JOURNAL OF COMPUTER AND SYSTEM SCIENCES: 3, 361–386 (1969)

Regular Expressions and the Equivalence of Programs

DONALD M. KAPLAN*

Department of Computer Science,† Stanford University, Stanford, California

Received October 14, 1968

If we assume that the study and detection of equivalence for ALGOL-like programs holds certain pragmatic interest, then it seems reasonable to pursue these matters despite the well-known undecidability of this property.

Various efforts have been made to isolate decidable sub-cases of this equivalence problem (e.g., by Paterson [*12*] and this author [*6*]). Other efforts have been made to define weaker, and therefore decidable, sorts of equivalence (e.g., by Ianov [*4*] and Rutledge [*14*]). Our interest here is to develop equivalence detecting procedures applicable to programs for which equivalence is undecidable. These procedures always produce an answer when questioned as to the equivalence of two programs: either YES or MAYBE. We consider a sequence of these procedures, each more powerful than the preceding ones. Thus, if one procedure returns MAYBE, then perhaps a subsequent more powerful one will return YES.

ELEMENTAL PROGRAMS

We consider here the class of *elemental programs*. Each of these is simply a flowchart comprised of ALGOL-like *assignment statements* and two-way branches on the truth-value of quantifier-free formulas of the first order predicate calculus (abbreviated as *qffs*). To simplify the discussion here, we assume that elemental programs have but one *entrance* and one *exit*. An example of an elemental program is illustrated in Fig. 1.

Now for the syntactic details. An elemental program, $\mathfrak{A}$ say, is a special sort of directed labelled graph (i.e., a flowchart), which we define by a triple $\langle X, \Gamma, \mathscr{L} \rangle$. Here, $X = Y \cup \{¢\}$ is a finite set of nodes, such that $¢ \notin Y$ and $\$ \in Y$, where $\$$ and $¢$ are special nodes called the *entrance* and *exit* of $\mathfrak{A}$, respectively. Then, $\Gamma : Y \to X \cup X^2$ gives the flow of control for $\mathfrak{A}$, and $\mathscr{L} : Y \to \mathscr{A} \cup \mathscr{Q}$ gives a labelling for the nodes of $\mathfrak{A}$ with assignment statements from $\mathscr{A}$ and qffs from $\mathscr{Q}$. This labelling is restricted

* Present address: Department of Computer Science, University of Toronto, Toronto 5, Canada.

† The research reported here was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183) and is part of the National Research Council of Canada.

so that for all $y \in Y$, $\mathscr{L}(y) \in \mathscr{A} \Leftrightarrow \Gamma y \in X$ and $\mathscr{L}(y) \in \mathscr{Q} \Leftrightarrow \Gamma y \in X^2$; thus qffs label only two-way branches, and assignment statements label only one-way branches. In the sequel, we denote the label of a node x by $[x]$.

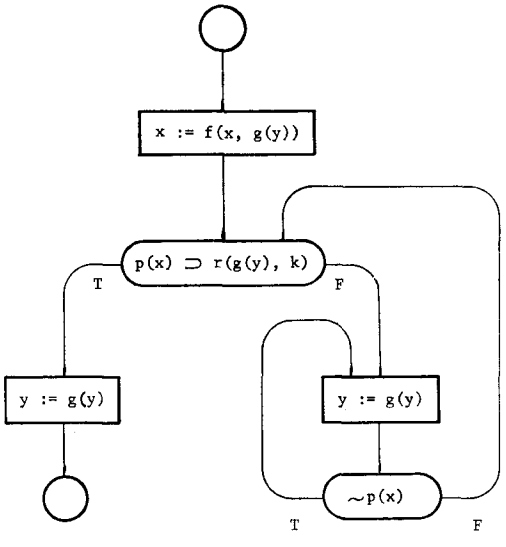

FIG. 1. The elemental program $\mathfrak{C}$. Here, x and y are variables; f and g are function letters; p and r are relation letters; and k is a constant letter.

Before defining the sets $\mathscr{A}$ and $\mathscr{Q}$ of assignment statements and qffs respectively, let us define terms. A *term* is either one of a set $\{v_i\}_{i<\omega}$ of *variables*; one of a set $\{k_i\}_{i<\omega}$ of constant letters; or an expression of the form $f(\sigma,..., \tau)$, where f is one of a countable set of *function letters* and $\sigma,..., \tau$ are terms.

Then an assignment statement is an expression of the form $(u := \tau)$, where τ is a term, and u is a variable, in the sequel called the *assigned variable.*

A qff is either an expression of the form $r(\sigma,..., \tau)$, where r is one of a countable set of *relation letters*, and $\sigma,..., \tau$ are terms; or an expression of the form $(p \supset q)$ or $(\sim p)$, where p and q are qffs.

Now for the semantic details. Meaning is given to an elemental program through a mathematical system, called a *computing structure*, of the sort usually used to give interpretation to qffs of the first-order predicate calculus. A computing structure is a domain and a mapping of the function letters, relation letters and constant letters into functions, relations and constants on that domain. We use D_0 to denote the domain of a computing structure D and $D(l)$ to denote the interpretation in D of the letter l.

The semantics of terms, assignment statements, qffs and elemental programs is defined with respect to a computing structure, D say, and some sequence $\xi : \omega \to D_0$

called a *state*. The state provides a mapping from the variables into values in the domain; thus, the variable v_i has $\xi(i)$ as its *value*.

In general, the *value* $\zeta[D, \xi]$ of a term ζ with respect to the computing structure D and state ξ is computed using[1]

$$\begin{aligned}\zeta[D, \xi] = \ & \textit{if } \zeta = v_i \textit{ then } \xi(i)\\ & \textit{else if } \zeta = k_i \textit{ then } D(k_i)\\ & \textit{else if } \zeta = f(\sigma,..., \tau) \textit{ then } D(f)(\sigma[D, \xi],..., \tau[D, \xi]).\end{aligned}$$

The new state $(v_i := \tau)[D, \xi]$, which results from the execution of the assignment statement $(v_i := \tau)$ on the state ξ, is computed simply by replacing the ith element of ξ with $\tau[D, \xi]$. That is, the sequence $(v_i := \tau)[D, \xi] : \omega \to D_0$ is given by

$$(v_i := \tau)[D, \xi](j) = \textit{if } i = j \textit{ then } \tau[D, \xi] \textit{ else } \xi(j),$$

for all $j < \omega$.

The *truth-value* $s[D, \xi] \in \{T, F\}$ of a qff s with respect to the computing structure D and state ξ is computed using

$$\begin{aligned}s[D, \xi] = \ & \textit{if } s = r(\sigma,..., \tau) \textit{ then}\\ & \qquad \textit{if } D(r)(\sigma[D, \xi],..., \tau[D, \xi]) \textit{ then } T \textit{ else } F\\ & \textit{else if } s = (p \supset q) \textit{ then}\\ & \qquad \textit{if } p[D, \xi] = F \textit{ or } q[D, \xi] = T \textit{ then } T \textit{ else } F\\ & \textit{else if } s = (\sim p) \textit{ then}\\ & \qquad \textit{if } p[D, \xi] = F \textit{ then } T \textit{ else } F.\end{aligned}$$

The *output state* $\mathfrak{A}[D, \xi]$ of an elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L}\rangle$ *executed in* the computing structure D with *input state* ξ is computed by the (partial) *execution function* E. That is, $\mathfrak{A}[D, \xi] = E(\mathfrak{A}, D, \xi, \$)$, where for any $x \in X$

$$\begin{aligned}E(\mathfrak{A}, D, \xi, x) = \ & \textit{if } [x] \in \mathscr{A} \textit{ and } \Gamma x = y \textit{ then } E(\mathfrak{A}, D, [x][D, \xi], y)\\ & \textit{else if } [x] \in \mathscr{Q} \textit{ and } \Gamma x = \langle y, z\rangle \textit{ then}\\ & \qquad \textit{if } [x][D, \xi] = T \textit{ then } E(\mathfrak{A}, D, \xi, y) \textit{ else } E(\mathfrak{A}, D, \xi, z)\\ & \textit{else if } x = \mathfrak{c} \textit{ then } \xi.\end{aligned}$$

So we see that elemental programs are executed precisely as our intuition would indicate. If the exit is reached, then $\mathfrak{A}[D, \xi]$ is *determinate*; otherwise $\mathfrak{A}[D, \xi]$ is *indeterminate*. Models of computation similar to the one presented here have been studied by many authors, e.g., Ershov [2], Luckham and Park [8], Narasimhan [11], Engeler [1], Paterson [12], Manna [10] and this author [6].

[1] We use here and in the sequel the meta-formalism of McCarthy [9] for recursively defined functions. Implicit use is also made throughout of his axioms for manipulating the conditional expressions appearing in these definitions.

Strong Equivalence

We say that the elemental programs $\mathfrak{A}$ and $\mathfrak{B}$ are *strongly equivalent*, and write $\mathfrak{A} \simeq \mathfrak{B}$, if and only if for all computing structures D and input states $\xi : \omega \to D_0$ we have $\mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$, i.e., either both $\mathfrak{A}[D, \xi]$ and $\mathfrak{B}[D, \xi]$ are indeterminate or both are determinate and $\mathfrak{A}[D, \xi] = \mathfrak{B}[D, \xi]$.

There is no effective procedure for determining whether or not two elemental programs are strongly equivalent. Luckham and Park [*8*], Paterson [*12*] and this author [*6*] have all shown this. Our aim here is to develop techniques of analysis that will to some extent alleviate this lack of an overall effective procedure for detecting strong equivalence.

Regular Expression Representation

To aid in the development of these techniques we introduce a *regular expression representation* for elemental programs. Such a representation is obtained in a very simple fashion. For example, consider the following regular expression representation of the elemental program in Fig. 1:

$$f(\sim(p \supset r)\, g(\sim pg)^* \sim \sim p)^*(p \supset r)\, g,$$

where $x := f(x, g(y))$ has been abbreviated to simply f, $p(x)$ to p, $r(g(y), k)$ to r, and $y := g(y)$ to g. This representation not only captures the graph theoretic properties of elemental programs, but, in addition, faithfully characterizes the T and F branching at nodes labelled with qffs.

Before proceeding, we will repeat here the basic definitions associated with regular expressions. This material is also given by Harrison [*3*], Salomaa [*15*] and many others; we include it here only to avoid notational misunderstandings.

Let $\Sigma = \{a, b, \ldots, z\}$ be a finite *alphabet*; here, each *letter* in the alphabet is some formal expression, i.e., perhaps a sequence of symbols over some lower level alphabet. This possibility does not concern us just now, however. Then, a regular expression over Σ is either one of the symbols 0 or 1; a letter in Σ; or an expression of the form $(\alpha \vee \beta)$, α^* or $(\alpha \cdot \beta)$, where α and β are regular expressions. In practice we usually omit the "$\cdot$" and certain parentheses, with the understanding that the operations are performed in the order "$*$", "$\cdot$", "$\vee$". Thus, $\alpha \vee \beta\gamma^*$ will be written instead of $(\alpha \vee (\beta \cdot \gamma^*))$.

The semantics of a regular expression over Σ is called a *regular event* and is a sub-set of Σ^*, the set of all finite *words* (including the empty word Λ) over the alphabet Σ. The regular event $|\gamma|$ associated with the regular expression γ is computed using

$$\begin{aligned} |\gamma| = {} & \textit{if } \gamma = 0 \textit{ then } \varnothing \text{ (i.e., the empty set)} \\ & \textit{else if } \gamma = 1 \textit{ then } \{\Lambda\} \\ & \textit{else if } \gamma \in \Sigma \textit{ then } \{\gamma\} \end{aligned}$$

$$\textit{else if } \gamma = \alpha \vee \beta \textit{ then } | \alpha | \cup | \beta |$$
$$\textit{else if } \gamma = \alpha^* \textit{ then } | 1 | \cup | \alpha | \cup | \alpha\alpha | \cup | \alpha\alpha\alpha | \cup \cdots$$
$$\textit{else if } \gamma = \alpha\beta \textit{ then } \{ab : a \in | \alpha | \ \& \ b \in | \beta |\}.$$

With an elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$, where $R(\mathscr{L})$ denotes the range of $\mathscr{L} : X \to \mathscr{A} \cup \mathscr{Q}$, we associate the finite alphabet

$$\Sigma_{\mathfrak{A}} = R(\mathscr{L}) \cup \{\sim p : p \in R(\mathscr{L}) \cap \mathscr{Q}\}.$$

In the sequel, we often write $\bar{p}$ instead of $\sim p$ for qffs p.

