SIMPLE PROGRAMS ON STRINGS
AND THEIR DECISION PROBLEMS

Giorgio Ausiello

TR 75-263

November 1975

Department of Computer Science
Cornell University
Ithaca, New York 14853

This work was carried on while the author was visiting the
Department of Computer Science of Cornell University and
it has been partially supported by National Science Founda-
tion Grant GJ-33171X and Consiglio Nazionale delle Ricerche.

Author's Address: Dott. Giorgio Ausiello - Istituto di
Automatica, Centro di Studio dei Sistemi di Controllo e
Calcolo Automatici del C.N.R., Via Eudossiana, 18-00184-
Roma, Italy.
SUMMARY

1. Motivation and relation to other work.
2. Definitions and basic properties.
3. Classes of simple programs on strings.
4. Simple acceptors.
5. Decidability properties for simple programs on strings.
6. An undecidability result.
7. References.

APPENDIX A. Complexity of decidable properties for simple programs on integers.
APPENDIX B. Simple programs on strings with weak sum.

ABSTRACT

Classes of simple programs operating on strings are considered. Their power as acceptors and their power as generation devices are compared and consequences on upper bounds and lower bounds for several decision problems are derived. It is shown that even for such a small class of programs some problems are undecidable.
1. MOTIVATION AND RELATION TO OTHER WORK.

The interest for studying the properties of small classes of programs comes from different areas of Computer sciences. The first reason (at least from a historical point of view) is to characterize the computational power achieved by using particular data structures, particular primitives, particular control structures. Some relevant work in this direction has been done by Meyer and Ritchie (1967) with the introduction of the so-called LOOP programs and of the hierarchy of classes of programs \( L_n \) defined by depth of nesting of LOOP instructions. Particular by interesting for these classes of programs and for the corresponding classes of functions \( L_n \) was the fact that, starting from the second level of the hierarchy, the syntactic characterization has a neat correspondence to the computational complexity of the functions (for example in terms of computation time on register machines or Turing machines). Another remarkable property of the LOOP classes of programs is that for the programs of the class \( L_1 \), the equivalence is decidable (while it is not decidable for programs in \( L_n \), for \( n \geq 2 \)).

Strictly related to the syntactic characterization of classes of programs is the complexity of their decision problems. Like the problem of equivalence of regular expressions has been shown to become harder and harder as long as new, more powerful operators (squaring, intersection, complement) are allowed (see Meyer and Stockmeyer (1972) and Meyer and Stockmeyer (1973), the same can be expected for equivalence in small classes of programs: by adding new features to a programming language which either increase the power of the language or allow to wri
become harder to decide. In this direction the work of Jones and Muchnick (1975) deals exhaustively with the class of programs (FMP: Finite memory programs) which compute exactly the finite state mappings. Several problems, which are shown to be exactly of linear nondeterministic space complexity, become quadratic or even exponential when a variable word size or subscripted identifiers are allowed.

Another reason for exploring this research area comes from the point of view of the semantics of programs. Since we know what are the theoretical and practical difficulties of automatically proving properties of programs it would be reasonable to look for small classes of programs which correspond to non trivial classes of functions, but whose properties are decidable. Example of such classes are the class $\mathcal{L}_1^+$ (Cherniavsky (1974)) which gives an exact realization of the theory of integer addition (Presburger arithmetic): given any relation $R(x_1, \ldots, x_n)$ described by a formula $R[x_1, \ldots, x_n]$ in the language of the theory there is a program $\pi$ in the programming language $\mathcal{L}_1^+$ such that $\pi(a_1, \ldots, a_n)$ halts iff $R[a_1, \ldots, a_n]$ is valid (also a notion of functional realization of Presburger arithmetic is given, and it is satisfied by the same programming language).

Unfortunately even if, in all cases, the semantics of the classes of programs is very restricted, the properties are already intractably hard (for example, in the last case, the equivalence requires at least exponential time).

A different approach has been followed by Cook (1975) whose aim is to characterize the class $F^P_2$ of feasibly (polynomial time) computable functions by providing a logical theory whose formulas are equations of the form $t = u$
(where t and u are terms built out of variables, constants and function symbol ranging over $E^2_2$) and whose theorems are conjectured to be exactly the formulas which have a polynomially long verification (that is the number of steps for verifying an instance $t(x) = u(x)$ is uniformly polynomially bounded in $|x|$). An interpretation of Cook's results in terms of properties of programs, though, is not possible because no definition is known of a programming language which allows to define programs for exactly the functions in $E^2_2$ and, besides, some relevant properties such as equivalence are already undecidable at this level.

In this paper small classes of programs for string manipulation (analogous to the simple programs on integers) are introduced and their properties are analyzed under various points of view. First of all the computational power is considered with respect to known classes of programs and transducers and it is shown to be somewhere between the class of functions computable by finite automata and $E^2_1$. Some decision problems are then examined and lower bounds and upper bounds are derived for their complexity. Surprisingly, the classes exhibit already some undecidable problems.

For sake of completeness, in Appendix A some results on the decidability and the complexity of problems relative to simple programs on integers are summarized. In Appendix B it is shown how the lower bound for the equivalence problem is increased by enlarging the instruction set of the simple programs (with the operation of the "weak sum").
2. DEFINITIONS AND BASIC PROPERTIES

In this paragraph we will give the definitions of classes of programs for manipulating strings, based on the LOOP control structure. Here and in the following paragraphs we will assume the reader to be familiar with the definitions of LOOP classes of programs (L_4) and functions of the integers (L_4: Meyer and Ritchie (1967)), of Grzegorczyk classes of functions of the integers (F_4:Grzegorczyk (1953)), of Grzegorczyk classes of string functions (F_4: Henke, Indermark and Weihrauch (1972)).

Let \( I = \{ \sigma_1, ..., \sigma_n \} \) be a finite alphabet. Let us define the following classes of programs (LOOP programs on strings)

DEFINITION 1. A program \( \pi \) is in the class \( L_0^F \) of LOOP-programs on strings if it is a finite sequence of statements of the form:

i) \( < \text{id}_1 > = \sigma_j \text{- SUC} \text{C} < \text{id}_1 > \quad 1 \leq j \leq n \)

ii) \( < \text{id}_1 > = \varepsilon \) \quad \((\varepsilon \text{ is the empty word})\)

iii) \( < \text{id}_1 > = < \text{id}_2 > \)

where \( < \text{id}_1 >, < \text{id}_2 > \) are identifiers out of a countably infinite set of identifiers. A program \( \pi \) is in the class \( L_{i+1}^F \) (\( i \geq 0 \)) if

i) \( \pi \) is in \( L_i^F \)

ii) \( \pi \) is of the form \( \text{LOOP} < \text{id}_1 > \)

\[
\sigma_1 : \pi_1 \\
\vdots \\
\sigma_n : \pi_n \\
\text{END}
\]
where \(< id_1 >\) is an identifier and \(v_1, \ldots, v_n\) are in \(L_i^p\).

iii) \(\ast\) is the concatenation of two programs in \(L_{i+1}\).

