# DECISION PROCEDURES FOR FAMILIES OF DETERMINISTIC PUSHDOWN AUTOMATA

Leslie G. Valiant

Department of Computer Science,
University of Warwick.

A dissertation submitted for the degree of Doctor of Philosophy. July 1973.

# P R E F A C E

I should like to thank my supervisor, Michael Paterson for his continued interest and encouragement and for the numerous discussions with him that provided the stimulation for this research.

I am also grateful to Albert Meyer for bringing the regularity problem to my attention, to the Science Research Council for financial support, and to Jill Pladdys and Rosemary Went for typing this thesis.

All the work presented here is the author's, except where otherwise indicated, and except Chapter 5 which contains the result of joint work with Michael Paterson.

# ABSTRACT

The existence and complexity of decision procedures for families of deterministic pushdown automata are investigated, with special emphasis on positive decidability results for those questions, such as equivalence, which are known to become undecidable when the deterministic restriction is removed.

The equivalence problem is proved decidable for the following three deterministic families, all of which are already extensive enough to have undecidable inclusion problems:

(a) nonsingular automata - a realtime subfamily, which extends the largest corresponding classes with previously known equivalence tests,

(b) finite-turn automata - characterised by having a bound on the number of times the direction of the stack movement can change, and

(c) one-counter automata - defined by restricting the stack alphabet to just one symbol.

The problem of whether a language defined by a machine in one family, can be recognised by one in another, is a convenient formulation of numerous decidable or potentially decidable questions. We show that such questions as whether a deterministic context-free language can be recognised by a machine in any one of the above named classes, must be, if decidable at all, at least as difficult to decide as whether such a language is regular. We re-examine the regularity test of Stearns, and obtain an improved algorithm. We

do this by reducing by an exponential order the upper bound on the possible state complexity of regular sets recognised by deterministic pushdown automata of a given size, to a level close to one known to be achievable.

We pursue an application of this analysis to a schema theoretic problem. We consider the succinctness with which certain functional schemas can be used to express equivalent large flowchart schemas, and obtain closely matching upper and lower bounds for a measure of this.

C O N T E N T S

# INTRODUCTION

To understand the meaning of an executable computer program we have to relate its finite specification to its possibly infinite distinct behaviours when applied to different inputs. Finding general ways of doing this is a primary goal in the theoretical study of computations. The specific aims of such investigations are to gain insight into the computational behaviour of whole classes of programs, and hence to be able to analyse instances of various particular problems for any program within these classes. The recursive decidability of the problems concerned is an important criterion of the practical viability of such analysis: the existence of an effective decision procedure capable of determining the truth of any instance of a particular question for a given class of programs, means not only that this analysis can be automated, but usually also that an a priori bound can be placed on its difficulty. In contrast, if the problem is undecidable for the class, that is, there is no effective procedure for solving it in general, then new instances of it can always be found for which the solution is more difficult than before and requires further creative effort.

The computations we shall study are those that can be carried out by abstract deterministic automata that have only a pushdown stack and a finite-state control for storage. This is a formalisation of the stack concept which is widely used in practical programming, for example in writing syntax analysers, or in implementing recursion in compilers. Consequently our results will relate to areas outside automata theory, such as the syntax of programming languages, and certain formal models of recursive programs.

The observation primarily motivating our work is that, while these families of automata are wide enough to be relevant to practical computations, many of the important questions about their behaviour can be shown to be decidable. This is a very rare conjunction of attributes for computational models so far studied. It is generally found that for abstract models designed to describe useful classes of programs, nearly all the key properties can be shown to be undecidable. For example, if we extend any of the families we shall consider by introducing nondeterminism, or by allowing a second stack, then we already lose the decidability properties we desire most.

The actual machines we investigate are deterministic pushdown automata (dpda), which have been widely studied before[1,2,3], and certain restrictions of them. These are all acceptors in the sense that the input consists of a sequence of symbols on a tape, and the output of either a 'yes' or a 'no' depending on the configurations of the machine after the input tape has been scanned. Thus each machine defines a set of strings of symbols, or a language, namely that consisting of exactly the strings that are accepted, i.e. lead to a 'yes'. These languages are all context-free, and have the additional useful property that they can all be parsed easily in linear time.

We note that this input-output behaviour is not quite as restrictive as it would appear. Problems to do with related but superficially more general machines (e.g. transducers which output a symbol for each symbol read ) can often be reduced to corresponding questions about acceptors and languages.

The decision problem with which we are most concerned for these families of automata is that of equivalence, i.e. is there an effective test to determine whether two machines perform equivalent computations? The existence of such a test has the practical significance that it provides a most convenient model-independent mechanism for verifying correctness. Thus if we have an automaton which is optimised in some way, and therefore perhaps complicated, and we want to prove that it will perform exactly as it is intended, then we could build a second automaton which is more perspicuous in structure but possibly otherwise unsuitable, and test the two for equivalence. Other criteria of correctness would, in contrast, necessitate that a new distinct language for describing the intended computation be introduced and related to the automata formalism.

The inclusion problem (i.e. to decide whether one language is a subset of another ) is related to the equivalence problem in that any procedure for deciding the former would lead directly to one for the latter. Thus, as distinct from the existing analysis of such classes as the regular sets[3], bounded languages[4], and parenthesis languages[5], for all of which both inclusion and equivalence are decidable, positive solutions to the equivalence problem for families with an undecidable inclusion problem assume new significance.

Our main results for the equivalence problem are to show for three distinct families of automata that they are in just this category. The only comparable family we know of is that of two-tape acceptors[6,7]. For the class of LL(k) languages[8] (which include the simple[9] languages), although equivalence has been shown to be decidable, the inclusion problem is currently open. We will show that both two-tape acceptors and LL(k) languages correspond closely

to particular restrictions of two of our families, and hence that the decidability of the equivalence problem for both of them now also follows as corollaries to our results.

In addition to proving decidability results we are also concerned with giving bounds for the complexity of the derived decision procedures. A bound on the time complexity of a problem (i.e. on the time required by the best possible decision procedure in the worst case ) gives an indication not only of how much time may be needed for an automatic analysis, but also of the possible difficulty in an informal sense, of resolving particular instances of it. The complexities are expressed in terms of functions which have the parameters of the machine description as arguments. Thus if we have a polynomial time bound on a decision procedure for one problem, but know that an exponential time is necessary for deciding another, then this will indicate that for sufficiently large machines, it will be easier to solve a problem of the first kind than one for the second.

Valid measures of absolute complexity for decision procedures are numerous, and may depend additionally on the machine model on which one intends to execute them. For this reason we are usually content with answers specifying only the number of levels of exponentiation, if any, involved. Thus typically we may say that a procedure takes polynomial, or perhaps exponential time, without giving further details. This kind of classification is known to be robust enough not to depend on which one of the customary machine models is chosen.[10]

More satisfying results, which are now totally machine

independent, can be given, however, for the relative complexities of the various decision problems. Thus we show that, in a specified sense, certain decision problems must be equally difficult, and that some others must be at least as difficult as these. Such results, besides confirming our experience with known decision procedures, also throw light on several specific currently open problems, which we show to be at least as difficult to decide as a question for which the best procedure we have been able to find works in double exponential time.

Although the decidability and complexity results themselves yield considerable information, the techniques used in the proofs and the properties on which they depend are equally important in giving insight into the structure of these computations. Our presentation will therefore attempt to highlight informally the ideas we believe are new and most important. We shall omit details of arguments which are well known and occur elsewhere in the literature. Where, for the sake of completeness, we mention results not directly related to our main theme, we shall just sketch the main ideas from which a proof can be reconstructed.

We suggest the following as appropriate both for preliminary reading and for providing motivation to our work from diverse angles: A general exposition of the concepts of automata and decidability is given in Hopcroft and Ullman[3]. Basic results about dpda are also to be found there, as well as in Ginsburg and Greibach[1]. Various grammatical characterisations of families of deterministic languages and their relevance to parsing are described by Knuth[11], Korenjak and Hopcroft[9], Rosenkrantz and Stearns[8], and Harrison and Havel[12]. An introduction to program schemas, and a description of the relation-

ship between certain recursive schemas and pushdown automata can be found in Paterson[13].

# I PRELIMINARIES

This first section introduces the definitions, basic notions, and preliminary results which we shall use in subsequent chapters. From the start we take a more complexity conscious approach to the subject than is available elsewhere in the literature. Consequently the preliminary ideas we have to introduce, although mostly well known, have had to be reformulated so as to give the more precise constructions which we now require. In particular we have to bypass those of the widely used standard constructions that lead to exponential explosions.

We first define the class of deterministic pushdown automata. Our criterion of acceptance differs from the most popular one[3], but is the more natural for our purposes. The definition is very broad in the sense that for any dpda specified in one of the other standard ways, there is always an equivalent one in our formulation that is not much more complex. Further, by placing simple restrictions on this class, we can also define several important subclasses which we shall later study.

The size or complexity of a dpda we describe by means of several of the parameters of its description, which we go on to specify. We then investigate the tradeoffs that can be realised when a machine is transformed to another equivalent one in order to economise on one of these parameters at the expense of the others.

In the second chapter we give results about the complexity of problems already known to be decidable. We justify the claim that

these problems are 'easy' by verifying that simple decision procedures exist for solving them, which work in time depending only polynomially on the parameters of the tested machines. In the course of this we define a normal form for dpda, and illustrate its use by deriving some of the decidability results as trivial consequences of its existence.

Throughout we judge the relative power of the various subfamilies of machines by comparing the classes of languages to which they correspond. For simplicity, therefore, when this is not otherwise confusing, we often identify a family of machines with the class of languages it defines.

# Chapter 1 DEFINITIONS

## 1.1 Deterministic Pushdown Automata

A pushdown automaton (pda) is an abstract device whose memory consists of a pushdown stack and a finite state control. It can read a string of characters from a finite alphabet from its unique input tape, which can progress past the input head of the machine in only one direction, and only once. At any step of the computation the transitions which the machine can undergo are determined by the state of the finite state control, the contents of the top of the stack, and the character under the input head. An input word is either "accepted" or "rejected", depending on the "configurations" the machine can reach after having read its last character. Each such automaton defines a language i.e. the set of words it accepts. The languages which are defined by the class of pushdown automata are exactly the context-free languages of Chomsky[14,3].

We are here interested in the deterministic restriction of this class, namely the case where each combination of state, top stack symbol, and input character defines a unique machine transition. The class of such deterministic pushdown automata (dpda) we call D.

More formally, for a machine M $\epsilon$ D, let $\Sigma$ be its finite input alphabet {a, b, c, ...}, $\Gamma$ its finite stack alphabet {A, B, ...} and Q its finite state set $\{s_1, s_2, \ldots\}$. We denote strings of input characters by $\alpha$, $\beta$, etc., and strings of stack symbols by $\omega_1$, $\omega_2$ etc. We define $\epsilon$ and $\Lambda$ to be the null elements of $\Sigma^*$ and $\Gamma^*$

respectively. In addition we use the special symbol $\Omega$ to denote an empty stack.

A <u>configuration</u> $c = (s, \omega)$ is an element of $Q \times (\{\Omega\} \cup \Gamma^+)$ and describes the state and stack content (starting from the empty stack end) of the machine at some instant. The <u>mode</u> of a configuration c is an element from $Q \times (\{\Omega\} \cup \Gamma)$ and describes the state and top stack symbol of c. Thus the mode of $(s, \omega A)$ is $(s, A)$, and that of $(s, \Omega)$ is $(s, \Omega)$. The machine has a set $F \subset Q \times (\{\Omega\} \cup \Gamma)$ of distinguished <u>accepting modes</u>.

The set of <u>transition rules</u> $\Delta$ is a set of rules of the form

$$(s, A) \xrightarrow{\pi} (s', \omega')$$

where $\pi \in \Sigma \cup \{\varepsilon\}$, with the additional restriction that each mode is

<u>either</u> (i) an $\varepsilon$-mode i.e. occurs on the left in just one rule, and there with an $\xrightarrow{\varepsilon}$ transition.

<u>or</u> (ii) a <u>reading mode</u> i.e. has at most one $\xrightarrow{a}$ rule for each $a \in \Sigma$, and no $\xrightarrow{\varepsilon}$ rule.

The machine M makes the <u>move</u> $(s, \omega A) \xrightarrow{\pi} (s', \omega\omega')$ from the one configuration to the other if and only if one of the rules is $(s, A) \xrightarrow{\pi} (s', \omega')$, where $\pi \in \Sigma \cup \{\varepsilon\}$. If $\pi \in \Sigma$ then $\pi$, the input character currently under the input head on the tape, is considered to have been read, and the head moves to the next position on the tape. Otherwise the head does not move.

A <u>derivation</u> $c \longrightarrow c'$ is a sequence of moves starting from configuration c and ending with $c'$. It is an <u>$\alpha$-derivation</u>, $c \xrightarrow{\alpha} c'$,

if the execution of it is accompanied by the reading of the word $\alpha$ on the input tape.

There is a distinguished <u>starting configuration</u> $c_s$, which for simplicity we assume to belong to $Q \times (\{\Omega\} \cup \Gamma)$. We say that a configuration $c'$ is <u>reachable from c</u> iff there is some derivation $c \longrightarrow c'$. When we say that $c'$ is reachable we mean that it is reachable from $c_s$.

An input string $\alpha$ is <u>accepted</u> by M iff $c_s \xrightarrow{\alpha} c$ for some c whose mode belongs to F, and <u>rejected</u> iff it is not accepted. Two configurations c, $c'$ are <u>distinguished by $\alpha$</u> if $\alpha$-derivations can take one to an accepting mode but not the other. The <u>rank</u> of two configurations c and $c'$ (denoted by rank (c, $c'$),) is defined as the length of a shortest string distinguishing the two, if one exists, and $\infty$ otherwise. Two configurations are said to be <u>equivalent</u>, i.e. $c \equiv c'$, iff rank (c, $c'$) = $\infty$. Two machines M, $M'$ are <u>equivalent</u> iff their starting configurations are equivalent.

We denote the set of strings which can take the machine from a configuration c to accepting modes, by L(c). Then clearly $c \equiv c'$ iff $L(c) = L(c')$. For a machine M we denote $L(c_s)$ by L(M), and for a class X of such machines, we will abbreviate $\{L(M) \mid M \in X\}$ to L(X).

Thus, to summarise, each dpda (as well as the language it defines) is completely specified by a sextuple $M(\Sigma, \Gamma, Q, F, \Delta, c_s)$. The following illustrates the main points:

Example 1.1

For machine M let $\Sigma = \{a, b, c\}$, $\Gamma = \{A_0, A, B\}$, $Q = \{s_1, s_2, s_3\}$, $F = Q \times \{\Omega\}$, $c_s = (s_1, A_0)$ and $\Delta$ the set of transitions:

$(s_1, A_0) \xrightarrow{a} (s_1, A)$ $\qquad$ $(s_1, A) \xrightarrow{a} (s_1, AA)$

$(s_1, A) \xrightarrow{b} (s_1, B)$ $\qquad$ $(s_1, B) \xrightarrow{b} (s_1, BB)$

$(s_1, B) \xrightarrow{a} (s_2, \Lambda)$ $\qquad$ $(s_1, B) \xrightarrow{c} (s_3, \Lambda)$

$(s_2, B) \xrightarrow{\varepsilon} (s_2, \Lambda)$ $\qquad$ $(s_3, B) \xrightarrow{c} (s_3, \Lambda)$

$(s_2, A) \xrightarrow{a} (s_2, \Lambda)$ $\qquad$ $(s_3, A) \xrightarrow{\varepsilon} (s_3, \Lambda)$

Then it can be easily verified that

$$L(M) = \{a^n b^m a^n \mid n, m \geq 1\} \cup \{a^n b^m c^m \mid n, m \geq 1\}.$$

For our notation we follow standard conventions for sets. For a string x we denote the reversal of x by $x^R$, the length of x by $|x|$, and the concatenation of n copies of x by $x^n$. For Z a set of strings, $Z^*$ and $Z^+$ will denote its transitive closure under concatenation, with and without the null string respectively, and $\emptyset$ will denote the empty set. The symbol \$ will always stand for an input character which, by convention, does not belong to the $\Sigma$ being discussed.

## 1.2 Elementary Properties

We first note that the more customary definition of acceptance is by states rather than modes[1,3]. It can be however easily verified (Hopcroft and Ullman[3], Lemma 12.3) that the classes of languages defined by the two methods are identical. Further, as we shall now

indicate, all the key properties on which theorems particular to dpda depend, are a consequence of determinism, and are independent of the manner of acceptance chosen.

One effect of the deterministic restriction is to ensure that once a sequence of $\varepsilon$-moves has started, the subsequent computation is uniquely determined by the original configuration, and can depend on the input tape again only when the $\varepsilon$-sequence has terminated. Thus if the latter never terminates, then it is clearly redundant and can be eliminated. If it is finite and gives rise to the derivation $(s, \omega A) \xrightarrow{\varepsilon} (s', \omega\omega' B)$ where $(s', B)$ is a reading mode, then such $\varepsilon$-derivations can be eliminated by replacing the $\varepsilon$-rule for $(s, A)$ by rules of the form $(s, A) \xrightarrow{a} (s'', \omega'\omega'')$ for each rule $(s', B) \xrightarrow{a} (s'', \omega'')$. Thus we conclude that $\varepsilon$-moves can only be essential in a dpda for derivations which cause the stack to decrease in height.

Similarly, but for other reasons, having a multiplicity of states cannot be essential in derivations where the stack is increasing. State information can always be coded into the top stack symbol and thus transmitted indefinitely, as long as no top symbol is ever removed. Thus in Example 1.1, while the distinctness of states $s_2$ and $s_3$ is essential, the information which $s_1$ carries, i.e. that the stack is in the increasing phase, could be coded into the stack symbols. We could therefore modify M to obtain another equivalent one with state set just $\{s_2, s_3\}$, but with an enlarged stack alphabet $\{A_0, A, A', B, B'\}$, if we replaced $c_s$ by $(s_2, A_0)$, and replaced the transition rules involving $s_1$ by the following:

$$(s_2, A_0) \xrightarrow{a} (s_2, A') \qquad (s_2, A') \xrightarrow{a} (s_2, AA')$$

$(s_2, A') \xrightarrow{b} (s_2, B')$ $\qquad$ $(s_2, B') \xrightarrow{b} (s_2, BB')$

$(s_2, B') \xrightarrow{a} (s_2, \Lambda)$ $\qquad$ $(s_2, B') \xrightarrow{c} (s_3, \Lambda)$

Another consequence of determinism that is useful for showing that certain languages cannot be recognised by any machine in a given class, is the following observation, of which a formal version is stated and proved by Ginsburg and Greibach[1]:

Observation 1.1

The effect of input strings periodic in a given word $\alpha$ (i.e. of the form $\alpha^n$ for $n \geq 1$) on a dpda is

either (a) to cause the stack to never grow larger than some constant height,

or (b) to create stacks which are periodic in some stack word everywhere except for parts of bounded length at the top and bottom.

Using this we can formally verify that $\varepsilon$-moves, multiple states, and a multiple stack alphabet, are all essential features of any dpda recognising the language of Example 1.1. Formal arguments of this kind are given by Ginsburg and Greibach[1], and by Harrison and Havel[12].

We note that the normal form given by Greibach for context-free languages shows that in the nondeterministic case, $\varepsilon$-moves and multiple state sets are inessential[15].

## 1.3 Subfamilies

We can define some important subfamilies of D by specifying simple syntactic restrictions on the machine descriptions:

One-counter machines (C) are those dpda with just one stack symbol. In these the stacks can only be used to count, and hence a configuration is best described as (s, n) where $s \in Q$, $n \geq 0$. Such counters are closely related to the registers of Minsky[16].

Realtime machines (R) are those with no $\varepsilon$-modes. They recognise strings in real time, as opposed to merely linear time.

Stateless machines (S) are those with just one state. This state can therefore be omitted from the description of the configurations or transition rules. A consequence of the restriction is that no information can be transmitted during stack decreasing derivations.