We will define $\alpha_{\mathfrak{A}}$, a regular expression over $\Sigma_{\mathfrak{A}}$ that serves to represent the elemental program $\mathfrak{A}$, by utilizing a nondeterministic finite automaton $M_{\mathfrak{A}}$ (as introduced by Rabin and Scott [*13*]) whose *behavior* is just $| \alpha_{\mathfrak{A}} |$. That is, given a word $x \in \Sigma_{\mathfrak{A}}{}^*$ as input, $M_{\mathfrak{A}}$ *accepts* x, i.e., stops in the *final state*, if and only if $x \in | \alpha_{\mathfrak{A}} |$. Let us define these ideas in more detail.

From the elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$, we effectively construct the nondeterministic finite automaton $M_{\mathfrak{A}} = \langle X, T, \Sigma_{\mathfrak{A}} \rangle$. In this context, X is a finite set of *automaton states* (or simply *states* if no confusion with states as sequences over a domain results). As well, \$ and ¢ in X are now called the *start state* and *final state* respectively. The (partial) *transition function* $T : X \times \Sigma_{\mathfrak{A}} \to X$ is defined by

$$T(x, \sigma) = \textit{if } [x] \in \mathscr{A} \textit{ and } \Gamma x = y \textit{ and } \sigma = [x] \textit{ then } y$$
$$\textit{else if } [x] \in \mathscr{Q} \textit{ and } \Gamma x = \langle y, z \rangle \textit{ then}$$
$$\textit{if } \sigma = [x] \textit{ then } y \textit{ else if } \sigma = \sim[x] \textit{ then } z.$$

The state-transition diagram for the nondeterministic finite automaton corresponding to the elemental program $\mathfrak{C}$ in Fig. 1 is shown in Fig. 2. We see that, in fact, the formation of $M_{\mathfrak{A}}$ from $\mathfrak{A}$ is really a trivial operation.

The behavior of the automaton $M_{\mathfrak{A}}$ is simply the subset of $\Sigma_{\mathfrak{A}}{}^*$ that $M_{\mathfrak{A}}$ *accepts*, i.e., those words that cause $M_{\mathfrak{A}}$ to go from the start state of the final state via the

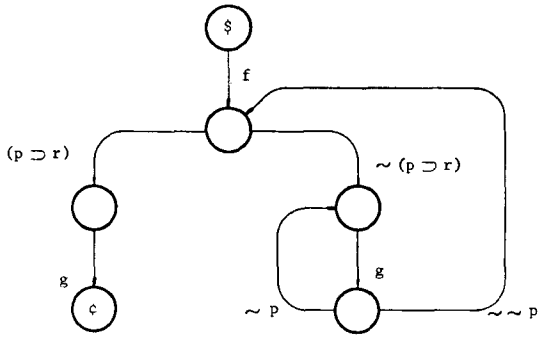

FIG. 2. The state-transition diagram for the nondeterministic finite automaton corresponding to the elemental program $\mathfrak{C}$ in Fig. 1. Here, the abbreviations mentioned earlier in the text are used.

transition function. The (partial) *acceptor function* $T^* : X \times \Sigma_{\mathfrak{A}}{}^* \to X$ is defined by

$$T^*(x, w) = \textit{if } w = \Lambda \textit{ then } x$$
$$\textit{else if } w = \sigma u \textit{ and } \sigma \in \Sigma_{\mathfrak{A}} \textit{ then } T^*(T(x, \sigma), u).$$

Then the *behavior* of $M_{\mathfrak{A}}$ is $B_{\mathfrak{A}} = \{w \in \Sigma_{\mathfrak{A}}{}^* : T^*(\$, w) = \not\in\}$.

THEOREM 1. *There exists an effective procedure, which for any elemental program* $\mathfrak{A}$ *constructs a regular expression* $\alpha_{\mathfrak{A}}$ *such that* $| \alpha_{\mathfrak{A}} | = B_{\mathfrak{A}}$.

Proof. We have shown how to effectively construct the nondeterministic finite automaton $M_{\mathfrak{A}}$ from $\mathfrak{A}$. Rabin and Scott [*13*, Theorem 11] show that $B_{\mathfrak{A}}$, the behavior of $M_{\mathfrak{A}}$, is also the behavior of a certain effectively constructed *deterministic* finite automaton $M_{\mathfrak{A}}'$. Then, using Kleene's result [7], we can effectively construct from $M_{\mathfrak{A}}'$ a regular expression $\alpha_{\mathfrak{A}}$ such that $| \alpha_{\mathfrak{A}} |$ is the behavior of $M_{\mathfrak{A}}'$, i.e., such that $| \alpha_{\mathfrak{A}} | = B_{\mathfrak{A}}$. ∎

A regular expression $\alpha_{\mathfrak{A}}$ such that $| \alpha_{\mathfrak{A}} |$ is the behavior of $M_{\mathfrak{A}}$ is called a *regular expression representation* of the elemental program $\mathfrak{A}$.

DETECTION OF STRONG EQUIVALENCE: PROCEDURE R

We say that the expression $\alpha = \beta$, where α and β are regular expressions over the same alphabet, is a *well-formed R-formula* (abbreirated as *R-wff*). Just in case $| \alpha | = | \beta |$, we say that α is *R-equivalent* to β and write $\models_R \alpha = \beta$ to indicate that the R-wff $\alpha = \beta$ is then *R-valid.*

The first procedure for detecting strong equivalence is based on the fact that if the regular expression representations of two elemental programs are R-equivalent, then the programs are strongly equivalent. To arrive at this results, we need a definition, four lemmas and a theorem.

With each execution of an elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{P} \rangle$ we can associate a (possibly infinite) word over $\Sigma_{\mathfrak{A}}$. We build up this word by starting with the empty word and adding, on the right, letters from $\Sigma_{\mathfrak{A}}$ as execution proceeds. If an assignment statement $f \in \Sigma_{\mathfrak{A}}$ is encountered, then we add f; at a qff p, if the T branch is taken then we add $p \in \Sigma_{\mathfrak{A}}$; if the F branch is taken, then we add $\bar{p} \in \Sigma_{\mathfrak{A}}$. More specifically, we associate with the output state $\mathfrak{A}[D, \xi]$ a word $\mathfrak{A}^*[D, \xi] = W(\mathfrak{A}, D, \xi, \$)$, where for any $x \in X$,

$$W(\mathfrak{A}, D, \xi, x) = \textit{if } [x] \in \mathscr{A} \textit{ and } \Gamma x = y \textit{ then } [x]\ W(\mathfrak{A}, D, [x][D, \xi], y)$$
$$\textit{else if } [x] \in \mathscr{Q} \textit{ and } \Gamma x = \langle y, z \rangle \textit{ then}$$
$$\textit{if } [x][D, \xi] = T \textit{ then } [x]\ W(\mathfrak{A}, D, \xi, y) \textit{ else}$$
$$\sim[x]\ W(\mathfrak{A}, D, \xi, z)$$
$$\textit{else if } x = \not\in \textit{ then } \Lambda.$$

Appropriately enough, the definition of W is much akin to the definition of the execution function E given earlier. Two results stem immediately from this fact.

LEMMA 1. *For any elemental program* $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$ *with regular expression representation* $\alpha_{\mathfrak{A}}$, *computing structure* D *and state* $\xi : \omega \to D_0$, *if* $\mathfrak{A}[D, \xi]$ *is determinate then* $\mathfrak{A}^*[D, \xi] \in | \alpha_{\mathfrak{A}} |$.

LEMMA 2. *For any two elemental programs* $\mathfrak{A}$ *and* $\mathfrak{B}$, *computing structure* D *and state* $\xi : \omega \to D_0$, $\mathfrak{A}^*[D, \xi] = \mathfrak{B}^*[D, \xi] \Rightarrow \mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$.

The first result is obvious once we note that the function W, when building up the word $\mathfrak{A}^*[D, \xi]$, adds just those letters that will keep the automaton $M_{\mathfrak{A}}$ "in the track"; that is, given $\mathfrak{A}^*[D, \xi]$ as input, $M_{\mathfrak{A}}$ never enters a state $x \neq \text{¢}$ from which a transition cannot be made, Thus, if execution of $\mathfrak{A}$ by E, and so word building by W reaches the exit ¢, then $M_{\mathfrak{A}}$ reaches the final state ¢, and $\mathfrak{A}^*[D, \xi] \in | \alpha_{\mathfrak{A}} |$. In [6, Theorem 19] a detailed proof by induction is given for this result.

The second result is equally straightforward. For, a comparison of definitions for the functions W and E shows that $\mathfrak{A}^*[D, \xi] = \mathfrak{B}^*[D, \xi]$ implies $\mathfrak{A}[D, \xi]$ and $\mathfrak{B}[D, \xi]$ arise from the application of identical sequences of assignment statements to the initial state ξ. Thus, the final states $\mathfrak{A}[D, \xi]$ and $\mathfrak{B}[D, \xi]$, if determinate, are identical. In [6, Theorem 20] a detailed proof by induction is given for this result.

We want now to obtain a necessary and sufficient condition for $w \in | \alpha_{\mathfrak{A}} |$ to be the word associated with a given execution of the elemental program $\mathfrak{A}$. To do this, we first generalize the notion of regular expression representation and then introduce *initial conditions.*

For any elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$, Theorem 1 guarantees the existence of a regular expression $\alpha_{\mathfrak{A}}$ such that

$$| \alpha_{\mathfrak{A}} | = \{w \in \Sigma_{\mathfrak{A}}^* : T^*(\$, w) = \text{¢}\}.$$

A simple generalization of this result assures, for each $x \in X$, the existence of a regular expression $\alpha_{\mathfrak{A}}(x)$ such that $| \alpha_{\mathfrak{A}}(x) | = \{w \in \Sigma_{\mathfrak{A}}^* : T^*(x, w) = \text{¢}\}$. Notice that we can take $\alpha_{\mathfrak{A}}(\$)$ as $\alpha_{\mathfrak{A}}$ and $\alpha_{\mathfrak{A}}(\text{¢})$ as 1.

A word $w \in | \alpha_{\mathfrak{A}}(x) |$ can be regarded as an alternating sequence of letters from $\mathscr{A}$ and words from $\mathscr{Q}^*$, the set of all finite words over $\mathscr{Q}$.

$$w = PfQg \cdots hR \begin{cases} P, Q, \ldots, R \in \mathscr{Q}^* \\ f, g, \ldots, h \in \mathscr{A}. \end{cases}$$

For a word $S \in \mathscr{Q}^*$, let us write S' for the logical conjunction[2] of the letters in S; if S is the empty word, then S' can be any tautology. Also, for any $(u := \tau) \in \mathscr{A}$ and $p \in \mathscr{Q}$, let us write $(u := \tau) \to p$ for the qff obtained from p by a syntactic

[2] For example, the logical conjunction of the qffs p and q can be expressed as $\sim(p \supset \sim q)$.

substitution[3] of the term τ for each occurrence in the qff p of the variable u. A useful property of this substitution operation is given by

LEMMA 3. *For any assignment statement* $f \in \mathscr{A}$, qff $p \in \mathscr{Q}$, *computing structure* D *and state* $\xi : \omega \to D_0$, $f \to p[D, \xi] = T \Leftrightarrow p[D, f[D, \xi]] = T$.