In order to express the operational semantics of LOOP programs on strings we have to think of each identifier as being the name of a register which contains an arbitrary word in \(\Sigma^*\). Then the statements have the following meaning:

\(< id_1 > = \varepsilon\) : clear register \(< id_1 >\);
\(< id_1 > = < id_2 >\) : transfer the content of \(< id_2 >\) in register \(< id_1 >\);
\(< id_1 > = \sigma_j - \text{SUCCE} < id_1 >\) : compute the \(\sigma_j\) right (left) successor of the word which is in \(< id_1 >\) and put the result in \(< id_1 >\);

The LOOP control structure is interpreted by the following informal program:

- transfer the content of \(< id_1 >\) in the register \(< \text{control} >\);

- while \(< \text{control} > \neq \varepsilon\) execute

  \(v_1\) if the leftmost (rightmost) character of \(< \text{control} >\) is \(\sigma_1\);

  erase the leftmost (rightmost) character of \(< \text{control} >\).

As we can see, every LOOP program on strings can be interpreted in four possible ways according to the interpretations of the successor operation and of the LOOP.
control structure:

LL : left successor and leftward scanning of the <control>;
LR : " rightward "
RR : right " "
RL : " leftward "

Accordingly, we have the following definitions:

DEFINITION 2.1. A LOOP program on strings is in the class \( L_1 \) (resp. \( L_1^{LR} \), \( L_1^{RR} \), \( L_1^{RL} \)) if it is in the class \( L_1 \) and it has to be executed according to the interpretation of the statements defined LL (resp. LR, RR, RL)

\( \forall x \in \mathbb{Z} \setminus 0 \), A function \( F : \mathbb{Z}^m \rightarrow \mathbb{Z}^k \) is said to be in the class \( L_1^{LL} \) (resp. \( L_1^{LR} \), \( L_1^{RR} \), \( L_1^{RL} \)) if there is a program \( P \) in \( L_1^{LL} \) (resp. \( L_1^{LR} \), \( L_1^{RR} \), \( L_1^{RL} \)) such that if it is started with \( x_1, \ldots, x_m \) in \( m \) specified input registers\(^*\) it terminates with \( y \in \mathbb{Z}^k \) in a specified output register if and only if \( f(x_1, \ldots, x_n) = y \)

In order to simplify the notation we will denote RLOOP' (LLOOP) the LOOP construct where the <control> has to be scanned rightward (leftward) and we denote \( \sigma_1 \circ \sigma_1 \sigma_1 \) (\( \sigma_1 \circ \sigma_1 \)) the \( \sigma_1 \) right (left) successor of the word stored in register \( \text{id}_1 \). The input and output registers are declared at the beginning of a program.

Example 1

\[
\begin{align*}
\text{IN (X)} ; \text{OUT Y} \quad f(x) = x \quad \text{(copy)} \\
\text{RLOOP X} \\
O : Y = Y \circ O \\
1 : Y = Y \circ 1 \\
\text{END}
\end{align*}
\]

\(^*\) All other registers are initialized to \( \varepsilon \)
Example 2

\[ f(x) = \bar{x} \] (reverse)

LLOOP X
0 : Y = Y \rightarrow O
1 : Y = Y \rightarrow 1
END

Example 3

\[ f(x) = \overline{xx} \] (reflect)

W = X
RLOOP X
RLOOP W
0 : Y = \varepsilon
A = B
B = B \rightarrow O
Y = Y \rightarrow O
1 : Y = \varepsilon
A = B
B = B \rightarrow 1
Y = Y \rightarrow 1
END
W = A
LOOP Y
0 : Z = Z \rightarrow O
1 : Z = Z \rightarrow 1
END
END
RLOOP X
0 : Z = Z \rightarrow O
1 : Z = Z \rightarrow 1
END

In future we will allow the notation \( X = f(Z, Y) \) to stand for the program which computes the function \( f \) of the content of registers \( Z \) and \( Y \) and puts the result in \( X \); for example \( X = \text{reverse}(X) \), \( X = \text{conc}(Y, Z) \), etc.
The following basic properties are immediate consequences of the definitions and of results of Henke, Indermark and Weihrauch (1972):

**Basic properties**

i) For $|\Sigma| = 1$, $L_n^{LL,\Sigma} = L_n^{LR,\Sigma} = L_n^{RR,\Sigma} = L_n^{RL,\Sigma}$ for all $n \geq 0$ and $f \in L_n$ if and only if $\lambda x_1 \ldots \lambda x_m[\sigma_i^f(|x_1|, \ldots, |x_m|)] \in L_n^\Sigma$ (all LOOP programs on one letter strings correspond exactly to LOOP programs on integers).

ii) For all $\Sigma$, $n \geq 2$, $L_n^{LL,\Sigma} = L_n^{LR,\Sigma} = L_n^{RR,\Sigma} = L_n^{RL,\Sigma} = E_{n+1}^\Sigma$ from the second LOOP level, analogously to what happens for LOOP programs on integers, the LOOP hierarchy and the Grzegorczyk - like hierarchy on strings coincide. The result for $L_n^{LL,\Sigma}$ is proved by Henke, Indermark and Weihrauch (1972). The results for the other hierarchies are immediate because the simulation among the LL, LR, RR, RL classes can be performed within two LOOP levels.

iii) For all $\Sigma$ and all $n$ the LL(RR) LOOP classes coincide with the classes of functions defined by depth of nesting of multiple primitive left (resp. right) recursion on notation with the left (resp. right) successor as basic function.

For example, the simultaneous right recursion on notation is realized by an RR-LOOP program in the following way: let $f_1, \ldots, f_m$ be defined as follows

$$f_i(\bar{x}, r) = g_i(\bar{x}) \quad i = 1, \ldots, m; \quad j = 1, \ldots, n$$

$$f_i(\bar{x}, w \sigma_j) = h_{ij}(\bar{x}, w, f, (\bar{x}, w), \ldots, f_m(\bar{x}, w))$$

then the function $f_i$ is computed by the following
program:

IN (\overline{X}, W); OUT F_4
F_1 = g_1(\overline{X})
...
F_m = g_m(\overline{X})
Y = \epsilon
RLOOP W
...
\sigma_j : F_1 = h_{1j}(\overline{X}, Y, F_1, \ldots, F_m)
...
F_m = h_{mj}(\overline{X}, Y, F_1, \ldots, F_m)
Y = Y \circ \sigma_j
...

