in which the two types of nodes are _links_ and _actors_. There are two types of links and five types of actors, as shown in Figure 1. _Data links_ are represented by solid dots and _control links_ by open dots (Figure 1a); the arcs between actors and links are data arcs or control arcs, according to the type of link.

Figure 1b illustrates the various types of actors. Of these, two deserve comment:

An operator has a single _input_ data arc and a single _output_ data arc. The data link from which the input arc emanates is the _input link_ of the operator; the data link at which the output arc terminates is the operator _output link_. Each operator is labelled with a function letter selected from a set F of function letters for the schema; at least one operator is labelled with each letter in F.

A _decider_ has a single input data arc and a single output control arc. Input links and output links for deciders are defined in manners analogous to those for operators. Each decider in a schema is labelled with a predicate letter selected from a set P of predicate letters for the schema; each letter in P labels at least one decider in the schema.

A UDFS S has an ordered set IN(S) of schema _input links_, and an ordered set OUT(S) of schema _output links_. No arc terminates on any input link of S,

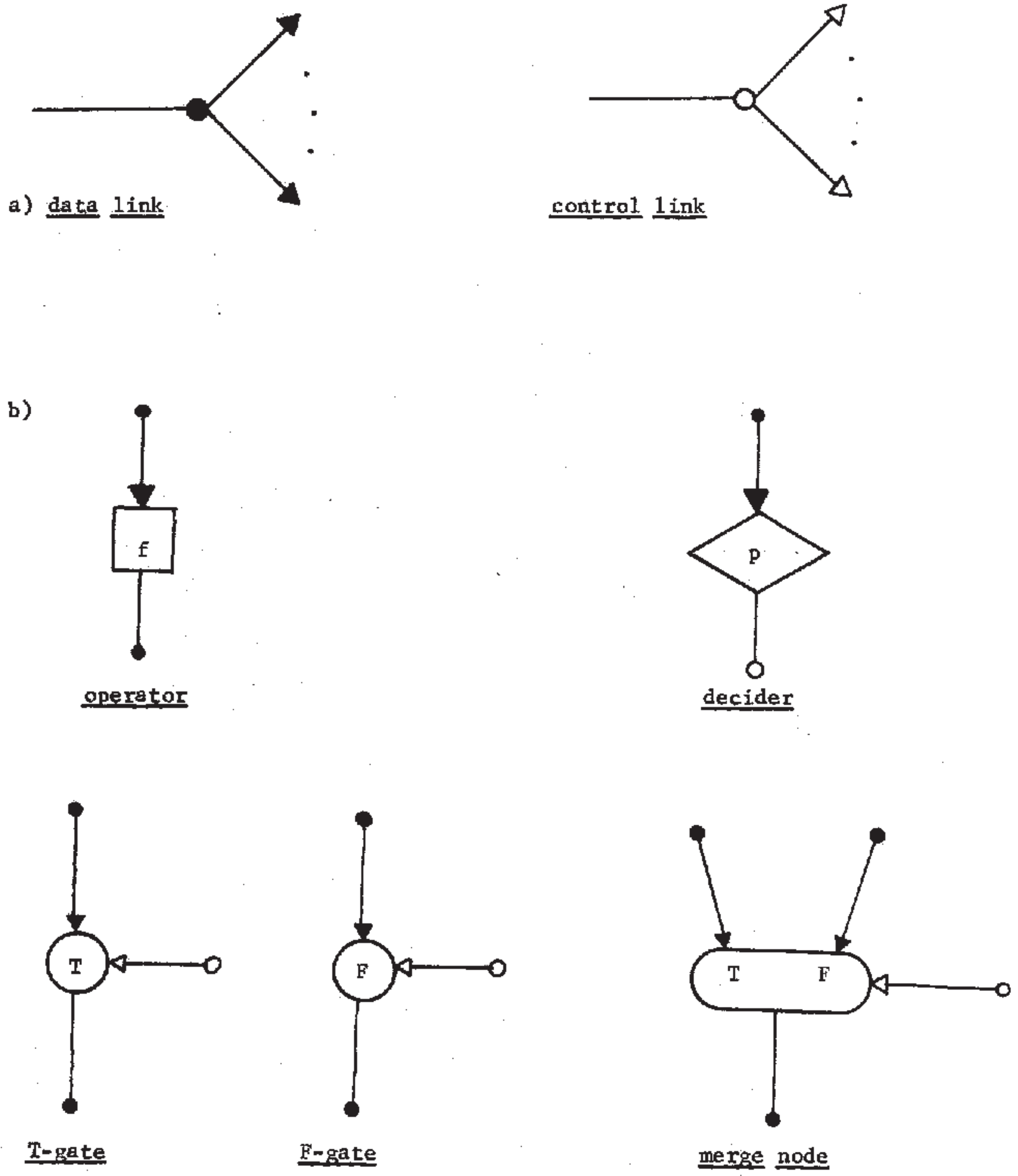

a) data link
control link
b)
f
operator
p
decider
T
T-gate
F
F-gate
T F
merge node

FIGURE 1.

while at least one arc terminates on each non-input link of S; at least one arc originates at each nonoutput link of S. Only data links may be in IN(S) or OUT(S), and the sets need not be disjoint. If card(IN(S)) = m and card(OUT(S)) = n, we say that S is an (m,n) - UDFS.

An interpretation I for a UDFS S with function letters F_S and predicate letters P_S consists of:

i) A domain D of values.

ii) An assignment of a total function $\varphi_f: D \rightarrow D$ to each f in F_S.

iii) An assignment of a total predicate $\Pi_p: D \rightarrow$ {true, false} to each p in P_S.

iv) An assignment of a value $v_i \in D$ to the ith input link of S for each i, $1 \leq i \leq$ card(IN(S)).

Let S be an (m, n)-UDFS with function letters F_S, and let L be the set of symbols $\Delta_1, \Delta_2, \ldots, \Delta_m$. Then the expression set of S, EXP(S), is the set $F_S^* \cdot L \cup$ {null}.

Let S be a UDFS. A configuration of S consists of:

i) An association of an expression in EXP(S) with each data arc of the schema.

ii) An association of one of the expressions {true, false, null} with each control arc of S.

A computation by S is a (possibly infinite) sequence of configurations $\nu_0, \nu_1, \ldots, \nu_k, \nu_{k+1}, \ldots$ where each ν_{i+1} is obtained from ν_i by firing some enabled node in the schema, and ν_0 is the initial configuration of S. The firing rules for the various types of nodes are depicted in Figure 2. (The condition for which a node is enabled, i.e. firable, is indicated by an asterisk, and the result of firing the enabled node is shown to the right.) As computational concurrency is an important aspect of the data flow model, several nodes may be enabled in a given configuration, any one of which may be fired to produce a successor configuration of a computation.

A computation C by a UDFS S is complete if either C is infinite, or C is finite and no node in S is enabled in the final configuration in C; otherwise C is a partial computation. Unless noted otherwise, "computation" shall refer to a complete computation.

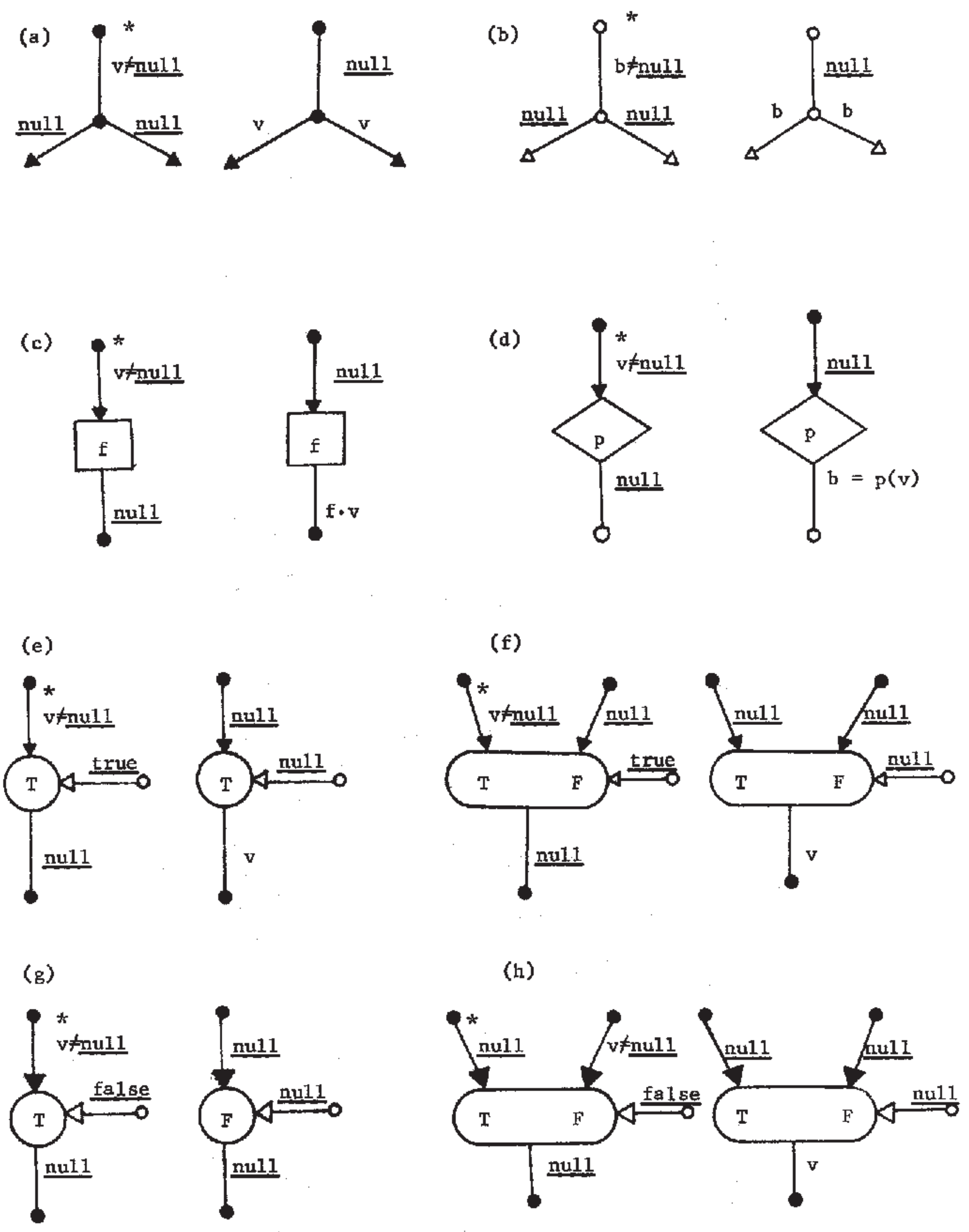

(a)
*
v≠null
null
null
null
v
v
(b)
*
b≠null
null
null
null
b
b
(c)
*
v≠null
f
null
null
f
f·v
(d)
*
v≠null
p
null
null
p
b = p(v)
(e)
*
v≠null
T
true
null
null
T
null
v
(f)
*
v≠null
null
T F
true
null
null
null
T F
null
v
(g)
*
v≠null
T
false
null
null
F
null
null
(h)
*
null
v≠null
T F
false
null
null
null
T F
null
v

FIGURE 2.

The *initial configuration* ν_0 is that in which the expression Δ_i is associated with the data arc(s) emanating from the ith input link of S, the expression *false* is associated with certain control arcs in S (in a manner described later), and the expression *null* is associated with all other arcs in S.

Each actor in a data flow schema is a determinate system, although the schema itself need not be, since the merge node is not persistent (i.e. once enabled it may be disabled without firing if non-null expressions become associated with *both* data input arcs). By adopting a few simple rules regarding the interconnection of these systems, determinacy is assured for the schema as a whole (see, for example [3]). The rules for constructing "well-formed" data flow schemas are as follows:

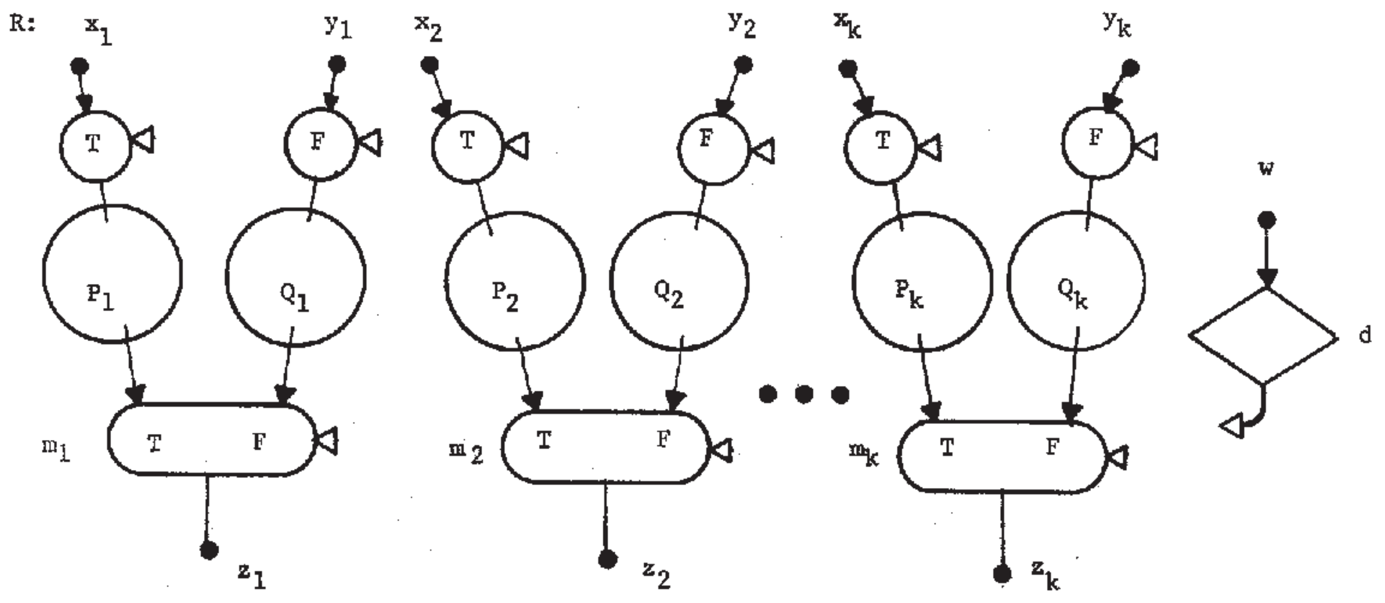
A *well-formed* schema (WFS) is a UDFS formed by an acyclic composition of component UDFS's, where each component is an operator, a conditional schema, or an iteration schema.