That the languages defined by each of these three classes is properly contained in L(D) follows from the previously made observation that the language defined in Example 1.1 cannot be recognised by any machine in C, R or S.

For all such classes we can have the additional restrictions that

(a) $F \subset Q \times \{\Omega\}$

or that (b) no rule is defined for modes in $Q \times \{\Omega\}$.

For the classes D, C, R and S, we respectively define the sub-

classes $D_0$, $C_0$, $R_0$ and $S_0$ by imposing both restrictions (a) and (b), and $D_1$, $C_1$, $R_1$ and $S_1$ by imposing just the first of these. That each of the implied containments is proper can be easily verified by observing that if X is any one of D, C, R or S then $\Sigma^* \in L(X_1)$ but $\Sigma^* \notin L(X_0)$, and $\{a^n b^m \mid m \leq n\} \in L(X)$ but $\notin L(X_1)$. Although $X_1$ thus properly extends $X_0$, it does not use the stack in any more general way, and consequently many of its properties can be deduced directly from those of the corresponding $X_0$. For simplicity, therefore, we shall only study the classes X and $X_0$.

We note that classes equivalent to $D_0$ and $R_0$ have been studied by Harrison and Havel[12] (who call them strict, and realtime strict deterministic respectively), and the class $S_0$ by Korenjak and Hopcroft[9]. To show that $S_0$ is just the 'simple' machines of the latter, we have to demonstrate that $\varepsilon$-moves are redundant in it. This is obvious since stack symbols with the property that $A \xrightarrow{\varepsilon} \Lambda$ can clearly be eliminated from the rules, while $\varepsilon$-moves which cause increases in stack height can be removed in the manner indicated in §1.2.

## 1.4 Some Family Properties

There are some further restrictions on families of machines and languages that are related to the definitions above.

A language $L \subset \Sigma^*$ is said to have the prefix property iff $\beta \in \Sigma^+$ and $\alpha \in L \Rightarrow \alpha\beta \notin L$. From the definition of $D_0$, $C_0$, $R_0$ and $S_0$ it is immediate that these all have this property. For $L(D_0)$ we can

say, further, that it contains all the languages of L(D) that have this property. For consider an M ε D s.t. L(M) has the prefix property, and modify it to an equivalent M' ε $D_0$ as follows: Introduce a special Ω-simulating symbol at the bottom of the stack to ensure that no accepting derivation has an intermediate empty stack configuration. Then for each accepting mode of M introduce an ε-move to a new state from which any stack will be emptied by further ε-moves. The modified machine clearly has the required properties[12].

The quasirealtime[12] property is a relaxation of the realtime restriction. ε-moves are now allowed to occur, but only a bounded number of times consecutively. If this bound is say n, then only up to n stack characters can be removed in a single ε-derivation. Thus, by changing the stack alphabet so that each stack is now specified by symbols encoding blocks of n + 1 old symbols (with a block of possibly fewer symbols at the top of the stack), and changing the transition rules so that these blocks are manipulated correctly, the old machine can be simulated by a new one with the same states, but with no ε-modes. A more formal proof of a similar statement is given by Harrison and Havel[12]. Thus we can conclude the following:

Observation 1.2

The power of the quasirealtime machines is no larger than that of R.

Finally we mention a syntactic restriction on the transition rules. A dpda is conservative iff in any rule (s, A) ⟶ (s', ω) where $|\omega| > 1$, it is the case that the first symbol of ω is A.

Lemma 1.3 will show directly that this is not a proper restriction on the power of D, C or of the quasirealtime machines. However S and $S_0$ are properly restricted to the symmetric languages, Sy and $Sy_0$ respectively. An immediate property of these symmetric machines is that they cannot transmit information even while the stack is increasing. Thus the language $\{a^n\$a^n \mid n \geq 1\}$ cannot be recognised by such a machine since the latter could not distinguish between being in the upward phase from being in the downward phase of the computation. Consequently $Sy_0 \not\subseteq S_0$ and $Sy \not\subseteq S$.

## 1.5 Derivations

We now introduce some terminology and notation for describing the geometrical movements of the stack, that are in addition useful concepts in talking about derivations.

The height $|c|$ of a configuration c is the length of its stack i.e. $|c| = |\omega|$ if $c = (s, \omega)$ for some s.

The derivation $c \xrightarrow{\alpha} c'$ is a stacking derivation iff $|c| < |c'|$ and every intermediate configuration in the derivation has height greater than $|c|$. It is then written as $c \uparrow(\alpha)\ c'$. The derivation is a popping derivation iff $|c| > |c'|$ and every intermediate configuration in the derivation has height greater than $|c'|$. It is then written as $c \downarrow(\alpha)\ c'$. (N.B. The notation $\uparrow$, $\downarrow$ is used by Stearns[2] but with the converse meanings.) It is a positive derivation, written as $c \xrightarrow[+]{\alpha} c'$, iff no intermediate configuration has empty stack.

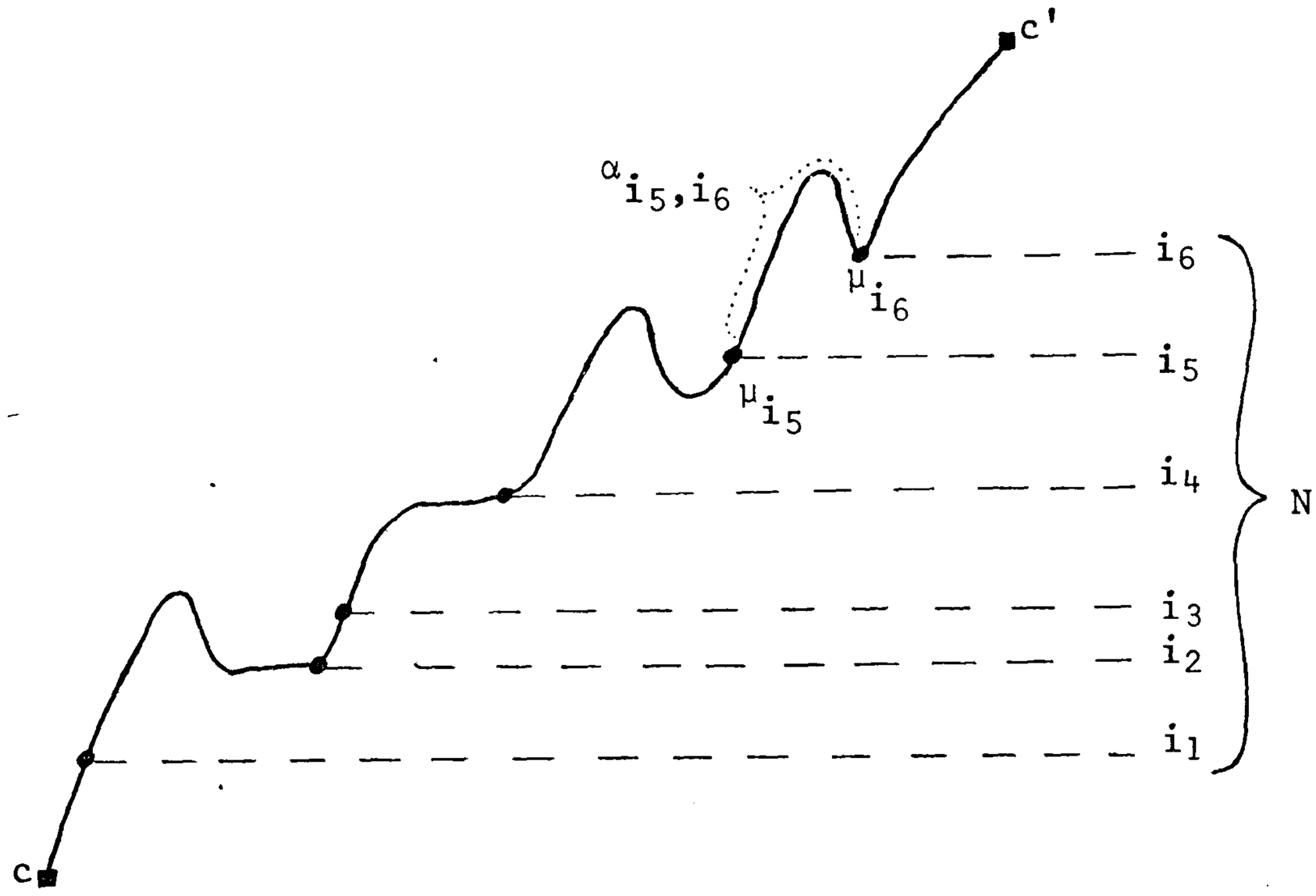
We define the stacking sequence of a stacking derivation as a sequence of modes $\mu_i$. Each $\mu_i$ is the mode of the unique configuration of height $(i + 1)$ in the derivation that is not followed by any configuration of height $\leq (i + 1)$ subsequently in the derivation. Note that if $(s, \omega) \uparrow(\alpha) (s', \omega')$ then $\mu_i$ is not defined unless $|\omega| \leq i+1 \leq |\omega'|$, and may not be defined for all intermediate values (e.g. if the derivation consists of a single move corresponding to a transition rule of the form $(s, A) \longrightarrow (s', \omega_1)$ where $|\omega_1| > 2$.) The significance of $\mu_i$ for the configuration corresponding to it, is that it contains all the information about the previous computation that may influence the subsequent part of the derivation.

We define analogously the popping sequence for a popping derivation to be a sequence of states $\sigma_i$. Each $\sigma_i$ is the state of the unique configuration of height i in the derivation that is not preceded by any configuration with height $\leq$ i.

For $\omega \in \Gamma^*$, $j > i \geq 0$ we define $\omega_{i,j}$ to be the substring of $\omega$ starting from the $(i + 1)$th character and ending with the jth. An index set N is an ordered subset of the positive integers. Thus any N induces a natural segmentation of $\omega$ into words $\omega_{i,j}$, where $i < j$ and $i, j \in N$.

The stacking and popping sequences w.r.t. index set N (Fig. 1) are just the subsequences of these corresponding to the elements of N. We define the segmentation of $\alpha$ in the stacking derivation $c \uparrow(\alpha) c'$ w.r.t. an index set $N = \{i_1, \ldots i_n\}$, as the unique sequence of input words $\alpha_{i_1, i_2}, \ldots, \alpha_{i_{n-1}, i_n}$ with the properties that there exist $\alpha_{i_0, i_1}, \alpha_{i_n, i_{n+1}}$ where $i_0 + 1 = |c|$, $i_{n+1}+1 = |c'|$, s.t.

Trace of c $\uparrow(\alpha)$ c'




Trace of c $\downarrow(\alpha)$ c'

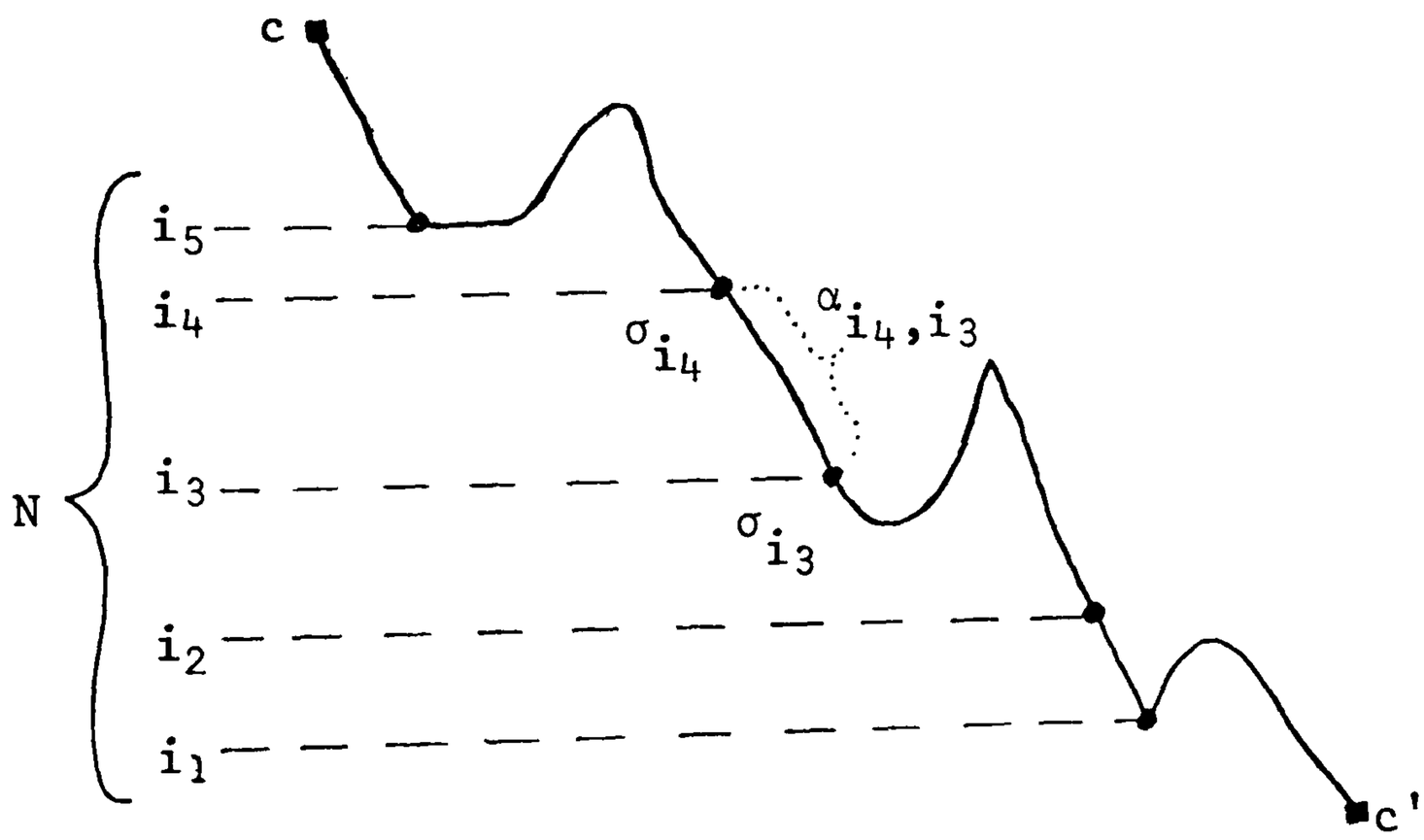



Fig. 1

$$\alpha = \alpha_{i_0,i_1}\alpha_{i_1,i_2} \cdots \alpha_{i_n,i_{n+1}},$$

and that if $c'$ has stack $\omega$, then for the stacking sequence $\{(s_r, A_r)\}$ w.r.t. $\{i_0, \ldots i_{n+1}\}$,

$$(s_r, A_r) \uparrow(\alpha_{i_r,i_{r+1}}) (s_{r+1}, \omega_{i_r,i_{r+1}}A_{r+1})$$

for $1 \leq r \leq n$. In other words $\alpha_{i_r,i_{r+1}}$ is just that part of the input string which takes the derivation from the last configuration of height $i_r+1$ to the last one of height $i_{r+1}+1$ provided that neither is followed by configurations smaller than themselves. We can similarly define $\alpha_{i,j}$ for $i < j$ and $i, j \in N$.

Analogously the segmentation of a popping derivation w.r.t. N can also be defined. $\alpha_{i,j}$ is then the part of the input string that takes the derivation from the first configuration of height i, to the first configuration of height j, where now $i > j$.

## 1.6 Complexity Measures

We shall describe the size of a dpda in terms of the following parameters:

t = size of stack alphabet

q = size of state set

p = size of input alphabet

h = length of longest stack word occurring in the transition rules

u = number of transition rules.

We shall express the complexity measures relating to these machines in terms of functions with these parameters as arguments e.g. X(t, q). We shall be primarily concerned with the asymptotic values, as the arguments become large, of positive valued functions monotonic in each parameter. We introduce the following notation to describe such behaviour:

Definition

The function $X(x_1,\ldots, x_n)$ is of order $E^n$ iff n is the largest integer such that

$$\overline{\lim_{y\to\infty}} \frac{\log^{(n+1)}X(y,\ldots, y)}{\log y} > 0.$$

Then, for example, $x^2$, $\sqrt{x}$, $x^{\log x}$ will be $E^0$, $x_1 2^{\sqrt{x_2}}$ and $x!$ will be $E^1$, etc. For convenience we shall refer to an expression X, even if it is of several variables, as polynomial if it is bounded above by some multinomial.

When we are interested in the leading exponent in an expression, we use the following more detailed notation:

Definition

The function $X(x_1, \ldots x_m)$ is of order $E^n(Y(x_1, \ldots x_m))$ iff

$\exists$ k, k' > 0 s.t. for all sufficiently large $x_1, \ldots x_m$,

$$Exp^n(kY) < X < Exp^n(k'Y).$$

[Here $Exp^0(x) = x$, $Exp^{r+1}(x) = 2^{Exp^r(x)}$ for $r \geq 0$, and similarly for log, which we shall always take to the base 2.]

## 1.7 Tradeoffs in Description

The dpda definition allows equivalent machines to have very dissimilar specifications. For example any language recognised by a finite state automaton with n states can clearly be recognised by some dpda with q = n and t = 1, and also by some with q = 1 and t = n. We investigate an aspect of this flexibility of description by asking to what extent an arbitrary dpda can be modified to obtain an equivalent one, so as to economise on one of the parameters at the possible expense of the others.

We define $LD(n_1, n_2, n_3)$ as the class of languages specified by dpda with $q \leq n_1$, $t \leq n_2$ and $h \leq n_3$. Then the questions we want to ask can be naturally phrased as inclusion problems for these classes. We note, however, that only the first of the following lemmas depends on determinism.

Lemma 1.1

$$LD(q, t, h) \not\subseteq LD(q-1, n_2, n_3) \text{ for any } n_2, n_3.$$

Harrison and Havel[12] show that the set

$$\{a^m b^k a^m b^k \mid 1 \leq k \leq n,\ 1 \leq m\}$$

can be recognised by a dpda with n states, but not by one with fewer. □

Lemma 1.2

$LD(q, t, h) \subset LD(q, t^h, 2)$ for $t > 1$.

Proof

We give a simple construction for converting an arbitrary dpda to an equivalent one which simulates it closely, but does not require long right hand sides in the rules, or extra $\varepsilon$-moves. We leave the state set unchanged but enlarge the stack alphabet to consist of all strings in $\Gamma^+$ of length less than h, each typically denoted by $[AB..C]$, in the following way:

Replace each rule $(s, A) \xrightarrow{\pi} (s', B_1 \ldots B_n)$, where $\pi \in \Sigma \cup \{\varepsilon\}$ and $n \geq 2$, by the set of rules

$$(s, [\omega A]) \xrightarrow{\pi} (s', [\omega B_1][B_2 \ldots B_n]),$$

one for each $\omega \in \Gamma^*$ s.t. $|\omega| < h - 1$, and make the analogous replacements for the rules with $n < 2$.

Then clearly the new stack alphabet will be of size $t + t^2 + \ldots + t^{h-1} < t^h$, and if acceptance is defined in the obvious way, then the new machine will be equivalent to the old one. □

Note 1

The new stack symbols that are not redundant in the modified machine are exactly those of the form $[\omega A]$ where $B\omega\omega'$ appears in some rule of the old machine for some B and $\omega'$. Consequently the new stack alphabet is essentially of size at most thu, and so really depends only polynomially on the original machine description. From

this it follows that the number of transition rules, and hence all of the description, need only be increased polynomially by this construction.

Note 2

The construction preserves the conservative property. Thus even symmetric languages have a normal form with h = 2.

Lemma 1.3

$$LD(q, t, h) \subset LD_{cons}(qt^h, t, 2) \text{ for } t > 1$$

and $$LD(q, 1, h) \subset LD(qh, 1, 2).$$

Proof

At the expense of introducing new $\varepsilon$-moves and increasing q we can reduce h to 2 and make the machine conservative.