END

3. CLASSES OF SIMPLE PROGRAMS ON STRINGS

In what follows we will concentrate on the classes of functions computable with only one LOOP level. In the case of LOOP programs on integers Tsichritzis (1970) characterizes the one LOOP level class of functions in terms of basic functions and closure under composition ("simple functions") and from this characterization derives decidability properties for the class of programs $L_1$ ("simple programs"). In order to establish decidability results for the programs in the classes $L^e_1$ we examine more carefully some of their properties. Let $|\Sigma| \geq 2$

FACT 1 1) $L^e_0, RR = L^e_0, RL$ and $L^e_0, LL = L^e_0, LR$
12.

ii) $L_{o}^{L, RR} \neq L_{o}^{L, LL}$

PROOF 1) obvious

ii) All functions in $L_{o}^{L, RR}(L_{o}^{L, LL})$ are of the form $f(x_1, \ldots, x_n) = x_1 \circ w$ (where $w \in \mathcal{L}^*$), or $f(x_1, \ldots, x_n) = w$.

hence $f_R(x) = x \circ w$ is in $L_{o}^{L, RR}$ and not in $L_{o}^{L, LL}$

while $f_L(x) = w \circ x$ is in $L_{o}^{L, LL}$ and not in $L_{o}^{L, RR}$

FACT 2

$L_{o}^{L, RR} \cup L_{o}^{L, LL} \not\subset L_{1}^{L, RR} \cap L_{1}^{L, LL}$

PROOF

The containment is immediately proved. The fact that it is proper derives from the fact that the concatenation is in any of $L_{1}^{L}$ but in none of $L_{o}^{L}$.

Q E D

The relative properties of the classes defined by one LOOP level programs are summarized in the following result:

**THEOREM 3**

i) $L_{1}^{L, RR} \subset L_{1}^{L, RL}$, $L_{1}^{L, LL} \subset L_{1}^{L, LR}$

ii) $f \in L_{1}^{L, RR}$ iff $f \in L_{1}^{L, LL}$

where

$f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$

iii) $L_{1}^{L, RL} = L_{1}^{L, LL}$

PROOF

1) The "reverse" function is in both $L_{1}^{L, RL}$ and $L_{1}^{L, LR}$ but in none of $L_{1}^{L, LL}$ and $L_{1}^{L, RR}$. On the other side any function in $L_{1}^{L, RR}(L_{1}^{L, LL})$ is computable by RL(LR) programs because the
sequence

R LOOP X

\* END \* can be realized by

y = reverse (X)

L LOOP X

\* END \*

ii) By induction on the structure of the definition of a function in \( L_1^{L,RR} \) we can prove that \( f \in L_1^{L,RR} \) implies \( f \in L_1^{L,LL} \) (the other part of the statement has a symmetrical proof).

In order to prove the result we show the following *Claim*: given any program \( \pi \) in \( L_1^{L,RR} \) there is a program \( \hat{\pi} \) in \( L_1^{L,LL} \) such that if at the beginning of the execution of \( \pi \) the content of all registers is \( x_1, \ldots, x_n \) and at the end of the execution of \( \pi \) the content of all registers is \( z_1, \ldots, z_n \) then when the registers of \( \hat{\pi} \) are initialized to \( \hat{x}_1, \ldots, \hat{x}_n \), the content of the registers at the end of the execution of \( \hat{\pi} \) is \( \hat{z}_1, \ldots, \hat{z}_n \).

a) \( \pi \) is in \( L_0^{L,RR} \). Let \( \hat{\pi} \) be obtained by replacing every right successor in \( \pi \) with a left successor. Every register \( X_i \) at the end of the execution of \( \pi \) will contain \( z_i = x_j \circ w_i \) or \( z_i = w_i \) where \( x_j \) is the content of a register \( X_j \) at the beginning of the execution of \( \pi \). Clearly the content of register \( X_i \) at the end of the execution of \( \hat{\pi} \) will be \( \hat{z}_i = \hat{x}_j \circ \hat{w}_i \) or \( \hat{z}_i = \hat{w}_i \).

b) \( \pi \) is of the form...
R LOOP X
 ...
 \sigma_1 : \tau_1
 ...
 END

Let \tau be the following program:
L LOOP X
 ...
 \sigma_1 : \tau_1
 ...
 END

where any \tau_1 is the program in L^LL_0 obtained by replacing right successors with left successors in \tau_1. Without loss of generality we may assume that X does not occur in the LOOP body. We may prove the claim by induction on the length of the content of the word x contained in X.

If \|x\| = 0 the LOOP body is never executed and the claim is obviously true.

If x = y \circ \sigma_h and \|y\| = k by inductive hypothesis the claim is supposed to be true for \tau and \tau when X is initialized to y and to y respectively. If \tau_h is such that at the end of the execution of \tau_h \nabla_i = x_j \circ w_i or \nabla_i = w_i at the end of the execution of \tau (with X initialized to x) we have

\nabla_i = \nabla_j \circ w_i or \nabla_i = w_i

where \nabla_i is the content of register \nabla.
ialized to \( y \). By inductive hypothesis the content of \( X_j \), after the execution of \( \tilde{y} \), when all registers \( x_1, \ldots, x_n \) are initialized to \( \tilde{x}_1, \ldots, \tilde{x}_n \) and \( X \) is initialized to \( \tilde{y} \), is \( \tilde{z}_j \). Hence the content of \( X \) after the execution of \( \tilde{y} \) when \( X \) is initialized to \( \tilde{x} = \sigma_h \cdot \tilde{y} \) is

\[
\tilde{x}_1 \cdot \tilde{z}_j = \tilde{z}_1 \text{ or } \tilde{x}_1 = \tilde{z}_1
\]

(c) \( \tilde{x} \) is the concatenation of two programs in \( L_1^{\text{RR}} \). The claim may be proved by induction on the length of \( \tilde{x} \), assuming as bases of the induction the cases in which \( \tilde{x} \) is of the form a) or is of the form b).

iii) The proof is very similar to the proof of part ii). First it can be proved that \( f \) is in \( L_1^{\text{RL}} \) iff \( \tilde{f} \) is in \( L_1^{\text{LR}} \). Then it can be observed that \( f \) is in \( L_1^{\text{RL}} \) (\( L_1^{\text{LR}} \)) iff \( \tilde{f} \) is in \( L_1^{\text{RL}} \) (\( L_1^{\text{LR}} \)) because the reversal can be defined in \( L_1^{\text{RL}} \) (\( L_1^{\text{LR}} \)).