The schema R shown in Figure 3a is a *conditional schema*, provided that P_i and Q_i are well-formed UDFS's, $1 \leq i \leq k$. The links w, x_1, y_1, x_2, y_2, ..., x_k, y_k (which are not necessarily distinct) are the *input links* of the conditional schema, links z_1, z_2, ..., z_k are the *output links*; the schemas P_1, Q_1, P_2, Q_2, ..., P_k, Q_k comprise the *body* of the conditional schema, and nodes m_1, m_2, ..., m_k are the *conditional* merge nodes of the schema.

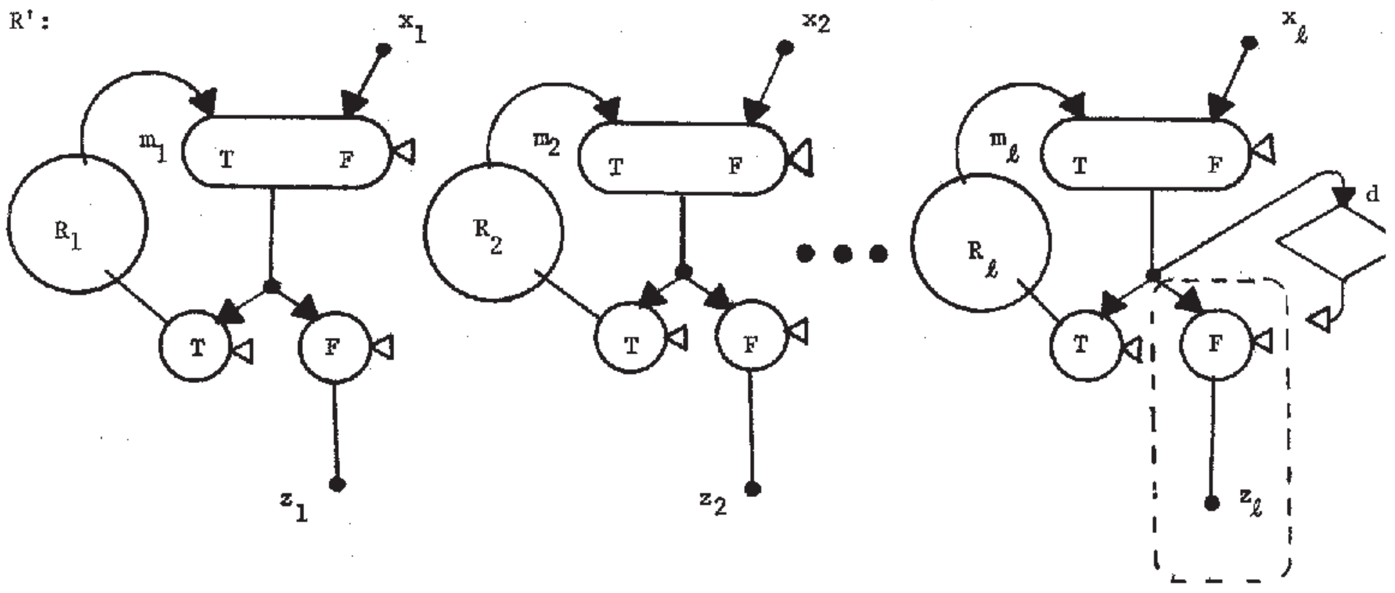
The schema shown in Figure 3b is an *iteration schema*, provided that R_i is a well-formed UDFS, $1 \leq i \leq \ell$. (The link z_ℓ and the preceding F-gate may or may not exist.) The links x_1, x_2, ..., x_ℓ (not necessarily distinct) are the *input links* of the iteration schema, the links z_1, z_2, ..., $z_{\ell-1}$ and z_ℓ (if it exists) are the *output links* of the iteration schema; the UDFS's R_1, R_2, ..., R_ℓ comprise the *body* of the iteration schema, and nodes m_1, m_2, .., m_ℓ are the *iteration* merge nodes of the schema.

The decider associated with a conditional schema R is a *conditional decider* and is said to *control* schema R; that associated with an iteration schema R' is an *iteration decider* and is said to *control* R'. The input control arcs of iteration merge nodes in a WFS are precisely those with which the value *false* is associated in the initial configuration of the schema.

Examples of well-formed schemas and the "programs" they represent are given in the Appendix.



(a) Conditional schema.



(b) Iteration schema.

FIGURE 3.

II. Equivalence of WFS's

Let S be a WFS and C some computation by S. Then for each data link x in S, the expression history of x is the sequence of non-null expressions associated with the incident arc of x during computation C.

When an enabled node of a schema fires during a computation, the subsequent non-null expression associated with the emanating arcs of the node is completely determined by those associated with the incident arcs, unless the node is a decider. If the schema has been provided an interpretation, then the output expression resulting from the decider's firing is also determined by the input arc's expression, since a total predicate will be associated with the decider and, in an obvious fashion, a value from the domain will be associated with each element in the expression set. Hence, while the specification of an interpretation for a WFS S does not determine a unique computation for the schema, each (complete) computation by an interpreted schema defines the same expression history for each data link in the schema. In fact, such is still the case if, rather than providing an interpretation for the schema, we provide, for each predicate letter p appearing in the schema, a total predicate Π_p: (EXP(S)) $\rightarrow$ {true, false}. In general, however, we will still have specified more than is needed to determine the expressions associated with the output arcs of S at the conclusion of any finite computation by S, since each predicate will be evaluated at only finitely many input expressions. This motivates the following definitions:

Let S be a WFS and d a decider in S. Then a test by d is a pair (E, d) where E is an element of EXP(S). Each firing of a decider during a computation by S defines a test by that decider in the obvious manner: the expression E is simply the expression associated with the input arc of the decider at the time of the firing.

If C is a computation by S, the logic sequence of C is constructed as follows:

We begin with the empty sequence; each time a decider fires during computation C, we append the pair (τ, N) where τ is the test defined by the firing and N $\in$ {true, false} is the outcome of the test, i.e. the control value associated with the output arc of the decider immediately after firing. The resulting sequence is the logic sequence of C.

Let S and S' be WFS's (S' not necessarily distinct from S), and let τ and τ' be tests made by deciders d and d' in S and S', respectively. Then τ and τ' are _similar_ tests if their first components are identical and their second components are labelled with the same predicate letter. Let L be the logic sequence of a computation C by S and L' the logic sequence of a computation C' by S'. Then L and L' _conflict_ at tests τ, τ' if (τ, N) is in L and (τ', N^c) is in L', where $N \in \{$_true_, _false_$\}$, N^c is the logical complement of N, and τ and τ' are similar tests. Logic sequences L and L' are _consistent_ if they do not conflict at any pair of tests. A computation is _proper_ if its logic sequence is self-consistent.

Let S and S' be (m,n)-WFS's, and let C and C' be proper computations by S and S', respectively. Then C and C' are (output) _equivalent_ if either both are infinite, or both are finite and each defines the same expression history for the ith schema output link, $1 \leq i \leq n$. The schemas S and S' are (strongly) _equivalent_ if, for all proper computations C by S and C' by S', the logic sequences of C and C' are consistent only if C and C' are equivalent.

It is convenient at this time to introduce a notion of equivalency among data links of WFS's. Let x be a data link of a WFS S and y a data link of a WFS S'. Then links x and y are _equivalent links_ if, for each proper computation C by S and consistent proper computation C' by S', the expression history of x defined by C is the same as that of y defined by C', whenever C and C' are finite. A WFS S is _reduced_ if it contains no pair of equivalent data links.

III. Free Schemas

A WFS S is a free WFS (FWFS) if, for each computation C by S, tests τ and τ' are made during C only if the first components of τ and τ' differ, or the predicate letters labelling the second components of τ and τ' differ. (Intuitively, a schema S is free if no predicate is ever applied twice to the same expression during a computation by the schema.) We note that freeness of S ensures that each computation by S is proper, regardless of the outcome of any test made during the computation.

IV. Decider and Schema Productivity

Intuitively, a test τ made during a computation C by a WFS S is productive if the outcome of the test affects the output behavior of the schema for that computation. Formally: Let S be a WFS, and let τ be a test made during a proper computation C by S. Then τ is productive if there exists a computation C' by S such that the logic sequences of C and C' conflict only at tests τ, τ' (for some τ' made during C) and C and C' are not equivalent.

A decider d in a WFS S is productive if d makes a productive test during some computation by S. A schema S is decider productive if each decider in S is productive.

A WFS S is said to be in standard form if it is reduced and decider productive.

V. Decidability of Link Equivalence in FWFS's

In this section, we prove that it is decidable, for any pair of data links x and y in a FWFS S, whether or not x and y are equivalent. (This result is a corollary of the result reported in [1]; the alternative proof given here is far simpler, although not readily generalizable to the entire class of Dennis-Fosseen schemas.)

Some additional notation shall prove useful. Let S be a FWFS and let $\mathcal{N}$ be the set of nodes of S. We may define a partial ordering "$>$" among the elements of $\mathcal{N}$ as follows:

$$n > n' \text{ if and only if:}$$

(i) n' is a decider of S and n is a node other than a decider; or

(ii) n and n' are deciders controlling schemas R and R' in S, respectively, such that neither decider is within the body of the schema controlled by the other, and some input link of R' lies on a path from some output link of R to some output link of S.

(Note that the well-formedness of S ensures that "$>$" is indeed a partial ordering.)

A computation C by a FWFS S is <u>properly</u> <u>ordered</u> if no node n fires in a configuration ν of C if a node n' is enabled in ν such that $n' > n$. We note that for each computation C by S there is an equivalent computation C' by S such that C' is properly ordered.

Intuitively, properly ordered computations have the property that the firing of deciders is held up until only deciders are enabled. Also, if a decider is fired at some point in a computation by a schema, then no loop-free path from any schema input link to that decider contains a node controlled by a decider which is also enabled at that point in the computation -- otherwise, this other decider would be fired first.

Let S be an FWFS and C a computation by S. Then the <u>outcome</u> <u>sequence</u> of C is the sequence of ordered pairs obtained from the logic sequence of C by deleting the first component of each test appearing in any pair in the sequence; i.e. if $(\tau_1, N_1), (\tau_2, N_2), \ldots, (\tau_k, N_k), (\tau_{k+1}, N_{k+1}), \ldots$ is the logic sequence of C, then the outcome sequence of C is the sequence $(\xi(\tau_1), N_1), (\xi(\tau_2), N_2), \ldots, (\xi(\tau_k), N_k), (\xi(\tau_{k+1}), N_{k+1}), \ldots$, where $\xi(\tau_i)$ denotes the second component of τ_i. We note that the freeness of S ensures that the set of outcome sequences of computations by S is a regular set.

<u>Theorem</u> <u>1</u>: Let S be a FWFS and let x and y be data links in S. Then it is decidable whether or not x and y are equivalent links.

<u>Proof</u>: As is the case with logic sequences, we note that many computations by an FWFS may have the same outcome sequence, but that all computations having the same outcome sequence define the same expression histories for each schema data link.

Let L_x be the language $\{\alpha\$\beta\$ | \alpha$ is the outcome sequence of some finite computation C by S, $\beta \in EXP(S)$ is the last element in the expression history of x defined by C (β is the empty string if the expression history of x defined by C is empty), and \$ is a special symbol not appearing in EXP(S)$\}$.

Let L_y be the language defined in a similar fashion for link y. We note that the freeness of S implies that x and y are equivalent links if and only if $L_x = L_y$, since if the expression histories of x and y differ for some computation by S, then in particular the last elements of the histories differ for some computation by S.