A straightforward proof by induction on the structure of the qff p is given in [6, Theorem 6] for this result. Then, for any word $w \in |\alpha_{\mathfrak{A}}(x)|$ of the form $PfQg \cdots hR$, we define the *initial condition* of w to be

$$I(w) = (P' f \to Q' \ \cdots \ f \to g \cdots h \to R')'.$$

That $I(w)$ is the sought after necessary and sufficient condition on w is expressed in

LEMMA 4. *For any elemental program* $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$ *with regular expression representation* $\alpha_{\mathfrak{A}}$, *computing structure* D, *state* $\xi : \omega \to D_0$ *and word* $w \in |\alpha_{\mathfrak{A}}|$, *$w$ is the word associated with the execution of $\mathfrak{A}$ in D with initial state ξ if and only if the initial condition on w has truth value T with respect to D and ξ, or in symbols,* $\mathfrak{A}^*[D, \xi] = w \Leftrightarrow I(w)[D, \xi] = T$.

Proof. We prove by induction the more general result that for any $x \in X$ and $w \in |\alpha_{\mathfrak{A}}(x)|$,

$$W(\mathfrak{A}, D, \xi, x) = w \Leftrightarrow I(w)[D, \xi] = T.$$

Case (i): $x = \phi$. Here, $\alpha_{\mathfrak{A}}(\phi) = 1$ so that $|\alpha_{\mathfrak{A}}(\phi)| = \{\Lambda\}$. Now, $I(\Lambda)$ is some tautology, so that $I(\Lambda)[D, \xi] = T$ independently of D and ξ. As well, $W(\mathfrak{A}, D, \xi, \phi) = \Lambda$ independently of D and ξ. Hence, $W(\mathfrak{A}, D, \xi, \phi) = w \Leftrightarrow I(w)[D, \xi] = T$, for any $w \in |\alpha_{\mathfrak{A}}(\phi)|$, i.e., for $w = \Lambda$.

Case (ii): $[x] \in \mathscr{A}$. Here, $\alpha_{\mathfrak{A}}(x) = [x]\,\alpha_{\mathfrak{A}}(\Gamma x)$, since any other initial letters for words in $|\alpha_{\mathfrak{A}}(x)|$ would not permit a transition of $M_{\mathfrak{A}}$ to another state, and so the final state ϕ would not be reached. Thus, $w \in |\alpha_{\mathfrak{A}}(x)|$ is of the form $[x]u$, where $u \in |\alpha_{\mathfrak{A}}(\Gamma x)|$. The induction hypothesis here is that

$$W(\mathfrak{A}, D, [x][D, \xi], \Gamma x) = u \Leftrightarrow I(u)[D, [x][D, \xi]] = T.$$

Let us write $w \in |\alpha_{\mathfrak{A}}(x)|$ in the form $PfQg \cdots hR$, where $P = \Lambda, f = [x], Q, \ldots, R \in \mathscr{Q}^*$ and $g, \ldots, h \in \mathscr{A}$. Then, using the fact that P' is a tautology so that $P'[D, \xi] = T$, we have

$$\begin{aligned} I(w)[D, \xi] = T & \\ \Leftrightarrow I(PfQg \cdots hR)[D, \xi] = T & \qquad \text{def}^n\ w \\ \Leftrightarrow (P' f \to Q' \ \cdots \ f \to g \cdots h \to R')'[D, \xi] = T & \qquad \text{def}^n\ I \end{aligned}$$

[3] Notice that $f \to g \cdots h \to p$, where $f, g, \ldots, h \in \mathscr{F}$ and $p \in \mathscr{Q}$, is the denotation for $(f \to (g \to (\ldots \to (h \to p) \ldots)))$.

$$\Leftrightarrow P'[D, \xi] = f \to Q'[D, \xi] = \cdots = f \to g \cdots h \to R'[D, \xi] = T \quad \text{def}^n\,'$$
$$\Leftrightarrow f \to Q'[D, \xi] = \cdots = f \to g \cdots h \to R'[D, \xi] = T \quad P'[D, \xi] = T$$
$$\Leftrightarrow Q'[D, f[D, \xi]] = \cdots = g \cdots h \to R'[D, f[D, \xi]] = T \quad \text{Lemma 3}$$
$$\Leftrightarrow (Q' \cdots g \cdots h \to R')'[D, f[D, \xi]] = T \quad \text{def}^n\,'$$
$$\Leftrightarrow I(u)[D, f[D, \xi]] = T \quad \text{def}^n\,I$$
$$\Leftrightarrow W(\mathfrak{A}, D, [x][D, \xi], \Gamma x) = u \quad \text{ind. hyp.}$$
$$\Leftrightarrow W(\mathfrak{A}, D, \xi, x) = [x]\, W(\mathfrak{A}, D, [x][D, \xi], \Gamma x) \text{ and } W(\mathfrak{A}, D, [x][D, \xi], \Gamma x) = u \quad \text{def}^n\,W$$
$$\Leftrightarrow W(\mathfrak{A}, D, \xi, x) = [x]u \quad \text{def}^n =$$
$$\Leftrightarrow W(\mathfrak{A}, D, \xi, x) = w \quad \text{def}^n\,w$$

Case (iii): $[x] \in \mathscr{Q}$. Here, if $\Gamma x = \langle y, z \rangle$, then

$$\alpha_{\mathfrak{A}}(x) = [x]\, \alpha_{\mathfrak{A}}(y) \vee {\sim}[x]\, \alpha_{\mathfrak{A}}(z),$$

since any other initial letters for words in $|\alpha_{\mathfrak{A}}(x)|$ would not permit a transition of $M_{\mathfrak{A}}$ to another state and so the final state $\notin$ would not be reached. Thus, $w \in |\alpha_{\mathfrak{A}}(x)|$ is either of the form $[x]u$, where $u \in |\alpha_{\mathfrak{A}}(y)|$, or ${\sim}[x]v$, where $v \in |\alpha_{\mathfrak{A}}(z)|$. The induction hypothesis here is that

$$\textit{if } w = [x]u \textit{ then } W(\mathfrak{A}, D, \xi, y) = u \Leftrightarrow I(u)[D, \xi] = T$$
$$\textit{else if } w = {\sim}[x]v \textit{ then } W(\mathfrak{A}, D, \xi, z) = v \Leftrightarrow I(v)[D, \xi] = T.$$

Let us write $w \in |\alpha_{\mathfrak{A}}(x)|$ in the form $P^f Qg \cdots hR$, where $P = pS$, $p \in \{[x], {\sim}[x]\}$, $S \in \mathscr{Q}^*$, $f, g, \ldots, h \in \mathscr{A}$ and $Q, \ldots, R \in \mathscr{Q}^*$. Then,

$$I(w)[D, \xi] = T$$
$$\Leftrightarrow I(P^f Qg \cdots hR)[D, \xi] = T \quad \text{def}^n\,w$$
$$\Leftrightarrow (P'\, f \to Q' \cdots f \to g \cdots h \to R')'[D, \xi] = T \quad \text{def}^n\,I$$
$$\Leftrightarrow P'[D, \xi] = f \to Q'[D, \xi] = \cdots = f \to g \cdots h \to R'[D, \xi] = T \quad \text{def}^n\,'$$
$$\Leftrightarrow p[D, \xi] = S'[D, \xi] = f \to Q'[D, \xi] = \cdots = f \to g \cdots h \to R'[D, \xi] = T \quad \text{def}^n\,P$$
$$\Leftrightarrow p[D, \xi] = T \textit{ and } (S'\, f \to Q' \cdots f \to g \cdots h \to R')'[D, \xi] = T \quad \text{def}^n\,'$$
$$\Leftrightarrow \textit{if } w = [x]u \textit{ then } [x][D, \xi] = T \textit{ and } I(u)[D, \xi] = T$$
$$\textit{else if } w = {\sim}[x]v \textit{ then } {\sim}[x][D, \xi] = T \textit{ and } I(v)[D, \xi] = T \quad \text{def}^n\,w$$
$$\Leftrightarrow \textit{if } w = [x]u \textit{ then } W(\mathfrak{A}, D, \xi, x) = [x]\, W(\mathfrak{A}, D, \xi, y) \textit{ and } I(u)[D, \xi] = T$$
$$\textit{else if } w = {\sim}[x]v \textit{ then } W(\mathfrak{A}, D, \xi, x) = {\sim}[x]\, W(\mathfrak{A}, D, \xi, z) \textit{ and } I(v)[D, \xi] = T \quad \text{def}^n\,W$$

$\Leftrightarrow$ *if* $w = [x]u$ *then* $W(\mathfrak{A}, D, \xi, x) = [x]\ W(\mathfrak{A}, D, \xi, y)$
and $W(\mathfrak{A}, D, \xi, y) = u$
else if $w = \sim[x]v$ *then* $W(\mathfrak{A}, D, \xi, x) = \sim[x]\ W(\mathfrak{A}, D, \xi, z)$
and $W(\mathfrak{A}, D, \xi, z) = v$ ind. hyp.
$\Leftrightarrow$ *if* $w = [x]u$ *then* $W(\mathfrak{A}, D, \xi, x) = [x]u$
else if $w = \sim[x]v$ *then* $W(\mathfrak{A}, D, \xi, x) = \sim[x]v$ def^n =
$\Leftrightarrow W(\mathfrak{A}, D, \xi, x) = w.$ def^n =

This completes the induction.

If we take $x = \$$, then we have that for $w \in |\ \alpha_{\mathfrak{A}}(\$)|$,

$$W(\mathfrak{A}, D, \xi, \$) = w \Leftrightarrow I(w)[D, \xi] = T.$$

Since $\alpha_{\mathfrak{A}}(\$) = \alpha_{\mathfrak{A}}$ and $\mathfrak{A}^*[D, \xi] = W(\mathfrak{A}, D, \xi, \$)$, we have therefore that for $w \in |\ \alpha_{\mathfrak{A}}\ |$,

$$\mathfrak{A}^*[D, \xi] = w \Leftrightarrow I(w)[D, \xi] = T. \quad \blacksquare$$

The first strong equivalence-detecting procedure is then based on the following

THEOREM 2. *For any two elemental programs* $\mathfrak{A}$ *and* $\mathfrak{B}$ *with regular expression representations* $\alpha_{\mathfrak{A}}$ *and* $\alpha_{\mathfrak{B}}$ *formed over the alphabet* $\Sigma_{\mathfrak{A}} \cup \Sigma_{\mathfrak{B}}$, *if* $\alpha_{\mathfrak{A}}$ *is R-equivalent to* $\alpha_{\mathfrak{B}}$, *then* $\mathfrak{A}$ *is strongly equivalent to* $\mathfrak{B}$, *or in symbols,* $|\ \alpha_{\mathfrak{A}}\ | = |\ \alpha_{\mathfrak{B}}\ | \Rightarrow \mathfrak{A} \simeq \mathfrak{B}$.

Proof. Let us execute $\mathfrak{A}$ and $\mathfrak{B}$ in an arbitrary computing structure D with an arbitrary initial state $\xi : \omega \to D_0$. To show $\mathfrak{A} \simeq \mathfrak{B}$, it suffices to show that $\mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$.

Suppose $\mathfrak{A}[D, \xi]$ is determinate so that $\mathfrak{A}^*[D, \xi] \in |\ \alpha_{\mathfrak{A}}\ |$ by Lemma 1. Then, by Lemma 4, $I(\mathfrak{A}^*[D, \xi])[D, \xi] = T$. By hypothesis, $|\ \alpha_{\mathfrak{A}}\ | = |\ \alpha_{\mathfrak{B}}\ |$, so that $\mathfrak{A}^*[D, \xi] \in |\ \alpha_{\mathfrak{B}}\ |$ as well. Then, again by Lemma 4, $\mathfrak{A}^*[D, \xi] = \mathfrak{B}^*[D, \xi]$. Finally, Lemma 2 gives $\mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$.