The new state set consists of

$$Q \cup \{[s, \omega] \mid s \in Q, \omega \in \Gamma^+, |\omega| < h\}$$

Each transition rule $(s, A) \xrightarrow{\pi} (s', B_1 \ldots B_n)$, with $n \geq 2$, we replace by the set:

$$(s, A) \xrightarrow{\pi} ([s', B_2 \ldots B_n], B_1)$$

$$([s', B_i \ldots B_n], B_{i-1}) \xrightarrow{\varepsilon} ([s', B_{i+1} \ldots B_n], B_{i-1}B_i) \text{ for } 1 < i < n$$

$$([s', B_n], B_{n-1}) \xrightarrow{\varepsilon} (s', B_{n-1}B_n)$$

This construction merely ensures that instead of a large stack word being added in one move, it is built up in a succession of specially

created ε-moves. □

Note

The non-redundant subset of the new state set depends only polynomially on the original parameters and is bounded by uh. The machine description as a whole is again increased only polynomially.

Lemma 1.4

$$LD(q, t, h) \subset LD(qt, 2, h.\lceil \log t \rceil).$$

Proof

We code each stack symbol into binary words of fixed length $\lceil \log t \rceil$. For each old state of the machine, the new one has to be able to interpret the top coded word in the stack. To do this it traces down, via ε-moves, a path of a binary tree of depth $\lceil \log t \rceil$ while popping this top word. Thus the new machine needs a tree with about t nodes in its state diagram for each original state. Thus the state set is now of size qt, and h has increased to $h.\lceil \log t \rceil$. □

We conclude by relating the other parameters p and u to q, t and h. By the deterministic condition on the transition rules, u is clearly bounded above by qtp. On the other hand the input alphabet may, in theory, be arbitrarily large. However, the number of characters which produce distinct behaviour from the automaton is bounded by the other parameters in the following way: Each input character can occur in at most qt rules, each of which has one of no more than $qt^{h+1}$ different possible right hand sides. Thus the

machine can only distinguish up to $(qt^{h+1})^{qt}$ different input characters, and if there are more, then at least some of them can be identified with each other without influencing the computations.

## 1.8 Family Relationships

We conclude by summarizing in fig. 2 the relative power of the classes of automata with which we shall be most concerned. We use the following relations for arbitrary classes X and Y:

(i) $X \longrightarrow Y$ only if $\forall\, M \in Y,\ \exists\, M' \in X$ s.t. $L(M) = L(M')$

(ii) $X \dashrightarrow Y$ only if $\forall\, M \in Y,\ \exists\, M' \in X$ s.t. $L(M)\$ = L(M')$

where \$ is a distinguished symbol.

For the definitions of $N_0$, T and dB - S, see chapters 3, 4 and 8 respectively. All the relationships in the diagram are immediate from the definitions or well known, with the exception of the position of $N_0$, which we prove in Chapter 3, that of 2-tape acceptors, explained in Chapter 4, and that of S, for which we omit the proof.

Family Relationships

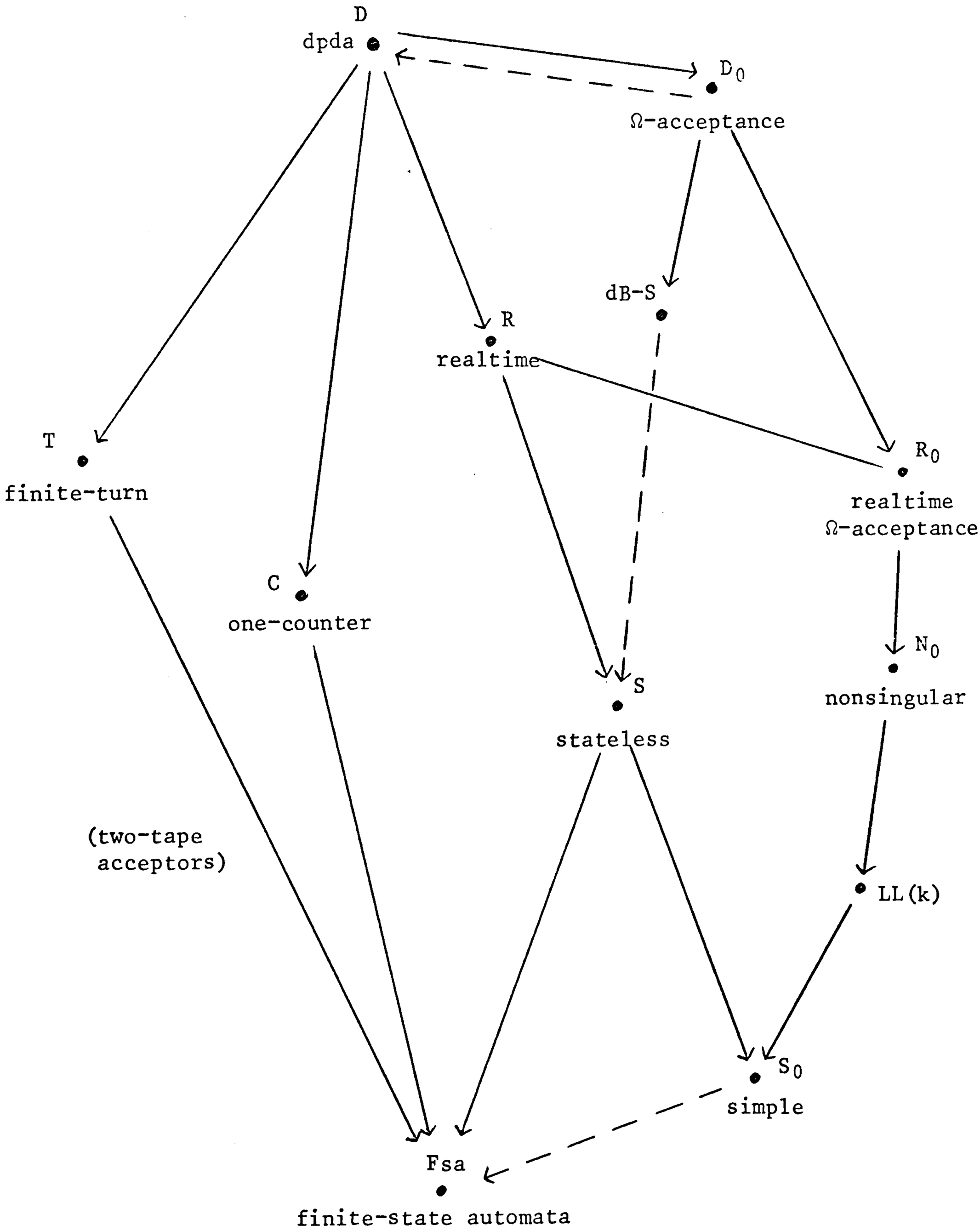



Fig. 2

# Chapter 2 EASILY DECIDABLE PROBLEMS

## 2.1 Introduction

Here we shall discuss some properties of deterministic languages which have been previously known to be decidable. We give decision procedures for these, and show them to be easier than the procedures for equivalence and regularity that we derive in later chapters, in the sense that they all require only polynomial time.

These "easily decidable" problems for $L \in L(D)$ include

(i) membership - does $\alpha \in L$?

(ii) emptiness - is there some $\alpha$ s.t. $\alpha \in L$?

(iii) finiteness - is L finite or infinite?

(iv) totality - is there some $\alpha$ s.t. $\alpha \notin L$?

(v) equivalence with regular set $L'$ - is $L = L'$?

(vi) has L the prefix property?

The first three of these are decidable for context-free languages in general but the last three rely on the deterministic restriction. Techniques for proving such decidability results are given by Bar-Hillel, Perles and Shamir[17] and by Ginsburg and Greibach[1]. Here we shall concentrate on the complexity of just some of them.

We know of no arguments giving interesting lower bounds for the time complexity of any of these decision problems, and, for this

reason, we shall not linger to give exact upper bounds either. However, for each problem there is often an alternative natural measure of complexity. For some of these we can give upper and lower bounds which differ by only a modest amount. Examples of such particular measures of complexity include, for the emptiness problem, the maximum length of shortest strings accepted by non-empty machines of a given size, and, for the regularity problem, the maximum size of the smallest finite-state automata equivalent to some dpda of a given size.

## 2.2 Emptiness

We shall consider from two viewpoints the complexity of this problem, which plays a crucial role in subsequent decision procedures.

The proof of decidability of emptiness given by Bar-Hillel, Perles, and Shamir[17] for context-free languages depends on the derivation tree[3] of the grammar for a shortest accepted string. If the grammar has x non-terminals and right hand sides never longer than h, then in any derivation, starting with a non-terminal, of a shortest string, no non-terminal need repeat along a path in the tree, which therefore can be taken to have depth $\leq$ x. We now define the depth of each non-terminal to be the depth of the shallowest derivation tree of which it is a root. Then, knowing which non-terminals have depth < i, we can find all those that have depth < i + 1 simply by looking for productions in which all the symbols on the right hand side have depth < i. Thus by going through the list of productions at most x times, and establishing the depth of

all the non-terminals, we can determine whether the starting symbol has a defined depth i.e. whether the language is empty.

Thus it follows that

(a) there is a decision procedure for emptiness which works in time depending only polynomially on the grammar description, and

(b) the shortest accepted string can be no longer than $h^x$.

The standard construction[3] for a pda with q states and t stack symbols gives a grammar with $q^2t$ non-terminals. Further if (and only if) h is taken to be a constant, the number of productions in the grammar will depend only polynomially on the parameters. However, from the notes to Lemmas 1.2 and 1.3 we already know that we can modify any pda to an equivalent one with h = 2, with only polynomial growth. Thus we can conclude that:

Lemma 2.1

Emptiness for pushdown automata can be tested in polynomial time. □

Since the standard construction of the grammar from the machine does not change h, it also follows that:

Lemma 2.2

If M has parameters $h > 1$, q and t, and L(M) is not empty, then $\exists\, \alpha \in L(M)$ s.t. $|\alpha| \leq h^{q^2t}$. □

We can show that for q = 1 the order of this bound is achievable.

Example 2.1

Let $M \varepsilon S_0$ be the simple machine defined by

$$A_i \xrightarrow{a} A_{i+1}^{h} \quad \text{for } 1 \leq i < t$$

$$A_t \xrightarrow{a} \Lambda$$

where the starting symbol is $A_1$ and acceptance is by empty stack.

Let $\min(A_i)$ be the length of the unique string generated from $A_i$.

Then from the productions it is clear that

$$\min(A_i) = 1 + h.\min(A_{i+1}) \text{ for } 1 \leq i < t.$$

Since $\min(A_t) = 1$, it follows that $\min(A_i) = \sum_{r=0}^{t-i} h^r$. Thus M accepts just one word, and this is of length of order $E^1(t.\log h)$. □

For q > 1 we can find a class of machines which achieves a bound of order $E^1(qt.\log h)$.

Example 2.2

Let $M \varepsilon R_0$ have states $\{s_1, \ldots s_q\}$, stack symbols $\{A_1, \ldots A_{t-1}, B\}$, and transition rules:

$$(s_i, A_j) \xrightarrow{a} (s_i, A_{j+1}^{h}) \qquad 1 \leq i \leq q,\ 1 \leq j < t-1$$

$$(s_i, A_{t-1}) \xrightarrow{a} (s_{i+1}, BA_1) \quad 1 \leq i < q$$

$$(s_q, A_{t-1}) \xrightarrow{a} (s_q, \Lambda)$$

$$(s_i, B) \xrightarrow{a} (s_{i-1}, \Lambda) \qquad 1 < i \leq q$$

Let the starting configuration be $(s_1, A_1)$ and the unique accepting mode be $(s_1, \Omega)$.

This is a generalisation of the previous example. Each occurrence of $A_1$ proliferates to produce an exponential number of $A_{t-1}$'s, but now instead of these being removed by a single input character, they are each allowed to proliferate themselves all over again for a different state. For any configuration the depth reached in this sequence of renewals is given both by the state and the number of occurrences of a B in the stack. By a second induction process, similar to that given in the previous example, it follows that the unique string accepted by M is of length of order $E^1(qt.\log h)$. □

We leave unresolved the gap between this lower bound and the previously derived upper bound of $E^1(q^2t.\log h)$.

## 2.3 Finiteness

Bar-Hillel, Perles, and Shamir[17] show that a context-free language is infinite if and only if there is some derivation tree which has a path along which some non-terminal repeats and in so doing generates distinct terminal strings. Thus to test for finiteness we produce a polynomial size grammar, eliminate redundant non-terminals (i.e. those that do not generate terminal words, or cannot be derived from the starting symbol), and then test for the proper embedding property. This last test can be done by looking at each of the remaining productions that have some terminal on the right hand

side, and searching through the other productions to see whether a self-embedding cycle of productions can be found for this given non-terminal. As all this can be done in polynomial time it follows that:

Lemma 2.3

Finiteness for pushdown automata can be tested in polynomial time. □

## 2.4 A Normal Form

There is a well known[3] normal form for dpda, which is very useful for our purposes and fits in naturally with our definitions.

Definition

A dpda is loop-free[1] iff every input word can be read in a finite number of machine moves.

This condition excludes the possibilities of either (i) moves from some reachable configuration being undefined, or (ii) there occurring an infinite sequence of consecutive ε-moves, thus preventing further inputs from being read.

Lemma 2.4

Every dpda M has an equivalent loop-free M' with the same stack alphabet, and just one extra state.

Proof

We first augment M by introducing a new state $s'$ and new transition rules $(s', A) \xrightarrow{a} (s', A)$ for each $a \in \Sigma$ and $A \in \Gamma$, without changing F.

For each reading mode we add the rules $(s, A) \xrightarrow{a} (s', A)$ whenever $(s, A) \xrightarrow{a}$ does not previously occur. Thus instead of stopping, the machine will now continue to read the tape and reject it.

To exclude the other possibility mentioned above, we first have to note that in any infinite $\varepsilon$-derivation there must be some configuration which has a smallest stack. If the mode of this is $(s, A)$, then by replacing the transition rule for this by the set $(s, A) \xrightarrow{a} (s', A)$ for each a, we again ensure that the rest of the tape is read and rejected. These modes can be effectively found by appropriate emptiness tests. □

All the new rules introduced serve the same role and therefore do not create additional complexity in any significant sense. Consequently we shall from now on assume that the loop-free machine is of essentially the same size, in every way, as the original one.

Definition

A dpda is in normal form iff it is loop-free, and every accepting mode is a reading mode.

Lemma 2.5

For each M ε D with q states and t stack symbols, there is an equivalent M' in normal form with the same stack alphabet and 2q + 1 states.

Proof

We first make M loop-free by introducing a distinguished state s' as in the previous lemma. For each other state $s_i$ ε Q we introduce an $s_i''$. The role of $s_i''$ is the same as that of $s_i$, except that the former indicates in a derivation that an accepting mode of M has been passed since the last time an input was read.

Thus for each accepting ε-mode of M, we replace

$(s_i, A) \xrightarrow{\varepsilon} (s_j, \omega)$ by $(s_i, A) \xrightarrow{\varepsilon} (s_j'', \omega)$.

For each reading rule $(s_i, A) \xrightarrow{a} (s_j, \omega)$ we add the extra rule $(s_i'', A) \xrightarrow{a} (s_j, \omega)$, and for each ε-rule $(s_i, A) \xrightarrow{\varepsilon} (s_j, \omega)$ we add the extra rule $(s_i'', A) \xrightarrow{\varepsilon} (s_j'', \omega)$.

This new machine clearly simulates M and accepts the same strings if we define the new set of accepting modes by

$$F' = \{(s_i'', A) \mid (s_i, A) \text{ is a reading mode}\}$$

$$\cup \{(s_i, A) \mid (s_i, A) \in F \text{ and } (s_i, A) \text{ is a reading mode}\}.$$

□

## 2.5 Totality

As corollaries to the lemma above, decidability proofs for properties which are known to be undecidable in the nondeterministic case[3], can now be easily derived.

Corollary 2.1

For each $M \in D, \exists M' \in D$ with about twice as many states, which accepts exactly the complement of L(D).

Proof

We change M to $M''$ in normal form. Then we redefine the accepting modes to be just those reading modes which are not accepting modes in $M''$. □

Ccrollary 2.2

Totality for dpda can be tested in polynomiaī time.

Proof

Construct the complement machine and test for emptiness. □

Corollary 2.3

For $M \in D$, and $M'$ a finite state automaton, the following can

be tested in time depending polynomially on the parameters of M and M': $L(M) = L(M')$, $L(M) \cap L(M') = \emptyset$, $L(M) \subset L(M')$, $L(M') \subset L(M)$.

Proof

If M' has state set Q' of size q', and M state set Q of size q, then both machines can be simulated together by a dpda which resembles M, except that it has state set $Q \times Q'$ and is able to mimic M' in the finite state control at the same time as simulating M. Provided that M is in normal form, each of the above questions can be tested by defining acceptance suitably in the new machine, and testing for emptiness. □

# II THE EQUIVALENCE PROBLEM

We have already observed that an equivalence test for a class of automata can provide a method of verifying the correctness of a given machine by comparing it with another of the same kind. In addition, we can attribute other useful properties to classes for which equivalence is decidable, that give us increased confidence in the possibility of systematically analysing and handling particular machines contained in them.

We first notice that for all automata with a decidable membership problem, inequivalence is partially decidable. For we can enumerate all strings over the input alphabet and test whether the two machines behave differently in each one. Then if the machines are inequivalent, we are sure to recognise this when we reach the first offending input string.

We then observe that all the automata in a syntactically defined class, such as D, can be enumerated in some lexicographic manner. It follows that if we can decide equivalence, then for any automaton in such a class we can enumerate all other equivalent ones. Conversely, if we can enumerate all machines equivalent to any particular one then, since inequivalence is partially decidable, we can decide equivalence by running this enumeration simultaneously with the partial decision procedure.

Thus we arrive at the following intuitive interpretation of the significance of this problem: equivalence is decidable if and only if for any automaton in the class, we can enumerate all other

ways of doing equivalent computations. Undecidability then means that the relationship between the automaton descriptions and their computational behaviour has a distinctly higher order of complexity, and we can no longer hope to be able to describe adequately all the different ways in which the freedom available within the class can be used to perform a given task.

The partial decidability of inequivalence has also the following immediate consequence for all the classes we shall study:

Observation

If equivalence is partially decidable, then it is decidable.

The decision procedure consists of simply running the partial decision procedures, for equivalence and inequivalence concurrently.

Thus, except when we can derive meaningful results about the complexity of an equivalence test, we shall, for the sake of simplicity, be content with proving decidability by showing partial decidability.

This section consists of three chapters in each of which a distinct subfamily of D is shown to have a decidable equivalence problem. There is an essential unity in the methods used for all three, for which the basic inspiration comes from Rosenkrantz and Stearns[8]. They show that for dpda recognising the LL(k) languages, there is an easily derived canonical form with the property that any two equivalent machines must, after reading the same input word,

have stack heights differing by less than a certain constant. Two such machines can therefore be simulated by a single dpda with a two-track tape, and equivalence verified by a suitable emptiness test. What we shall show is that even for more general classes, where no such close relationship between the stack movements need occur, suitable pushdown automata can be devised to simulate equivalent machines.

# Chapter 3 A REALTIME FAMILY

## 3.1 Introduction

The syntax of programming languages is customarily defined in part as a general context-free grammar. Most frequently, however, the languages generated by such grammars are not only deterministic, but also expressible as $R_0$ languages, i.e. those recognised by realtime deterministic pda accepting only by empty stack, and even subclasses of these. Such subclasses as the simple languages[9], and the LL(k) languages[8] which generalise them, have been studied with exactly this motivation, and equivalence tests have been found for these particular ones.

The principal restriction on LL(k) recognisers is that they require essentially only one state, and therefore cannot transmit information during stack decreasing derivations. Thus, for example, in Algol 60, after some expression '((..(( Expression ))..))' with arbitrarily nested matching brackets is scanned, no such machine can remember whether the contents of the innermost brackets was an arithmetic or a boolean expression. Since 'if ((..(( Expression ))..))' has distinct valid sequels depending on just this condition, we conclude that no essentially one state machine can recognise arithmetic expressions as specified in that language.