We may construct a deterministic pushdown automaton M_x (see, for example [4]) which recognizes L_x as follows:

M_x will have stored in its finite state control a description of the schema S. Scanning an input string of the form $\alpha\$\beta$, M_x will push α into its stack, ensuring as it does so that α is indeed an outcome sequence of some properly ordered computation by S. After scanning the special symbol \$, M_x will begin to "trace a path" from link x backward through the schema S. As merge nodes are encountered in the path, symbols of α are popped from the stack to determine which of the possible paths are followed. (The fact that α is the outcome sequence of a properly ordered computation ensures that the required outcomes are stored in the stack in the correct order.) As operators are encountered in the path, symbols of β are scanned to ensure that β is the correct expression for the path followed, i.e. as each operator is encountered, the next symbol of β is scanned to ensure that it is the function symbol labelling the operator. Finally, if and when the ith input link of S is encountered, the last two symbols of β are scanned to ensure that they are $\Delta_1\$$.

In a similar manner we may construct an automaton M_y which recognizes the language L_y. The automata M_x and M_y have the property that in accepting or rejecting any input string, the direction of the stack head changes only once; hence the work of Valiant [4] implies that the equivalence of M_x and M_y (and hence that of links x and y) is decidable.

Before stating the next result, we introduce additional notation: For any FWFS S, let the <u>boundary links</u> of S, BOUND(S), be the union of OUT(S) and the set of data links which are input links to iteration deciders. The well-formedness of S ensures that each node of S must lie on a path from IN(S) to BOUND(S). (Note that a node need <u>not</u> lie on a path from IN(S) to OUT(S).)

<u>Corollary 1.1</u>: Let S be a FWFS. Then we may effectively construct from S an equivalent FWFS S' such that S' is reduced.

<u>Proof</u>: For each pair of equivalent links x and y in S, we replace all arcs emanating from link x by arcs emanating from link y and delete link x. We then delete all nodes no longer on a path from IN(S) to BOUND(S). This procedure is repeated until no pair of equivalent links remains. The result schema is S'.

VI. <u>Decidability of Productivity for AFWFS 's</u>.

It has been shown [2] that the equivalence problem for the class of WFS's is recursively undecidable. It follows immediately that decider productivity is an undecidable property for the class, since the schema S of Figure 4 is a WFS if S_1 and S_2 are WFS's, and decider d is productive if and only if S_1 and S_2 are not equivalent. It is currently an open problem whether or not productivity is a decidable property for the class of FWFS's, since the ability to decide productivity implies the ability to determine the equivalence of arbitrary FWFS's. (Note that the schema S of Figure 4 is free if S_1 and S_2 are free schemas.)

We are able to show, however, that productivity <u>is</u> a decidable property of a subclass of the FWFS's:

Let S be an arbitrary FWFS. Then S satisfies <u>Property A</u> if each decider in S is labelled with a predicate letter not appearing elsewhere in the schema; in such a case, we say that S is an <u>AFWFS</u>.

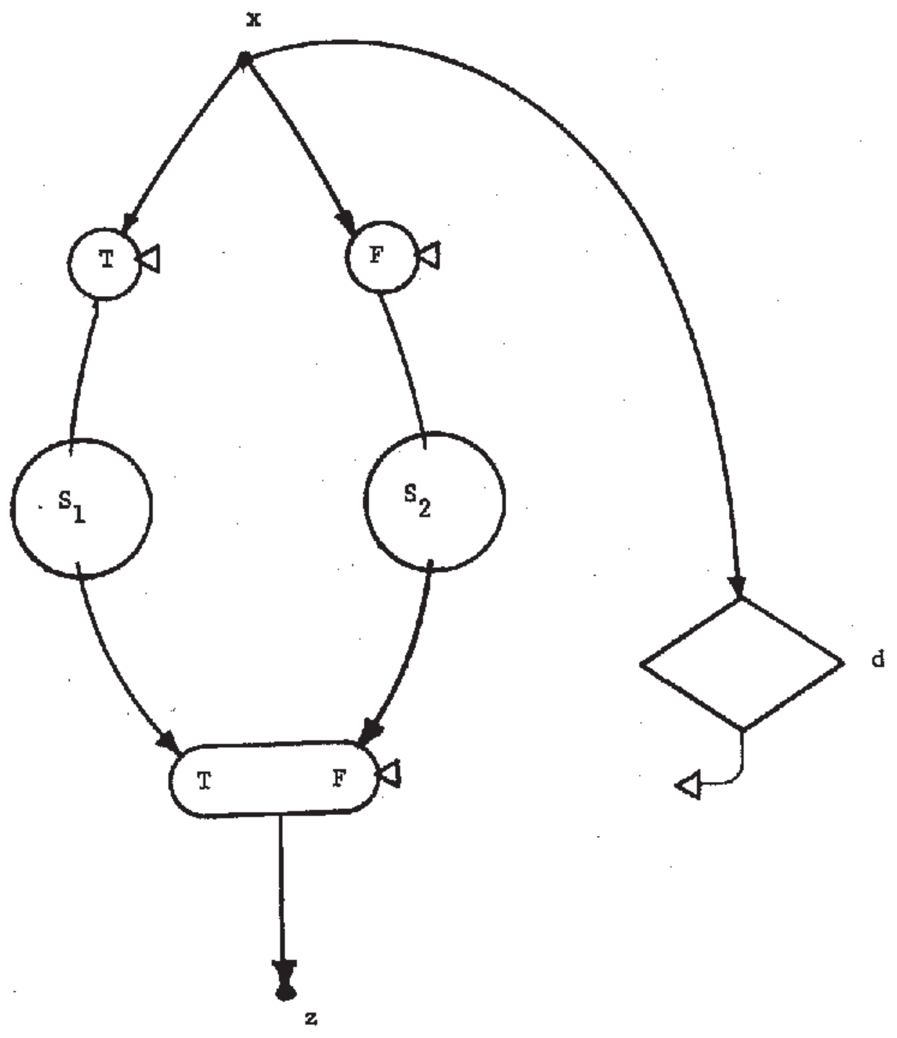
In this section of the paper we show that it is decidable whether or not a decider in an AFWFS is productive. Unfortunately, this does not directly imply the decidability of equivalence for the class of AFWFS's, since the schema S of Figure 4 is not, in general, an AFWFS, even if both S_1 and S_2 are.

We note that if S is an AFWFS and d is an iteration decider in S, then each test made by d is productive. Thus, it is sufficient to prove that it is decidable whether or not a conditional decider d in an arbitrary AFWFS is productive.

Some additional notation is useful:

Let S be a FWFS and d a conditional decider in S. Let m be a merge node controlled by d. Then m is <u>null</u> if, whenever C and C' are computations by S conflicting only at tests made by d, the expression history of the output link of m defined by C is the same as that defined by C'. (Informally, m is null if the sequence of non-null expressions associated with the output arc of m is independent of the outcomes of tests made by d during any computation by S.)

S:



IN(S) = $\{x\}$
OUT(S) = $\{z\}$

FIGURE 4.

We note that there is no reason to extend the concept of null node to the merge nodes controlled by iteration deciders, since such nodes cannot be null: the length of the expression history of such a node's output link depends on the number of times the controlling decider fires.

Clearly, any conditional decider which controls only null nodes must be non-productive. Hence, the identification and elimination of such nodes is a necessary step in the identification of non-productive deciders. Unfortunately, while it is an easy task to eliminate null merge nodes from a schema, the elimination of such nodes is not in itself sufficient to ensure the productivity of each decider in the schema, as demonstrated by the schema of Figure 5. Schema S contains no null merge nodes, yet decider d is non-productive: the output of the merge node m is used solely as input to decider d", and m exhibits <u>non-null behavior</u> (i.e. the expression history for the output link of m defines the outcome of the test by decider d) for precisely those computations during which m" exhibits null behavior. It would seem that a necessary and sufficient condition to ensure that a schema S is decider productive is that there exist a computation by S during which each conditional merge node exhibits non-null behavior; as we shall see, this is nearly the case.

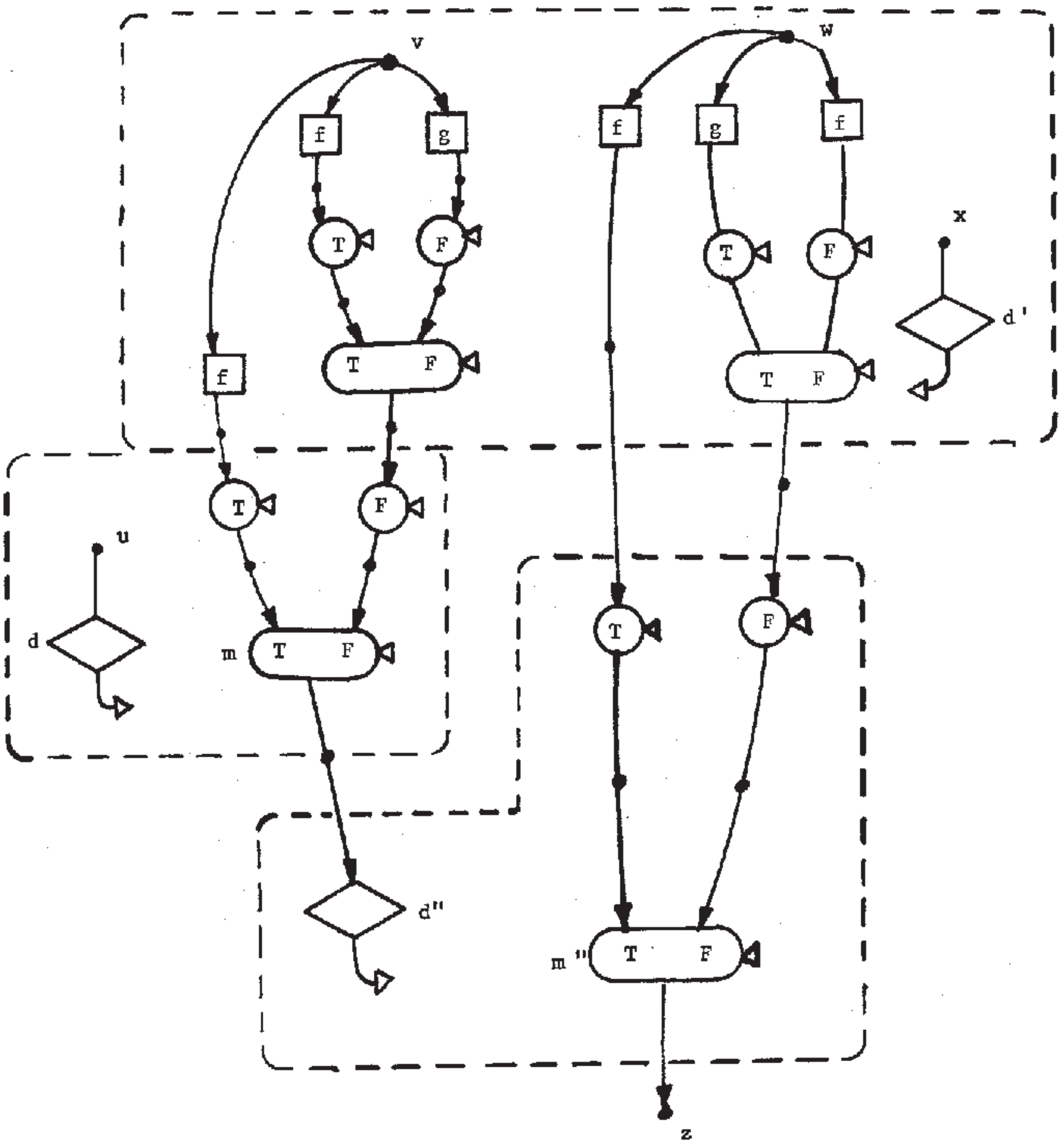
Before presenting the next lemma, we consider a simple transformation which may be applied to AFWFS's:

Let R be a (portion of) a conditional schema within S as shown in Figure 6a. (We say that R is a <u>conditional construct</u> controlled by d; schema P is the <u>true alternative</u> of R, schema Q the <u>false alternative</u>.) Then <u>Transformation T</u> consists of moving gate t_1 past schema P, and moving gate f_1 past schema Q, as illustrated in Figure 6b. We note that if P and Q are free of iteration schemas, then the application of Transformation T to R results in an AFWFS which is equivalent to S; in such a case, we say that T is <u>applicable</u> to R.

<u>Lemma 2.1</u>: Let S be an AFWFS. Then for each merge node controlled by a conditional decider of S, it is decidable whether or not the node is null.