A similar argument in case $\mathfrak{B}[D, \xi]$ is determinate yields $\mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$ as well. Of course, if neither $\mathfrak{A}[D, \xi]$ nor $\mathfrak{B}[D, \xi]$ is determinate, then $\mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$ here too. This covers all cases. $\blacksquare$

It is well-known that R-equivalence of regular expressions is a decidable property; in fact, Salomaa [*15*] has given a complete formal theory for R-equivalence. We will denote this theory as $\mathscr{T}_R$; the axiom schemata and rules of inference are as follows:

S1:	$\alpha \vee (\beta \vee \gamma) = (\alpha \vee \beta) \vee \gamma$	*S7:*	$\alpha 1 = \alpha$
S2:	$\alpha(\beta\gamma) = (\alpha\beta)\gamma$	*S8:*	$\alpha 0 = 0$
S3:	$\alpha \vee \beta = \beta \vee \alpha$	*S9:*	$\alpha \vee 0 = \alpha$
S4:	$\alpha(\beta \vee \gamma) = \alpha\beta \vee \alpha\gamma$	*S10:*	$\alpha^* = 1 \vee \alpha\alpha^*$
S5:	$(\alpha \vee \beta)\gamma = \alpha\gamma \vee \beta\gamma$	*S11:*	$\alpha^* = (1 \vee \alpha)^*$
S6:	$\alpha \vee \alpha = \alpha$		

R1: $\alpha = \beta \Rightarrow \beta = \alpha$

R2: $\alpha = \beta \Rightarrow \gamma(\alpha) = \gamma(\beta)$, where $\gamma(\beta)$ arises from $\gamma(\alpha)$ by a syntactic substitution of β for one or more occurrences of α in $\gamma(\alpha)$.

R3: $\alpha = \gamma \vee \beta\alpha \Rightarrow \alpha = \beta\gamma^*$, provided $\Lambda \notin | \beta |$; this is a decidable side condition.

Just in case an R-wff $\alpha = \beta$ is finitely derivable in $\mathscr{T}_R$, we write $\vdash_R \alpha = \beta$. For any theory and associated notion of validity, we say that the theory is *sound* if and only if all derivable wffs are valid, and *adequate* if and only if all valid wffs are derivable.

THEOREM 3. *The theory $\mathscr{T}_R$ is both sound and adequate, or in symbols, $\vdash_R \alpha = \beta \Leftrightarrow \models_R \alpha = \beta$ for any R-wff $\alpha = \beta$.*

This is the result obtained by Solomaa [*15*]. His method is to show how to construct a derivation in $\mathscr{T}_R$ of any R-valid R-wff. Thus, the theory $\mathscr{T}_R$ itself can serve as the basis for an R-equivalence decision procedure.

The first strong equivalence-detecting procedure can now be specified:

Procedure R: given elemental programs $\mathfrak{A}$ and $\mathfrak{B}$,

(i) *construct* $\alpha_{\mathfrak{A}}$ *and* $\alpha_{\mathfrak{B}}$ *(effective)*
(ii) *test for* $| \alpha_{\mathfrak{A}} | = | \alpha_{\mathfrak{B}} |$ *using* $\mathscr{T}_R$ *(effective)*
(iii) *if* $| \alpha_{\mathfrak{A}} | = | \alpha_{\mathfrak{B}} |$, *return* YES, *otherwise* MAYBE.

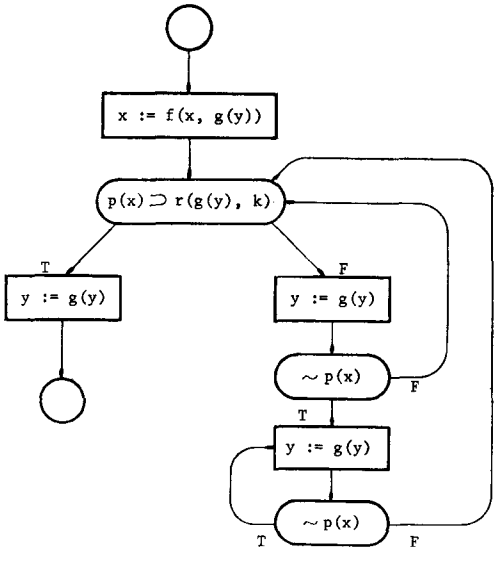

FIG. 3. An elemental program strongly equivalent to that in Fig. 1.

To consider an example, notice that Procedure R detects that the elemental program in Fig. 3 is strongly equivalent to that in Fig. 1. To see the limitations of Procedure R, notice that the elemental program in Fig. 4 is also strongly equivalent to that in Fig. 1, but that Procedure R fails to detect this fact. We now turn our attention to developing a second and somewhat more powerful procedure, which will be successful in this latter case as well.

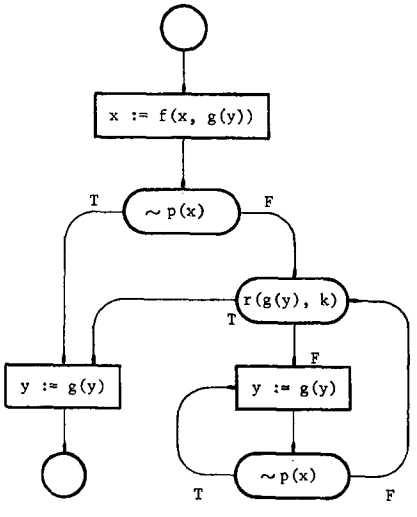

FIG. 4. Another elemental program strongly equivalent to that in Fig. 1.

K-EXPRESSION REPRESENTATION

The *K-expressions* are like regular expressions except that some letters have explicit propositional structure. Let us first give the basic definitions associated with K-expressions apart from their use in the detection of strong equivalence.

Let $\mathscr{F} = \{f, g, \ldots, h\}$ and $\mathscr{R} = \{p, q, \ldots, r\}$ be finite alphabets of *operators* and *propositional atoms* respectively. Then, a *proposition* over $\mathscr{R}$ is either one of the symbols 0 or 1; a letter in $\mathscr{R}$; or an expression of the form $(p \supset q)$ or $(\sim p)$, where p and q are propositions. We write $\mathscr{P}$ for the set of all such propositions. Then, a *K-expression* over $\mathscr{F} \cup \mathscr{R}$ is either an operator in $\mathscr{F}$; a proposition in $\mathscr{P}$; or an expression of the form $(\alpha \wedge \beta)$, α^* or $(\alpha \cdot \beta)$, where α and β are K-expressions. We omit parentheses and order the operations "*", "·", "∨" as in regular expressions.

The semantics of a K-expression over $\mathscr{F} \cup \mathscr{R}$ is called a *K-event* and is a subset of

$(\mathscr{F} \cup \mathscr{R})^*$, the set of all finite words over $\mathscr{F} \cup \mathscr{R}$. Playing a central role in the definition of K-event is the set $\mathscr{T}$ called *truth*:

$$\mathscr{T} = \{\hat{p}\hat{q} \cdots \hat{r} : \hat{p} \in \{p, \bar{p}\}, \hat{q} \in \{q, \bar{q}\}, \ldots, \hat{r} \in \{r, \bar{r}\}\},$$

where, recall, $\mathscr{R} = \{p, q, \ldots, r\}$. Thus, truth is just the set of all disjuncts in the full disjunctive normal form (abbreviated as *fdnf*) of a tautology in $\mathscr{P}$. Notice that for any computing structure D and state $\xi : \omega \to D_0$, there is one and only one $C \in \mathscr{T}$ such that $C'[D, \xi] = T$; we will denote this particular word of truth as $\mathscr{T}[D, \xi]$. The K-event $\| \gamma \|$ associated with a K-expression γ is computed using

$$\begin{aligned}
\| \gamma \| = {} & \textit{if } \gamma = 0 \textit{ then } \varnothing \\
& \textit{else if } \gamma = 1 \textit{ then } \mathscr{T} \\
& \textit{else if } \gamma \in \mathscr{R} \textit{ then } \{\hat{p}\hat{q} \cdots \hat{\gamma} \cdots \hat{r} \in \mathscr{T} : \hat{\gamma} = \gamma\} \\
& \textit{else if } \gamma = (p \supset q) \textit{ then } \|(\sim p)\| \cup \| q \| \\
& \textit{else if } \gamma = (\sim p) \textit{ then } \mathscr{T} - \| p \| \\
& \textit{else if } \gamma \in \mathscr{F} \textit{ then } \{A\gamma B : A, B \in \mathscr{T}\} \\
& \textit{else if } \gamma = \alpha \vee \beta \textit{ then } \| \alpha \| \cup \| \beta \| \\
& \textit{else if } \gamma = \alpha^* \textit{ then } \| 1 \| \cup \| \alpha \| \cup \| \alpha\alpha \| \cup \| \alpha\alpha\alpha \| \cup \cdots \\
& \textit{else if } \gamma = \alpha\beta \textit{ then } \{aCb : aC \in \| \alpha \| \ \& \ Cb \in \| \beta \| \ \& \ C \in \mathscr{T}\}
\end{aligned}$$

Notice that all words in $\| \gamma \|$ are of the form $AfBg \cdots hC$, where $A, B, \ldots, C \in \mathscr{T}$ and $f, g, \ldots, h \in \mathscr{F}$. As well, for any proposition $p \in \mathscr{P}$, $\| p \|$ is evidently just the set of all disjuncts in the fdnf of p. Notice too that the K-expression 1, in addition to standing for truth, acts as an identity operator, since $\| 1\gamma \| = \| \gamma 1 \| = \| \gamma \|$ for any γ.

Clearly, the regular expression representation $\alpha_{\mathfrak{A}}$ of an elemental program $\mathfrak{A}$ can equally well be regarded as a K-expression. That is, the regular expression $\alpha_{\mathfrak{A}}$ formed over $\Sigma_{\mathfrak{A}}$ is also a K-expression over $\mathscr{F}_{\mathfrak{A}} \cup \mathscr{R}_{\mathfrak{A}}$, where $\mathscr{F}_{\mathfrak{A}} = \Sigma_{\mathfrak{A}} \cap \mathscr{A}$, and where $\mathscr{R}_{\mathfrak{A}}$ is the set of all qffs of the form $r(\sigma, \ldots, \tau)$ occurring in the letters of $\Sigma_{\mathfrak{A}} \cap \mathscr{Q}$. In this context, we will write $\mathscr{T}_{\mathfrak{A}}$ for the set truth, $\mathscr{P}_{\mathfrak{A}}$ for the set of propositions, and say that $\alpha_{\mathfrak{A}}$ is a *K-expression representation* of $\mathfrak{A}$.

For example, consider the elemental program $\mathfrak{C}$ in Fig. 1. As mentioned earlier, using abbreviations, we can take

$$\alpha_{\mathfrak{C}} = f(\sim(p \supset r)\, g(\sim pg)^* \sim \sim p)^*(p \supset r)g.$$

Here, $\alpha_{\mathfrak{C}}$ is a K-expression over $\mathscr{F}_{\mathfrak{C}} \cup \mathscr{R}_{\mathfrak{C}}$, where $\mathscr{F}_{\mathfrak{C}} = \{f, g\}$ and $\mathscr{R}_{\mathfrak{C}} = \{p, r\}$. The set truth is then $\mathscr{T}_{\mathfrak{C}} = \{pr, p\bar{r}, \bar{p}r, \bar{p}\bar{r}\}$. An example of a word in the K-event $\| \alpha_{\mathfrak{C}} \|$ is $p\bar{r}fp\bar{r}gprgp\bar{r}$.

Engeler [*1*] has independently used a representation for programs similar in spirit to K-expressions in his study of program termination. Ito also has adopted this notation and his suggested [*5*] semantics for revealing propositional structure, though primitive in form, was a forerunner of the K-event semantics developed here and in [*6*].