Here we shall define a family $N_0$ of quasirealtime dpda such that $L(N_0)$ properly contains the LL(k) languages. $N_0$ machines may have an arbitrary number of states, and can easily accommodate the above

mentioned example. In addition, however, they extend the LL(k) languages in a much more significant way, thus indicating that our generalisation of the equivalence test is an important one. The procedure for LL(k) languages given by Rosenkrantz and Stearns[8] relies on the fact that min(c), the length of the shortest string accepted from a configuration c, is a very well behaved function of the stack contents. The value of this function for any configuration can be deduced to within an additive constant from the contents of the stack alone, and, further, can only change a bounded amount from one configuration to the next in a derivation. This property does not hold for some languages in $L(N_0)$, such as $\{a^n bc^n\} \cup \{a^n de^{2n}\}$. The concept of 'thickness' of stack words, central to the proof of Rosenkrantz and Stearns, is no longer applicable here.

While the inclusion problem for simple and LL(k) languages is at present open, we can already prove this to be undecidable for $N_0$. In fact we shall show that $L_1 \subset L_2$ is undecidable where $L_1$ is a simple language and $L_2 \in L(N_0)$ has just two states.

We conjecture that the property on which the validity of the equivalence test for $N_0$ depends holds also for $R_0$ in general, and believe that a proof of this would be illuminating.

The emphasis in this chapter is on the decision procedure itself, rather than the class $N_0$, which we use merely to demonstrate the increased generality of our technique over previous ones.

## 3.2 Nonsingular Automata

We define the class $N_0$ of nonsingular machines as follows.

**Definition**

$M \in D_0$ is nonsingular iff $\exists\ m \geq 0$ s.t.

$\forall\ \omega, \omega' \in \Gamma^*, s, s' \in Q$ where $|\omega| > m$,

$L(s, \omega'\omega) = L(s', \omega') \Rightarrow L(s', \omega') = \phi$.

**Theorem 3.1**

(i) $LL \not\subseteq L(N_0)$ where LL is the class of all LL(k) languages.

(ii) $L(N_0) \not\subseteq L(R_0)$.

**Proof**

(i) For a configuration c let min(c) be the length of a shortest string in L(c). Then from Lemma 8 of Rosenkrantz and Stearns[8], we can deduce for certain canonical LL(k) recognisers, when translated into our terminology, that there exist positive constants $\ell$, $\ell'$, s.t. for all s, s', $\omega$, and $\omega'$,

(a) $\quad |\ \min(s, \omega'\omega) - \min(s', \omega') - \min(s, \omega)\ | \leq \ell$, and

(b) $\quad \ell'.\ |\omega| > \min(s, \omega) \geq |\omega|$,

provided that min is defined throughout.

It follows directly that any dpda with these properties

belongs to $N_0$, with nonsingularity constant m equal to $\ell$. We note that for the simple[9] machines this constant can always be taken to be zero.

To show that containment is proper, consider the following dpda M recognising the language $\{a^n bc^n\} \cup \{a^n de^{2n}\}$ for $n \geq 1$.

$\Delta$: $(s_1, A) \xrightarrow{a} (s_1, AA)$

$(s_1, A) \xrightarrow{b} (s_2, \Lambda)$, $(s_2, A) \xrightarrow{c} (s_2, \Lambda)$

$(s_1, A) \xrightarrow{d} (s_3, \Lambda)$, $(s_3, A) \xrightarrow{e} (s_4, A)$, $(s_4, A) \xrightarrow{e} (s_3, \Lambda)$

$c_s = (s_1, A)$, $F = Q \times \{\Omega\}$.

In M for any i, j and m, n where $m \neq n$,

$L(s_i, A^n) \neq L(s_j, A^m)$.

Thus $M \in N_0$. However, it is implicit in conditions (a) and (b) above that the value of min can only change by up to a bounded amount from one configuration to the next in a derivation. For any dpda accepting the above language, this cannot be true for the set of moves in which a symbol d is read. We therefore conclude that $L(M) \notin LL$.

(ii) If a machine is nonsingular, then the appropriate constant m clearly sets an upper bound on the amount the height of a configuration can change in the course of any sequence of $\varepsilon$-moves, for any two configurations in such a derivation must necessarily be equivalent. Thus, from an argument given in §1.4, it follows that the same language can be accepted by a realtime machine.

To show that the converse is not true, consider the language $\{a^n bc^n\} \cup \{a^n dc^{2n}\}$, which clearly belongs to $L(R_0)$. Let a dpda M recognising this language reach the equivalent configurations $(s, \omega)$ and $(s', \omega')$ after reading $a^{2n}b$ and $a^n d$ respectively for some large n. Then from Observation 1.1 (§1.2) we know that $|\omega|$ is about twice $|\omega'|$, and further that the effect of $c^k$ for some "small" k is to reduce $(s', \omega')$ to a configuration whose stack is periodic right to the top. Thus for an infinite set of distinct values of n, some string $c^k$ takes the two equivalent configurations to new ones with stacks still differing in height by about n, but with one stack being a prefix of the other. Thus $M \notin N_0$. □

## 3.3 Alternate Stacking

We now describe a way of constructing a single stack machine, $M'$, for simulating two realtime dpda M, $\overline{M}$ together.

A configuration of $M'$ can be specified by a pair of states, one from each of M and $\overline{M}$, and a stack which consists of alternate segments of words from the stack alphabets (assumed distinct) of the two machines. The simulated configurations of M and $\overline{M}$ can be recovered by taking the corresponding state component, and concatenating the appropriate set of alternate segments of the stack of $M'$, to obtain the original stack. On reading each input symbol, $M'$ simulates M and $\overline{M}$ by simultaneously manipulating the two topmost segments in its stack according to the respective transition rules of the component machines.

More formally, distinguishing the notation for M and $\bar{M}$ by over-lining everything concerned with the latter, so that M' has stack alphabet $\Gamma \cup \bar{\Gamma}$ and state set $Q \times \bar{Q}$, we describe a typical configuration of M' by

$$([s, \bar{s}], \omega_1\bar{\omega}_2\omega_3\bar{\omega}_4 \ldots \bar{\omega}_{2n-2}\omega_{2n-1}\bar{\omega}_{2n}),$$

where only the last segment, $\omega_{2n}$, may be empty.

Without loss of generality we shall assume that the two topmost non-null segments of the stack are $\omega_{2n-1} = \omega A$ and $\bar{\omega}_{2n} = \bar{\omega}\bar{A}$. If the transition rules of M, $\bar{M}$ specify that

$$(s, A) \xrightarrow{a} (s_1, \pi\omega') \text{ and } (\bar{s}, \bar{A}) \xrightarrow{a} (\bar{s}_1, \bar{\pi}\bar{\omega}'),$$

(where $\pi \in \Gamma$ if $\omega'$ is non-null, and $\pi \in \Gamma \cup \{\Lambda\}$ otherwise, and similarly for $\bar{\pi}$) then for input a in M', the segment $\omega A\bar{\omega}\bar{A}$ will change to $\omega\pi\bar{\omega}\bar{\pi}\omega'\bar{\omega}'$, and the state from $[s, \bar{s}]$ to $[s_1, \bar{s}_1]$.

An important implication of this notation is that if, for example, $\bar{\omega}\bar{\pi} = \Lambda$, then the words $\omega\pi$ and $\omega'$ will merge into one segment since they are from the same alphabet.

## 3.4 Main Results

The alternate stacking construction is only useful to us if we can ensure that the simulating machine is itself a pushdown automaton.

Definition

Alternate stacking for a pair of machines, for a given set

of input strings is said to succeed iff the simulation of them together for these strings by the above construction produces stack segments of only bounded size.

We define an input string $\alpha$ to be live for a dpda M iff it is the prefix of some accepted string, and a configuration c to be live iff $L(c) \neq \phi$.

Theorem 3.2

For M, $\bar{M}$ equivalent nonsingular dpda, alternate stacking succeeds for all live inputs.

Proof

We shall show that in the simulating machine M' constructed as described in §3.3, no input $\alpha$ that is live in M, $\bar{M}$ can lead to a configuration

$$([s, \bar{s}], \omega_1\bar{\omega}_2 \ldots \omega\bar{\omega})$$

with $|\bar{\omega}| > z(2z + m)$. Here z is the maximum of the lengths of the shortest strings accepted by dpda of size no larger than M and $\bar{M}$, and m is the larger of the nonsingularity constants of these two machines, which are taken to be realtime.

Let us assume the contrary. Consider the configurations c and $\bar{c}$ of M and $\bar{M}$ respectively at the moment when the bottom symbol of the segment $\bar{\omega}$ was actually placed there. Let $\alpha$ be the input read since that time, which has taken M, $\bar{M}$ to the present live configura-

tions $c_1$, $\bar{c}_1$ respectively. By our choice of $\bar{c}$ and $\alpha$, all the configurations in the $\alpha$-derivation in $\bar{M}$ have height $\geq |\bar{c}|$. Also, by virtue of the alternate stacking construction, all the configurations in the $\alpha$-derivation in M have height $\geq |c_1|$, for what is finally the $\bar{\omega}$ segment in M' has been in existence throughout the $\alpha$ derivation, and hence the segment below it cannot have been increased in the meantime (Fig. 3).

Let $\beta$ be a shortest string in $L(\bar{c}_1)$, and let $\beta = \beta_1\beta_3$ where $\bar{c}_1 \downarrow(\beta_1)\ \bar{c}_2$ and $|\bar{c}_2| = |\bar{c}|$. Then $|\beta_1| \geq |\bar{\omega}|$ by the realtime property. Also $\beta$ must be a shortest string in $L(c_1)$. But then $\beta$ is the concatenation of segments induced by the popping sequence in M, each one taking some $(s_1,A)$ to some $(s_2, \Lambda)$. If $\beta$ is minimal, each such segment must also be minimal and of length no more than z. Thus $\beta_1$ consists of a sequence of such minimal segments terminated possibly by a proper prefix of another such segment. Let $\beta_2$, a prefix of $\beta_3$, be the completion of this last segment. Then let $c_2 \xrightarrow{\beta_2} c_3$ and $\bar{c}_2 \xrightarrow{\beta_2} \bar{c}_3$. Clearly

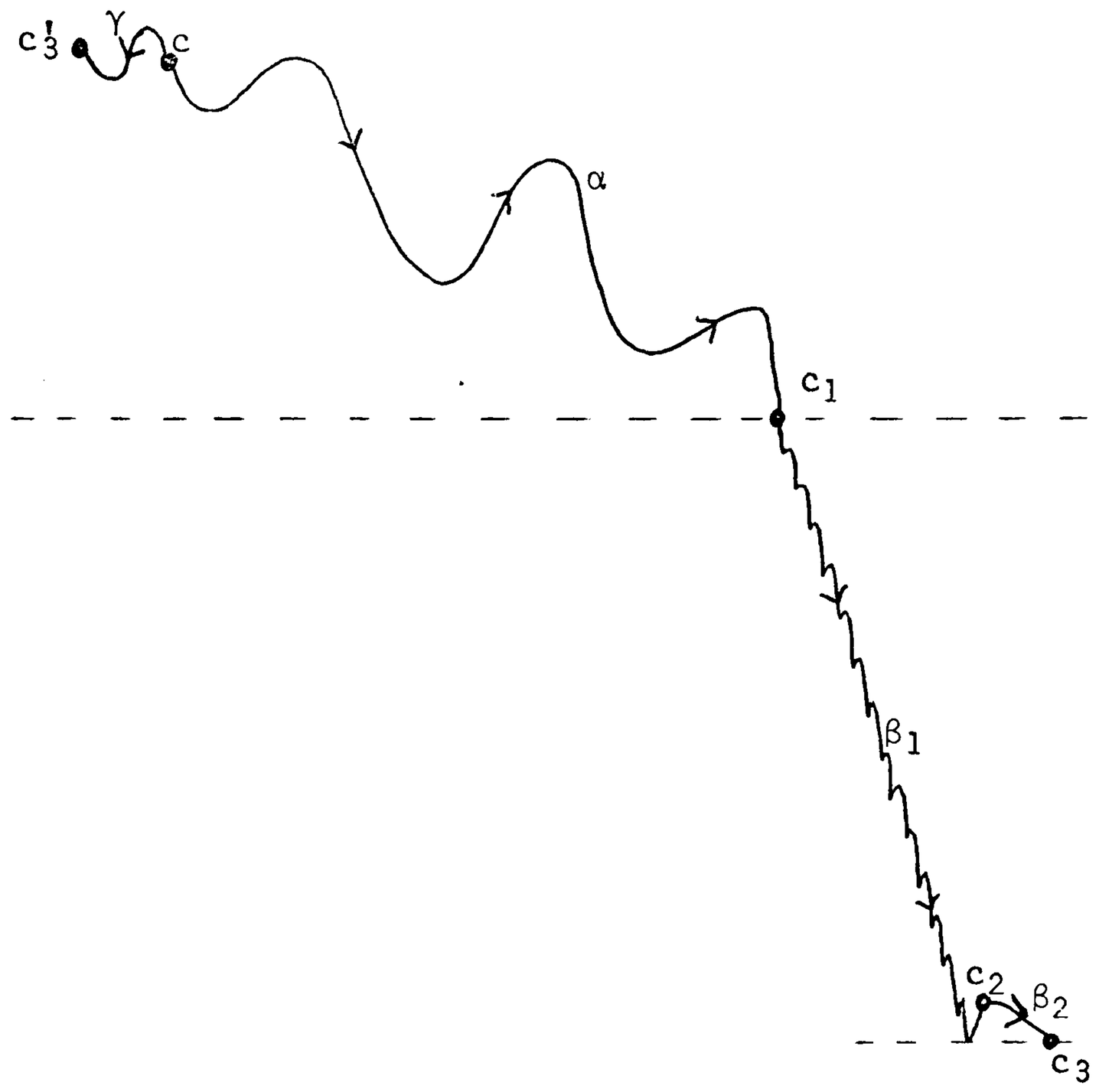
$$|c_1| - |c_3| \geq |\beta_1|/z. \qquad \text{(a)}$$

Now consider a shortest string $\gamma$ taking $\bar{M}$ from $\bar{c}$ to $\bar{c}_3$. Then the definitions of $\bar{c}_2$, $\bar{c}_3$ ensure that $|\gamma| \leq z + |\beta_2| \leq 2z$. Let $c \xrightarrow{\gamma} c'_3$. Then

$$|c| - |c'_3| \leq 2z. \qquad \text{(b)}$$

But if M and $\bar{M}$ are equivalent then by construction $c_3 \equiv c'_3$. But from (a), (b) and the observations that $|c| \geq |c_1|$ and $|\beta_1| \geq |\bar{\omega}|$ we conclude that

M:




$\overline{M}$:

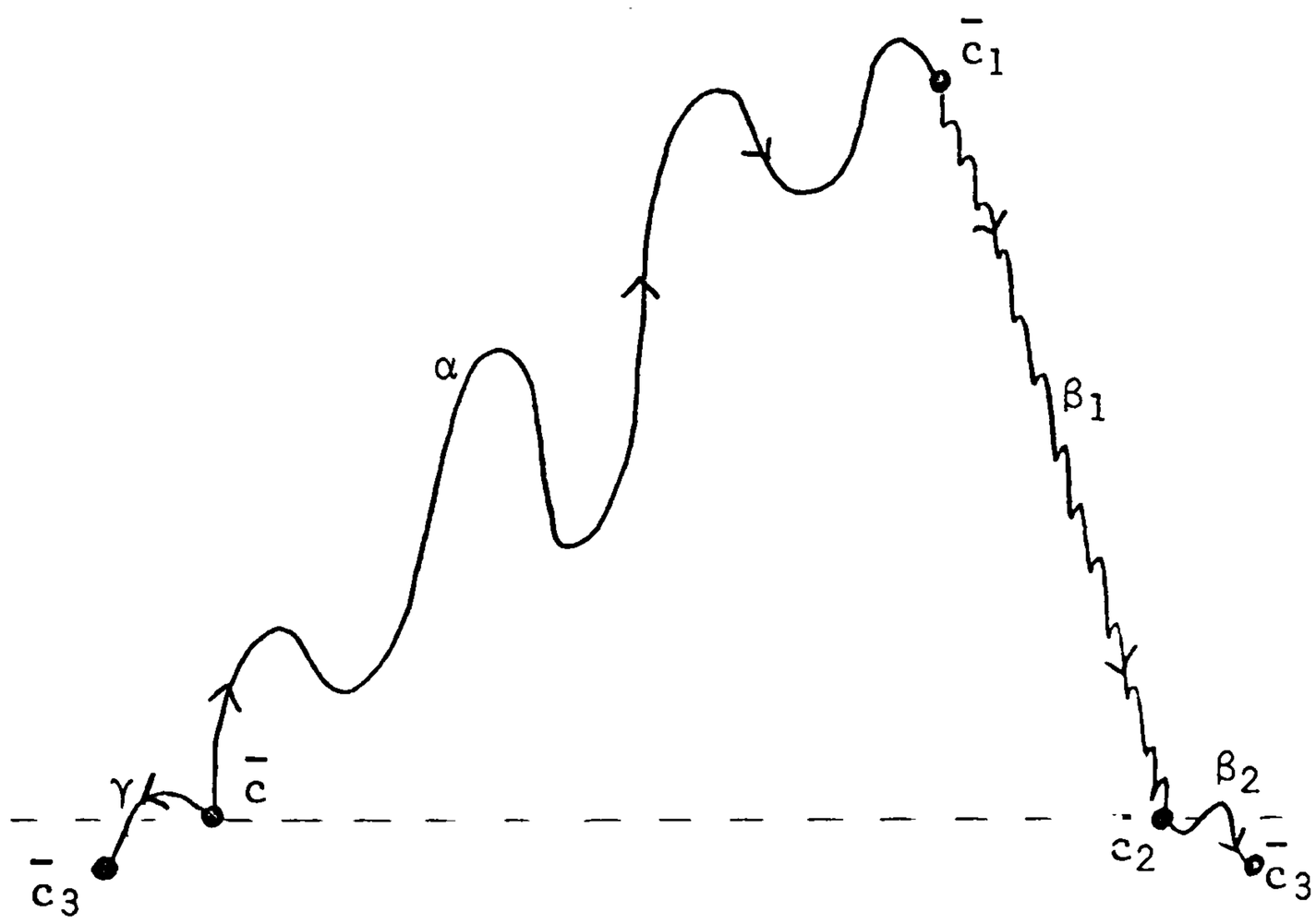



Fig. 3. Derivations in Proof of Theorem 3.2 .

$$|c_3| \leq |c'_3| + 2z - |\bar{\omega}|/z.$$

Thus if $|\bar{\omega}|/z - 2z > m$ then we have a contradiction to the non-singularity condition.

We therefore have to conclude that for equivalent nonsingular automata, alternate stacking succeeds for all live inputs. □

Theorem 3.3

If alternate stacking succeeds for all live inputs for all pairs of machines $M \equiv \overline{M}$ in some class $X \subset D$, then the equivalence problem is decidable for X.

Proof

Assume, as is permitted by Lemma 2.5, that M, $\overline{M}$ are in normal form. First suppose that we know the bound to which stack segments can grow in the alternate stacking simulating machine for M and $\overline{M}$. Then we can effectively construct a pda M'' with the property that L(M'') is empty iff $M \equiv \overline{M}$. M'' mimics the alternate stacking machine M' for M, $\overline{M}$ by encoding the top segment of M' in its finite state control. As long as this top segment never gets larger than the given bound, M'' accepts iff exactly one of the configurations it is simulating is in an accept mode. When the bound is exceeded, M'' proceeds nondeterministically to mimic one of M or $\overline{M}$, and accepts as the appropriate machine would.

By assumption , if $M \equiv \overline{M}$ then alternate stacking will succeed for live inputs, and the stack segment bound will only be exceeded

once nothing more can be accepted by M and $\overline{M}$. Thus $M''$ will be empty by construction.

Conversely, if $M''$ is empty, then clearly no input can produce different behaviour in M and $\overline{M}$, which are therefore equivalent.

Thus, if we have an a priori bound, we can test equivalence by constructing this pda and testing it for emptiness. However, even if we do not know this bound, by enumerating and testing for emptiness the possible candidate machines, we can obtain a partial decision procedure. That is, we construct pda of the form $M''$ for assumed segment bounds of 1, 2, ... successively. If M, $\overline{M}$ are not equivalent, then none of these constructed machines can be empty, while if they are, then one of them must be.