<u>Proof</u>: Let S" be the AFWFS obtained from S by applying T wherever applicable in the schema. It is clear that if T is not applicable to a conditional construct associated with a merge node m in the schema, then m cannot be null. (Since at least one alternative of the construct contains iteration deciders which might diverge if enabled, the expression history of m's output link depends in general on the outcomes of tests made by the controlling decider.) Hence, the only candidates for null nodes in S" are those condi-

S:



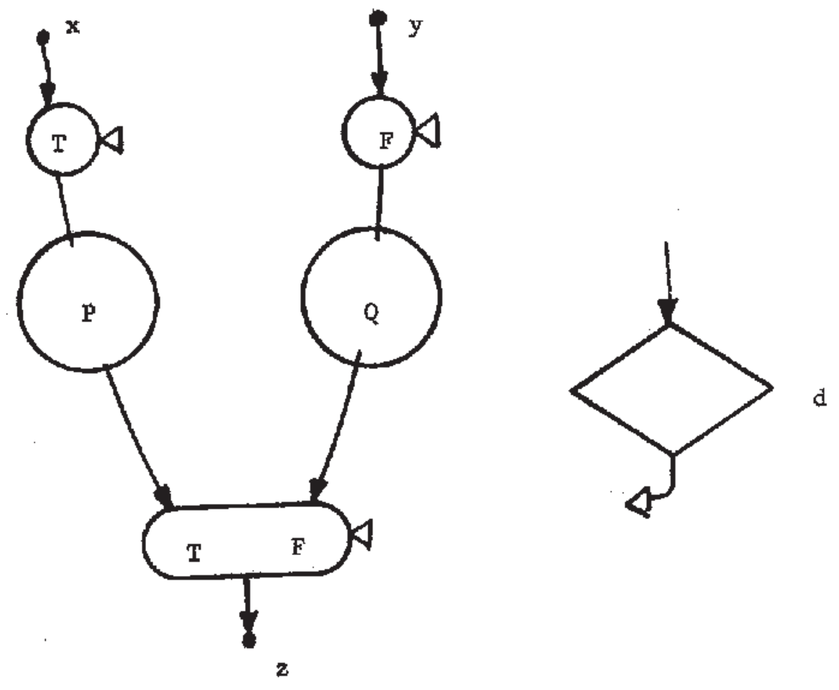
IN(S) = {u,v,w,x}

OUT(S) = { z }

FIGURE 5.

R:

(a)



(b)

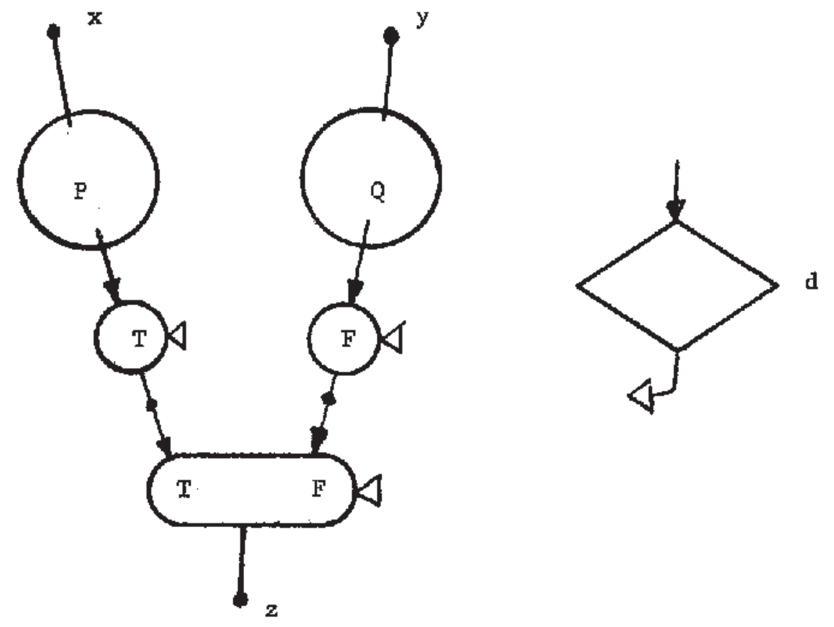


FIGURE 6.

tional merge nodes in which the paths from the associated gates to the merge consist of a single data arc. But it is clearly decidable whether or not such a node is null, since it is null if and only if the input links of the associated T and F gates are equivalent.

Corollary 2.1.1: Let S be an AFWFS. Then we may construct an AFWFS S' from S such that S' is equivalent to S and S' is free of null merge nodes.

Proof: For each null merge node m in S, we merge the output link of m with the input link of either associated gate, and then delete both associated gates and node m. We then delete all nodes no longer on a path from IN(S) to BOUND(S); the remaining schema is S'.

Lemma 2.2: Let S be an AFWFS. Suppose that Transformation T is not applicable to a conditional construct associated with a merge node m driven by a decider d. Then d is productive.

Proof: Since Transformation T is not applicable to the construct, one alternative of the construct (say the true alternative) contains an iteration decider labelled with some predicate letter p. Property A ensures that the false alternative cannot contain a similarly labelled decider. Since S is free, the p-labelled decider can diverge (i.e. perform an infinite series of tests, each with outcome true) in response to a true outcome of a test made by d, but not in response to a false outcome. Thus d is productive.

Lemma 2.3: Let S be an AFWFS free of null merge nodes. Suppose that x is an output link of a merge node driven by a conditional decider d, and that a data path exists from link x to BOUND(S). Then d is productive.

Proof: By hypothesis, there exist finite computations C and C' by S such that C and C' conflict only at a test τ made by d, and such that the expression history of x defined by C differs from that of x defined by C'. Moreover, since there is a data path from x to a link $y \in$ BOUND(S), then C and C' can be chosen in such a way as to ensure that the expression history of y defined by C differs from that of y defined by C'. If $y \in$ OUT(S), the productivity of d is immediate; if y is the input link of an iteration decider d' in S, the productivity of d follows directly from Property A and the freeness of the schema: since a test τ' is made by d' during C which is not made by d' during C' (or vice versa), we may certainly construct a computation C" by S such that C" is infinite and conflicts with one of the finite computations C and C' only at test τ, thus implying the productivity of decider d.

The proof of the following result is a straightforward, albeit tedious exercise, and is left to the reader:

<u>Lemma</u> <u>2.4</u>: Let $\tau_1, \tau_2, \ldots, \tau_k, \tau_{k+1}$ be 2n-tuples $(n > 0)$ of words over an alphabet V, let $R_1, R_2, \ldots, R_k$ be sets of 2n-tuples of words over V, and let L be the language $\tau_1 \cdot R_1^* \cdot \tau_2 \cdot R_2^* \cdot \ldots \cdot \tau_k \cdot R_k^* \cdot \tau_{k+1}$. (The concatenation operation is extended to tuples of words in the obvious manner: if $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_\ell)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_\ell)$ are tuples of character strings, then $\alpha \cdot \beta$ is the tuple $(\alpha_1 \cdot \beta_1, \alpha_2 \cdot \beta_2, \ldots, \alpha_\ell \cdot \beta_\ell)$.)

Then: $((\forall \omega \in L)(\exists i, 1 \le i \le n)$ (components 2i-1 and 2i of ω are identical)) $\Rightarrow$ $((\exists i, 1 \le i \le n) (\forall \omega \in L)$ (components 2i-1 and 2i of ω are identical)).

Some additional terminology is needed before presenting an important corollary of the above Lemma:

Let S be a WFS. Then the set of <u>main</u> <u>deciders</u> of S, MAIND(S), is the set of deciders in S which do not occur within the body of some iteration schema in S; the set of <u>main</u> <u>links</u> of S, MAINL(S), is the set of links in S which do not occur within the body of some iteration schema in S, less the output links of merge nodes controlled by main iteration deciders of S. We note that if x is a link in MAINL(S), the expression history of x defined by any computation C by S consists of at most a single element.

<u>Corollary</u> <u>2.4.1</u>: Let S be a reduced FWFS and let $X = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ be an ordered set of data links in MAINL(S) such that no merge node driven by a conditional decider in MAIND(S) lies on a data path from IN(S) to an element of X. Then there exists a computation C by S such that for all i, $1 \le i \le n$, the expression history of x_i defined by C is not the same as that of y_i defined by C.

<u>Proof</u>: Let $E = (d_1, d_2, \ldots, d_k)$ be an enumeration of the iteration deciders in MAIND(S) such that if $d_i \succ d_j$ (where $\succ$ is the partial ordering of the nodes of S defined previously), then $i > j$; let $\mathcal{C}$ denote the class of finite, properly ordered computations by S. Then for any data link x in X, the expression history of x defined by any computation C in $\mathcal{C}$

consists of a single word ω of the form
$\alpha_{k+1}(x) \cdot \beta_k(x, C) \cdot \alpha_k(x) \cdot \beta_{k-1}(x,C) \cdot \ldots \cdot \alpha_2(x) \cdot \beta_1(x,C) \cdot \alpha_1(x)$ where $\beta_i(x, C)$ denotes the (possibly empty) portion of ω due to the firings of operators in the iteration schema controlled by d_i; $\alpha_i(x)$ denotes the fixed portion of ω due to the operators which fire between the last firing of d_i and the first firing of d_{i+1}, $1 \le i \le k$; and $\alpha_{k+1}(x)$ denotes the portion of ω due to the operators which fire after the last firing of d_k. (Note that the α's are the same in all computations in $\mathcal{C}$.)

For each i, $1 \le i \le k+1$, let τ_i be the 2n-tuple $(\alpha_i(x_1), \alpha_i(y_1), \alpha_i(x_2), \alpha_i(y_2), \ldots, \alpha_i(x_n), \alpha_i(y_n))$.

For each j, $1 \le j \le k$, let R_j be the set of 2n-tuples $\{(\beta_j(x_1, C), \beta_j(y_1, C), \beta_j(x_2, C), \beta_j(y_2, C), \ldots, \beta_j(x_n, C), \beta_j(y_n, C)) | C \in \mathcal{C}\}$.

Finally, let $L = \tau_{k+1} \cdot R_k^* \cdot \tau_k \cdot R_{k-1}^* \cdot \ldots \cdot \tau_2 \cdot R_1^* \cdot \tau_1$. (Intuitively, $L = \{(\mathcal{E}_{x_1}, \mathcal{E}_{y_1}, \mathcal{E}_{x_2}, \mathcal{E}_{y_2}, \ldots, \mathcal{E}_{x_n}, \mathcal{E}_{y_n}) |$ for some computation C in $\mathcal{C}$, $\mathcal{E}_\omega$ is the (singleton) expression history of link ω defined by C, $\omega \in \{x_1, y_1, x_2, y_2, \ldots, x_n, y_n\}\}$).

We have:

((∀ computations C in $\mathcal{C}$)(∃i ≤ n)(the expression histories of links x_i and y_i defined by C are the same)) ⇔ ((∀$\omega \in L$)(∃i ≤ n) (components 2i-1 and 2i of ω are identical)) ⇔ ((∃i ≤ n)(∀$\omega \in L$)(components 2i-1 and 2i of ω are identical)) ⇔ ((∃i ≤ n) (∀ computations C in $\mathcal{C}$)(the expression histories of x_i and y_i defined by C are the same)) ⇔ S is not reduced.

<u>Corollary 2.4.2</u>: Let S be a FWFS free of null merge nodes, and let M = $m_1, m_2, \ldots, m_n$ be a set of merge nodes in S such that for each i, $1 \le i \le n$, m_i satisfies two properties:

i) m_i is controlled by a conditional decider in MAIND(S); and

ii) no merge node controlled by a conditional decider in MAIND(S) lies on a data path from IN(S) to m_i.

Then there exists a computation C by S such that each element of M exhibits non-null behavior during C.

Proof: We may assume, without loss of generality, that Transformation T is not applicable to the conditional constructs associated with the elements of M. For each i, $1 \leq i \leq n$, let x_i and y_i denote the input data links of the gates associated with merge node m_i. By the preceeding corollary, there is a computation C by S such that the expression history of x_i and that of y_i defined by C differs for all i, and the result follows immediately.

Theorem 2: Let S be an AFWFS and d a decider in S. Then it is decidable whether or not d is productive.

Proof: 1. We first show that productivity of d is decidable if d is in MAIND(S):

We may assume that S is in standard form, and that Transformation T is not applicable in S. Decider d must be productive if d fails to satisfy the following conditions:

i. d is a conditional decider, by Lemma 2.1.

ii. All merge nodes controlled by d have the property that the paths from the associated gates to the merge nodes consists of single data arcs (otherwise by Lemma 2.2, Transformation T is applicable), and by Lemma 2.3, there are no data paths from the output link of any merge node controlled by d to BOUND(S), i.e. each path from the output link of a merge controlled by d to BOUND(S) contains at least one main control link.

Assume that decider d satisfies the above conditions. Let D be the set of conditional deciders in MAIND(S) to which paths exist from the output links of merge nodes controlled by d. Let Σ be the set of AFWFS's obtained from S by fixing, in all possible combinations, the outcomes of tests made by the main conditional deciders not in D, and replacing the associated conditional constructs by the appropriate alternatives. Let Σ' be the set of AFWFS's obtained from Σ by removing null merge nodes from the schemas, as outlined in the proof of Corollary 2.1.1. Clearly, d is productive in S if and only if d is productive in some element of Σ'. But d is productive in some element of Σ' if it appears at all in some element of Σ':

Suppose d appears in some element of Σ'. Then in particular, it must appear in some schema S' in Σ' in which there is a path ρ from the output link of a merge node driven by d to BOUND(S'), such that no merge node driven by a conditional decider in MAIND(S') lies on a path from IN(S') to an input link of any main merge node in path ρ. By Corollary 2.4.2, there is a computation C by S' such that each merge node in ρ driven by a conditional decider in MAIND(S') exhibits non-null behavior during C; hence, the test made by d during C is productive and the productivity of d in schema S is thus ensured.