Detection of Strong Equivalence: Procedure K

We say that an expression of the form $\alpha = \beta$, where α and β are K-expressions over the same alphabet, is a *K-wff*. Just in case $\| \alpha \| = \| \beta \|$, we say that α is *K-equivalent* to β and write $\models_K \alpha = \beta$ to indicate that the K-wff $\alpha = \beta$ is therefore *K-valid*.

The second procedure for detecting strong equivalence is based on the fact that if the K-expression representations of two elemental programs are K-equivalent, then the programs are strongly equivalent. To arrive at this result, we proceed as we did for the first procedure.

With each execution of an elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$, we can associate a (possibly infinite) word over $\mathscr{F}_{\mathfrak{A}} \cup \mathscr{R}_{\mathfrak{A}}$. Specifically, we associate with the output state $\mathfrak{A}[D, \xi]$ a word $\mathfrak{A}^{**}[D, \xi] = V(\mathfrak{A}, D, \xi, \$)$, where for any $x \in X$,

$$
\begin{aligned}
V(\mathfrak{A}, D, \xi, x) = {} & \textit{if } [x] \in \mathscr{A} \textit{ and } \Gamma x = y \textit{ then } \mathscr{T}_{\mathfrak{A}}[D, \xi][x]\ V(\mathfrak{A}, D, [x][D, \xi], y) \\
& \textit{else if } [x] \in \mathscr{Q} \textit{ and } \Gamma x = \langle y, z \rangle \textit{ then} \\
& \qquad \textit{if } [x][D, \xi] = T \textit{ then } V(\mathfrak{A}, D, \xi, y) \textit{ else } V(\mathfrak{A}, D, \xi, z) \\
& \textit{else if } x = \mathfrak{c} \textit{ then } \mathscr{T}_{\mathfrak{A}}[D, \xi].
\end{aligned}
$$

As with the function W, the definition of V is much akin to the definition of the execution function E given earlier. Two results stem immediately from this fact.

Lemma 5. *For any elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$ with K-expression representation $\alpha_{\mathfrak{A}}$, computing structure D and state $\xi : \omega \to D_0$, if $\mathfrak{A}[D, \xi]$ is determinate then $\mathfrak{A}^{**}[D, \xi] \in \| \alpha_{\mathfrak{A}} \|$.*

Lemma 6. *For any elemental programs $\mathfrak{A}$ and $\mathfrak{B}$, computing structure D and state $\xi : \omega \to D_0$, $\mathfrak{A}^{**}[D, \xi] = \mathfrak{B}^{**}[D, \xi] \Rightarrow \mathfrak{A}[D, \xi] \cong \mathfrak{B}[D, \xi]$.*

The first result follows from Lemma 1 by a straightforward argument (although a detailed proof by induction from first principles is also possible). Suppose that the word $\mathfrak{A}^{*}[D, \xi] \in | \alpha_{\mathfrak{A}} |$ is $PfQg \cdots hR$, where $P, Q, \ldots, R \in \mathscr{Q}^*$ and $f, g, \ldots, h \in \mathscr{A}$. A comparison of definitions for the functions W and V then shows that the word $\mathfrak{A}^{**}[D, \xi]$ must be of the form $AfBg \cdots hC$, where $A \in \| P' \|$, $B \in \| Q' \|, \ldots, C \in \| R' \|$. On the other hand, since regular expressions and K-expressions are both structured

with "$\vee$", "$*$" and "$\cdot$" at the outer level, and since "$\cdot$" distributes over "$\vee$" in the definition of K-event, we have, therefore, that $PfQg \cdots hR \in | \alpha_{\mathfrak{A}} |$ implies

$$\{\alpha f \beta g \cdots h\gamma : \alpha \in \| P' \|, B \in \| Q' \|, \ldots, \gamma \in \| R' \|\} \subseteq \| \alpha_{\mathfrak{A}} \|.$$

But $AfBg \cdots hC$ belongs to this subset, so that $\mathfrak{A}^{**}[D, \xi] \in \| \alpha_{\mathfrak{A}} \|$, as required.

The second result follows precisely as did Lemma 2, although, again, a detailed proof by induction is possible.

Then, with initial condition defined as it was earlier, we have

LEMMA 7. *For any elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$ with K-expression representation $\alpha_{\mathfrak{A}}$, computing structure D, state $\xi : \omega \to D_0$ and word $w \in \| \alpha_{\mathfrak{A}} \|$, w is the word associated with the execution of $\mathfrak{A}$ in D with initial state ξ if and only if the initial condition on w has truth value T with respect to D and ξ, or in symbols,*

$$\mathfrak{A}^{**}[D, \xi] = w \Leftrightarrow I(w)[D, \xi] = T.$$

Proof. We prove by induction the more general result that for any $x \in X$ and $w \in \| \alpha_{\mathfrak{A}}(x) \|$,

$$V(\mathfrak{A}, D, \xi, x) = w \Leftrightarrow I(w)[D, \xi] = T.$$

Case (i): $x = \mathfrak{c}$. Here, $\alpha_{\mathfrak{A}}(\mathfrak{c}) = 1$ so that $\| \alpha_{\mathfrak{A}}(\mathfrak{c}) \| = \mathscr{T}_{\mathfrak{A}}$. Now, for $w \in \mathscr{T}_{\mathfrak{A}}$, $I(w)[D, \xi] = T \Leftrightarrow w = \mathscr{T}_{\mathfrak{A}}[D, \xi]$ by the definition of $\mathscr{T}_{\mathfrak{A}}[D, \xi]$. As well,

$$V(\mathfrak{A}, D, \xi, \mathfrak{c}) = \mathscr{T}_{\mathfrak{A}}[D, \xi]$$

by the definition of V. Hence, $V(\mathfrak{A}, D, \xi, \mathfrak{c}) = w \Leftrightarrow I(w)[D, \xi] = T$, for $w \in \| \alpha_{\mathfrak{A}}(\mathfrak{c}) \|$.

Case (ii): $[x] \in \mathscr{A}$. Here, $\alpha_{\mathfrak{A}}(x) = [x]\, \alpha_{\mathfrak{A}}(\Gamma x)$, so that $w \in \| \alpha_{\mathfrak{A}}(x) \|$ is of the form $A[x]u$, where $A \in \mathscr{T}_{\mathfrak{A}}$ and $u \in \| \alpha_{\mathfrak{A}}(\Gamma x) \|$. The induction hypothesis here is that

$$V(\mathfrak{A}, D, [x][D, \xi], \Gamma x) = u \Leftrightarrow I(u)[D, [x][D, \xi]] = T.$$

Let us write $w \in \| \alpha_{\mathfrak{A}}(x) \|$ in the form $AfBg \cdots hC$, where $f = [x], A, B, \ldots, C \in \mathscr{T}_{\mathfrak{A}}$ and $g, \ldots, h \in \mathscr{F}_{\mathfrak{A}}$. Then,

$$\begin{aligned}
&I(w)[D, \xi] = T \\
&\quad \Leftrightarrow I(AfBg \cdots hC)[D, \xi] = T && \text{def}^n\ w \\
&\quad \Leftrightarrow A'[D, \xi] = B'[D, f[D, \xi]] = \cdots \\
&\qquad\qquad = g \cdots h \to C'[D, f[D, \xi]] = T && \text{def}^n\ w, \\
&\qquad\qquad && \text{def}^n\ ', \text{ Lemma 3} \\
&\quad \Leftrightarrow A = \mathscr{T}_{\mathfrak{A}}[D, \xi] \text{ and } I(u)[D, f[D, \xi]] = T && \text{def}^n\ \mathscr{T}_{\mathfrak{A}}[D, \xi], \\
&\qquad\qquad && \text{def}^n\ I
\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow A = \mathscr{T}_{\mathfrak{A}}[D, \xi] \text{ and } V(\mathfrak{A}, D, [x][D, \xi], \Gamma x) = u && \text{ind. hyp.}\\
&\Leftrightarrow V(\mathfrak{A}, D, \xi, x) = \mathscr{T}_{\mathfrak{A}}[D, \xi][x]\ V(\mathfrak{A}, D, \xi, \Gamma x) \text{ and} &&\\
&\quad A = \mathscr{T}_{\mathfrak{A}}[D, \xi] \text{ and } V(\mathfrak{A}, D, [x][D, \xi], \Gamma x) = u && \text{def}^n\ V\\
&\Leftrightarrow V(\mathfrak{A}, D, \xi, x) = w && \text{def}^n\ w, \text{def}^n =
\end{aligned}$$

Case (iii): $[x] \in \mathscr{Q}$. Here, if $\Gamma x = \langle y, z \rangle$, then $\alpha_{\mathfrak{A}}(x) = [x]\ \alpha_{\mathfrak{A}}(y) \vee \sim[x]\ \alpha_{\mathfrak{A}}(z)$, so that $w \in \| \alpha_{\mathfrak{A}}(x) \|$ is either of the form Au, where $A \in \|[x]\|$ and $Au \in \| \alpha_{\mathfrak{A}}(y) \|$, or of the form Bv, where $B \in \| \sim[x] \|$ and $Bv \in \| \alpha_{\mathfrak{A}}(z) \|$. The induction hypothesis here is that

$$\begin{aligned}
&\textit{if } w = Au \textit{ then } V(\mathfrak{A}, D, \xi, y) = Au \Leftrightarrow I(Au)[D, \xi] = T\\
&\textit{else if } w = Bv \textit{ then } V(\mathfrak{A}, D, \xi, z) = Bv \Leftrightarrow I(Bv)[D, \xi] = T.
\end{aligned}$$

Then, using the fact that the definitions of A and B give $A'[D, \xi] = T \Rightarrow [x][D, \xi] = T$ and $B'[D, \xi] = T \Rightarrow [x][D, \xi] = F$, we have

$$\begin{aligned}
&I(w)[D, \xi] = T &&\\
&\quad \Leftrightarrow \textit{if } w = Au \textit{ then } I(Au)[D, \xi] = T &&\\
&\quad \textit{else if } w = Bv \textit{ then } I(Bv)[D, \xi] = T && \text{def}^n\ w\\
&\quad \Leftrightarrow \textit{if } w = Au \textit{ then } A'[D, \xi] = T \textit{ and } I(Au)[D, \xi] = T &&\\
&\quad \textit{else if } w = Bv \textit{ then } B'[D, \xi] = T \textit{ and } I(Bv)[D, \xi] = T && \text{def}^n\ I\\
&\quad \Leftrightarrow \textit{if } w = Au \textit{ then } [x][D, \xi] = T \textit{ and } V(\mathfrak{A}, D, \xi, y) = Au &&\\
&\quad \textit{else if } w = Bv \textit{ then } [x][D, \xi] = F \textit{ and } V(\mathfrak{A}, D, \xi, z) = Bv && \text{def}^n\ A, B,\\
&\qquad\qquad && \text{ind. hyp.}\\
&\quad \Leftrightarrow \textit{if } w = Au \textit{ then } V(\mathfrak{A}, D, \xi, x) = V(\mathfrak{A}, D, \xi, y) &&\\
&\qquad\qquad \textit{and } V(\mathfrak{A}, D, \xi, y) = Au &&\\
&\quad \textit{else if } w = Bv \textit{ then } V(\mathfrak{A}, D, \xi, x) = V(\mathfrak{A}, D, \xi, z) &&\\
&\qquad\qquad \textit{and } V(\mathfrak{A}, D, \xi, z) = Bv && \text{def}^n\ V\\
&\quad \Leftrightarrow \textit{if } w = Au \textit{ then } (\mathfrak{A}, D, \xi, x) = Au &&\\
&\quad \textit{else if } w = Bv \textit{ then } (\mathfrak{A}, D, \xi, x) = Bv && \text{def}^n\ =\\
&\quad \Leftrightarrow (\mathfrak{A}, D, \xi, x) = w && \text{def}^n\ =
\end{aligned}$$

This completes the induction.