QED

As far as the relation among the \( L_1^{\text{L}} \) classes and other known classes of string functions and transductions, the following facts hold:

FACT 4 1) \( L_1^{\text{RL}} \subseteq E_0^{\text{RL}} \)

ii) \( L_1^{\text{RL}} \not\subseteq E_1^{\text{RL}} \)

PROOF 1) Concatenation is in \( L_1^{\text{RL}} \) but not in \( E_0^{\text{L}} \). On the other side Rose and Weihrauch(1973) show that if \( f \) is in \( L_1^{\text{LL}} \) then there exists a \( k \) such that for all \( z \in \text{range} \ (f) \) \( \exists x_1, \ldots, x_n \) such that \( f(x_1, \ldots, x_n) = z \) and \( \forall \ \ell \in [1, k] \)
The same can be easily seen to hold for $L_1^{L, RL}$. Since the function $\sqrt{x}$ such that for all $x$

$$|\sqrt{x}| = \left\lfloor \sqrt{|x|} \right\rfloor$$

is definable in $L_0^L$ and is not in $L_1^{L, RL}$ the proof is completed.

ii) For all functions $f$ in $L_1^{L, RL}$ it can be shown that there are constants $c_1, \ldots, c_n$ such that

$$|f(x_1, \ldots, x_n)| \leq \Sigma c_i |x_i|.$$ 

Hence all functions in $L_1^{L, RL}$ can be defined in $L_1^L$ by primitive recursion on notation limited by concatenation. On the other side fact 4 i) implies proper containment.

Q.E.D.

**Definition 4.** A *generalized sequential machine* is a 6-tuple $M = <K, \Sigma, \Delta, \delta, \lambda, q_o>$ where $K$ is the finite set of states, $\Sigma$ is the input alphabet, $\Delta$ is the output alphabet, $\delta : K \times (\Sigma \cup \{\varepsilon\}) \rightarrow K$ is the state transition function, $\lambda : K \times (\Sigma \cup \{\varepsilon\}) \rightarrow \Delta^*$ is the output function, $q_o$ is the initial state.

**Definition 5.** Let $M$ be a generalized sequential machine; let $\overline{\lambda}, \overline{\delta}$ be the output function and the state transition function extended to $\Sigma^+$. Then we denote $g_M$ the mapping defined by $g_M(x) = \overline{\lambda}(q_o, x)$ for every $x \in \Sigma^*$. $G$ is the class of mappings defined by generalized sequential machines.

**Theorem 5.** (Rose and Weihrauch(1973)) $G \subseteq L_1^{L, LL}$

**Proof** For any generalized sequential mapping $g$ a program $\pi \in L_1^{L, LL}$ can be defined such that $\pi$ computes $g_M$; for any transition rule $\delta(q_i, \sigma_j) = q_{ij}$ and any output rule $\lambda(q_i, \sigma_j) = a_{ij}$
in $N$, $\pi_j$ realizes the mapping $Q_i = a_{ij} o Q_{ij}$ where $Q_{ij}$ is the content of register $X_{ij}$ at the beginning of the execution of program $\pi_j$ and $Q_i$ is the content of register $X_i$ at the end of the execution of program $\pi_j$. Then $\pi$ is $L$ LOOP $X$

...  

$\sigma_j : \pi_j$

...  

END

and the output is in $X_0$ (the register corresponding to the initial state). On the other side a gsm mapping preserves regular sets. Since a $L_1^{E, LL}$ program can map $\Sigma^*$ into $\{a^n b^n | n \geq 0\}$ there are functions in $L_1^{E, LL}$ which are not gsm mappings.

QED

**THEOREM 6**  
$G \subseteq L_1^{E, LL} \cap L_1^{E, RR}$

**PROOF.** If $g \in G$ then $\lambda x [\text{reverse} (g (\text{reverse} (x)))] \in G$. Hence, by theorem 3 ii) and theorem 5, $G \subseteq L_1^{E, RR}$. On the other side the mapping from $\Sigma^*$ into $\{a^n b^n | n \geq 0\}$ can be defined both in $L_1^{E, LL}$ and $L_1^{E, RR}$.

QED

In the next paragraph we will consider $L_1^E$ programs as acceptors. Here we consider two simple basic properties of ranges of $L_1^E$ functions:

**FACT 7  i)** If $|\Sigma| \geq 2$ there is a function $f$ in $L_1^{E, LL}$ such that range $(f)$ is not a bounded language.

**ii)** For all $\Sigma$, for all $f \in L_1^{E, RL}$, range $(f)$ is a context sensitive language

**PROOF.**  

**i)** For $|\Sigma| \geq 2$ $\Sigma^*$ is not a bounded language.

**ii)** any program in $L_1^{E, RL}$ can be simulated in linear space by a deterministic Turing machine
Fig. 1. Subelementary classes of recursive functions.

$E_1$ : Grzegorczyk classes on integers
$L_1$ : Loop classes on integers (Meyer-Ritchie)
$L_1^{+, \text{tot}}$ : Total presburger functions (Cherniavsky)
$L_1^{ER, RE}$ : $gsm$ mappings
$E_1^\Sigma \Sigma$ : Grzegorczyk classes on strings (Weihrauch)
$L_1^{\Sigma \Sigma}$ : Loop classes on strings (Weihrauch)
$L_1^{\Sigma, RR, \Sigma, RL}$ : Simple classes defined by RR and RL rules
$P$ : Polynomial time computable functions (Cobham)
and, hence \( z \in \text{range}(f) \) for \( f \in L^1_{\text{RL}} \) can be recognized by simulating the program for all inputs \( x_1, \ldots, x_n \) which satisfy \( \Sigma_1 |x_i| \leq k (|z| + 1) \) (see proof of fact 4 i) and this can be done in deterministic linear space.

QED

REMARK Fact 7 i) is particularly interesting since the graphs of \( L^1_{\text{RL}} \) functions on one letter alphabets are bounded c.f languages (Theorem 1, Sistelss (1970)) and from this fact decidability of equivalences of programs and its complexity can be derived.

4. SIMPLE ACCEPTORS.

As it will be more clear in the following pages, one typical aspect of programs in \( L^1_{\text{RL}} \) (we will make a distinction among programs of type RR, LL, RL only when needed) is the large difference among the recognition power and the generation power of these programs. Within one LOOP level, in fact it is not possible to recognize a much wider class than regular languages (the problem whether it is actually possible to recognize any non regular language is open), while it is possible to generate even a context sensitive language such as \( \{a^n b^n c^n | n \geq 1\} \). A consequence of this fact will be that problems about languages accepted by simple programs are indeed easier than problems about functions computed by simple programs. In this paragraph we consider simple programs as acceptors, in the following sense.