2. It remains to be shown that the productivity of d is decidable if d is not a main decider of S. Some additional notation is useful:

Let S be a WFS and r a decider in S. The __level__ of r in S is 0 if r is in MAIND(S), and is k+1 if r is in MAIND(R), where R is an iteration schema in S driven by a decider of level k in S.

We now show that if d is a conditional decider of level $k > 0$ in an AFWFS S, then the decidability of productivity for d reduces to that for no more than two conditional deciders of level $k - 1$ in an AFWFS S' constructed from S; the theorem then follows immediately by induction on the level of a conditional decider.

Let S be an AFWFS and let d be a conditional decider of level $k > 0$ in S. Let R be the iteration schema in S of which d is a main node, and let d' be the decider driving R. Decider d lies in one "loop" of the iteration schema controlled by d', i.e. a loop free data path exists to d from one merge node m controlled by d' (Figure 7a); let x be the output link of m. If there is a path from X to d', then d is productive in S if and only if it is productive in the body of schema R (which is decidable, since d is a main conditional of R). If no such path exists, d is productive in S iff the outcomes of its firings affect the expression history of x, and the expression history of x affects the output expressions of the schema, i.e. if and only if d is productive in R and decider d" is productive in the schema resulting from its insertion in S as shown in Figure 7b. (In the figure, f is a new function letter not appearing in S and ρ is a new predicate letter not appearing in S.) Again d is a main decider of R and since d" is of level $k - 1$ in S, the result follows.

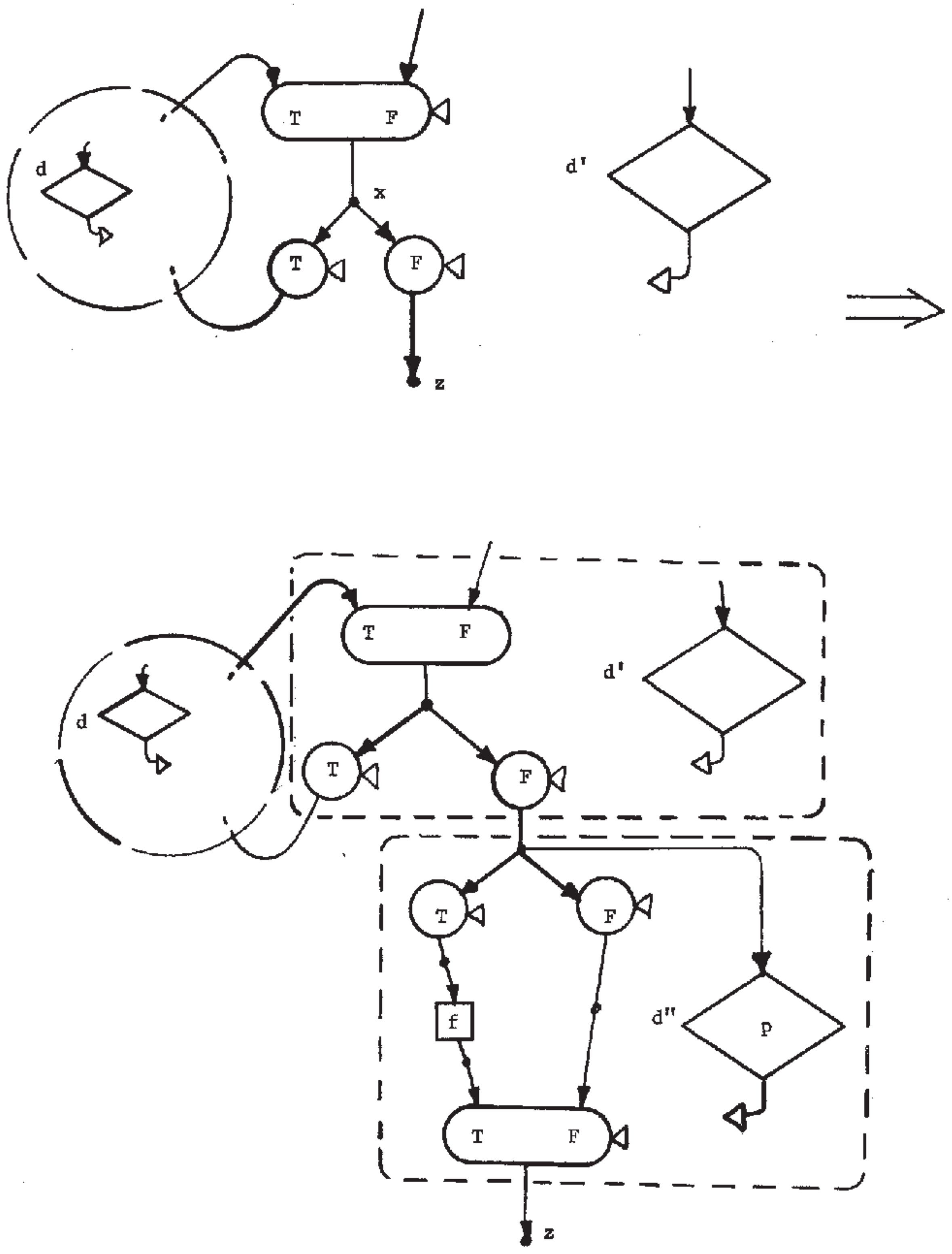


FIGURE 7.

Corollary 2.1: Let S be an AFWFS. Then we may construct from S an AFWFS S' such that S and S' are equivalent, and S' is decider productive.

Proof: We simply eliminate the merge nodes controlled by any non-productive decider d in S exactly as if they were null nodes. The resultant schema is S'.

VII. Decidability of Equivalence for AFWFS's

In this section we prove the main result of this paper: equivalence is decidable for the class of AFWFS's.

The following Lemma and its Corollaries provide a basis for the proof:

Lemma 3.1: Let S and S' be equivalent reduced AFWFS's, and let C be a finite computation by S. Then there exists a computation C' by S' such that the logic sequences of C and C' are consistent and such that for each productive test τ made during C, a similar test τ' is made during C'.

Proof: Let S and S' be as above and let C be any computation by S. Let $\tau_1, \tau_2, \ldots, \tau_n, \tau_{n+1}, \ldots$ be an enumeration of the productive tests made during C, and let $\mathcal{C}$ be the set of computations by S' which have logic sequences consistent with that of C. The following procedure may be used to select the required computation C':

i. Set i = 1, set $\mathcal{C}_i = \mathcal{C}$.

ii. Choose an element C" from $\mathcal{C}_i$. If a test τ'_i similar to test τ_i is made during C", go to step (iv).

iii. By definition of productivity, there exists some computation C_i' by S, whose logic sequence conflicts with that of C only at test τ_i and which is not equivalent to C. This computation cannot be equivalent to C", and hence its logic sequence must conflict with that of C" at a test τ such that no test similar to τ is made during C. Let $\mathcal{C}'$ be the subset of $\mathcal{C}_i$ consisting of those elements whose logic sequences conflict with that of C" only at test τ. (Note that $\mathcal{C}'$ must be non-empty.) Set $\mathcal{C}_i$ to $\mathcal{C}'$ and go to step (ii).

iv. Let $\mathcal{C}''$ be the subset of $\mathcal{C}_i$ consisting of those elements in which test τ_i' is made. Set $\mathcal{C}_{i+1} = \mathcal{C}''$. Set $i = i + 1$. Go to step (ii).

Corollary 3.1.1: Let S, S' be equivalent productive AFWFS's. Let P_I and P_I' be the sets of predicate letters labelling the iteration deciders in S and S' and let P_C and P_C' be the sets of predicate letters labelling the conditional deciders in S and S', respectively. Then $P_I = P_I'$ and $P_C = P_C'$.

Proof: The first equality follows from the fact that each test made by an iteration decider in an AFWFS is productive, the second from the productivity of each conditional decider in S and S'.

Corollary 3.1.2: Let S and S' be equivalent productive AFWFS's. Let P_I^M and $P_I^{M\prime}$ be the sets of predicate letters labelling iteration deciders in MAIND(S) and MAIND(S'), respectively, and let P_C^M and $P_C^{M\prime}$ be the sets of predicate letters labelling the conditional deciders in MAIND(S) and MAIND(S'). Then $P_I^M = P_I^{M\prime}$ and P_C^M and $P_C^{M\prime}$.

Proof: Again, the first equality follows directly from the productivity of each test made by an iteration decider in either schema; the second follows from the observation that if d is a productive conditional decider in an iteration subschema of an AFWFS, then d can make a productive test each time the body of the subschema is executed.

The previous Corollaries are important because they imply that similarly labelled deciders are similarly "nested" within iteration subschemas in equivalent AFWFS's. In particular, within the main deciders of equivalent AFWFS's we are assured of finding similarly labelled conditional and iteration deciders.

Before proceeding to the next result, we introduce some additional notation:

Let S be an AFWFS, and let S_e be an AFWFS constructed from S by creating a new output link for each main iteration decider as shown in Figure 8(a) if the gate g already exists in S, or as shown in Figure 8(b) otherwise. Then

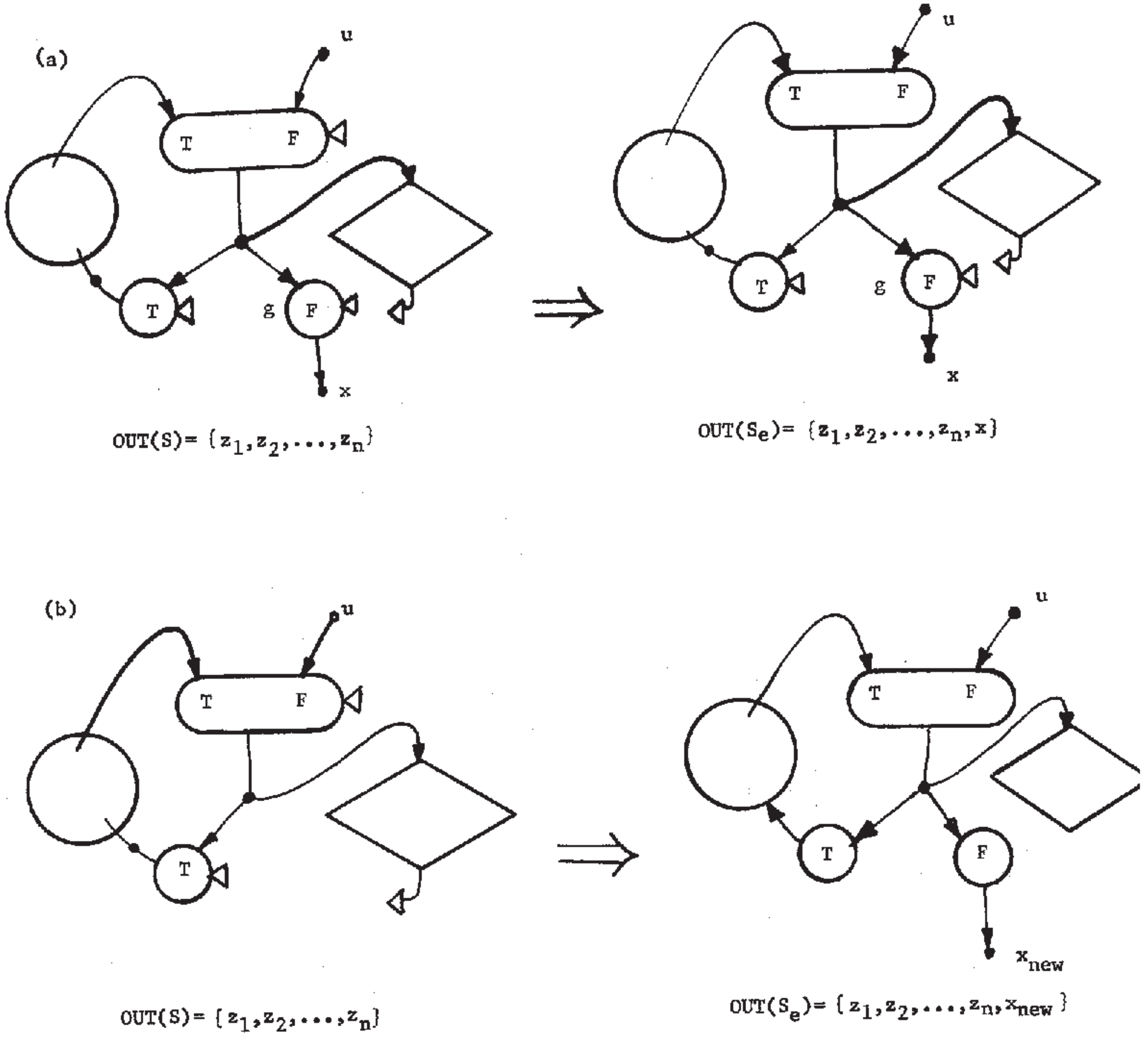

(a)
u
T
F
T
g
F
x
OUT(S)= {z1,z2,...,zn}
u
T
F
T
g
F
x
OUT(Se)= {z1,z2,...,zn,x}
(b)
u
T
F
T
OUT(S)= {z1,z2,...,zn}
u
T
F
T
F
xnew
OUT(Se)= {z1,z2,...,zn,xnew}

FIGURE 8.

the schema S_e is a main extension of S. We note that if x is the new output link associated with a decider d in S_e, then the expression associated with the incident arc of x at the termination of any finite computation by S is the last element of the expression history of the input link of d.