We therefore have partial decidability and hence, by an earlier observation, decidability. □

Corollary

The equivalence problem for nonsingular machines is decidable.

Proof

Immediate from Theorems 3.2 and 3.3. □

Note 1

Nondeterminism in the simulating machine is an inessential convenience, which can be avoided. M and $\overline{M}$ could have been preprocessed to recognise instantly whether a configuration reached is

live, and to enter a distinguished dead state if it is not. A simulating deterministic pda could then be easily constructed.

Note 2

We can show that the order of the bound on the segment growth derived in Theorem 3.2 is achievable for simple machines. For these we have already observed that the nonsingularity constant m is zero, and, in §2.2, that z is $E^1(t.\log h)$. Thus the bound $z(2z + m)$ is of order $E^1(t.\log h)$ also.

Let $A_1$ be the starting symbol for the simple grammar of Example 2.1 generating the singleton string $\{a^n\}$ where n is of order $h^t$. Then consider the simple languages generated by B and C respectively:

$$B \xrightarrow{a} B_1B_2 \qquad C \xrightarrow{a} A_1C_1C_2$$

$$B_2 \xrightarrow{a} B_1B_2 \qquad C_2 \xrightarrow{a} C_1C_2$$

$$B_2 \xrightarrow{b} A_1 \qquad C_2 \xrightarrow{b} \Lambda$$

$$B_1 \xrightarrow{a} \Lambda \qquad C_1 \xrightarrow{a} \Lambda$$

Both B and C generate the set $\{a^rba^{n+r} \mid r \geq 1\}$. However, if $\alpha = a^nba^n$, then $B \xrightarrow{\alpha} B_1^n$, but $C \xrightarrow{\alpha} A_1$. Thus in an alternate stacking machine for these two, the top segment can grow to size n, which is of order $E^1(t.\log h)$.

Note 3

To illustrate a slightly different way in which the success of alternate stacking can be used, we outline a proof that the equivalence problem for symmetric dpda, $Sy_0$ ( 1.4), can be solved in polynomial time.

We recall from the notes to Lemma 1.2 (§1.7) that any such machine can be transformed to one, only polynomially increased in size, with h = 2. Since alternate stacking clearly succeeds, and the emptiness test is itself polynomial, it remains only to prove that the simulating machine is itself of only polynomial size. The key observation here is that, instead of having to memorize the whole top segment, it is now sufficient for the finite state control to remember just the top symbol of the next-to-top segment, and to treat the top segment itself in the normal stack-like way. The property of $Sy_0$ machines from which this can be derived is that any move that causes the stack to increase, also causes the value of the function min( ) to increase.

We note, however, that for the class $S_0$, and therefore also $Sy_0$, algebraic properties can be derived[9] which give more detailed insight into these restricted classes than our more macroscopic approach. Thus for $Sy_0$ grammars with a fixed terminal alphabet and with h = 2, an equivalence test on the lines of Korenjak and Hopcroft[9] can be obtained, that works in time cubic in the number of non-terminals.

## 3.5 Undecidability of Inclusion

Friedman[18] proves the undecidability of the inclusion problem for dB - S schemas (chapter 8) by showing that for each instance of the Post Correspondence Problem[3], a pair of appropriately related dpda $M_1$, $M_2$ can be constructed with the property that $L(M_1) \subset L(M_2)$ iff it has no solution. The construction is valid for certain

classes (e.g. dB - S) which accept by empty stack but are essentially not realtime, and also for realtime families that do not have this empty stack restriction (e.g. S).

We use a similar formulation of the problem but, by taking a refinement of the correspondence problem that is implicit in the customary proof of its unsolvability, we can show that inclusion is undecidable even for a realtime class with empty stack acceptance. Because of the slight novelty involved, we shall digress here to give this undecidability proof.

Theorem 3.4

For $M_1 \in S_0$, $M_2 \in N_0$ it is undecidable whether $L(M_1) \subset L(M_2)$.

Proof

The instances of the correspondence problem that we take are just those obtained directly from Turing machine computations in the standard proofs of its undecidability. Thus Hopcroft and Ullman[3] show that for each Turing machine a correspondence problem with the following properties can be constructed:

Let $X = x_1, \ldots x_k$, $Y = y_1, \ldots y_k$ be the sequences of non-null words over a finite alphabet $\Sigma$. We define a sequence of positive integers $i_1, \ldots i_n$ (all $\leq k$) to be a solution (or partial solution respectively) for X, Y iff $x_1 x_{i_1} \ldots x_{i_n}$ is equal to (or a proper prefix of, respectively) the string $y_1 y_{i_1} \ldots y_{i_n}$. Then the construction is such that a solution exists for X, Y if and only if the Turing machine has a terminating computation, but no partial solution

for Y, X exists under any circumstances.

We construct two dpda $M_1$, $M_2$ which both reject strings not of the form $\alpha^R\beta\$$ where $\alpha$ is a string over the integers {1, ... k}, $\beta \in \Sigma^*$ and \$ is a distinguished terminating character. Both machines initially place a special symbol A at the bottom of the stack.

$M_1$ is a simple ($S_0$) machine which accepts exactly the strings $\alpha^R\beta\$$ where $\beta$ is the word indexed in Y by the sequence $\alpha$. Thus $M_1$ records $\alpha^R$ in its stack, matches it with $\beta$, and accepts iff the matching never fails and the \$ is read when there is only the symbol A left in the stack. It rejects all other strings by placing a special symbol on the stack which permanently freezes its motion once matching fails.

$M_2$ also tries to match $\alpha$ and $\beta$, but now as specified by X. If matching is successful and the \$ is read when the symbol A is reached at the bottom of the stack, then rejection occurs. Otherwise, if matching fails, $M_2$ goes into a second state, in which the effect of any $\Sigma$-input is to pop the stack without changing the state. When the A is reached in this second state, all input symbols leave the configuration unchanged, except for \$ which pops the A. If acceptance is defined by empty stack then clearly the length of a shortest string accepted from a configuration is just the height of the configuration. Thus it follows that $M_2 \in N_0$.

If there is a solution to the chosen instance of the correspondence problem then the input string which specifies it is, by definition, accepted by $M_1$ but not by $M_2$. Tnus $L(M_1) \subset L(M_2)$ implies that it has no solution. Conversely, if it has no solution,

and $\alpha^R\beta\$$ is any string accepted by $M_1$, then there can be at most a partial solution specified by an initial segment of $\alpha$. However, by our particular choice of X and Y, we have ensured that at the point where the $\alpha$, $\beta$ fail to match in $M_2$, the stack of $M_2$ has fewer symbols left than that of $M_1$. Since $M_2$ proceeds to accept any string from $\Sigma^*\$$ that is long enough to empty its stack, it must accept $\alpha^R\beta\$$ in particular, since $M_1$ will take at least as long to accept the remainder of this string as $M_2$ will take to reach the bottom of its stack. Thus if there is no solution then $L(M_1) \subset L(M_2)$.

Since for any Turing machine we can construct an instance of the correspondence problem, and hence also the machines $M_1$, $M_2$ with the above properties, it follows that if we could decide inclusion for such dpda, then we could decide whether Turing machines had terminating computations, which, however, we cannot. □

## 3.6 A Conjecture about $R_0$

We conjecture that Theorem 3.2 holds for $M, \overline{M} \in R_0$, and hence, by Theorem 3.3, that the equivalence problem is decidable for $R_0$. Here we shall outline the proof of a property of $R_0$ which, though not strong enough to prove the conjecture, may throw some light on it.

Lemma

There exists a function $F(q, t, h, \ell)$, asymptotically linear in $\ell$, with the property that for two machines $M, \overline{M} \in R_0$ with

parameters appropriately bounded above by q, t, and h, if c, $\bar{c}$ are equivalent configurations of M, $\bar{M}$ respectively, then

if $c \uparrow(\alpha)\ c_1$ where $c_1$ is live and $|c_1| - |c| > \ell$, then

$\exists \alpha'$ s.t. $c \uparrow(\alpha')\ c_1$ and $\bar{c} \xrightarrow{\alpha'} \bar{c}_1$ where $|\bar{c}_1| - |\bar{c}| > F(q, t, h, \ell)$.

Outline Proof

Choose the $\alpha'$ to give the shortest stacking derivation from c to $c_1$. If c', c" are any two configurations occurring in this derivation with $|c'| - |c"| > z$, (where z is as in Theorem 3.2), then it is easy to verify that $\min(c') > \min(c")$. Thus $\alpha'$ can be segmented into lengths of no more than $z^2$, such that each further segment in it takes c, and therefore also $\bar{c}$, to new configurations with increased values of min. The main argument then is to show, using this segmentation, that no subderivation of the $\alpha'$ derivation in $\bar{M}$ can produce a stack drop of more than G(q, t, h), where G is an $E^1$ function. This is done by assuming the opposite and inducing on the number of states that can be reached at this lowest level by popping derivations from the previous configurations in the subderivation. The observation used is that if $\min(s_1, \omega\omega') < \min(s_2, \omega\omega")$ for some $s_1$, $s_2$, $\omega$, $\omega'$, $\omega"$, such that $|\omega'|/|\omega"| > z$, then $\exists s$ s.t. (s, $\omega$) is not reachable from $(s_2, \omega\omega")$. Since the reading of the successive segments of $\alpha'$ must lead to a set of pairwise inequivalent configurations, the statement of the lemma then easily follows. □

# Chapter 4 FINITE-TURN AUTOMATA

## 4.1 Introduction

We consider a family of deterministic pushdown automata on which the only restriction is one on the movement of the stack. Using a technique related to the one given in the previous chapter, we show that the equivalence problem is decidable. Again we build a pda to try to simulate the tested machines, but now this is an inherently nondeterministic one. Since two equivalent machines in this family may have totally unrelated stack movements, a deterministic simulation by a pda is no longer possible.

A special case are the 1-turn machines. The undecidability of the inclusion problem for these is well known, and is also implicit in our proof of Theorem 3.4. A further restriction gives a class which is intimately related to the two-tape acceptors of Rabin and Scott[6], for which equivalence has already been proved decidable by Bird[7]. From our proof, therefore, another equivalence test can be abstracted that is not directly related to that of Bird and, while possibly less efficient, involves a technique of apparently more general applicability.

Although the finite-turn property is essential to the proof of the effectiveness of our main construction, we know of no pair of equivalent dpda for which a simulating machine of a broadly similar nature cannot be found. Thus it is possible that, when more detailed knowledge about the structure of dpda in general becomes

available, our methodology may be extended to prove decidability for the unrestricted case.

## 4.2 Definitions

We define the class T ⊂ D of finite-turn dpda as a deterministic analogue of the nondeterministic class studied by Ginsburg and Spanier[19].

Definition

A derivation in a dpda is a stroke if either no single move in it decreases the stack (i.e. an upstroke) or if no single move increases the stack (i.e. a downstroke).

Definition

A dpda M belongs to the class n - T, for some $n \geq 0$, iff every derivation in M from the starting configuration can be segmented into no more than n + 1 strokes alternating in direction.

Definition

A dpda $M \in T$ iff $M \in n - T$ for some $n \geq 0$.

In other words, the restriction we impose on T is that in the set of all derivations from the starting configuration of a machine, there is a bound on the number of times the direction of the stack movement can change.

Clearly the language $\{a^r b^r \mid r \geq 1\}^m$ belongs to $L([2m - 1] - T)$ but not to $L([2m - 2] - T)$. Further, in general, $L([2n] - T) = L([2n - 1] - T)$. This is because after an even number of turns the stack is increasing, and so after its last turn a $[2n] - T$ machine can only proceed to recognise a regular set. Thus the last turn in such a machine cannot be essential for recognising the language. In particular we note that $L(0 - T)$ are just the regular sets.

We give two properties of finite-turn automata which can be derived from analogous properties given by Ginsburg and Spanier[19]. They both depend on the fact that for any pda M, an equivalent one M' can be constructed to mimic M in every way and, in addition, to remember in its state set whether the derivation it has been doing is in an upstroke or a downstroke.

Observation 4.1

It is decidable for $M \in D$ whether $M \in T$.

Proof

For M we construct a nondeterministic pda M'' which mimics all the stack movements of M by $\varepsilon$-moves instead of reading the normal inputs. In addition M'' remembers in its finite-state control the direction of the current stroke, and reads a character \$ whenever this changes. Then $M \in T$ iff $L(M'')$ has strings of only bounded length (i.e. iff $L(M'')$ is finite). Thus to test M for the finite-turn property, we test $L(M'')$ for finiteness. □

Definition

A machine $M \in n - T$ is ordered iff

(i) its state set is the disjoint union of sets $Q_0, \ldots Q_n$, and

(ii) a state reached in a derivation from the starting configuration is in $Q_i$ iff the derivation has undergone exactly i turns.

Definition

A state s of an ordered machine is of order i iff $s \in Q_i$.

Observation 4.2

For any $M \in n - T$ we can construct an equivalent ordered $M' \in n - T$.

Proof

We first modify M so as to be able to remember the stroke direction, and then make n + 1 copies of the transition rules, each referring to a distinct state set $Q_i$, for $0 \leq i \leq n$. We obtain $M'$ by slightly modifying this new set of rules. $M'$ starts in the starting configuration corresponding to $Q_0$, and mimics M in every way except that whenever it is in a state $s \in Q_i$ and the current stroke is terminated by a move in the opposite direction, the appropriate transition occurs to a state of $Q_{i+1}$, instead of $Q_i$. The computation continues within the new copy of the transition rules, with acceptance defined as in M, until further directional changes occur. $M'$ is clearly deterministic and has the required properties. □

We notice that we can test easily whether $M \varepsilon n - T$ for a particular n, by building the candidate ordered $M' \varepsilon [n + 1] - T$ equivalent to it, and testing whether any derivation in $M'$ can reach a state of order $n + 1$. The latter can be done by suitably redefining acceptance in $M'$ and testing for emptiness.

## 4.3 Proof Strategy

We shall prove that equivalence is decidable for T by showing that it is partially decidable. In particular, we shall show that for each pair $M_1$, $M_2 \varepsilon T$, a family P of pushdown automata can be constructed with the properties that

(i) if $L(M_1) = L(M_2)$ then for some $M' \varepsilon P$, $L(M') = \phi$, and

(ii) if $L(M_1) \neq L(M_2)$ then for all $M' \varepsilon P$, $L(M') \neq \phi$.

Thus the enumeration and testing for emptiness of all the machines in P constitutes the required partial decision procedure.

For simplicity of presentation, we shall form a machine M from the disjoint union of the two tested machines $M_1$, $M_2$ as follows. M initially reads a character from the input tape and, depending on what this is, moves to the starting configuration of one of $M_1$ or $M_2$, which it then proceeds to simulate precisely. Clearly, if $M_1$, $M_2 \varepsilon T$, then $M \varepsilon T$ also. The advantage we gain by this construction is that now we need only talk about the equivalence of configurations in a single finite-turn machine.

In our constructions we shall further assume that this machine

M is:

(i) in normal form (§2.4), and

(ii) ordered (§4.2).

Lemma 2.5 and Observation 4.2 ensure that any finite-turn automaton can be transformed into this form.

We shall present our proof in a number of stages.

## 4.4 Parallel Stacking

We shall first outline the general form of the nondeterministic pda $M'$ that is to simulate M.

A configuration of $M'$ has a stack which can be thought of as having a left track and a right track, the top of each one being associated with a state of M. The stack is segmented into lengths of bounded size by special symbols, called "ceilings", occupying both tracks. The finite-state control is able to manipulate directly the top segment (i.e. the part of the stack above the top-most ceiling).

In each segment both tracks contain stack words of length greater than one. Into each ceiling is encoded the following information about the previous history of the computation:

(1) a quadruple $(s_1, A_1, s_2, A_2)$ which states that at the time the ceiling was created, the two tracks had modes $(s_1, A_1)$ and $(s_2, A_2)$ respectively.

(2) an <u>indicator pair</u> from $\{L, R\}^2$ specifying the connection of the tracks above the ceiling to the ones immediately below. For example, (L, L) will mean that both the tracks above are to be associated with the left track below.

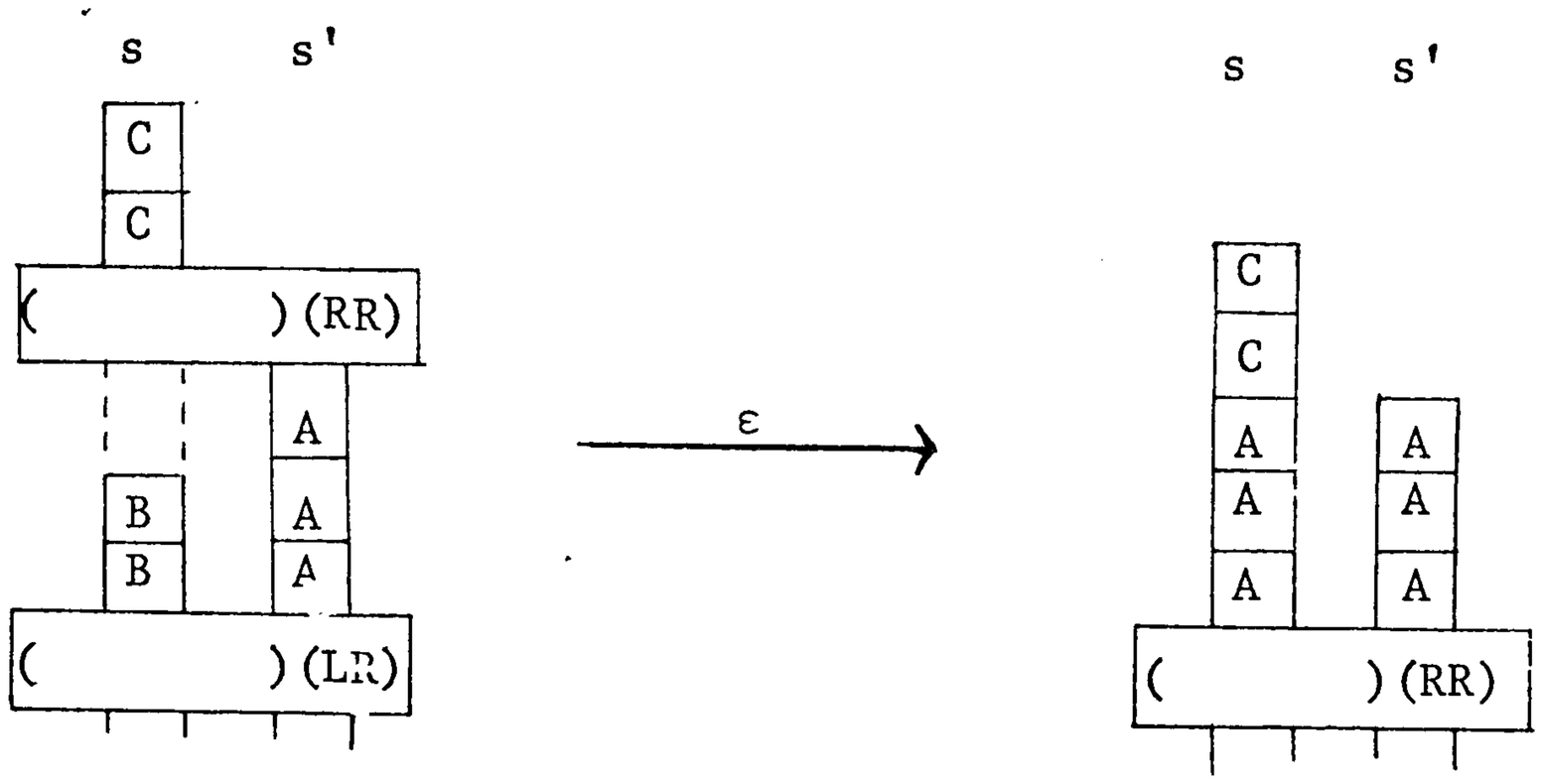
Each configuration of $M'$ is to be interpreted as corresponding to two configurations of M in the obvious way, i.e. the M configurations can be recreated by taking each track in the top segment of $M'$ and concatenating it with the appropriate words in the segments below as specified by the indicator pairs in the ceilings, and by adopting the corresponding state.