DEFINITION 6. Let \( \pi \) be a program in \( L^1_{\text{RL}} \) and let \( X, Y \) be two registers. We define \( L(\pi, X, Y) \) to be the language accp
ted by program \( \pi \) if \( L(\pi, X, Y) = \{ x \in \mathbb{L}^* | f(x) \neq \varepsilon \} \) where \( f \) is the function computed by \( \pi \) with input in \( X \) and output in \( Y \). When \( X \) and \( Y \) are fixed we will write \( L(\pi) \).

DEFINITION 7. Simple languages are the languages accepted by programs in \( L_1^{\varepsilon, \text{RR}} \) we will denote the class of simple languages by \( S_1^\varepsilon \).

The fact that we chose the acceptance to be characterized by a non-empty output (and rejection characterized by empty output) does not restrict us in any sense. In fact we can prove that acceptance and rejection can be equivalently expressed by characteristic functions with output 0 (rejection) or 1 (acceptance).

FACT 8. Given any program \( \pi \in L_1^\varepsilon \) that accepts a language \( L \), there is a program \( \pi' \in L_1^\varepsilon \) which computes the characteristic function of \( L \) and vice versa.

PROOF. Immediate.

As a first step toward considering the properties of sets of \( L_1^{\varepsilon, \text{RR}} \) programs we will consider closure properties of the class of simple languages.

THEOREM 9 \( S_1^\varepsilon \) is closed under

i) complementation

ii) union

iii) concatenation by finite sets

PROOF i) Let \( \pi \) accept \( L \). Then \( \pi' \) accepts \( L' = \mathbb{L}^\varepsilon - L \) where \( \pi' \) is the programs

\[
\begin{align*}
\pi & \\
Z &= 1 \\
\text{RLOOP } Y & \\
0 : Z &= \varepsilon \\
1 : Z &= \varepsilon \\
\text{END}
\end{align*}
\]

ii) Let \( \pi' \) by the following program:
\[ X_1 = X \]
\[ X_2 = X \]
\[ \pi_1 \]
\[ \pi_2 \]
\[ Y = Y_1 \]
\[ Y = \text{conc}(Y, Y_2) \]

Then \( L(\pi', X, Y) = L(\pi_1, X_1, Y_1) \cup L(\pi_2, X_2, Y_2) \)

iii) Let \( F = \{ w_1, \ldots, w_n \} \) be a finite subset of \( \Sigma^* \); then
given any \( w_i = \sigma_1 \ldots \sigma_{ik_1} \) if \( \pi \) is a program for \( L, \pi \) is a program for \( L \circ w_i \):

\[ X_{k_1} = \text{last digit of } X \]
\[ X = X \text{ without last digit} \]
\[ Z = Z^0(\text{if } X_{k_1} = \sigma_{ik_1} \text{ then } 1 \text{ else } 0) \]
\[ \ldots \]
\[ X_1 = \text{east digit of } X \]
\[ X = X \text{ without last digit} \]
\[ Z = Z^0(\text{if } X_1 = \sigma_{i1} \text{ then } 1 \text{ else } 0) \]
\[ \pi \]
\[ W = 0 \]
\[ \text{RLOOP } Y \]
\[ 0 : W = 1 \]
\[ 1 : W = 1 \]
\[ \text{END} \]

\[ Z = \text{copy } (W, Z) \]
\[ Z = \epsilon \text{ if } Z \notin \{ 1 \}^+ \]
Since simple languages are closed under union the proof easily follows.

QED

COROLLARY $S^L$ is closed under intersection.

Remember that as a consequence of theorem 6 we also have

FACT 10. $S^L$ contains the regular languages.

6. DECIDABILITY PROPERTIES FOR SIMPLE PROGRAMS ON STRINGS.

Our main goal will be to establish decidability properties for programs in $L^L_{1, RR}$. In order to do so and to establish lower and upper bounds for the complexity of decidable properties we will relate sets of programs in the following way:

DEFINITION 8. i) If $A$ and $B$ are two sets of programs in $L^L_{1, RR}$ we define $A \leq B$ if there exists a function $f : L^L_{1, RR} \rightarrow L^L_{1, RR}$ such that

- $\pi \in A$ iff $f(\pi) \in B$

- $|f(\pi)| \leq c |\pi|$ (the length of a program takes into account the number of statements and the length of identifiers).

- $f$ is Turing computable within $\log$ (possibly constant) space.

ii) $A \equiv B$ if $f A \leq B$ and $B \leq A$

Let us now consider the following sets of programs in $L^L_{1, RR}$.
DEFINITION 9 \( \text{ACC}\emptyset = \{ \pi | L(\pi) \text{ is empty} \} \)

\( \text{ACCEPT} = \overline{\text{ACC}} \emptyset = \{ \pi | (\exists x) [\pi \text{ accepts } x] \} \)

\( \text{EQACC} = \{ <\pi_1, \pi_2 > | L(\pi_1) = L(\pi_2) \} \)

\( \text{INEQACC} = \overline{\text{EQACC}} = \{ <\pi_1, \pi_2 > | L(\pi_1) \neq L(\pi_2) \} \)

\( \text{EQUIV} = \{ <\pi_1, \pi_2 > | (\forall x_1, \ldots, x_n) [f_{\pi_1}(x_1, \ldots, x_n) = f_{\pi_2}(x_1, \ldots, x_n)] \}

\( \text{INEQUIV} = \overline{\text{EQUIV}} = \{ <\pi_1, \pi_2 > | (\exists x_1, \ldots, x_n) [f_{\pi_1}(x_1, \ldots, x_n) \neq f_{\pi_2}(x_1, \ldots, x_n)] \}

\( \text{RINT} = \{ <\pi_1, \pi_2 > | \text{range}(f_{\pi_1}) \cap \text{range}(f_{\pi_2}) \neq \emptyset \} \)

\( \text{GRINT} = \{ <\pi_1, \pi_2 > | \text{graph}(f_{\pi_1}) \cap \text{graph}(f_{\pi_2}) \neq \emptyset \} \)

DEFINITION 10 \( \text{PCP} = \text{Post's correspondence problem} = \)

\( = \{ <u_1, \ldots, u_n | v_1, \ldots, v_n > | |u_i|, |v_i| \leq |\Sigma|, |\Sigma| > 2, \Sigma^* \} \)

PCP is well known to be undecidable.