Lemma 3.2: Let S and S' be (m,n)-AFWFS's such that for each iteration decider in MAIND(S) there is a similarly labelled iteration decider in MAIND(S'), and vice versa. Let S_e and S_e' be main extensions of S and S' such that the order in which the new output links of S_e and S_e' are created (in terms of the predicate letters labelling the main iteration deciders) are the same in each case. Then S and S' are equivalent iff whenever C and C' are consistent computations by S_e and S_e' such that the corresponding main iteration deciders of the schemas fire the same number of times during the computations, then C and C' are equivalent.

Proof: Let S, S', S_e, S_e' be as above and assume that S_e and S_e' satisfy the conditions of the Lemma.

"If": We note that for each computation by a schema there is a computation by its main extension possessing the same logic sequence, and vice versa; we note also that if C_e and C_e' are equivalent computations by S_e and S_e', then C and C' are equivalent computations by S and S', where C is the computation by S possessing the same logic sequence as the computation C_e by S_e and similarly for C' and C_e'. We now simply observe that if S_e and S_e' appear equivalent for all pairs of consistent computations in which corresponding main iteration deciders fire an equal number of times, then they must, in fact, be equivalent since the last pair of tests performed by corresponding main iteration deciders (and thus each pair of tests performed) during any pair of consistent computations must be the same. The equivalence of S and S' follows immediately.

"Only if": If S and S' are equivalent, then the expression histories of similarly labelled iteration deciders defined by any pair of finite, consistent computations by the schemas must be the same, otherwise we could easily alter one so that it diverged without violating the consistency of the computations. Thus S_e and S_e' are guaranteed equivalent.

The significance of Lemma 3.2 is this: if we wish to determine whether or not a pair of AFWFS's are equivalent, we may construct from them a pair of main extensions and test this pair of AFWFS's for equivalence _under the assumption_ that corresponding main iteration deciders must always fire an equal number of times during pairs of consistent computations. If the extensions are equivalent under this assumption, the Lemma guarantees that the original schemas are equivalent.

It is convenient at this point to introduce a notion of size for well-formed schemas. The following definition, while not the most obvious, will prove to be quite useful:

Let S be a WFS. Then the _size_ of S, SIZE(S), is the number of merge nodes controlled by iteration deciders in the schema.

Our proof of the decidability of equivalence for AFWFS's will involve an induction on the size of the schemas being compared. (We note that if S is an AFWFS of size 0, then S can be equivalent only to another AFWFS of size 0; moreover, equivalence is trivially decidable in such a case since the number of distinct computations by the schemas is finite and an exhaustive analysis is sufficient.)

Because of the length and nature of the argument needed to prove the next lemma, the proof of the lemma will be deferred to the next section of the paper:

Lemma 3.3: Let S and S' be AFWFS's such that the labelling of deciders in the schemas satisfies the conditions required of equivalent AFWFS's by Corollaries 3.1.1 and 3.1.2. Then the problem of deciding whether or not S and S' are equivalent can be reduced to the problem of deciding equivalence for no more than i pairs of AFWFS's, each of size no greater than k, such that each schema is free of main conditional deciders, where i is a constant bounded by $(\ell + 1)(3^{\ell})^2$, ℓ is the number of main conditional deciders in S or S', and k is the maximum of SIZE(S) and SIZE(S').

<u>Proof</u>: (See Section VIII.)

To demonstrate the decidability of equivalence for AFWFS's, it is sufficient to show that if S and S' are AFWFS's such that MAIND(S) and MAIND(S') are free of conditional deciders, and k is the maximum of SIZE(S) and SIZE(S'), then the problem of deciding whether or not S and S' are equivalent reduces to the problem of deciding equivalence for two pairs of AFWFS's, each schema of size less than k. To this end we present the following Lemma:

<u>Lemma 3.4</u>: Let X, X', Y, Y' and Z' be sets of words over some alphabet T, and let γ be a word in T^*. Let $f_1: X \to X'$ and $f_3: X \to Z'$ be total functions; let $f_2: Y^* \to (Y')^*$ be a total function such that for each $\psi, \rho \in Y^*$ we have $f_2(\psi) \cdot f_2(\rho) = f_2(\psi\rho)$. Suppose we have, for each $\alpha \in X$ and each $\beta \in Y$, the following equalities:

1. $\gamma \cdot \alpha = f_3(\alpha) \cdot f_1(\alpha)$
2. $\gamma \cdot \beta \cdot \alpha = f_3(\alpha) \cdot f_2(\beta) \cdot f_1(\alpha)$

Then for each $\omega \in Y^*$ we have $\gamma \cdot \omega \cdot \alpha = f_3(\alpha) \cdot f_2(\omega) \cdot f_1(\alpha)$.

<u>Proof</u>: We know from (1) and (2) that the assertion is valid for $\omega = \epsilon$ and for $\omega \in Y$. Assume the assertion is valid for all $\omega \in Y^i$, $0 \le i < i_0$. Then the assertion is valid for all $\omega \in Y^{i_0}$ as follows:

Let $\omega = \omega_1 \cdot \omega_2$, $\omega_1 \in Y^{i_0-1}$, $\omega_2 \in Y$. We consider three cases:

<u>Case 1</u>. $\alpha = f_1(\alpha)$, $\gamma = f_3(\alpha)$. We have by assumption $\gamma \cdot \omega_1 \cdot \alpha = \gamma \cdot f_2(\omega_1) \cdot \alpha$ and $\gamma \cdot \omega_2 \cdot \alpha = \gamma \cdot f_2(\omega_2) \cdot \alpha$, from which we have $\omega_1 \cdot \omega_2 = f_2(\omega_1) \cdot f_2(\omega_2) = f_2(\omega_1 \cdot \omega_2)$ and thus $\gamma \cdot \omega_1 \cdot \omega_2 \cdot \alpha = f_3(\alpha) \cdot f_2(\omega_1 \cdot \omega_2) \cdot f_1(\alpha)$.

<u>Case 2</u>. $\alpha = \mu \cdot f_1(\alpha)$, $\mu \ne \lambda$, $\gamma \cdot \mu = f_3(\alpha)$. Then:

$$\begin{aligned}
\forall\rho[&(\gamma \cdot \rho \cdot \alpha = f_3(\alpha) \cdot f_2(\rho) \cdot f_1(\alpha)) \Rightarrow \\
&(\gamma \cdot \rho \cdot \mu \cdot f_1(\alpha) = f_3(\alpha) \cdot f_2(\rho) \cdot f_1(\alpha)) \Rightarrow \\
&(\gamma \cdot \rho \cdot \mu = f_3(\alpha) \cdot f_2(\rho)) \Rightarrow \\
&(\gamma \cdot \rho \cdot \mu = \gamma \cdot \mu \cdot f_2(\rho)) \Rightarrow \\
&(\rho \cdot \mu = \mu \cdot f_2(\rho))]
\end{aligned}$$

Substituting ω_1 and ω_2 for ρ yields:

$$\omega_1 \cdot \mu = \mu \cdot f_2(\omega_1)$$
$$\omega_2 \cdot \mu = \mu \cdot f_2(\omega_2)$$

Then:

$$\begin{aligned}
\gamma \cdot \omega \cdot \alpha &= \gamma \cdot \omega \cdot \mu \cdot f_1(\alpha) \\
&= \gamma \cdot \omega_1 \cdot \omega_2 \cdot \mu \cdot f_1(\alpha) \\
&= \gamma \cdot \omega_1 \cdot \mu \cdot f_2(\omega_2) \cdot f_1(\alpha) \\
&= \gamma \cdot \mu \cdot f_2(\omega_1) \cdot f_2(\omega_2) \cdot f_1(\alpha) \\
&= f_3(\alpha) \cdot f_2(\omega_1 \cdot \omega_2) \cdot f_1(\alpha) \\
&= f_3(\alpha) \cdot f_2(\omega) \cdot f_1(\alpha)
\end{aligned}$$

<u>Case 3</u>. $\mu \cdot \alpha = f_1(\alpha)$, $\mu \neq \lambda$, $\gamma = f_3(\alpha) \cdot \mu$.

The proof for Case 3 is similar to that for Case 2 and is left to the reader. □

The following Lemma provides the remaining needed result:

<u>Lemma 3.5</u>: Let S and S' be standard form (m,n)-AFWFS's such that each is free of main conditional deciders. Let k be the maximum of SIZE(S) and SIZE(S'). Then either it is decidable if S and S' are equivalent, or the problem of deciding whether or not S and S' are equivalent can be reduced to the problem of deciding equivalence of two pairs of AFWFS's, each of size less than k.

<u>Proof</u>: If the size of either schema is 0, equivalence is trivially decidable. Suppose that the size of both schemas is greater than 0:

Let x denote the first output link of S for which there exists a path from IN(S) to x containing a merge node controlled by an iteration decider. (Some such link exists.) Let x' denote the corresponding output link of S'. Because of the absence of main conditional deciders in S, we have that there exists in S a single loop-free data path from IN(S) to x, as shown in Figure 9. (In the figure, the O_i's represent operator schemas, the R_i's iteration constructs.) We have by hypothesis N > 0, and thus may consider the final iteration construct R_N, encountered in the path from IN(S) to x.

Since S is reduced, the expression associated with the incident arc of x at the conclusion of a complete computation by S depends, in general, on the number of times the decider d_N controlling R_N fires during the computation. Hence if x' is to be equivalent to x

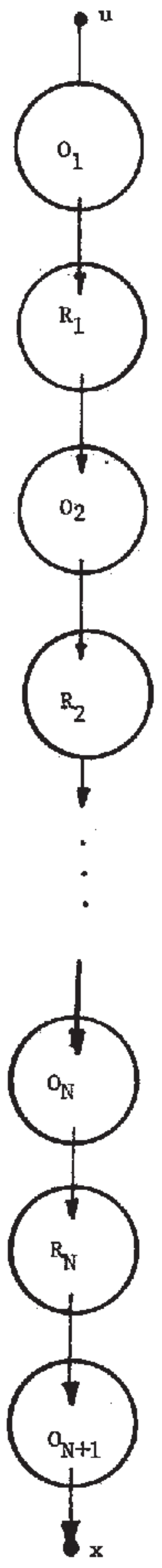


FIGURE 9.

we must encounter an iteration construct R' controlled by a similarly labelled decider d' on the path from IN(S') to x'.

Let C be a finite computation by S and C' a consistent computation by S'. We can write the expression associated with the incident arc of x at the conclusion of C as $\gamma\beta\alpha$, where α and $\beta\alpha$ are the expressions associated with the input and output arcs, respectively, of R_N during C. Similarly, we can write the expression associated with the incident arc of x' during computation C' as $\gamma'\beta'\alpha'$, where α' and $\beta'\alpha'$ are the expressions associated with the input and output arcs of R'. We note that if x and x' are equivalent links, the expression α' is completely determined by the expression α. Less obvious, perhaps, is the fact that γ' is also completely determined by α: let τ' be a test made during C'. Then if γ' is dependent on the outcome of τ', it must be the case that a test τ similar to τ' is also made during computation C and that either α or β (γ is fixed) is dependent on the outcome. But β is dependent only on tests made by deciders in the body of R_N, and hence it must be α which is dependent on the outcome of τ. Thus γ' must be completely determined by α, this in turn implying that β' is a function of β. From Lemma 3.4, therefore, we may conclude that if x and x' appear equivalent for all computations in which the deciders controlling R_N and R' fire no more than once with outcome true, then x and x' are equivalent iff d_N and d' have equivalent input links. We thus have the following result: let S_e S_e' be main extensions of S and S' constructed as in Lemma 3.2. Let S_{e_1} be the schema obtained from S_e by merging the false data input link of node m_N (the merge node associated with construct R_N) with its output link and deleting R_N; let S_{e_2} be the schema obtained from S_e be replacing the construct R_N by a copy of its body as shown in Figure 10. Let $S_{e'_1}$ and $S_{e'_2}$ be the schemas obtained in like manner from S_e'. Then link x is equivalent to link x' iff the output link corresponding to link x' in $S_{e'_1}$ is equivalent to the output link corresponding to link x in S_{e_1} and the output link corresponding to link x' in $S_{e'_2}$ is equivalent to that corresponding to link x in S_{e_2}. This in turn implies that S is equivalent to S' iff S_{e_1} is equivalent to $S_{e'_1}$ and S_{e_2} is equivalent to $S_{e'_2}$. Since each of the AFWFS's S_{e_1}, $S_{e'_1}$, S_{e_2}, $S_{e'_2}$ is of size no greater than k-1, the result follows.