If we take $x = \$$, then we have for $w \in \| \alpha_{\mathfrak{A}}(\$) \|$, $V(\mathfrak{A}, D, \xi, \$) = w \Leftrightarrow I(w)[D, \xi] = T$. Since $\alpha_{\mathfrak{A}}(\$) = \alpha_{\mathfrak{A}}$ and $\mathfrak{A}^{**}[D, \xi] = V(\mathfrak{A}, D, \xi, \$)$, we have therefore that for $w \in \| \alpha_{\mathfrak{A}} \|$,

$$\mathfrak{A}^{**}[D, \xi] = w \Leftrightarrow I(w)[D, \xi] = T. \qquad \blacksquare$$

The second strong equivalence-detecting procedure is then based on the following

THEOREM 4. *For any two elemental programs* $\mathfrak{A}$ *and* $\mathfrak{B}$ *with K-expression representations* $\alpha_{\mathfrak{A}}$ *and* $\alpha_{\mathfrak{B}}$ *formed over the alphabet* $\mathscr{F}_{\mathfrak{A}} \cup \mathscr{R}_{\mathfrak{A}} \cup \mathscr{F}_{\mathfrak{B}} \cup \mathscr{R}_{\mathfrak{B}}$, *if* $\alpha_{\mathfrak{A}}$ *is K-equivalent to* $\alpha_{\mathfrak{B}}$, *then* $\mathfrak{A}$ *is strongly equivalent to* $\mathfrak{B}$, *or in symbols,* $\| \alpha_{\mathfrak{A}} \| = \| \alpha_{\mathfrak{B}} \| \Rightarrow \mathfrak{A} \simeq \mathfrak{B}$.

Lemmas 5, 6, and 7 can be used to prove this result in precisely the same manner as Lemmas 1, 2 and 4 were used to prove Theorem 2.

As a preliminary to obtaining the decidability of K-equivalence, let us first extend the theory $\mathscr{T}_R$ by adding seven new axiom schemata to form the theory $\mathscr{T}_K$. Here p and q are any propositions.

$$\begin{aligned} &C1: \quad pp = p && C5: \quad (p \supset q) = \bar{p} \vee q \\ &C2: \quad p\bar{p} = 0 && C6: \quad \sim(p \supset q) = p\bar{q} \\ &C3: \quad pq = qp && C7: \quad \sim\sim p = p \\ &C4: \quad p \vee \bar{p} = 1 \end{aligned}$$

Of course, axiom schemata *S1*,..., *S11* and rules *R1*, *R2* and *R3* now refer to K-wffs. Furthermore the side condition on *R3* is changed from $\Lambda \notin |\beta|$ to $\mathscr{T} \not\subseteq \|\beta\|$; this latter side condition is also decidable. Just in case a K-wff $\alpha = \beta$ is finitely derivable in $\mathscr{T}_K$, we write $\vdash_K \alpha = \beta$.

When later (Theorem 6) we prove the adequacy of $\mathscr{T}_K$, we depend heavily upon the notion of *normal forms* for K-expressions. A K-expression β is a *normal form* of the K-expression α if and only if

(i) $\| \alpha \| = \| \beta \|$

(ii) β is a regular expression over $\mathscr{F} \cup \mathscr{T}$

(iii) $\| \beta \| = | \beta |$, where the regular event $| \beta |$ is evaluated considering β as a regular expression over $\mathscr{F} \cup \mathscr{T}$.

We will write $N(\alpha)$ for a normal form of α.

For example, referring to the elemental program $\mathfrak{C}$ in Fig. 1, we can give a normal form for its K-expression representation

$$\alpha_{\mathfrak{C}} = f(\sim(p \supset r)\, g(\sim pg)^* \sim \sim p)^*(p \supset r)g$$

as

$$\begin{aligned} N(\alpha_{\mathfrak{C}}) = {} & (pr \vee p\bar{r} \vee \bar{p}r \vee \bar{p}\bar{r})\, f(pr \vee \bar{p}r \vee \bar{p}\bar{r} \vee (p\bar{r}g(\bar{p}rg \vee \bar{p}\bar{r}g)^*)^* pr) \\ & g(pr \vee p\bar{r} \vee \bar{p}r \vee \bar{p}\bar{r}). \end{aligned}$$

Illustrated in Fig. 5 is the state transition diagram for a nondeterministic finite automaton whose behavior is $|N(\alpha_{\mathfrak{C}})|$, where $N(\alpha_{\mathfrak{C}})$ is considered to be a regular expression over $\{f, g\} \cup \{pr, p\bar{r}, \bar{p}r, \bar{p}\bar{r}\}$.

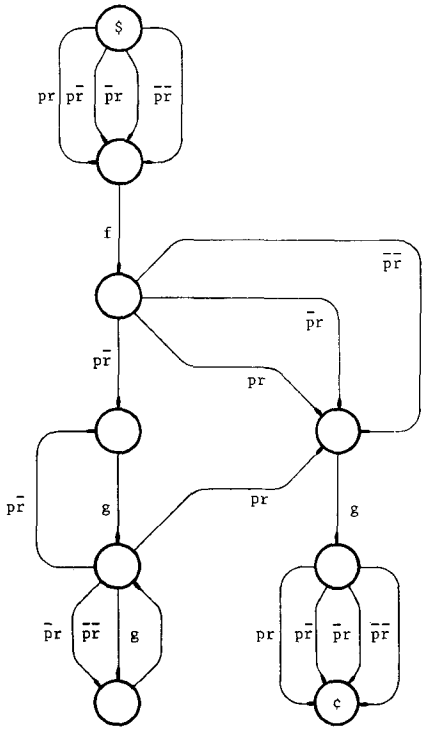

FIG. 5. The state-transition diagram for a nondeterministic finite automaton whose behaviour is $|N(\alpha\mathfrak{C})|$.

Before proceeding, let us consider one or two notational matters. If A is a set of K-expressions, then $\vee A$ will denote the disjunction formed with "$\vee$" of the elements in A; if A is the empty set, then $\vee A$ is just 0. If A is a set of K-expressions each of the form p or pzq for some $p, q \in \mathscr{T}$, then the K-expression $\vee A$ is said to be *standard*. If $\vee A$ and $\vee B$ are standard, then by $\vee A \otimes \vee B$ we mean $\vee\{xpy : xp \in A \,\&\, py \in B \,\&\, p \in \mathscr{T}\}$. Evidently, we have

LEMMA 8. *For any standard K-expressions* α *and* β, $\vdash_K \alpha\beta = \alpha \otimes \beta$.

Let us now give the main result concerning normal forms for K-expressions.

THEOREM 5. *For any K-expression* α, *there exists a normal form* $N(\alpha)$ *such that* $\vdash_K \alpha \doteq N(\alpha)$.

Proof. We give a proof by induction on the structure of α. First we show $\vdash_K \alpha = N(\alpha)$, where α is *non-iterative*, i.e., contains no "*". Then, with the induction hypothesis that $\vdash_K \beta = N(\beta)$, we show $\vdash_K \beta^* = N(\beta^*)$.

For α non-iterative, evidently we can take $N(\alpha)$ to be $\vee \| \alpha \|$; this is straightforward to verify. Notice that $\vee \| \alpha \|$ is a standard K-expression.

Consider now finding a normal form of β^*, where $\vdash_K \beta = N(\beta)$ by induction hypothesis. As an additional induction hypothesis, we assume $N(\beta)$ to be standard, say of the form $(a_0 \vee a_1 \vee \cdots \vee a_{m-1})$, where $m < \omega$. Thus, it sufficies to find a normal form for $(a_0 \vee a_1 \vee \cdots \vee a_{m-1})^*$, i.e., for $N(\beta)^*$, since now $\vdash_K \beta^* = N(\beta)^*$.

Since $\vdash_K (\alpha \vee \beta)^* = \alpha^*(\beta\alpha^*)^*$, we therefore have $\vdash_K N(\beta)^* = \theta_0\theta_1 \cdots \theta_{m-1}$, where θ_0 is $a_0{}^*$ and θ_n is $(a_n\theta_0\theta_1 \cdots \theta_{n-1})^*$, for $0 < n < m$. For example,

$$\vdash_K (a \vee b \vee c \vee d)^* = a^*(ba^*)^*\,(ca^*(ba^*)^*)^*\,(da^*(ba^*)^*\,(ca^*(ba^*)^*)^*)^*.$$

Here, $m = 4$, θ_0 is a^*, θ_1 is $(ba^*)^*$, θ_2 is $(ca^*(ba^*)^*)^*$, and θ_3 is

$$(da^*(ba^*)^*(ca^*(ba^*)^*)^*)^*.$$

Thus, it suffices to find a normal form for $\theta_0\theta_1 \cdots \theta_{m-1}$, since now $\vdash_K \beta^* = \theta_0\theta_1 \cdots \theta_{m-1}$.

Our aim now is to show $\vdash_K \theta_i = N(\theta_i)$, for all $i < m$. This we do by induction. Since θ_0 is $a_0{}^*$, where a_0 is of the form p or pzq for some $p, q \in \mathscr{T}$, the appropriate instance of

$$\vdash_K (s \vee sxs \vee syt)^* = \vee\mathscr{T} \vee sx(sx)^*s \vee s(y \vee x(sx)^* sy)t, \qquad (**)$$

where $s, t \in \mathscr{T}$ and are distinct, gives that $\vdash_K \theta_0 = N(\theta_0)$. Moreover, the normal form $N(\theta_0)$ obtained from $(**)$ is standard. Now assume $\vdash_K \theta_i = N(\theta_i)$, for all $i < n < m$, and that the $N(\theta_i)$ are all standard. Recall that θ_n is $(a_n\theta_0\theta_1 \cdots \theta_{n-1})^*$. The induction hypothesis then gives $\vdash_K \theta_n = (a_n N(\theta_0)\, N(\theta_1) \cdots N(\theta_{n-1}))^*$, and Lemma 8 gives $\vdash_K \theta_n = (a_n \otimes N(\theta_0) \otimes N(\theta_1) \otimes \cdots \otimes N(\theta_{n-1}))^*$, since a_n and θ_i, $i < n$, are all standard. Since a_n is of the form pzq for some $p, q \in \mathscr{T}$, all of the disjuncts in $a_n \otimes N(\theta_0) \otimes N(\theta_1) \otimes \cdots \otimes N(\theta_{n-1})$ have $p \in \mathscr{T}$ as their leftmost word of truth. Therefore, a straightforward generalization of $(**)$ can be applied to give $\vdash_K \theta_n = N(\theta_n)$. Notice that this $N(\theta_n)$ will be standard. This completes the induction and so $\vdash_K \theta_i = N(\theta_i)$, for all $i < m$. Thus, it suffices to find a normal form for $N(\theta_0)\, N(\theta_1) \cdots N(\theta_{m-1})$, since now $\vdash_K \beta^* = N(\theta_0)\, N(\theta_1) \cdots N(\theta_{m-1})$.

But, of course, $\vdash_K N(\theta_0)\, N(\theta_1) \cdots N(\theta_{m-1}) = N(\theta_0) \otimes N(\theta_1) \otimes \cdots \otimes N(\theta_{m-1})$ by Lemma 8, and this latter K-expression will evidently serve as the required standard normal form $N(\beta^*)$. Thus, $\vdash_K \beta^* = N(\beta^*)$, and this completes the induction. ∎

Now for the main result concerning the theory $\mathscr{T}_K$.

THEOREM 6. *The theory $\mathscr{T}_K$ is both sound and adequate, or in symbols, $\vdash_K \alpha = \beta \Leftrightarrow \models_K \alpha = \beta$ for any K-wff $\alpha = \beta$.*

Proof. Soundness, i.e., $\vdash_K \alpha = \beta \Rightarrow \models_K \alpha = \beta$ is easily verified by showing that each axiom schema in $\mathscr{T}_K$ gives rise to K-valid K-wffs, and that each rule preserves K-validity.