The following reductions are consequences of properties of \( \text{L}_1^{*}, \text{RR} \) programs established in the preceding paragraphs.

THEOREM 11 1) \( \text{ACC}\emptyset \equiv \text{EQACC} \)

ii) \( \text{ACC}\emptyset \leq \text{EQUIV} \)

PROOF 1) \( \text{ACC}\emptyset \leq \text{EQACC} \)

Let \( \pi \emptyset \) be the program which accepts the empty set \( \pi \).

\( \text{ACC}\emptyset \leq \text{EQACC} \)

\( <\pi_1, \pi_2 > \in \text{EQACC} \text{ iff } L(\pi_1) \equiv L(\pi_2) \)

\( \text{iff } (L(\pi_1) \cap L(\pi_2)) \cup (L(\pi_2) \cap L(\pi_1)) = \emptyset \)

where \( \pi' \) is the program such that (see theorem 9)

\( L(\pi') = (L(\pi_1) \cap L(\pi_2)) \cup (L(\pi_2) \cap L(\pi_1)) \)

It is easy to see that also in this case
$|\pi'| \leq c(|\pi_1| + |\pi_2|)$ and the function mapping $<\pi_1, \pi_2>$ into $\pi'$ does not require more than $\log n$ space to take care of renamings.

ii) $\textsc{ACC} \preceq \textsc{EQUIV}$

Let $\pi_o^0$ be the program which gives constant output 0 and let $\pi_C^0$ the program which computes the characteristic function of $L(\pi)$. Then

$\pi \in \textsc{ACC} \iff <\pi_C^0, \pi_o^0> \in \textsc{EQUIV}$

QED

In a similarly straightforward manner we can prove

**THEOREM 12**

i) $\textsc{ACCEPT} \equiv \textsc{INEQACC}$

ii) $\textsc{ACCEPT} \preceq \textsc{INEQUIV}$

**PROOF**

1) $\pi \in \textsc{ACCEPT} \iff$ the complement of $L(\pi)$ is not $L^c$; on the other side $L(\pi_1) \neq L(\pi_2)$ iff $(L(\pi_1) \cap L(\pi_2)) \cup (L(\pi_2) \cap L(\pi_1))$ is not empty

ii) $\pi \in \textsc{ACCEPT} \iff$ the characteristic program of $L(\pi)$ is not equivalent to the characteristic program which outputs the constant 0.

QED

![Diagram](image_url)

*Fig. 2. Reductions among programs in $L_1^{\Sigma, RR}$*
The following theorems state the decidability of the equivalence of simple acceptors and derive an upper bound and a lower bound for the decision procedure.

**THEOREM 13.** EQACC is decidable

**PROOF.** Since EQACC = INEQACC and INEQACC ≡ ACCEPT it will be sufficient to show that ACCEPT is decidable. \( \pi \in \text{ACCEPT} \) iff \( (\exists x)[x \in L(\pi)] \) and this is true iff \( \overline{\pi} \) rejects at least the word \( x \) (where \( L(\overline{\pi}) = \overline{L(\pi)} \)) which means \( x \in \text{range } (f_\pi) \). Since the range of a function in \( L_1^{\text{RR}} \) is decidable, ACCØ is decidable.

QED

**THEOREM 14.** The decision procedure for INEQACC requires at most exponential space.

**SKETCH OF THE PROOF.** Let \( k_\pi \) be the constant given by the already cited pumping lemma (such that for any \( z \in \text{range}(f_\pi) \) there exist \( x_1, \ldots, x_n \) such that \( f(x_1, \ldots, x_n) = z \) and \( \sum |x_i| \leq k_\pi (|z| + 1) \)). In order to decide the inequivalence of \( \pi_1, \pi_2 \) we have to derive \( \pi^* \) such that \( L(\pi^*) = \overline{L(\pi_1)} \) and \( L(\pi^*) = (\overline{L(\pi_1)} \cap L(\pi_2)) \cup (\overline{L(\pi_2)} \cap L(\pi_1)) \) and decide if \( \pi^* \) rejects any word \( x \) such that \( |x| \leq k_\pi^* \). Since \( k_\pi^* \) grows exponentially with the length of \( \pi^* \) (see Rose and Weihrauch (1973)) we need as much space to store the initial value of the registers on the tape of a Turing machine, while the simulation of \( \pi^* \) will bring only a linear increase on the amount of tape.

QED

On the other side the time lower bound for INEQACC can be easily stated by showing that the satisfiability of formulae of propositional calculus can be reduced to the equivalence of languages accepted by simple programs (the reduction being, also in this case, linear).
THEOREM 15. CNF \leq IN EQACC

PROOF. The proof can be given essentially by using a result of Constable, Hunt and Sahni (1974) who prove that CNF is reducible to the inequivalence of simple programs on integers. This would imply the reducibility of CNF to INEQUIV. By a slight modification we obtain the desired result. The direct proof works as follows.

Let \( w \) be any formula in CNF and let \( |w| = n \).

We can define a \( L^R,RR \) program \( \pi \) (whose length is \( \leq cn \)) such that \( \pi \) does not accept \( \varepsilon \) if and only if \( w \) is satisfiable.

Let \( w = C_1 \land \ldots \land C_k \), \( C_i = C_{i1} \lor \ldots \lor C_{in_i} \), \( C_{ij} = \frac{x_i}{x_j} \).

Let \( m \) be the number of proportional variables.

Then \( \pi \) is the following program (input in \( X \), output in \( Y \)).

\[
\begin{align*}
\text{set } X_j &= \begin{cases} 1 \text{ if } p_j \text{ divides } |X| \\ 0 & \text{ otherwise} \end{cases} \\
\text{INIT} & \\
C &= C \circ 0 \\
\pi_1 & \\
C &= C \circ 0 \\
\pi_2 & \\
& \ldots \\
\pi_k & \\
\text{RLOOP } C & \\
0 : S &= X \\
Y &= 1 \\
1 : \lor &= \end{align*}
\]

\[
\begin{align*}
\pi & \text{ adds } 1 \text{ to } C \text{ any time one of } \\
C_{ij} & \text{ is } "\text{true}". \text{ That is } \pi_1 \text{ is the } \\
& \text{ program } \\
\pi_1 & \\
\vdots & \\
\pi_{in_1} & \\
\text{RLOOP } X_j & (x_j) \\
0 & : - \\
1 : C &= C \circ 1 \\
& \ldots \\
\end{align*}
\]
The last RLOOP construct is needed to explore C and give output ϵ if and only if C contains a string of the form \((0 \, 1^+)^k\) that is if and only if w is satisfiable by the truth assignment determined by the input X.