The basic operations of $M'$ are to try to mimic simultaneously for all inputs the transitions appropriate to the two simulated configurations of M. In order to be able to do this, while at the same time maintaining an upper bound on the length of the segments that can arise, the machine $M'$ can also on occasions, depending on the contents of the top stack segment, do one of the following additional operations without reading inputs:

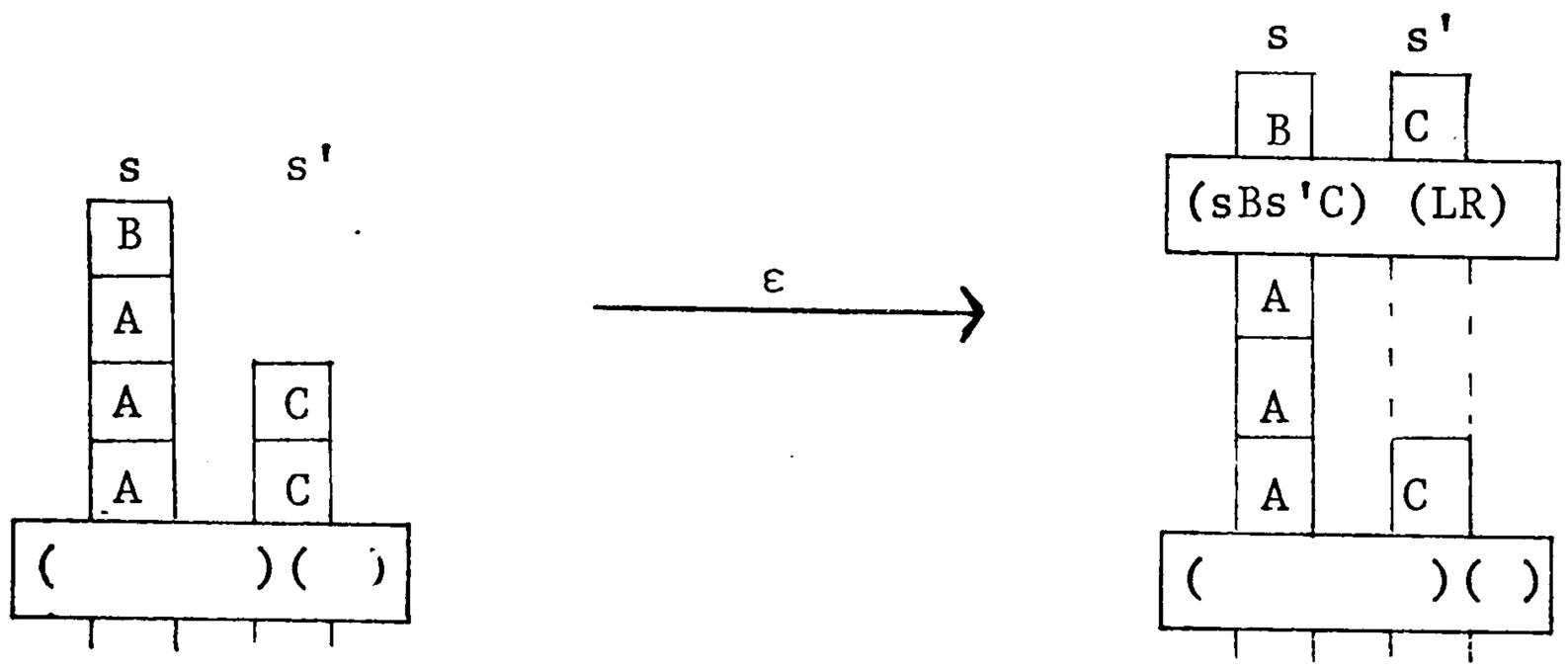
(a) if one of the tracks in the top segment is empty, and the other contains a word from a certain set of 'short' words, then the ceiling below it is removed, and the tracks formerly immediately below and above this ceiling are fused to form one segment, in the manner specified by the indicator pair.

(b) if both tracks have more than one symbol, then a ceiling is placed to be just below the top symbol of each track. The indicator pair (L, R), and the quadruple corresponding to the modes of the tracks are encoded into this ceiling.

move (a):




move (b):




move (c):

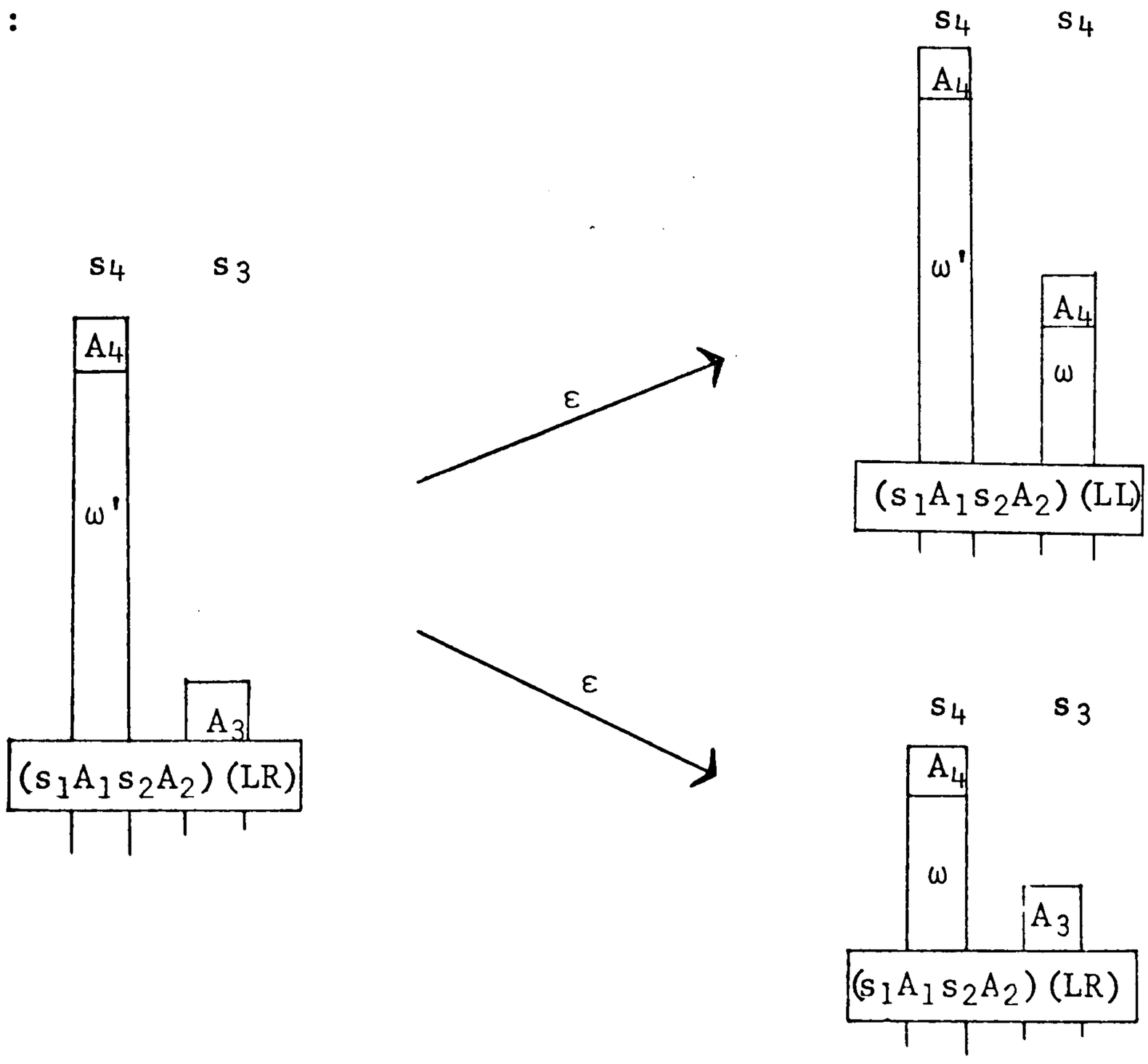



Fig. 4

(c) if one track is empty or has just one symbol, while the other contains a word from some specified set of 'long' words, then M' has the following nondeterministic choice of moves: A replacement stack word, with the same mode and indicator as the large track, is introduced to replace either one of the tracks in the top segment. The simulation is then to be allowed to continue to compare the newly introduced configuration with whichever one of the old ones is left. The replacement word is 'short' in the above sense, and is uniquely specified by the M' configuration.

Examples of each of these three kinds of moves are illustrated in fig. 4. We notice that once a ceiling has been created, though its quadruple cannot be modified, its indicator can be changed by moves of type (a) or (c).

We define <u>acceptance</u> to occur in M' when both tracks of the top segment are in reading modes, exactly one of which is an accepting mode of M.

## 4.5 <u>Existence of a Verifying Machine</u>

We have to show, that if M has two equivalent configurations, then there exists a simulating pda M' of the kind just described, which starts with these configurations in its tracks and accepts no input string.

Clearly any pair of configurations of M, reached via the same input string from a pair of equivalent configurations, will be equivalent. We have to show that when the tracks get too much out of step in M', we can, by making appropriate replacements, get them back within a finite bound while still maintaining the equivalence of the pairs of configurations simulated. Further, we have to verify that the replacements, and the whole simulation, can be carried out by a pda.

We now define the function Rep over

$$Q \times \Gamma \times Q \times \Gamma \times Q \times (\Gamma \cup \{\Lambda\}) \times Q \times \Gamma \times \{1, 2\}$$

to determine these replacements.

Definition

$Rep(s_1, A_1, s_2, A_2 : s_3, A_3 : s_4, A_4, \theta) = \omega$, where $\omega$, if defined, is a shortest non-null stack word with the properties that

(i) $(s_1, A_1) \xrightarrow[+]{} (s_4, \omega A_4)$, and

(ii) $\forall\, \omega_1, \omega_2 \in \Gamma^*$, if $(s_1, \omega_1 A_1) \equiv (s_2, \omega_2 A_2)$

then $(s_4, \omega_1 \omega A_4) \equiv (s_3, \omega_\theta A_3)$.

Without having to construct this function or to determine the arguments for which it is defined, we can observe that since it has a finite domain, we can define $\rho$ to be the maximum of the lengths of the stack words in its range.

In M', if the top ceiling contains $(s_1, A_1, s_2, A_2)$ and (L, R), and the right track above contains $(s_3, A_3)$, and the left one some

$(s_4, \omega' A_4)$ where $(s_4, A_4)$ is a reading mode, then we make the non-deterministic replacement by $(s_4, \omega A_4)$ where $Rep(s_1, A_1, s_2, A_2 : s_3, A_3 : s_4, A_4, 2) = \omega$, provided that $|\omega| < |\omega'|$. This is illustrated in fig. 4(c). If the indicator were (L, L) then the value of Rep for $\theta = 1$ would have been appropriate, while if the top tracks were interchanged we would treat them in a similar way by symmetry.

It is easy to see that under the circumstances specified for the replacement, the function Rep must be defined, for $\omega'$ is itself a suitable candidate for its value: We first notice that the simplest way of realising the necessary conditions is if there is some $\alpha$ s.t.

$$(s_1, A_1) \xrightarrow[+]{\alpha} (s_4, \omega' A_4) \text{ and}$$

$$(s_2, A_2) \xrightarrow[+]{\alpha} (s_3, A_3).$$

Then $\omega'$ clearly has the properties required of a value of Rep (except possibly minimality). In addition, all other ways of reaching the specified situation in M' (i.e. if nondeterministic replacements have been made in the meantime) involve only valid equivalence preserving replacements which are defined completely by the ceiling concerned and subsequent computations, irrespective of the stack contents below. Thus by induction on the number of such replacements made during a derivation in M', we can deduce that a valid replacement is available whenever it is necessary.

A finite-state control can carry out these replacements since Rep is finite in every sense, provided that it can always appropriately manipulate the whole of the top segment. Thus it remains to prove that a bound x exists such that no track segment can become

longer than x in any computation. It is here that the finite-turn property is required.

We first consider in turn the various eventualities that may arise:

(1) If both tracks are steadily increasing then new ceilings will be frequently added and no segment will become large.

(2) If one track is increasing steadily in an upward stroke, while the other one is stationary (i.e. with stack height not changing) then after a stack increase of $q^2t^2h$ in the one, a valid replacement must be possible. For there must be two intermediate points in this derivation at which the mode of the growing track and the configuration of the stationary one both repeat. Thus a replacement word can be obtained by cutting out a segment from the growing track. Therefore, for however long this track is trying to grow, the effect on $M'$ will be to repeatedly make nondeterministic moves so as to keep the segment lengths bounded.

(3) If one track is decreasing then situations can arise in which the length of the segment created depends in a bounded way on the size of the previously existing segments, and is not a priori bounded by $\rho + 1$. There are just two ways in which a track can grow "out of bounds" without immediately being arrested by a nondeterministic replacement:

(i) If one track in the top segment is empty, but the other one is not quite long enough to qualify for a replacement, then the ceiling below may be removed by an (a)-move. Thus the resulting fusion may create a top segment which is suddenly longer by y than before, where y is the previous bound on the segments occurring in

the configuration.

(ii) A track may grow larger and larger without a replacement being possible, if the other track does not become empty or a singleton in the meantime. This second track can be assumed to be steadily decreasing since, as in (2), stationary periods do not contribute to growth. Thus if the decreasing track is initially of height y, then this may cause to arise a segment of length no more than $q^2t^2yh$.

The crucial observation for both (i) and (ii) is that the gain in length achieved by each method can only be exploited to achieve further gains by the same method, once the order of at least one of the simulated configurations of M has increased (i.e. after a turn has been made).

More precisely, we define the order of an M' configuration to be the pair (i, j) when i, j are the orders of the states of the left and right track respectively. Thus the effect of a replacement on a configuration of M' of order (i, j) is to create one of order (i, j) or (i, i) or (j, j), while an ordinary simulating move would lead to one of order (i, j) or (i + 1, j) or (i, j + 1) or (i + 1, j + 1). We have to ensure that the simulating machine cannot enter any "loop" which could cause the segments to increase in size indefinitely. The case we have to consider is that of derivations from configurations of M' of a given order to others of the same order, via ones of different orders. Clearly these must involve replacements of the kind that substitute for the empty or singleton track. Since by definition the modes of the tracks are the same after any such replacement, either both tracks will be in an upstroke, or both in a downstroke. In the former case it is obvious that before the two

tracks can again exhibit different behaviour, turns must be made by both tracks. In the latter case the height of the top segment can be exploited by method (ii) for further growth, after a turn has been made by just one of the tracks. However, the new gains will only be achieved in the track that has undergone a turn, and cannot be exploited for further gains until a turn has occurred in the other track also.

Thus the times when successive gains can be made in the size of segments in excess of the bound $\rho$, can be regarded as occurring at periods during which $M'$ has configurations whose orders form a monotonic increasing sequence under the ordering defined by:

$(i, j) > (i', j')$ iff:

$i, j > i'$, or $i, j > j'$,

or $i > i'$ and $j \geq j'$, or $j > j'$ and $i \geq i'$.

If M can only make n turns, then this sequence can have no more than about $n^2/2$ pairs. Hence we conclude that there is a bound on the maximum size of a segment that may occur in any computation of $M'$. Consequently a finite-state control is sufficient to specify for each stack word that may arise in a track of the top segment, whether moves of type (a), (b) or (c) are appropriate and what form these should take, and to carry out the normal simulation with inputs otherwise.

## 4.6 The Decision Procedure

If a machine $M'$ is constructed for the tested automata $M_1$, $M_2$

as described above, with an arbitrarily guessed replacement function and segment bound, and if it is found to be empty, then we have a verification of equivalence. For let us assume to the contrary that the starting configurations of $M_1$ and $M_2$ are inequivalent. Then at each point in the computation of $M'$ we can follow the shortest string distinguishing the simulated configurations. Then as each input character is read, the rank of the simulated pair decreases. When nondeterministic replacements are made then, clearly, at least one of the new pairs has rank no greater than that of the old one. Further, the replacements all occur in reading modes to ensure that further progress along the shortest distinguishing string can be made immediately. The only exception to this is the situation that occurs when successive replacements without intermediate inputs have to be made because of a long stack decreasing $\varepsilon$-derivation. However this must once come to an end since the stack is finite. Thus we conclude that if $M_1 \not\equiv M_2$ then for some input string we will reach an accepting configuration of $M'$ after a finite number of steps.

Conversely, we have already shown that if $M_1 \equiv M_2$ then some $M'$ exists which always simulates pairs of equivalent configurations of M, and therefore accepts no string.

Thus enumerating such simulating machines for $M_1$, $M_2$, for all possible replacement functions and appropriate finite state controls, and testing each for emptiness is a partial decision procedure for equivalence.

The main result therefore follows.

Theorem 4.1

Equivalence of deterministic finite-turn pushdown automata is decidable. □

Corollary

Equivalence of deterministic two-tape acceptors[6] is decidable.

Proof

For a two-tape acceptor M' with input alphabet $\Sigma'$, and words $(\alpha, \beta)$ on its input tapes, construct a dpda M, with $\Gamma = \Sigma'$, and $\Sigma = \Sigma' \cup \{\$\}$, that accepts exactly the strings $\alpha^R\$\beta$ such that $(\alpha, \beta)$ is accepted by M'. The $\alpha^R$ part is read by M, and stored symbol by symbol on the pushdown stack. Once the $ is read the finite state control treats the input tape and the stack exactly as M' treats its pair of input tapes. M can be made to accept whenever M' accepts, whichever of the various criteria of acceptance is taken (e.g. Rabin and Scott[6], Bird[7]). M is evidently a one-turn machine. □

# Chapter 5 ONE-COUNTER AUTOMATA

## 5.1 Introduction

Counters have been studied in a variety of computational contexts as simple and natural mechanisms for unbounded storage. A counter can be conveniently regarded as a stack with an alphabet of just one symbol. The gain in simplicity achieved by this restriction is that a counter has essentially just two behaviours, depending on whether it is empty or not.

Despite the apparent severity of this limitation, it is well known that several basic problems that are undecidable for automata with stacks, remain so even when these are restricted to counters. Such undecidability results can be deduced from the theorem of Minsky[16] that any Turing machine can be simulated by a 2-register machine. This latter can be regarded as a deterministic automaton with two counters, a finite state control, and transitions reading no inputs. The computation executed by such a machine can be described by a sequence of triples, each giving the state and the emptiness conditions of the two counters, of the successive configurations reached. Since a deterministic one-counter automaton (doca) can check whether successive triples correspond to valid transitions, and also whether the overall action induced in one of the two counters is consistent with the sequence, the termination problem for 2-register machines (and hence Turing machines) can be reduced to the nullity of intersection problem for doca. Thus this latter problem must be undecidable.

Since the class of doca (C) can be shown to be closed under complementation by the same arguments as used in §2.4, §2.5 for dpda, it can be easily deduced that totality, and hence equivalence, for nondeterministic one-counter automata, and, more pertinently, inclusion for the deterministic case, are all undecidable.

Against this background we are nevertheless able to develop a detailed analysis of the structural properties of the computations of doca, and hence derive a decision procedure for equivalence.

We have seen that any doca can be transformed into normal form (§2.4) with h = 2 (§1.7) with no more than a polynomial increase in $q$. We shall assume that the machines are all in this form, so that we can express all the derived properties in terms of just the one parameter q.

The decision procedure takes the form of a nondeterministic simulation, as in the previous chapter, but now the properties we deduce are sufficient to specify explicitly the nondeterministic replacements required. We note, however, that, as in Chapter 3, nondeterminism here is a convenience that is not essential. A procedure could also have been obtained in the form of a <u>finite</u> number of deterministic simulations.

## 5.2 Preliminary Results

The function S(q) (which also gives the maximal order of an element in the symmetric group on q elements) will be the dominating factor in the bounds we derive.