To obtain adequacy, note that

$$\begin{aligned}
\models_K \alpha = \beta &\Rightarrow \| \alpha \| = \| \beta \| && \text{def}^n \models_K \\
&\Rightarrow \| N(\alpha) \| = \| N(\beta) \| && \text{def}^n N(\ ) \\
&\Rightarrow | N(\alpha) | = | N(\beta) | && \text{def}^n N(\ ) \\
&\Rightarrow \vdash_K N(\alpha) = N(\beta) && \text{Theorem 3} \\
&\Rightarrow \vdash_K \alpha = \beta && \text{Theorem 5,}
\end{aligned}$$

where the normal forms $N(\alpha)$ and $N(\beta)$ are those generated by the construction used to prove Theorem 5. Note that $N(\alpha)$ and $N(\beta)$ are considered to be regular expressions over $\mathscr{F} \cup \mathscr{T}$ when evaluating $| N(\alpha) |$ and $| N(\beta) |$. Therefore, the derivation $\vdash_K N(\alpha) = N(\beta)$ will only use those instances of the axiom schemata and rules of $\mathscr{T}_K$ that are likewise constituted.

In the proofs of Theorems 5 and 6, we have shown how to construct a derivation in $\mathscr{T}_K$ of any K-valid K-wff. Thus, the theory $\mathscr{T}_K$ itself can serve as the basis for a K-equivalence decision procedure.

The second strong equivalence-detecting procedure can now be specified:

Procedure K: given elemental programs $\mathfrak{A}$ *and* $\mathfrak{B}$

(i) *construct* $\alpha_{\mathfrak{A}}$ *and* $\alpha_{\mathfrak{B}}$ *(effective)*
(ii) *test for* $\| \alpha_{\mathfrak{A}} \| = \| \alpha_{\mathfrak{B}} \|$ *using* $\mathscr{T}_K$ *(effective)*
(iii) *if* $\| \alpha_{\mathfrak{A}} \| = \| \alpha_{\mathfrak{B}} \|$, *return* YES, *otherwise* MAYBE.

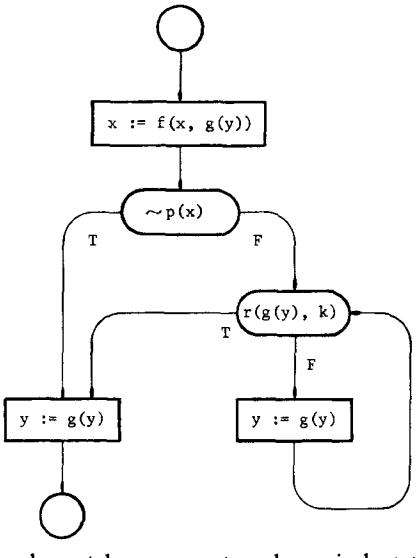

FIG. 6. Yet another elemental program strongly equivalent to that in Fig. 1.

To consider an example, notice that Procedure K detects that the elemental program in Fig. 4 is strongly equivalent to that in Fig. 1. To see the limitations of Procedure K, notice that the elemental program in Fig. 6 is also equivalent to that in Fig. 1, but that Procedure K fails to detect this fact. We now turn our attention to developing a third and somewhat more powerful procedure, which will be successful in this latter case as well.

SHIFT K-EVENTS

As we have seen, Procedure K depends upon a test for equality of K-events. Justification for making this test is Theorem 4, and the proof of this Theorem (which has precisely the same form as the proof of Theorem 2) depends essentially on the fact that if $\mathfrak{A}[D, \xi]$ is determinate, then $\mathfrak{A}^{**}[D, \xi] \in \| \alpha_{\mathfrak{A}} \|$ (which is just the statement of Lemma 5). For two elemental programs $\mathfrak{A}$ and $\mathfrak{B}$, if $\| \alpha_{\mathfrak{A}} \| \neq \| \alpha_{\mathfrak{B}} \|$ then Procedure K returns MAYBE. But suppose we could throw away certain words and so define the subsets $A \subseteq \| \alpha_{\mathfrak{A}} \|$ and $B \subseteq \| \alpha_{\mathfrak{B}} \|$ such that Lemma 5 still held, i.e., such that if $\mathfrak{A}[D, \xi]$ were determinate, then $\mathfrak{A}^{**}[D, \xi] \in A$, and similarly for $\mathfrak{B}$ and B. Then, if $A = B$, we would have $\mathfrak{A} \simeq \mathfrak{B}$ by Theorem 4 once again. Thus, we can strengthen the ability of Procedure K to detect strong equivalence if we can find a way to cast out of the K-events associated with elemental programs words that can never be associated with halting executions of those programs.

We will call such whittled down K-events *shift K-events*. The word *shift* serves to emphasize the connection between shift K-events and the work of Ianov [*4*] and Rutledge [*14*].

Suppose we are considering K-expressions defined over the alphabets $\mathscr{F} = \{f, g, .., h\}$ of operators and $\mathscr{R} = \{p, q, ..., r\}$ of propositional atoms. Then, a *shift distribution* is defined to be any mapping $\mathscr{S} : \mathscr{F} \to \| \vee \mathscr{F} \|$, such that for any $x \in \mathscr{F}$, $\mathscr{S}(x) \subseteq \| x \|$. (Intuitively speaking, $\mathscr{S}(x)$ delimits the possibilities for the truth-values of the propositional atoms before and after execution of the operator x.) The shift K-event $\| \gamma \|_{\mathscr{S}}$ associated with the K-expression γ is computed using

$$
\begin{aligned}
\| \gamma \|_{\mathscr{S}} = {} & \textit{if } \gamma \in \mathscr{P} \textit{ then } \| \gamma \| \\
& \textit{else if } \gamma \in \mathscr{F} \textit{ then } \mathscr{S}(\gamma) \\
& \textit{else if } \gamma = \alpha \vee \beta \textit{ then } \| \alpha \|_{\mathscr{S}} \cup \| \beta \|_{\mathscr{S}} \\
& \textit{else if } \gamma = \alpha^* \textit{ then } \| 1 \|_{\mathscr{S}} \cup \| \alpha \|_{\mathscr{S}} \cup \| \alpha\alpha \|_{\mathscr{S}} \cup \| \alpha\alpha\alpha \|_{\mathscr{S}} \cup \cdots \\
& \textit{else if } \gamma = \alpha\beta \textit{ then } \{aCb : aC \in \| \alpha \|_{\mathscr{S}} \,\&\, Cb \in \| \beta \|_{\mathscr{S}} \,\&\, C \in \mathscr{T}\}
\end{aligned}
$$

Notice that $\| \gamma \|_{\mathscr{S}}$ is just $\| \gamma \|$ with all those words cast out that are not permitted by the shift distribution $\mathscr{S}$.

For example, consider the elemental program $\mathfrak{C}$ in Fig. 1. As mentioned earlier, using abbreviations, we can take

$$\alpha_{\mathfrak{C}} = f(\sim(p \supset r)\, g(\sim pg)^* \sim \sim p)^*(p \supset r)g.$$

Notice that since the assigned variable in the assignment statement f does not occur in the qff r, execution of f cannot change the truth-value of r. Thus, no execution of $\mathfrak{C}$ can give rise to the word $\bar{p}rfp\bar{r}gprgp\bar{r} \in \| \alpha_{\mathfrak{C}} |'$, which shows r going from T to F on executing f. If, however, we exclude $\bar{p}rfp\bar{r}$ from $\mathscr{S}(f)$, then the unwanted word $\bar{p}rfp\bar{r}gprgp\bar{r}$ will not appear in the shift K-event $\| \alpha_{\mathfrak{C}} \|_{\mathscr{S}}$.

Any shift distribution that preserves Lemma 5 is said to be *consistent*. That is, with respect to an elemental program $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$ with K-expression representation $\alpha_{\mathfrak{A}}$, a shift distribution $\mathscr{S}$ is consistent if and only if for any computing structure D and state $\xi : \omega \to D_0$, if $\mathfrak{A}[D, \xi]$ is determinate then $\mathfrak{A}^{**}[D, \xi] \in \| \alpha_{\mathfrak{A}} \|_{\mathscr{S}}$. We return later to the problem of specifying consistent shift distributions for elemental programs.

Detection of Strong Equivalence: Procedure S

For any K-wff $\alpha = \beta$ and shift distribution $\mathscr{S}$, just in case $\| \alpha \|_{\mathscr{S}} = \| \beta \|_{\mathscr{S}}$, we say that α is *$\mathscr{S}$-K-equivalent* to β and write $\models_{K,\mathscr{S}} \alpha = \beta$ to indicate that the K-wff $\alpha = \beta$ is therefore *$\mathscr{S}$-K-valid*.

The third procedure for detecting strong equivalence is based on the fact that if the K-expression representations of two elemental programs are $\mathscr{S}$-K-equivalent for some consistent shift distribution $\mathscr{S}$, then the programs are strongly equivalent. In fact, because $\mathscr{S}$ being consistent preserves Lemma 5, we can state without proof the following

Theorem 7. *For any two elemental programs $\mathfrak{A}$ and $\mathfrak{B}$ with K-expression representations $\alpha_{\mathfrak{A}}$ and $\alpha_{\mathfrak{B}}$ formed over the alphabet $\mathscr{F}_{\mathfrak{A}} \cup \mathscr{R}_{\mathfrak{A}} \cup \mathscr{F}_{\mathfrak{B}} \cup \mathscr{R}_{\mathfrak{B}}$, if $\mathscr{S}$ is a shift distribution consistent with respect to both $\mathfrak{A}$ and $\mathfrak{B}$, and if $\alpha_{\mathfrak{A}}$ is $\mathscr{S}$-K-equivalent to $\alpha_{\mathfrak{B}}$,* then $\mathfrak{A}$ *is strongly equivalent to $\mathfrak{B}$, or in symbols,* $\| \alpha_{\mathfrak{A}} \|_{\mathscr{S}} = \| \alpha_{\mathfrak{B}} \|_{\mathscr{S}} \Rightarrow \mathfrak{A} \simeq \mathfrak{B}$.

As a preliminary to obtaining the decidability of $\mathscr{S}$-K-equivalence, let us first extend the theory $\mathscr{T}_K$ by adding a new axiom schema *C8* to form the theory $\mathscr{T}_{K,\mathscr{S}}$

$$C8: \quad x = \vee\mathscr{S}(x), \qquad \text{where} \quad x \in \mathscr{F}.$$

Just in case a K-wff $\alpha = \beta$ is finitely derivable in $\mathscr{T}_{K,\mathscr{S}}$, we write $\vdash_{K,\mathscr{S}} \alpha = \beta$.

Theorem 8. *For any shift distribution $\mathscr{S}$, the theory $\mathscr{T}_{K,\mathscr{S}}$ is both sound and adequate, or in symbols,* $\vdash_{K,\mathscr{S}} \alpha = \beta \Leftrightarrow \models_{K,\mathscr{S}} \alpha = \beta$ *for any K-wff* $\alpha = \beta$.

Proof. We only sketch a proof here. Soundness, i.e., $\vdash_{K,\mathscr{S}} \alpha = \beta \Rightarrow \models_{K,\mathscr{S}} \alpha = \beta$, is easily verified by showing that each axiom schema in $\mathscr{T}_{K,\mathscr{S}}$ gives rise to $\mathscr{S}$-K-valid K-wffs, and that each rule preserves $\mathscr{S}$-K-validity.

To obtain adequacy, we proceed precisely as in Lemma 8 and Theorem 5, except that now the normal form $N(x)$ of an operator $x \in \mathscr{F}$ is taken to be $\vee\mathscr{S}(x)$ (instead of $\vee \| x \|$ as in the proof of Theorem 5). Notice that *C8* gives $\vdash_{K,\mathscr{S}} x = N(x)$ as required.