QED

COROLLARY. INEQACC is NP-hard.

REMARK. The same result holds for EQACC.

Of course the same reductions can be used to show that, if decidable, the problems of equivalence and inequivalence of programs in \(L^e_{1,RR}\) are at least polynomially complex in nondeterministic time.

In Appendix B it is shown that if we allow the basic instruction

\[
<id_1> = <id_2> \times <id_3>
\]

(which puts the content of \(<id_1>\) to ϵ if the content of \(<id_2>\) and the content of \(<id_3>\) are both ϵ, 1 otherwise) the problem of the equivalence of programs becomes PTAPE hard.

7. AN UNDECIDABILITY RESULT.

As we have noticed before, the power of \(L^e_{1,RR}\) programs as generators is much stronger than their power as recognizers.

This results in a greater complexity of problems about ranges of simple functions on strings than of problems about sets recognized by simple programs.

THEOREM 17. i) For \(|E| \geq 2\), PCP \(\leq\) GRINT

ii) For \(|E| \geq 2\), PCP \(\leq\) RINT
REMARK. The reduction symbol $<$ is used also in this case where PCP is not a subset of $L^*_{\text{RR}}$ because the mapping we will use has analogous properties.

PROOF. 1) Let $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ be given words an $\mathbb{I} = \{0, 1\}$. Then we may construct two programs $\tau_u$ and $\tau_v$ in the following way:

\begin{align*}
\text{RLOOP } X & \\
0 : & Y_1 = Y_1 \circ u_1 \\
\ldots & \\
Y_n = Y_n \circ u_n \\
1 : & Z = Y_n \\
Y_n = Y_{n-1} \\
\ldots & \\
Y_2 = Y_1 \\
Y_1 = Z \\
\text{END}
\end{align*}

\begin{align*}
\text{RLOOP } X & \\
0 : & Y_1 = Y_1 \circ v_1 \\
\ldots & \\
Y_n = Y_n \circ v_n \\
1 : & Z = Y_n \\
Y_n = Y_{n-1} \\
\ldots & \\
Y_2 = Y_1 \\
Y_1 = Z \\
\text{END}
\end{align*}

Let the input be in $X$ and the output in $Y_1$. Suppose there is a sequence $i_1, \ldots, i_k$ such that

$$u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$$

then for

$$X = \underbrace{11 \ldots 1}_{i_1-1} \underbrace{0 1 \ldots 1 0}_{n-i_1+i_2} \ldots \underbrace{0 1 \ldots 1}_{n-i_k+1}$$

\begin{align*}
i_1 & = 11 \ldots 1 \\
i_2 & = n-i_1+i_2 \\
i_3 & = n-i_1+i_2 \ldots
\end{align*}
we have \( f_u(x) = f_v(x) = u_{i_1} \ldots u_{i_k} \).

On the other hand if the programs give the same output for the same input string then there is a sequence of integers \( i_1, \ldots, i_k \) which satisfies the given PCP.

In fact we may prove by induction on the length of \( X \) that for any \( X \), for any \( i \), registers \( Y_i \) in both programs contain sequences of words indexed by the same sequence of integers.

If \( |X| = 0 \) the claim is obviously true.

If \( |X| = k + 1 \) and \( w \) is any word of length \( k \), by inductive hypotheses with input \( w \) all registers \( Y_i \) in \( \pi_u \) contain sequences of words indexed by the same sequences of integers as registers \( Y_i \) in \( \pi_v \):

- if \( X = w \circ 1 \) then with input \( X \) registers are permuted in the same way in both programs and the property is preserved;

- if \( X = w \circ 0 \) then the string contained in any register \( Y_i \) in \( \pi_u \) gets concatenated to the string \( u_i \) and the string contained in any register \( Y_i \) in \( \pi_v \) gets concatenated to the string \( v_i \), so that the property is preserved also in this case.

ii) Let \( \pi_1, \pi_2 \) be as in part i).

Let \( \pi_i^j (i=1, 2) \) be the following programs on \( I \cup \{ \emptyset \} \) (output in \( Y_i^j \)):

\[
\pi_i^j \quad Y_i^j = X \circ \xi \circ Y_i
\]

Then \( \langle <u_1, \ldots, u_n>, <v_1, \ldots, v_n> \rangle \in \text{PCP} \) iff

\( \langle \pi_1, \pi_2 \rangle \in \text{GRINT} \) iff \( \langle \pi_1^1, \pi_2^1 \rangle \in \text{RINT} \).

Let \( \tau \) be the following homomorphism:

\[
\tau(0) = 101, \tau(1) = 10^2 1, \tau(\emptyset) = 10^3 1
\]

we can define \( \pi^\tau \) on \( I \) in the following way:
π₁

\[ Y_1^n = \tau(X) \circ 10^3 1 \circ \tau(Y_1) \]

Hence \( <\pi_1^n, \pi_2^n> \in \text{RINT} \) iff \( <\pi_1^n, \pi_2^n> \in \text{RINT} \).

QED

COROLLARY  

1) The graph intersection of simple functions is undecidable for \(|\Sigma| \geq 2\).

ii) The range intersection of simple functions is undecidable for \(|\Sigma| \geq 2\).

REMARK. The intersection problem for simple programs on integers is decidable (see Appendix A). This shows that the extension of LOOP programs from integers to strings is not trivial.

ACKNOWLEDGEMENTS I wish to thank Juris Hartmanis and Bob Constable for many interesting discussions and for making my visit so enjoyable.
REFERENCES

decidable logical theories. SUNY at
Stony Brook, Dept. of C.S.

computational complexity of scheme equi-
valence. Dept. C.S.-TR 74-201, Cornell
University.

propositional calculus. Proceedings of
VII ACM Symposium on Theory of Compu-
ting, Albuquerque, N.M.

functions. Rozprawy Matematyczne, 4.

chies of primitive recursive word fun-
tions and transductions defined by
automata. Automata, Languages and Pro-
gramming, Nivat Ed. North Holland.

complete for nondeterministic log space.
(S-TR-75). University of Kansas.

are hard to analyse. To be published in
JACM.

xity and program structure. IBM Research
RC-1817.

blem for regular expressions with squa-
ring requires exponential space. XIII Sym-
posium on Switching and Automata Theory.


APPENDIX A

Complexity of decidable properties for simple programs on integers.

In the following we state some known results and comments on the decidability of certain properties for simple programs on integers.

RESULT 1. The simple programs are the one LOOP level programs on the integers. The simple functions are the integers functions computable by simple programs; they can be expressed as composition of the following basic functions: successor, identity, zero, sum, predecessor, division by a constant, remainder by a constant and $\lambda x_1 \lambda x_2 \{ if \ x_2 = 0 \ then \ x_1 \ else \ 0 \}$.