**Definition**

$$S(q) = \max \{\ell.c.m. \{n_i\} \mid \Sigma n_i = q,\ n_i > 0\}.$$

**Lemma 5.1**

$$S(q) = e^{\sqrt{q.\log_e q}.1(q)} \text{ where } 1(q) \to 1 \text{ as } q \to \infty.$$

**Proof**

(Outline) Using standard number theoretic results, this asymptotic function can be easily obtained for the case where $\{n_i\}$ in the definition is additionally restricted to be a set of primes. To show that the same bound is valid for arbitrary $\{n_i\}$, it remains to prove that the possible extra contributions from prime powers is not significant. Details of this argument are given in Valiant and Paterson[20]. □

For a deterministic one-counter automaton M, we describe a configuration c with state s, and counter contents n ($n \geq 0$), by (s, n). For this c, we use c + m to denote the configuration (s, n + m), provided that $n + m \geq 0$.

Definition

The input word $\beta$ is a standard sequence for the configurations c, c' iff

(i) $\beta$ is a shortest string such that $c \xrightarrow[+]{\beta} c'$,

(ii) $\beta = \beta_1\beta_2^r\beta_3$ where $|\beta_1\beta_3| < q^2$, $|\beta_2| \leq q$, and $r > 0$, and

(iii) For some state $s_e$ and integers $w, d > 0$, $\forall v$ s.t. $0 \leq v \leq r$

$$c \xrightarrow{\beta_1\beta_2^v} (s_e, w - vd).$$

Lemma 5.2

For a doca with q states there is a positive integer Z, no greater than S(q), such that if $|c| - |c'| \geq q^2$, $|c'| \geq q^2$ and $c \xrightarrow[+]{} c'$, then there is a standard sequence for c, c' in which the loop drop d divides Z.

Proof

We define the efficiency of a state s to be the maximum value (possibly infinite) of $d/|\gamma|$, where, for all sufficiently large n, the derivation $(s, n + d) \xrightarrow[+]{\gamma} (s, n)$ exists, but repeats no state except s at the beginning and end. Clearly $d, |\gamma| \leq q$.

Suppose $\alpha$ is a shortest string such that $c \xrightarrow[+]{\alpha} c'$. We mark the last occurrence of one of the states with greatest efficiency

in this derivation. Let this state be $s_e$, and let its efficient simple loop be generated by $\gamma$, and cause a drop of d, where $d > 0$. Now excise from this derivation a set of (not necessarily simple) disjoint loops of maximal total length, such that the total drop due to them is a multiple of d, and the marked occurrence of $s_e$ is preserved. We can show that the length m of the remaining derivation is no more than $q^2 - 1$, by first observing that at least $k = \lceil (m + 2)/q - 2 \rceil$ disjoint simple loops, not containing the marked $s_e$ in their interior, must occur it. But $k \geq d$ would imply that some non-null subset of these loops accounts for a total drop that is a multiple of d. This subset could therefore have been removed in the original excision contrary to the maximality condition. Hence $(m + 2)/q - 2 \leq d - 1$, and so $m < q^2 - 1$ follows immediately if $d < q$. By a similar argument it can be shown that the length of the derivation after the marked $s_e$ is less than $q(q - 1)$, if $d < q$.

Let $\beta_1, \beta_3$ be the input strings for the parts of this remaining derivation before and after the chosen occurrence of $s_e$ respectively. Then, clearly, for some integer r

$$c \xrightarrow[+]{\beta_1\beta_3} c' + rd.$$

But $|c| - |c'| \geq q^2 > m$ implies that $r > 0$. Thus

$$c \xrightarrow[+]{\beta_1\gamma^r\beta_3} c',$$

since the bounds derived for the parts of this derivation in the case $d < q$, and simple special arguments in the trivial case $d = q$, ensure that the counter is at no stage empty. Also, since we have replaced

arbitrary loops by ones of at least the same efficiency, $\beta_1\gamma^r\beta_3$ must still be of minimal length.

We note that the case of a null $\gamma$ has not been excluded in this argument. We also note that a simpler argument would suffice if the bounds of $q^2$ were relaxed to $2q^2$.

To obtain a value for Z we investigate the set of possible values of d in the above construction. For each state s we select, if possible, a maximally efficient simple loop through s, and denote the set of states in this loop by Loop(s). Clearly, if $s' \varepsilon$ Loop(s), then the efficiency of s' is greater than or equal to the efficiency of s. Also, any standard sequence whose principal loop is based on s could be replaced by one based on s', by applying the construction to the derivation of the old sequence in which s' must occur since, by definition, $r > 0$.

Let $s' \geq s$ be the transitive closure of the relation defined by $s' \varepsilon$ Loop(s). Defining s, s' to be equivalent iff $s' \geq s$ and $s \geq s'$, the relation $\geq$ becomes a partial ordering on the equivalence classes. Let $s_1,..,s_k$ be a selection of representatives, one from each class that is maximal in this ordering. It is easily verified that the corresponding loops must be disjoint, and that standard sequences can always be based on some such loop. Then the drops due to all these must add up to no more than q, and each one must divide Z where

$$Z = \ell.c.m\ \{d_i \mid d_i \text{ is the drop due to Loop}(s_i)\}.$$

Thus $Z \leq S(q)$. □

That this bound for Z is achievable can be verified by examining Example 7.3.

As we are concerned only with asymptotic bounds, and as S(q) clearly dominates any fixed polynomial in q as q becomes large, it will be sufficient for our purposes to prove the existence of, rather than obtain specific expressions for, the various polynomials we derive. The proof of the following lemma introduces a useful technique.

Lemma 5.3

There is a polynomial $p_3$ such that for any configuration c with $|c| \geq p_3(q)$, and any positive multiple Y of Z,

(i) $\text{rank}\ (c, c + Y) + Y/q \leq \text{rank}\ (c + Y, c + 2Y) \leq \text{rank}\ (c, c + Y) + Yq$,

(ii) $c \equiv c + Y$ iff $c + Y \equiv c + 2Y$.

Proof

Assuming that c and c + Y can be distinguished, there must be a minimal distinguishing sequence $\beta\delta$, where

$$c \xrightarrow[+]{\beta} (s, q^2)$$

for some s. Provided that $p_3$ is sufficiently large, Lemma 5.2 ensures that $\beta$ may be taken to be in the form of a standard sequence $\beta_1\beta_2^r\beta_3$. Let the drop due to $\beta_2$ be d, where $d \geq 0$. Since $\beta\delta$

distinguishes c and c + Y, clearly $\beta_1\beta_2^{r + Y/d}\beta_3\delta$ distinguishes c + Y and c + 2Y. Since $|\beta_2| \leq q$ and $d \geq 1$, the right hand inequality follows.

In a similar fashion we can choose $\beta_1\beta_2^r\beta_3\delta$, where $r > Y/d$, to be a minimal string distinguishing c + Y and c + 2Y. Then $\beta_1\beta_2^{r - Y/d}\beta_3\delta$ distinguishes c and c + Y. But $|\beta_2| \geq 1$, for otherwise, since d divides Z and thus also Y, the sequence would not distinguish c + Y and c + 2Y. The left hand inequality then follows since $d \leq q$ also.

Statement (ii) is an immediate consequence of (i). □

## 5.3 Propriety

We now establish some relationships that hold for periodic sets of configurations.

### Definition

A configuration c is <u>improper</u> iff $c \equiv c + mZ$ for all integers m (not necessarily positive) such that

$$|c| + mZ \geq p_3(q).$$

### Lemma 5.4

If $c \equiv c + mZ$ for some $m > 0$ and $|c| > p_3(q)$, then c is improper.

Proof

It is easy to see that for any set of configurations $\{c_1,\ldots,c_n\}$,

$$\text{rank } (c_n, c_1) \geq \min_{1\leq j<n} \{\text{rank } (c_j, c_{j+1})\}.$$

Hence, if rank $(c, c + mZ) = \infty$, then

$$\text{rank } (c, c + Z) \geq \min_{1\leq i<m} \{\text{rank } (c + iZ, c + (i + 1)Z)\}.$$

By Lemma 5.3(i) it follows that these ranks must all be infinite, and therefore also, by Lemma 5.3(ii), that c is improper. □

Definition

A configuration is proper iff it is not improper.

That the period Z is optimal for the propriety condition can be seen from Example 7.3, where for any configuration c in the starting state, $c \equiv c + i$ iff i is a multiple of $Z = S(q - 1)$.

Lemma 5.5

There is a polynomial $p_5$ such that if $|c| > p_5(q).Z$, $|c'| < q^2$ and $c \equiv c'$, then c is improper.

Proof

Suppose that c is proper, and let $\beta_1\beta_2^r\beta_3\delta$ be a string distinguishing c and c + Z, constructed exactly as in the first part of the proof of

Lemma 5.3, but for the case Y = Z. We define $c_n$, $c'_n$ for $n \geq 0$ by

$$c \xrightarrow{\beta_1\beta_2^n} c_n \quad \text{and} \quad c' \xrightarrow{\beta_1\beta_2^n} c'_n,$$

where, in the case of $\varepsilon$-moves, maximal derivations (i.e. to reading modes) are taken. If $c \equiv c'$, then clearly $c_n \equiv c'_n$ for all n. The polynomial $p_5$ is chosen to ensure that r is sufficiently large for the following argument to work: Either in $c'_0,\ldots,c'_{(2q^4)}$ some configuration repeats, or else some $c'_k$ in this set has height not less than $2q^3$. In the latter case it is easy to verify that for some i, j such that $i < j \leq 2q^4$,

$$c'_i \xrightarrow[+]{\beta_2^{j-i}} c'_j \quad \text{and} \quad c'_j = c_i + w,$$

for some $w > 0$. In either case, for some i, j such that $i < j \leq 2q^4$, we have, putting $\ell = (j-i).Z$,

$$c'_{i+\ell} = c'_i + wZ$$

where now $w \geq 0$. By Lemma 5.3(i) if $w > 0$, and trivially if $w = 0$,

$$\text{rank } (c'_i, c'_{i+\ell}) \leq \text{rank } (c'_{i+\ell}, c'_{i+2\ell}).$$

However, if r is large enough then from the propriety of $c_i$ and Lemma 5.3(i),

$$\text{rank } (c_i, c_{i+\ell}) > \text{rank } (c_{i+\ell}, c_{i+2\ell}).$$

This contradicts the assumption that $c_n \equiv c'_n$ for all n. Thus c must be improper. □

We can now derive, as a consequence of this result, the property of equivalent configurations on which our decision procedure depends.

Definition

Integers m, n are (x, y)-rationally related iff there exist integers a, b with $0 < a, b \leq x$, such that

$$|ma - nb| \leq y.$$

Lemma 5.6

There exist polynomials $p_6$, $\overline{p}_6$ such that if $c \equiv c'$, $|c| > \overline{p}_6(q).Z$, and c is proper, then $|c|$, $|c'|$ are $(q^2, p_6(q).Z)$ - rationally related.

Proof

Suppose that $c \equiv c'$, that c is proper, and that $|c| > \overline{p}_6(q).Z$, for $\overline{p}_6$ sufficiently large for the following argument to work. Choose $\beta_1, \beta_2, \beta_3, \delta$, and define $c_n$, $c'_n$ as in the previous lemma. Let $\ell$ be the least n such that

$$\min (|c_n|, |c'_n|) < q^2.$$

$\ell$ must exist, for otherwise $\{|c_n|\}$ would be an infinite strictly decreasing sequence. $c_n$ is proper for all $n \leq \ell$, and therefore $c_n \neq c_n + Z$ if $|c_n| \geq p_3(q)$.

If $|c'_\ell| < q^2$ then by Lemma 5 and the propriety of $c_\ell$, we have that $|c_\ell| \leq p_5(q).Z$.

Alternatively suppose $|c_\ell| < q^2$. For some i, k' such that $0 \leq i < i + k' \leq q$, the states of $c'_i$, $c'_{i+k'}$ are the same, $c'_{i+k'} = c'_i - e$ say, where $|e| \leq q^3$. Then if k is a large enough multiple of k' to ensure that $|c_{\ell-kZ}| \geq p_3(q)$, the propriety of the $c_n$ gives

$$c_{\ell-kZ} \neq c_{\ell-kZ} + dkZ = c_{\ell-2kZ}.$$

Hence

$$c'_{\ell-kZ} \neq c'_{\ell-2kZ},$$

that is

$$c'_\ell + ekZ \neq c'_\ell + 2ekZ.$$

By lemma 5, $|c'_\ell| \leq p_5(q).Z$, for otherwise $|c'_\ell|$ would be proper, contrary to the previous statement.

Thus for a suitable choice of $p'_6$ we have in both cases

$$||c| - d\ell| < p'_6(q).Z \text{ and } ||c'| - e\ell/k| < p'_6(q).Z$$

Since $0 < d \leq q$ and $0 < e/k \leq q^2$, it follows for some $p_6$ that

$$||c|.e/k - |c'|.d| < p_6(q).Z. \square$$

## 5.4 Decision Procedure

Using the result of Lemma 5.6 we construct a nondeterministic one-counter automaton M' which is able to simulate, in a certain sense, the computations of a pair of equivalent doca. By taking the disjoint union of the states and transition rules of the two machines, we can regard the simulation as maintaining a representation of pairs of equivalent configurations, c and c', of the combined machine. We ensure that at each point in the simulation $a|c| - b|c'| = f$, for some a,b,f such that $0 < a, b < q^2$, and $|f| < p_6(q).Z$. Then M' can represent c and c' by holding $a|c|$ in its counter and remembering f,a,b, and the states of c and c', in its finite state control.

The action of m' is as follows. Let $p_o$ be some polynomial such that whenever m or n is greater than $p_o(q).Z$, and they are $(q^2, p_6(q).Z)$-rationally related, then they are related with respect to only one admissible rational ratio a/b. Whenever $|c|$, $|c'|$ are both less than $p_o(q).Z$, their values are stored in the finite state control. When a simulation step is about to exceed this bound, the finite state control determines the coefficients a,b, if any, and sets up the counter for the appropriate representation. When a simulation step would reach a pair of configurations not rationally related, say c is just too large for c', then by Lemma 5.6, if $c \equiv c'$, then c must be improper. Instead of continuing the simulation with (c, c'), a nondeterministic step is made <u>either</u> to the simulation of (c - Z, c') <u>or</u> to the simulation of (c, c - Z). Then, if $c \equiv c'$ and so also $c \equiv c - Z$, the simulation continues to be one for equivalent configurations in either case.

Since the original doca are in normal form, we can easily define acceptance in M' to occur if and only if exactly one of the simulated configurations is in an accept mode. Thus if the starting configurations are indeed equivalent, our discussion shows that no string is accepted by M'. On the other hand, if they are inequivalent, we can show, as in the previous chapter, that some string must be accepted. For if some $\alpha$ distinguishes the starting configurations, then either both derivations will be simulated directly to their different conclusions, or else the rational relationship must fail. In the latter case, if (c, c') is reached where $c \neq c'$, then the remainder of $\alpha$ distinguishes one of the new pairs created. The assumed normal form guarantees that any long $\varepsilon$-derivations will steadily reduce the counter. This, in turn, ensures that further progress along $\alpha$ can always be made in a finite number of moves, and therefore that $\alpha$ will eventually be accepted.

The construction and testing for emptiness of the simulating machine described therefore constitutes a decision procedure for equivalence. The number of states of this machine need be no more than $p(q).Z^2$ for some polynomial p, where q is the total number of states of the tested machines, and Z is bounded above by $S(q) \triangleq e^{\sqrt{q.\log_e q}}$. Assuming a fixed input alphabet, and recalling Lemma 2.1 we conclude that

Theorem 5.1

The equivalence problem for doca is decidable, and there is a decision procedure which, for q state machines, has a running time bounded above by

$$2^{k\sqrt{q.\log q}}$$

for some constant k. □

# III CONTAINMENT PROBLEMS

Certain restricted classes of deterministic pushdown automata have some important basic properties which do not hold for the whole class. For example, finite-state machines recognise a class of languages that are closed under the Boolean operations, one-counter machines, we have seen, have a rigid periodic structure, while for simple machines each configuration can be related very directly to the language it generates. Furthermore, within these and some other subclasses we know how to test for equivalence. It is also plausible that even for problems for which decidability can be proved in the unrestricted case, easier decision procedures can be found for these subclasses than for the whole class.

For any of these reasons we may want to determine whether for a given dpda there exists an equivalent one belonging to a particular subclass. Formally we ask the following containment problem: If X, Y are two classes of automata and $M \in X$, then is there an $M' \in Y$ such that $L(M) = L(M')$? We denote the containment problem for X, Y by (X:Y).

Emptiness, finiteness and totality can all be phrased as containment problems. Testing for the prefix property we have seen in §1.4 to be equivalent to the problem $(D:D_o)$. Of the more difficult problems mentioned above, the only one known to be decidable is that of regularity, i.e. (D:Fsa). A proof of this has been given by Stearns[2]. Without resolving the remaining open problems we shall nevertheless throw some light on their expected

difficulty by relating them to the regularity problem in the following way.

We shall define a very general notion of relative complexity with which one can compare the inherent difficulty of various containment problems. Results expressed in terms of this are of wide applicability. For example, our result that testing for emptiness, and for the prefix property are, in the defined sense, equally difficult, implies that if the time complexity for deciding these two problems on any machine model are polynomials in the parameters of the tested automata, then the leading terms of these polynomials will differ by only a multiplicative constant.

The results we then prove using this notion are that such containment problems as (D:C), (D:T), and ($D:S_o$) must be, if decidable at all, at least as difficult to decide as regularity.

With this as one source of motivation, we then proceed in the following chapter to investigate the regularity problem in detail. For a natural particular measure of complexity used by Stearns, we improve his upper bound from a treble to a double exponential level, which now closely approaches a known double exponential lower bound. As a consequence we can also significantly improve the upper bound on the time complexity of this problem.

We observe that all the containment problems in which we are here interested become undecidable if D is replaced by ND, the class

of nondeterministic pushdown automata. This can be deduced from a theorem of Korenjak and Hopcroft [9] who construct, for each instance of the Post Correspondence Problem, a context-free grammar over a terminal alphabet $\Sigma \cup \{\$\}$, with the properties that

(i) if the PCP has no solution the language generated is $\Sigma^*\$$,

(ii) if the PCP has some solution then the language is not deterministic.

From this we conclude that for any class $Y \subset D$ such that $\Sigma^*\$ \in L(Y)$, (ND:Y) is undecidable. It therefore follows that (ND:Y) is undecidable if Y is $D_0, S_0$, T, C, Fsa, etc.

# Chapter 6 RELATIVE COMPLEXITY

## 6.1 Introduction

We shall define a partial ordering on decision problems to express the relative difficulty of solving them. Thus $P_1 \geq P_2$ will mean that if, on any machine model, the problem $P_2$ needs time x to be decided, then $P_1$ will require at least about the same time. Furthermore, similar conclusions can then also be made about space requirements, and various other measures as well.

We could define such an ordering very simply by saying that $P_1 \geq P_2$ iff any procedure to decide $P_1$ is effectively able to decide $P_2$ also. For our applications to dpda problems, however, it is convenient to relax this condition slightly. We denote by $D(n_1,n_2,n_3,n_4,n_5)$ the class of dpda whose parameters q,t,h,p,u are respectively bounded above by $n_1,n_2,n_3,n_4$ and $n_5$. We define a transformation of a machine description to be direct iff it can be carried out by an algorithm that makes only one pass of the transition rules, (and modifies each one as necessary,) requires only a finite amount of memory additional to the capability of recognising accepting modes, and increases each parameter at most linearly. We then say that a procedure can decide problem P directly for a class X, iff there is a direct transformation which takes all machines in X to a form in which the application of the procedure effectively decides P.

Definition

$P_1 \geq P_2$ iff $\exists\, k > 0$ s.t. any procedure to decide $P_1$ for

$D(kn_1,kn_2,kn_3,kn_4,kn_5)$ can directly decide $P_2$ for $D(n_1,n_2,n_3,n_4,n_5)$.

Definition

$P_1 = P_2$ iff $P_1 \geq P_2$ and $P_2 \geq P_1$.