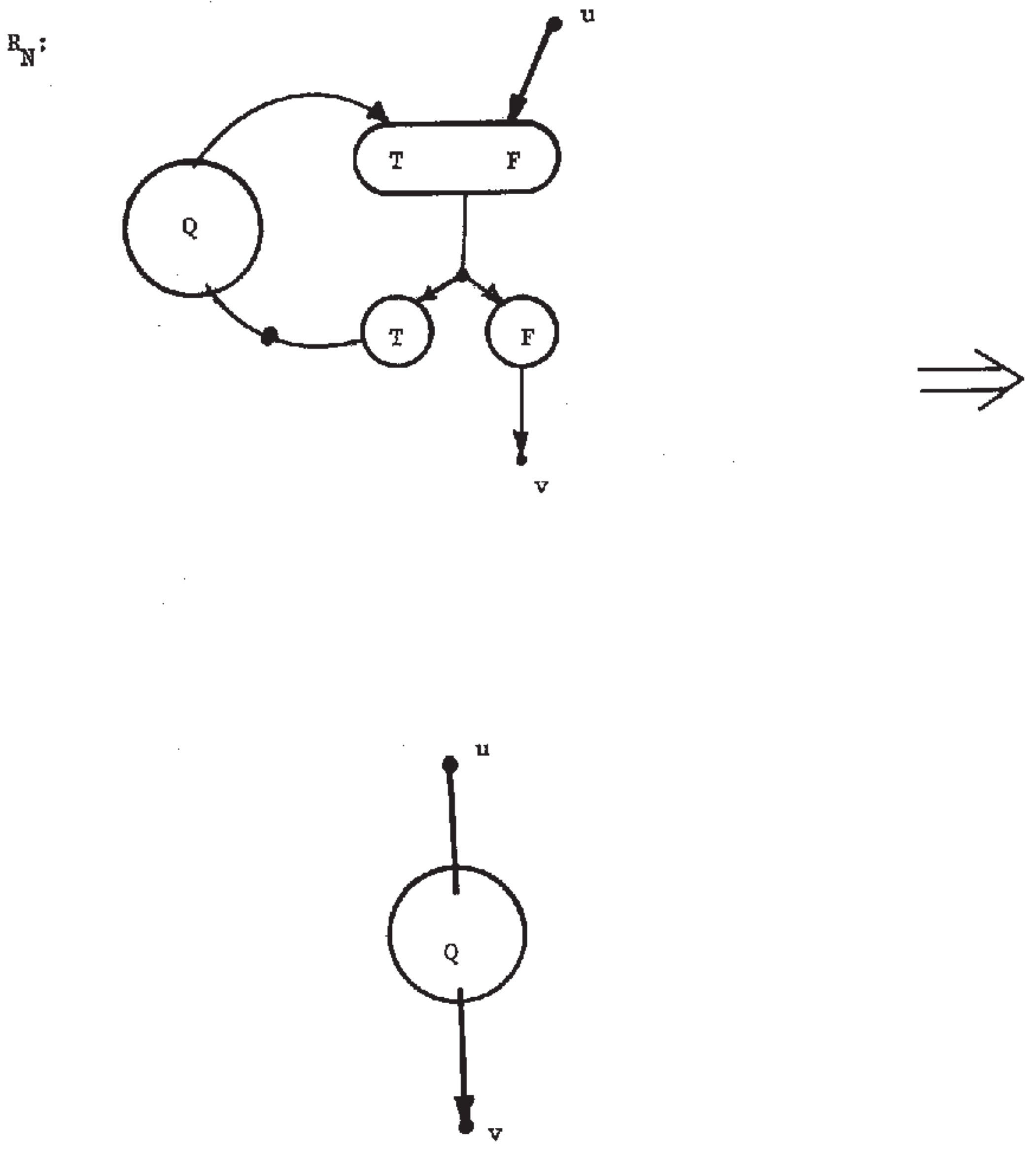


FIGURE 10.

<u>Theorem 3</u>: Let S and S' be AFWFS's. Then it is decidable if S and S' are equivalent.

<u>Proof</u>: We may assume that S and S' are in standard form. The result then follows from Lemmas 3.3 and 3.5 by induction on the maximum size of S and S'.

VIII. Proof of Lemma 3.3:

Our equivalence result will be complete once we have demonstrated the validity of Lemma 3.3.

The proof of the following result is similar to the proof of Corollary 2.4.1 and is left to the reader.

Lemma 3.3.1: Let S and S' be (m,n)-AFWFS's such that there are no conditional deciders in MAIND(S) or MAIND(S'). Let x and y be the ith and jth output links of schema S, $1 \le i, j \le n$, and let x' and y' be the ith and jth output links of schema S'. Suppose that x and x' are not equivalent and y and y' are not equivalent. Then there exist finite, consistent computations C by S and C' by S' such that the expression associated with the incident arc of link x at the conclusion of C differs from that associated with the incident arc of link x' at the conclusion of C', and the expression associated with the incident arc of link y at the conclusion of C differs from that associated with link y' at the conclusion of C'.

For convenience, the Lemma 3.3 is reproduced below:

Lemma 3.3: Let S and S' be (m,n)-AFWFS's such that the labelling of deciders in the schemas satisfies the conditions required of equivalent AFWFS's by Corollaries 3.1.1 and 3.1.2. Then the problem of deciding whether or not S and S' are equivalent can be reduced to the problem of deciding equivalence for no more than i pairs of AFWFS's, each of size no greater than k, such that each schema is free of main conditional deciders, where i is a constant bounded by $(\ell + 1)(3^{\ell})^2$, ℓ the number of main conditional deciders in S or S', and k is the maximum of SIZE(S) and SIZE(S').

Proof: Let S_e and S_e' be the main extensions constructed from S and S', respectively, as in Lemma 3.2. (We note that S_e and S_e' are (m,j)-AFWFS's for some $j \ge n$.) We shall assume that S_e and S_e' contain some main conditional deciders, otherwise the lemma is trivially true. Let c_o denote the number of such deciders, and assume that they have been ordered in some fashion, similarly (according to predicate letter) in each schema.

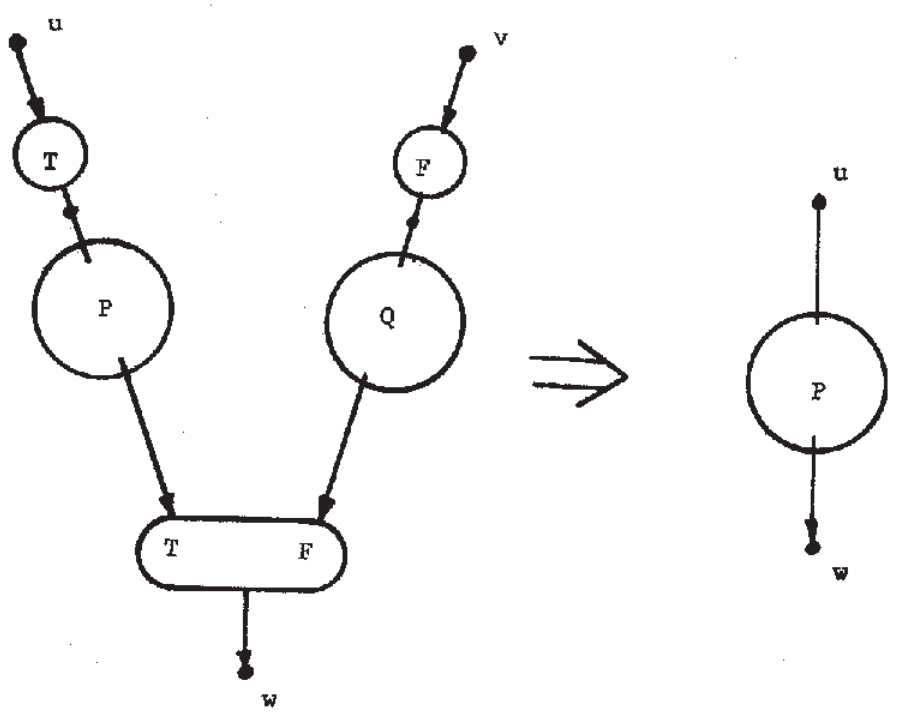
We define a <u>conditional assignment</u> for S_e or S_e' to be a c_o-tuple of expressions from the set {<u>null</u>, <u>true</u>, <u>false</u>} and note that each finite computation by either schema determines, in obvious fashion, a conditional assignment for the schema: the ith component of the tuple, $1 \leq i \leq c_o$, is <u>true</u> if the ith main conditional in the schema fires with outcome true during the computation, <u>false</u> if the ith main conditional fires with outcome false during the computation, or <u>null</u> if the ith main conditional does not fire at all during the computation. A conditional assignment is said to be <u>valid</u> if it is, in fact, determined by some finite computation by the schema.

Let V denote a valid conditional assignment for schema $S_e(S_e')$. Then we may construct a schema S(V) from V and $S_e(S_e')$ as follows: each conditional construct driven by the ith main conditional in $S_e(S_e')$ is replaced (as in Figure 11a) by its true alternative if the ith component of V is <u>true</u>, or by its false alternative (as in Figure 11b) if the ith component of V is <u>false</u>; all nodes no longer on a path to a boundary node are deleted. The decider is then deleted, and its input link made the j + ith output link of S(V). When this procedure has been carried out for each main conditional, we create a new j + kth output link for each k such that the kth decider in $S_e(S_e')$ does not appear in S(v), i.e. such that <u>null</u> is the kth component of V. The resultant schema is S(V).

Let Σ be the set of AFWFS's $\{S(V) | V$ is a valid conditional assignment for $S_e\}$ and let Σ' be the set $\{S(V') | V'$ is a valid conditional assignment for $S_e'\}$. (The cardinality of either set is less than 3^{ℓ}.) Now, S and S' are not equivalent if and only if the following condition holds:

For some S(V) in Σ and S(V') in Σ', there exists finite consistent computations C by S(V) and C' by S(V') such that the expressions associated with the jth output links of S(V) and S(V') differ at the conclusion of the computations, for some j ≤ j and <u>all</u> j between j+1 and j+c_o such that the jth components of V and V' differ.

(a)



(b)

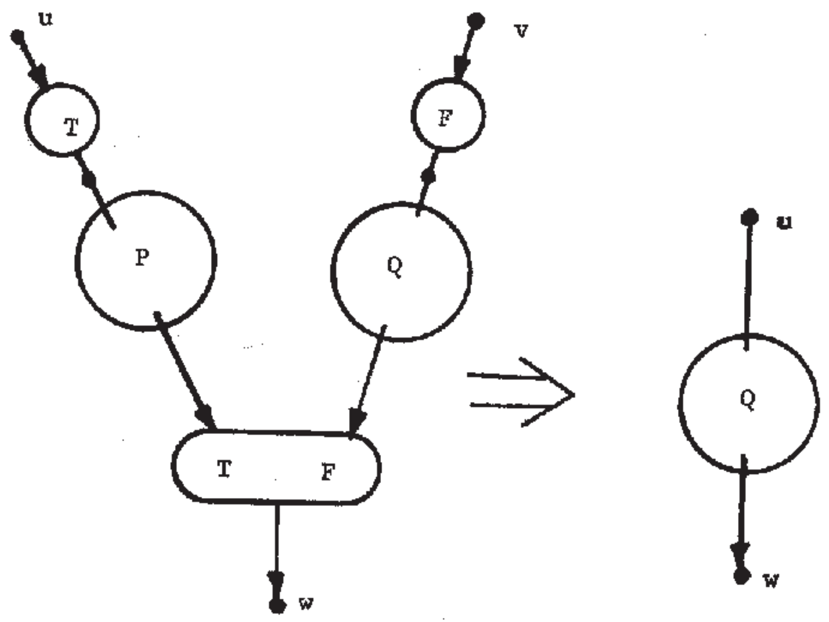


FIGURE 11.

By Lemma 3.1.1, we may compare for equivalence the pair of schemas formed by deleting from S(V) and S(V') all output links after the jth (and all links no longer on a path to a boundary link), and the pairs of schemas obtained from S(V) and S(V') by deleting, for each j such that the jth components of V and V' differ, all output links other than the $j+j$th output links. We thus compare for equivalence no more than $(\ell+1)$ pairs of schemas, each free of main conditionals, and we do these comparisons for each element in $\Sigma \times \Sigma'$; the result follows.

IX. Extensions of the Result

The result presented in this paper is a rather specialized result but suggests an approach to the problem of deciding equivalence in more general models. In particular, the following generalizations are suggested:

1. Demonstrating the decidability of equivalence for the class of FWFS's, i.e. elimination of Property A as a condition of the proof.
2. Demonstrating the decidability of equivalence for a class of schemas satisfying Property A in which operators with more than a single input link are permitted, and/or in which deciders with more than a single input link are permitted.
3. Demonstrating the decidability of equivalence for a class of AFWFS's in which the output links of deciders may be interconnected by a net of Boolean actors so that conditional and iteration constructs may be controlled by an interconnection of deciders, rather than by a single decider.

It is the opinion of the author that the first generalization is the most important of those suggested, since the FWFS's are capable of modelling the controls of some very interesting classes of automata; some progress in this direction has been made.

The second generalization is also of interest: an analysis of the results presented here demonstrate that the generalization to n-ary deciders presents little more than notational difficulties; the generalization to n-ary operators also appears straightforward.

It is not clear precisely what problems are caused by the existence of Boolean actors, but it is felt that, provided only free schemas are examined, the problems introduced will be minor.

References

1. Dennis, J. B., and J. B. Fosseen. An Introduction to Data Flow Schemas. Computation Structures Group Memo 81, Project MAC, M.I.T., Cambridge, Mass., September 1972.

2. Leung, C. Unpublished notes.

3. Patil, S. S. Closure properties of interconnections of determinate systems. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York, 1970, pp 107-116.