RESULT 2. Let a simple program be given. Let us express the function computed by this program as a simple function. Then let us define two parameters $M$ and $T$ in the following way:

- $M_1$, the number of occurrences of the function $\_ \ 1$,
- $(t_1)$, set of constants appearing in a division operation or in a remainder operation, allowing repetitions

$$T = \max \{ M_1, t_1, 1 \}$$

$$M = M_1 \cdot T + 1$$

Then the following is true:

1) for all $x_1', \ldots, x_n'$,

$$f(x_1', \ldots, x_n') = f(x_1, \ldots, x_n) + a_1(x_1' - x_1) + \ldots + a_n(x_n' - x_n)$$

if $x_1' = x_1$ or $x_1' = x_1 + bT$ and $x_1 > M$

(In the one dimensional case $f(x + bT) = f(x) + a \cdot bT$

if $x > M$

ii) The value of $f(x_1', \ldots, x_n')$ is completely specified from
the values the function takes in the region

\[ N = \{ <x_1, \ldots, x_n> | x_1 \leq M + 2T \} \]

\[ f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) + \frac{\sum_{r=1}^{n} (f(x_1^r, \ldots, x_n^r) - x_r - y_r)}{T} \]

where \( y_i = x_i \pmod{T} \)
\( M < y_1 < M + T \)
\( z_i^r = y_i \quad \forall r \quad M < z_i^r < M + T \)
\( z_r^r = y_i + T \quad M + T < z_r^r < M + 2T \)

(In the one dimensional case

\[ f(x) = f(y) + (f(z) - f(y)) \frac{x - y}{T} \quad M < y < M + T \]

that is

\[ f(x) = f(y) + ps \]

where

\[ f(z) = f(y) + p \quad x = y + sT \]

iii) Two simple programs are equivalent if the functions they compute coincide in any point of the region

\[ V = \{ <x_1, \ldots, x_n> | x_1 \leq M_1 + M_2 + 2T_1 T_2 \} \]

RESULT 3. The equivalence of simple programs on integers is NP complete.

PROOF. The inequivalence (and the equivalence) can be shown to be NP hard because we can reduce the satisfiability problem (the tautology problem) to it. On the other side the inequivalence can be shown to be in NP. By Result 2 iii) we nondeterministically examine the region \( V \) to check whether for some point \( x_1, \ldots, x_n \) the two programs give different output. On each point the program takes time at most polynomial in the length of the program.

QED

RESULT 4. The equivalence of simple programs takes at most time \( 2^\alpha \log_2 \)
PROOF For any program $i$ the parameter $T_i$ and $M_i$, according to the definition can be bounded as follows.

$$T_i = \max \{ r_1 \ k_i \cdot 1 \} \leq \ell \quad \text{every } k_j \leq \ell \quad \text{for } j \leq \ell$$
$$M_i = (c + 1) \cdot T_i + 1 \leq \ell^{\ell+1} \quad \text{for } i \leq \ell$$

Volume of the hypercube $V$

$$\leq (2 \ell^{\ell+1} + 2 \ell^{2\ell}) n \leq 4 \ell^{2n\ell} \leq 2c\ell \log \ell.$$

Computation time on any point takes time at most polynomial in the size of the input (log of coordinates of the point) and hence polynomial in the length of the program.

QED

RESULT 5. The intersection of the graphs of two simple functions $g_1$, $g_2$ is decidable.

PROOF Since Result 2,ii) holds both for $g_1$, $g_2$ it is enough:
- to compute $g_1$ and $g_2$ for all $x \leq \max \{ M_1 + 2T_1, M_2 + 2T_2 \}$
- given any pair $y_1, y_2 \leq M_1 \cdot y_1 \leq M_1 + 2T_1$ to check if

$$\exists s_1, s_2 > 1 \text{ such that }$$

$$\begin{cases}
    g_1(y_1) + p_1 s_1 = g_2(y_2) + p_2 s_2 \\
    y_1 + s_1 T_1 = y_2 + s_2 T_2
\end{cases}$$

where $p_1 = g_1(y_1 + T_1) - g_1$

which is clearly decidable. QED
APPENDIX B

Simple programs on strings with weak sum.

DEFINITION 1. A simple program with weak sum is a program in \( L_{1}^{\Sigma, RR} \) where also the basic instruction

\[
<\text{id}_1> = <\text{id}_2> \times <\text{id}_3>
\]

is allowed,

which means \( <\text{id}_2> \) contain

\[
\begin{cases} 
\text{if } <\text{id}_2> = \varepsilon \text{ and } <\text{id}_3> = \varepsilon \\
1 \text{ otherwise}
\end{cases}
\]

THEOREM 1. Let \( A_N = <\Sigma, \delta_N, \delta, F, Q_o> \) be a nondeterministic finite automaton. We can define a program \( \pi \) which accepts the same language as \( A_N \).

PROOF. Let \( K = \{q_1, \ldots, q_n\} \) and \( \Sigma = \{\sigma_1, \ldots, \sigma_m\} \). The program \( \pi \) makes use of the registers \( X, Y, Q_1, \ldots, Q_n \) and is defined as follows:

\`
\*INIZ
RLOOP X
...
\sigma_j : \pi_j
...
\*FIN
\`

where \( \*\text{INIZ} \) puts 1 in all \( Q_i \)

\`
\...
where \( \*\text{INIZ} \) puts 1 in all \( Q_i \)
\...
\`

such that \( q_i \in Q_o \),

where \( \pi_j \) is such that at the end of the execution of \( \pi_j \)

the content

of \( Q_k \) is

\[
\begin{cases} 
\varepsilon \text{ if for no } q_i \quad q_k \in \delta_N(q_i, \sigma_j) \\
X Q_i \quad \text{ for all } q_i \text{ such that } q_k \in \delta_N(q_i, \sigma_j)
\end{cases}
\]
and \( \pi_{\text{FIN}} \) puts in \( Y \) the weak sum of all register corresponding to final states.

COROLLARY. The equivalence of languages accepted by simple programs with weak sum is PTAPE hard.

PROOF. For any regular expression of length \( n \) there is a NDFA with \( K \leq 2n \) states and for any such NDFA there is a program of length \( \ell \leq 2K(c+1) \) where \( c \) is the max indegree of a state, that is \( \ell = O(n^2) \). Since the problem of the equivalence of regular expressions is PTAPE complete, this is the lower bound for the complexity of the problem of the equivalence of simple programs with weak sum. Q.E.D.