We may then proceed as in Theorem 6 to ultimately show

$$\models_{K,\mathscr{S}} \alpha = \beta \Rightarrow \vdash_{K,\mathscr{S}} \alpha = \beta. \qquad \blacksquare$$

In the proof of Theorem 8, we have indicated how to construct a derivation in $\mathscr{T}_{K,\mathscr{S}}$ of any $\mathscr{S}$-K-valid K-wff. Thus, the theory $\mathscr{T}_{K,\mathscr{S}}$ itself can serve as the basis for a $\mathscr{S}$-K-equivalence decision procedure.

The third strong equivalence-detecting procedure can now be specified:

Procedure S: given elemental programs $\mathfrak{A}$ *and* $\mathfrak{B}$

(i) *construct* $\alpha_{\mathfrak{A}}$ *and* $\alpha_{\mathfrak{B}}$ *(effective)*

(ii) *define a consistent shift distribution* $\mathscr{S}$

(iii) *test for* $\| \alpha_{\mathfrak{A}} \|_{\mathscr{S}} = \| \alpha_{\mathfrak{B}} \|_{\mathscr{S}}$ *using* $\mathscr{T}_{K,\mathscr{S}}$ *(effective)*

(iv) *if* $\| \alpha_{\mathfrak{A}} \|_{\mathscr{S}} = \| \alpha_{\mathfrak{B}} \|_{\mathscr{S}}$, *return* YES, *otherwise* MAYBE.

To consider an example, let us return to the problem of detecting that the elemental programs in Figs. 1 and 6 are strongly equivalent; recall that Procedure K fails to detect this fact. Using the abbreviations mentioned earlier, we have $\mathscr{F} = \{f, g\}$ and $\mathscr{R} = \{p, r\}$ so that $\mathscr{T} = \{pr, p\bar{r}, \bar{p}r, \bar{p}\bar{r}\}$. The, let us define $\mathscr{S} : \mathscr{F} \to \| \vee\mathscr{F} \|$ so that

$$\mathscr{S}(f) = \{prfpr, prf\bar{p}r, p\bar{r}fpr, p\bar{r}f\bar{p}r, \bar{p}rfpr, \bar{p}rf\bar{p}r, \bar{p}\bar{r}fpr, \bar{p}\bar{r}f\bar{p}\bar{r}\}$$

$$\mathscr{S}(g) = \{prgpr, prgp\bar{r}, p\bar{r}gpr, p\bar{r}gp\bar{r}, \bar{p}rgpr, \bar{p}rgp\bar{r}, \bar{p}\bar{r}gpr, \bar{p}\bar{r}gp\bar{r}\}$$

Since $\mathscr{S}$ is defined simply to reflect the fact that executing f can affect the truth-value of p but not r, and that executing g can effect the truth-value of r but not p, therefore $\mathscr{S}$ is certainly consistent. Furthermore, with this shift distribution, Procedure S successfully detects that the elemental programs in Figs. 1 and 6 are strongly equivalent.

Consistent Shift Distributions

In the example considered above, we constructed a consistent shift distribution by considering whether or not execution of the assignment statements involved could affect the truth-value of certain qffs. There are at least two other effects that can be

looked for. First, we can reflect in a consistent shift distribution the fact that, for example, the qffs $p(f(v))$ and $p(u)$ have the same truth-value after execution of the assignment statement $u := f(v)$. Second, we can reflect the fact that the truth-values of *all* qffs remain unchanged upon execution of, for example, the assignment statement $v := v$.

Let us now combine all of these into a single procedure for defining a consistent shift distribution for one or more elemental programs. Suppose we have defined a set $\mathscr{F}$ of operators (i.e., assignment statements) and a set $\mathscr{R}$ of propositional atoms (i.e., qffs of the form $r(\sigma,..., \tau)$). The set $\mathscr{T}$, or truth, is built up from $\mathscr{R}$ in the usual way.

We may then define a shift distribution $\mathscr{S}$ as follows: for each $x \in \mathscr{F}$, $\mathscr{S}(x) = \{sxt : s, t \in \mathscr{T} \ \& \ (s'(x \rightarrow t'))'$ is not a logical contradiction$\}$. That $\mathscr{S}$ defined this way is consistent is obvious once we notice that $(s'(x \rightarrow t'))'$ is just the initial condition, $I(sxt)$, for the word sxt. Referring to the statement of Lemma 7, we see that if the initial condition of sxt is a logical contradiction, i.e., if $I(sxt)[D, \xi] = F$, for all computing structures D and states $\xi : \omega \rightarrow D_0$, then there exists *no* elemental program with an execution of which we can associate the word sxt. Thus, all such words may safely be omitted when constructing the shift distribution, i.e., $\mathscr{S}$ as defined above is consistent.

Furthermore, we see that this $\mathscr{S}$ is the maximal consistent shift distribution, i.e., there are no more words of the form sxt that can be cast out and still leave $\mathscr{S}$ consistent. This is clear since if $I(sxt)$ is not a logical contradiction, then for some D and ξ, we can in fact associate the word sxt with the execution of an elemental program; this again by Lemma 7.

The mechanism of consistent shift distributions was devised to permit a casting out from a K-event $\| \alpha_{\mathfrak{A}} \|$ those words that cannot be associated with executions of the elemental program $\mathfrak{A}$. Other devices, tricks and heuristics may, of course, be employed in this endeavor. In fact, the more powerful our ability to cast out such words, the more powerful is our ability to detect strong equivalence.

It may be mere whimsy to point out that a maximal consistent shift distribution, defined as above, but of the form

$$\mathscr{S} : \bigcup_{n<\omega} \mathscr{F}^n \rightarrow \bigcup_{n<\omega} \| \vee \mathscr{F} \|^n,$$

where $\mathscr{F}^n = \mathscr{F} \times \mathscr{F}^{n-1}$ and $\| \vee \mathscr{F} \|^n = \| \vee \mathscr{F} \| \otimes \| \vee \mathscr{F} \|^{n-1}$, is in fact, adequate to the task of casting out *all* the unwanted words from a K-event. Here, of course, if $x = \langle f, g,..., h \rangle$, where $f, g,... h \in \mathscr{F}$, then

$$\mathscr{S}(x) \subseteq \| f \| \otimes \| g \| \otimes \cdots \otimes \| h \|,$$

and since $\mathscr{S}$ is maximal consistent, $\mathscr{S}(x)$ will contain only those words $pfqg \cdots hr$, where $p, q,..., r \in \mathscr{T}$, that can be associated with the execution of an elemental program.

Just how such a shift distribution would be used to perform the casting out of unwanted words is not yet understood.

A Decidable Subcase

To point up the relation between the approach developed here and the work of Ianov [4] and Rutledge [14], let us consider a special subclass of elemental programs. An elemental program $\mathfrak{A}$ is said to be *abstract* if and only if

(i) there are no constants occurring in $\mathfrak{A}$ and only one variable, v say,

(ii) no function letters occur in any qff of $\mathfrak{A}$,

(iii) assignment statements of $\mathfrak{A}$ are restricted to be of the form $v := f(v,..., v)$, where f is any function letter.

An example of an abstract elemental program is illustrated in Fig. 7.

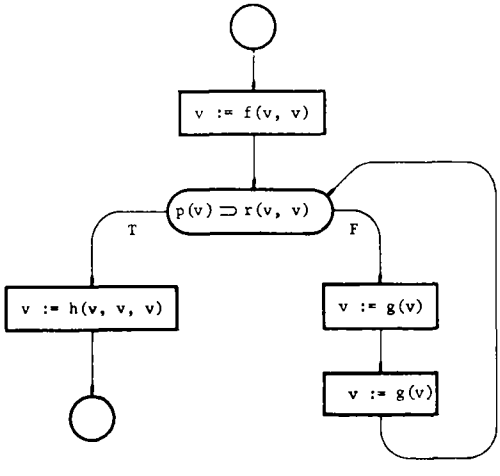

FIG. 7. An example of an abstract elemental program.

THEOREM 9. *For any two abstract elemental programs* $\mathfrak{A}$ *and* $\mathfrak{B}$ *with K-expression representations* $\alpha_{\mathfrak{A}}$ *and* $\alpha_{\mathfrak{B}}$ *formed over the alphabet* $\mathscr{F}_{\mathfrak{A}} \cup \mathscr{R}_{\mathfrak{A}} \cup \mathscr{F}_{\mathfrak{B}} \cup \mathscr{R}_{\mathfrak{B}}$, *just in case* $\alpha_{\mathfrak{A}}$ *is K-equivalent to* $\alpha_{\mathfrak{B}}$, *then* $\mathfrak{A}$ *is strongly equivalent to* $\mathfrak{B}$, *or in symbols,* $\| \alpha_{\mathfrak{A}} \| = \| \alpha_{\mathfrak{B}} \| \Leftrightarrow \mathfrak{A} \simeq \mathfrak{B}$.

Thus, strong equivalence is decidable for abstract elemental programs, since in this case strong equivalence is identical to K-equivalence, which we already know to

be decidable. Essentially, what we have done is to restrict the structure of abstract elemental programs so that *all* words in the K-event associated with such a program can be associated with some execution thereof. A detailed proof of this result is given in [6, Theorem 35].

Evidently, the class of abstract elemental programs contains the logical schemata of Ianov. We have, as required, the conditions that only syntactically identical sequences of operators are equivalent, and that each operator may affect the truth-value of each propositional atom. We say "contains," because here we have lifted the rather arbitrary restriction placed by Ianov that no operator may appear more than once in a logical scheme.

ACKNOWLEDGMENT

I am grateful to Z. Manna for his critical reading of the manuscript and subsequent helpful suggestions.

REFERENCES

1. E. ENGELER. Algorithmic properties of structures. Math. Systems Theory **1**, 183–195 (1967).
2. A. P. ERSHOV. Operator algorithms I. *Problems of Cybernetics* **3**, 697–763 (1962).
3. M. S. HARRISON. "*Introduction to Switching and Automata Theory*." McGraw-Hill, New York, 1965.
4. I. IANOV. The logical schemes of algorithms. *Problems of Cybernetics* **1**, 82–140 (1960).
5. T. ITO. Notes on theory of computation. Memo No. 61, Stanford Artificial Intelligence Project, Stanford University, May 1968.
6. D. M. KAPLAN. The formal theoretic analysis of strong equivalence for elemental programs. Thesis (Technical Report No. CS101), Stanford University, June 1968.
7. S. C. KLEENE. Representation of events in nerve nets and finite automata. *In* (Shannon, C. E., and McCarthy, J., eds.), "Automata Studies." Princeton University Press, Princeton, 1956.
8. D. LUCKHAM AND D. PARK. The undecidability of the equivalence problem for program schemata. Report No. 1141, Bolt, Beranek, and Newman, Inc., 1964.
9. J. MCCARTHY. A basis for a mathematical theory of computation. *In* (Braffort, P., and Hirschberg, D., eds.), "Computer Programming and Formal Systems." North-Holland Publ. Co, Amsterdam, 1963.
10. Z. MANNA. Termination of algorithms. Thesis, Carnegie-Mellon University, 1968.
11. R. NARASIMHAN. Programming languages and computers: a unified meta-theory. *Advances in Computers* **8**, 189–224 (1967).
12. M. S. PATERSON. Equivalence problems in a model of computation. Thesis, University of Cambridge, 1967.
13. M. O. RABIN AND D. SCOTT. Finite automata and their decision problems. *IBM J. Res. Develop.* **3** (1959),
14. J. D. RUTLEDGE. On Ianov's program schemata. *J. ACM* **11**, 1–9 (1964).
15. A. SALOMAA. Two complete axiom systems for the algebra of regular events. *J. ACM* **13**, 158–169 (1966).