4. Valiant, L. Decidability of equivalence for finite-turn deterministic pushdown automata. Symposium on Theory of Computing, ACM, April 1974.

APPENDIX

Examples of Well-Formed Unary Operator Data Flow Schemas

The following "programs" are represented by the schemas of Figure A1:

(a) S_1: $\{w\} \rightarrow \{z\}$

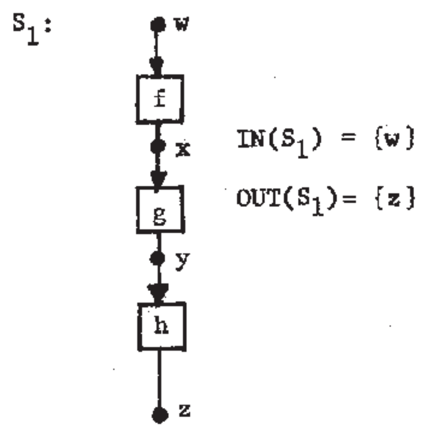
```
begin;
x:=f(w);
y;=g(x);
z:=h(y);
end;
```

(b) S_2: $\{u,v\} \rightarrow \{z\}$

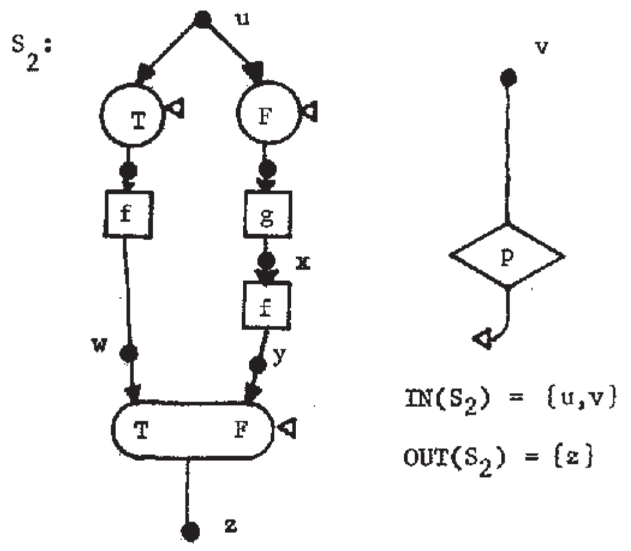
```
begin;
IF p(v) then do;
        w:=f(u);
        z:=w;
        end;
        else do;
        x:=g(w);
        y:=f(x);
        z:=y;
        end;
end;
```

(c) S_3: $\{w,x\} \rightarrow \{y,z\}$

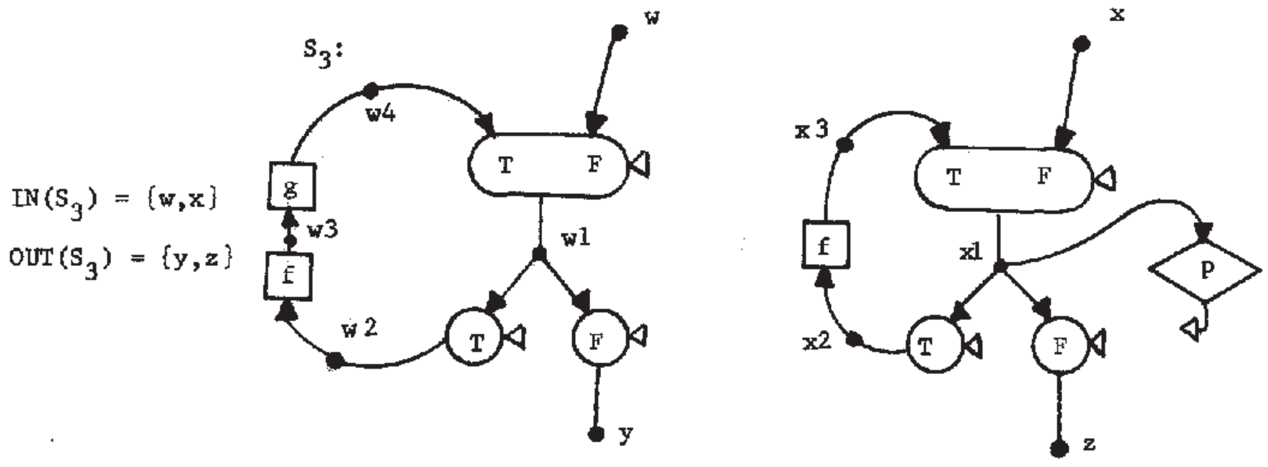
```
begin;
w1:=w;
x1:=x;
WHILE p(x1) do;
           w2:=w1;
           w3:=f(w2);
           w4:=g(w3);
           w1:=w4;
           x2:=x1
           x3:=f(x2);
           x1:=x3;
           end;
y:=w1;
z:=x1;
end;
```



(a) Operator Schema



(b) Conditional Schema

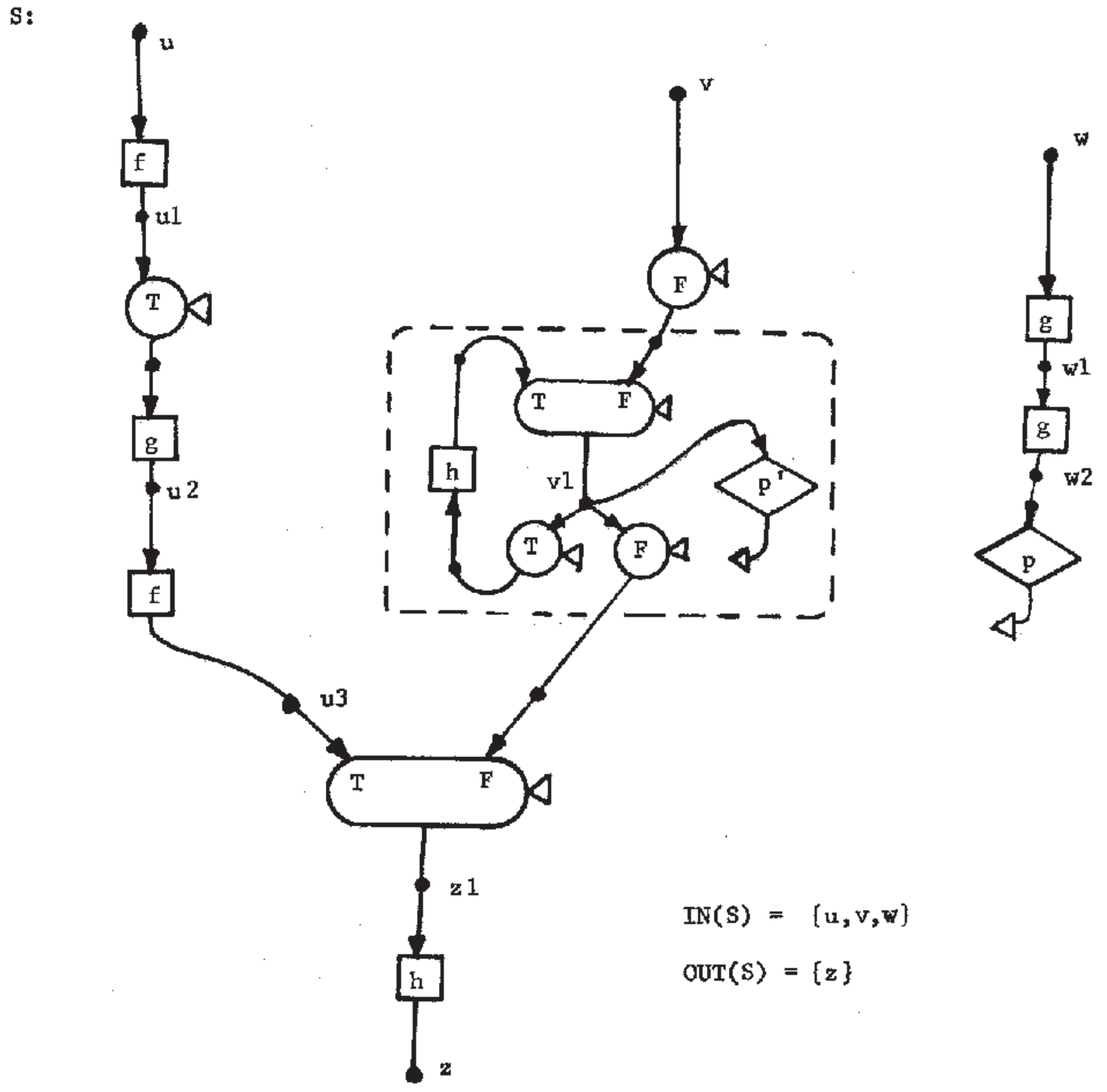


(c) Iteration Schema

FIGURE A1.

The following "program" is represented by schema S, Figure A2:

```
S:{ u,v,w } → { z }
begin;
u1:=f(u);
w1:=g(w);
w2;=g(w1);
IF p(w2) then do;
          u2:=g(u1);
          u3:=f(u2);
          z1:=u3;
          end;
        else do;
          v1:=v;
          WHILE p'(v1) do;
                v1:=h(v1);
                end;
          z1:=v1;
          end;
z:=h(z1);
end;
```

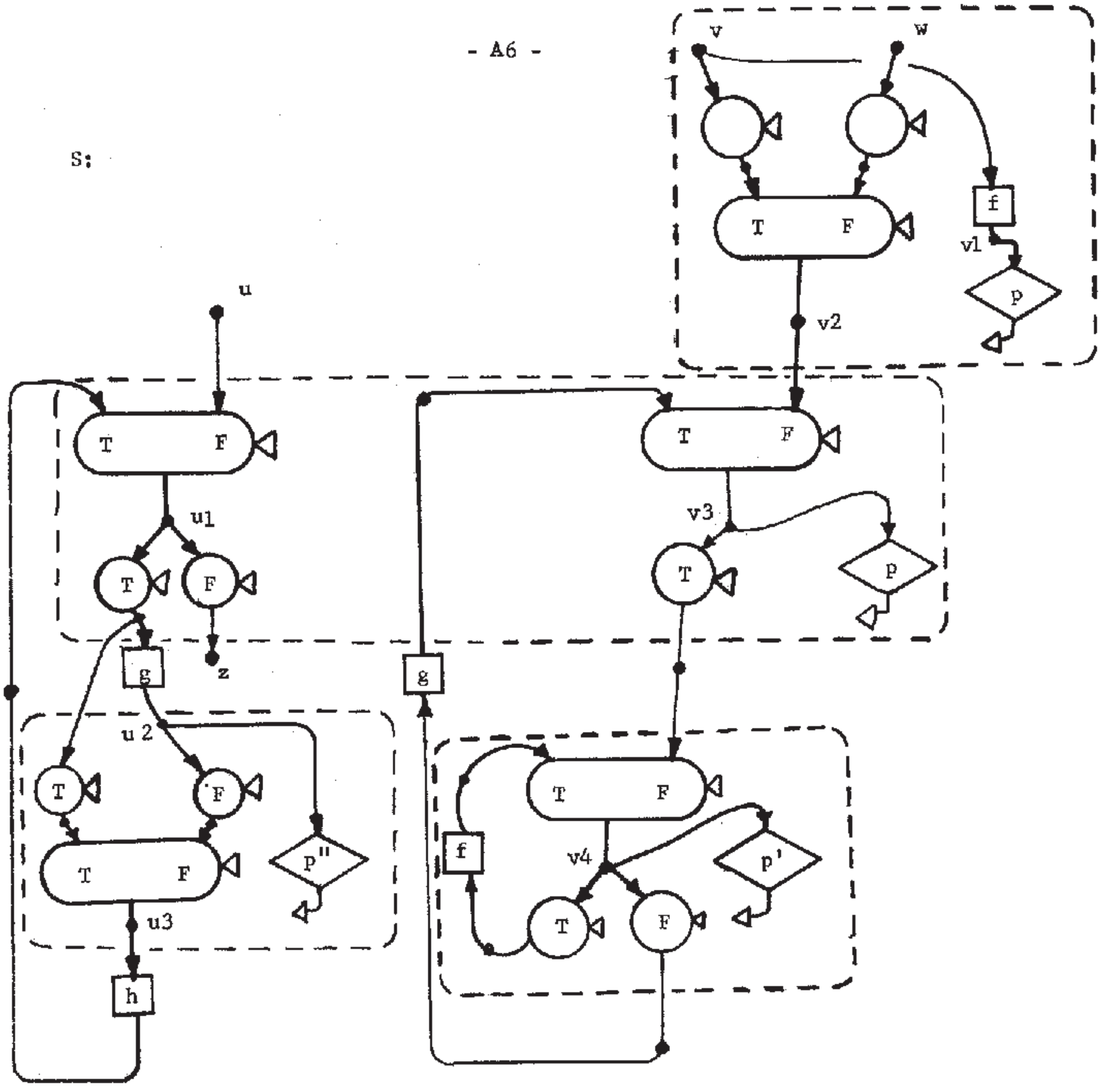


IN(S) = {u,v,w}

OUT(S) = {z}

FIGURE A2.

S:



IN(S) = {u,v,w}

OUT(S) = {z}

FIGURE A3.