It is immediate from the definition that for the classes of automata X,X',X'' the following relation holds among the containment problems:

(i) $X' \subset X'' \Rightarrow (X'':X) \geq (X':X)$.

We observe that, in order to decide any global property of a machine, we require operations at least as difficult as a direct transformation. Thus the relaxation of our definition of the ordering does not endanger its validity for our purposes.

## 6.2 Results

As is permitted by virtue of Lemma 2.5, we shall assume here, for convenience, that all dpda are in normal form.

Theorem 6.1

$(D:T) \geq (D:Fsa)$.

Proof

Let $L = L(M) \subset \Sigma^*$ where $M \in D$, and let $L' = (L\$)^*$ where

$\$ \notin \Sigma$. We shall show that $L' = L(M')$ for some $M' \in D$ which is only slightly larger than, and easily obtained from M, and, further, that $L' \in L(T)$ iff L is regular. This is clearly sufficient, for then any M can be tested for regularity by testing the appropriate $L(M')$ for containment in $L(T)$.

We create M' to recognise L' by modifying M in the following way. For every accepting mode of M we introduce a move on \$ input to a new special state that causes the stack to empty and the starting configuration of M to be restored, all via an $\varepsilon$-derivation. If we make the starting and accepting configurations of M' to be the starting configuration of M, then clearly $L(M') = (L\$)^*$. To do all this we need add no more than one state, and one input symbol, and we will at most have doubled the number of transition rules. Furthermore, M' can be obtained from M by a direct transformation.

To show that L regular $\Rightarrow L' \in T$, we simply observe that

L regular $\Rightarrow (L\$)^*$ regular $\Rightarrow L' \in O\text{-}T \Rightarrow L' \in T$.

To show the converse, we recall a comment made in §4.2 that $L(O\text{-}T) = L(Fsa)$. Thus, if L is not regular then in any recognising machine for $(L\$)^*$, turns may have to occur during the parsing of each substring between successive \$ characters. In that case the machine cannot be finite-turn. □

Theorem 6.2

$$(D:C) \geq (D:Fsa).$$

Proof

Let $L = L(M) \subset \Sigma^*$, where $M \in D$, and let $L' = \bigcup_{n=1}^{\infty} (L\$)^n \$_1^n$, where $\$, \$_1 \notin \Sigma$. We shall show that there is an $M' \in D$ recognising $L'$ that is only slightly larger than M, and that any procedure for testing $L' \in L(C)$ will automatically decide whether L is regular.

We construct $M'$ to simulate M repeatedly, and to count the number of strings from $L\$$ read, by keeping a string $A^m$ at the bottom of the stack, where A is a new symbol. Thus as in the previous theorem, whenever an accept mode of M is reached and a $\$$ immediately follows, the stack is emptied, but now only up to the topmost A, an extra A is added, and the starting mode is restored on top of this. When the $\$_1$ characters are read, they are checked one by one against the A's, and acceptance occurs iff they are equal in number. Such an $M'$ can clearly be produced from M by a direct transformation, and will not be much larger.

It remains to show that $L' \in L(C)$ iff L is regular. Clearly if L is regular then a recognising machine for $L'$ exists which only uses its stack to store the A's. Thus $L' \in L(C)$. Conversely suppose L is not regular. Any machine recognising $L'$ must have an infinite set of pairwise distinguishable configurations reached

via input strings terminating with a $. If it is a 1-counter machine and L is not regular, then to recognise some words from L from any such configuration, the machine will have to empty its stack, for otherwise, regularity would be implied. However once the stack is emptied, all but a finite amount of information about the number of instances of L$ already parsed, is lost. Thus $L' \notin L(C)$. □

Definition

For a language $L \subset \Sigma^*$:

$Sinit(L) = \Sigma^* - L\Sigma^+$.

Lemma 6.1

If L has the prefix property then:

L regular $\Leftrightarrow$ Sinit(L) is regular.

Proof

(i) Since Sinit(L) is defined by regularity preserving operations, if L is regular then so is Sinit(L).

(ii) If L has the prefix property then it consists of just those words in Sinit(L) that are not proper prefixes of other words in Sinit(L). Thus a finite state automaton for L can be obtained from one for Sinit(L) by simply removing all states from which any further strings can be accepted, from the set of accepting states. □

Theorem 6.3

For any class $X_0 \subset R_0$ such that $L(Fsa)\$ \subset L(X_0)$,

$$(D:X_0) \geq (D:Fsa)$$

Proof

Let $L = L(M) \subset \Sigma^*$ where $M \in D_0$, and let $L' = Sinit(L)\$$, where $\$ \notin \Sigma$. We modify M by replacing all $\Sigma$-transitions from accepting modes to ones to a special reject mode, and by adding a \$ transition for all reading modes (other than the reject mode) to a new mode which is now declared the sole accepting mode. Then this new machine, say M', recognises L', and so L(M') is regular iff L is regular (from Lemma 6.1).

If L(M') is regular then, by definition, $L(M') \in L(X_0)$. However, if L(M') is not regular, then Sinit(L) is not regular and so requires arbitrary large stacks to occur during recognition. But from any such live configuration we expect acceptance to be possible in M' with a further input of only a single character \$. Thus L(M') not regular $\Rightarrow L(M') \notin L(X_0)$.

We can therefore conclude that $L(M') \in L(X_0)$ iff L is regular, and hence that

$$(D_0:X_0) \geq (D_0:Fsa).$$

However, from property (i) in §6.1 we know that

$$(D:X_0) \geq (D_0:X_0).$$

Also, since any test for the regularity of L$ for any $L \subset \Sigma^*$ can be used as a test for the regularity of L by the now familiar argument, we also have that

$$(D_0:Fsa) \geq (D:Fsa).$$

By the transitivity of the ordering the result follows. □

Theorem 6.4

(D:empty) = (D:total) = $(D:D_0)$, where empty, total refer to the class of machines accepting nothing, and $\Sigma^*$ respectively.

Proof

From the argument in Corollary 2.1 to Lemma 2.5 it is immediate that a direct transformation exists to modify any machine M in normal form, to one that recognises exactly the complement of L(M). It follows that emptiness and totality are equally difficult to decide in our sense.

To show that $(D:empty) \geq (D:D_0)$ we recall that the latter is equivalent to testing for the prefix property. For any $M \in D$

we can construct an M' consisting essentially of two copies of the transition rules for M with distinguished state sets. These rules are modified so that transitions from accepting modes in the first copy lead to the appropriate states of the second copy. The starting configuration of M' is defined to be that of the first copy of M, while the accepting modes are those of the second copy. Clearly M' accepts just those strings of L(M) that have proper prefixes in L(M). Thus testing M' for emptiness is equivalent to testing whether $L(M) \in L(D_0)$.

To see that $(D:D_0) \geq (D:empty)$ we observe that any $M \in D$ can be modified by replacing all the transition rules from each accepting mode by a reading rule leaving the mode unchanged. Then testing this machine for the prefix property will effectively test M for emptiness. □

Theorem 6.5

$$(D:Fsa) \geq (D:finite) \geq (D:empty)$$

Proof

Let $L = L(M) \subset \Sigma^*$ where $M \in D$, and let $L' = L\$^*$.

Clearly M can be transformed to become a recogniser for L' with little change in size. Since L' is finite if and only if L is empty, L can be tested for emptiness by testing L' for finiteness.

To show the other inequality, we use the observation that for a dpda M', L(M') is infinite iff M' has some live derivation that repeats a configuration, or has a repeated mode in its stacking sequence. We introduce some new stack symbols, an $\overline{A}_i$ for each $A_i \in \Gamma$, and a special B. For each mode $(s,A_i)$ we replace each rule $(s,A_i) \overset{\pi}{\rightarrow} (s_1,w)$ by $(s,\overline{A}_i) \overset{\pi}{\rightarrow} (s_1,w)$ and add the rule $(s,A_i) \overset{\$}{\rightarrow} (s,B\overline{A}_i)$, and also, if it is an accepting mode, the rule $(s,A_i) \overset{\$_1}{\rightarrow} (s_a,\Lambda)$ where $s_a$ is a new special state, and \$, $\$_1$ are new input characters. We also add the transitions $(s,A) \xrightarrow{\$_1} (s,\Lambda)$ for any mode with $s = s_a$, or $A = B$, and add $(s_a,\Omega)$ to the set of accepting modes. Assuming that M' is in a form never requiring to empty its stack, it can be verified that the modified machine just described will recognise a regular set iff L(M') is finite. □

## 6.3 Comments

We have only obtained theorems for (X:Y) where X = D. However numerous comparable results can be derived by similar means for X equal to various subsets of D.

Thus we can relate the realtime property, and the stateless property to regularity. If we define $T^1$ to be the class of ordered deterministic one-turn machines, with only one state of order zero, then we can show that (T:R) ≥ ($T^1$:Fsa) and that (T:S) ≥ ($T^1$:Fsa). To do this, the construction we need for each M' ∈ $T^1$ is of an M ∈ T that accepts the language

$$\{\ \alpha\$\beta\$\gamma \mid \beta\gamma \in L(M')\ \} \cup \{\ \alpha\$\beta\$_1\gamma \mid \alpha\gamma \in L(M')\ \}$$

where each $\alpha$, $\beta$ is a turn-free prefix of $L(M')$. We note that despite the restricted nature of $T^1$, our analysis of regularity (Chapter 7) gives no indication of this problem being substantially easier to decide for $T^1$ than for D.

The equivalence problem is not a containment problem. However the above mentioned kind of argument shows immediately that it is no easier to decide for $D_0$ than for D. For to test for $L_1 \equiv L_2$ in D, we could test for $L_1\$ \equiv L_2\$$ in $D_0$. Even more trivially we notice that since emptiness is a particular instance of equivalence, it cannot be more difficult to decide.

We conclude by summarising our results in the following diagram, in which $P_1 \rightarrow P_2$ implies $P_1 \geq P_2$, and each language is assumed to be specified by a dpda in normal form.

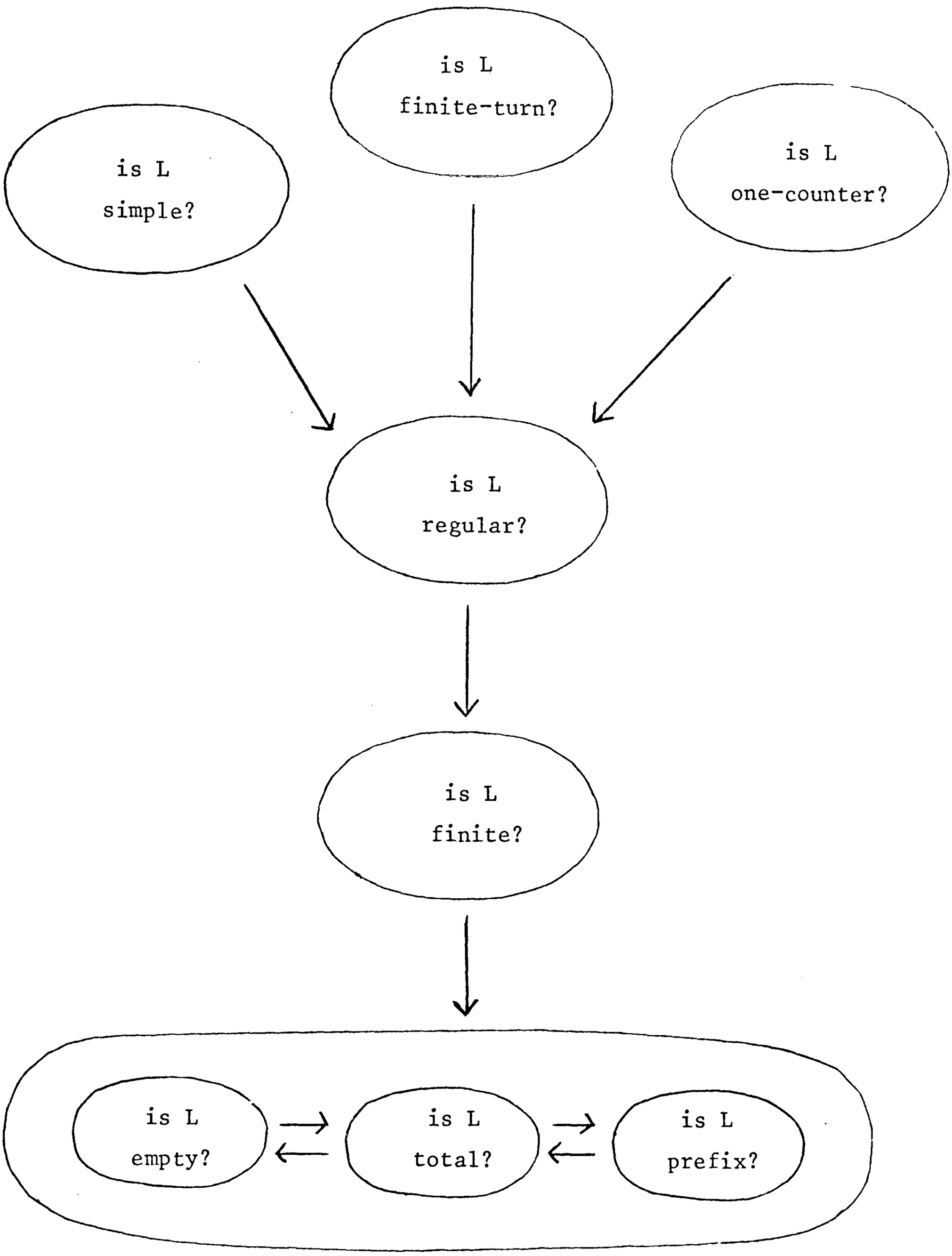



Fig. 5. Relative Complexity of Containment Problems.

# Chapter 7 THE REGULARITY PROBLEM

## 7.1 Introduction

Stearns[2] shows that the problem of whether a dpda accepts a regular set is decidable. To do this he proves that if a dpda of a certain form, with q states and t stack symbols, accepts a regular language L, then L is also recognised by some finite-state automaton, the states of which number no more than some $E^3$ expression in q and t. Meyer and Fischer[21] give an example to show that under these circumstances a finite-state automaton of $E^2$ size may indeed be necessary.

Our main result in this Chapter is to reduce the upper bound given by Stearns for this, by an exponential level, to an $E^2$ function which differs from the lower bound of Meyer and Fischer by only a multiplicative factor in the leading exponent. As a consequence we can also derive an $E^2$ time algorithm for testing dpda for regularity.

By similar analysis for the regularity of each of the classes R, C and $S_0$ we obtain distinct $E^1$ expressions, the orders of which in each case we can show to be valid as both upper and lower bounds.

Almost all the ideas we shall use can be found in Stearns' paper. However, in addition to the improvement in the final results, the following differences are noteworthy.

Our proof is for the general case allowing arbitrarily long right hand sides in the transition rules, while Stearns considers

only conservative machines with h = 2. Thus, although for some dpda q may have to grow exponentially when it is reduced to the h = 2 form, we shall show that this is not a source contributing an extra exponential in the main result. Furthermore, our analysis will be directly applicable to, for example, stateless and dB-S machines (Chapter 8), for which equivalent conservative machines do not exist in general.

The notions of null-transparent and $\ell$-invisible segments, introduced by Stearns, and the proofs of their existence in sufficiently large stacks, remain at the centre of the argument. However, we have unified the proofs of these theorems (Lemma 7.2) by applying the technique used by Stearns for the one (null-transparency) to obtain the now improved result for the other ($\ell$-invisibility) also. Our definition of the latter is a generalisation of that of Stearns, that contributes a further smaller improvement.

In addition we are rather more explicit about the phenomena that correspond to regularity and irregularity respectively. We exhibit the fact that our main construction picks out a family of computations that resembles a one-counter automaton in structure and has the same behaviour vis-a-vis regularity.

We shall use the notation introduced in §1.5.

## 7.2 Null-transparency and ℓ-invisibility

It will be convenient here to say that, in a derivation $c \vdash (\alpha)\ c'$, where $\alpha = \alpha_1\alpha_2\alpha_3$, "the $\alpha_2$ subderivation pops the segment $\omega_{i,j}$" in c iff $c \vdash (\alpha_1)\ (s,\omega_{o,j})$ and $(s,\omega_{o,j}) \vdash (\alpha_2)\ (s',\omega_{o,i})$ for some s, s'.

Definition

$(s,\omega) \vdash (\alpha)\ c'$ is a j-derivation w.r.t. index set N iff there are fewer than j pairs (m,n) of consecutive elements of N with the property that $\omega_{m,n}$ is popped by a non-null subsequence of $\alpha$.

Definition

The segment $\omega'$ is ℓ-invisible in $(s,\omega\omega'\omega'')$ w.r.t. index set N iff for any s', and any ℓ-derivation $(s,\omega\omega'\omega'') \vdash (\alpha)\ (s',\omega\omega')$, it is the case that $(s',\omega\omega') \vdash (\varepsilon)\ (s',\omega)$.

In other words, the existence of the segment $\omega'$ can only be detected in the configuration by derivations which pop by non-null input strings at least ℓ of the segments of $\omega''$ induced by N.

Definition

The stack word $\omega$ is null-transparent iff for all $s \in Q$,

$$(s,\omega) \vdash (\varepsilon)\ (s',\Lambda) \Rightarrow (s',\omega) \vdash (\varepsilon)\ (s',\Lambda).$$

A null-transparent segment therefore has the property that if $(s,\omega) \vdash (\varepsilon)\ (s'\Lambda)$, then for all $n \geq 1$, $(s,\omega^n) \vdash (\varepsilon)\ (s',\Lambda)$. Thus $\varepsilon$-derivations which pop sequences of a null-transparent word are incapable of distinguishing different numbers of occurrences of them.

Lemma 7.1

If $\omega$ is null-transparent, then for all s, $\omega_1$, no string $\alpha$ of length n can distinguish $(s,\omega_1\omega^m)$ from $(s,\omega_1\omega^{m'})$ for $m,m' > n$.

Proof

Consider $\alpha$ derivations from the two configurations. If these have at no stage popped the top n + 1 copies of $\omega$ in the stack, then clearly they cannot be distinguishing derivations. However, if they have, then at least one copy of $\omega$ must have been popped by an $\varepsilon$-subderivation. But then by null-transparency, the rest of the $\omega$ segments would also have been popped at the same time, without leaving any trace of their number. Thus we conclude that no $\alpha$ of length n can distinguish the given configurations. □

We now prove by an inductive argument the existence of both kinds of segments in sufficiently large configurations.

Lemma 7.2

For a configuration c with stack $\omega$ and an index set N of $\overline{N}$ elements all less than $|\omega|$,

(i) $\overline{N} > q!$ ⇒ some segment of $\omega$ induced by N is null-transparent,

(ii) $\overline{N} > \ell(\ell q)^q$ ⇒ some segment of $\omega$ induced by N is $\ell$-invisible in c, provided that $\ell, q > 2$.

Proof

For each part we produce an inductive assertion of the form $A(P_m, N_m)$ for $m = 0,1,2\ldots$, where $P_m \subset$ a finite set P, and $N_m \subset N$. We show that $A(P_o, N_o)$ is true for $P_o = \phi$ and $N_o = N$, and that, if $\overline{N}_m$ is sufficiently large and $A(P_m, N_m)$ holds, then <u>either</u> $N_m$ already induces the required segment, <u>or</u> else we can find $P_{m+1}$ and $N_{m+1}$ such that $A(P_{m+1}, N_{m+1})$ is true also. We ensure that the induction terminates, and thus guarantees to produce the required segment, by showing that $P_m \subsetneq P_{m+1}$ and that $\overline{N}_{m+1}$ is greater than some given function of $\overline{N}_m$. Then by picking N large enough initially, we can ensure that, though the sets $N_m$ may get successively smaller, they will always be large enough to enable the induction to continue until P is exhausted.

We take P to be Q, and the assertion $A(P_m, N_m)$ to be:

$$i,j \in N_m,\ s \in P_m \Rightarrow (s,\omega_{ij}) \nvdash (\varepsilon)\ (s,\Lambda).$$

Then $A(P_o, N_o)$ is trivially true.

(i) We assume that $A(P_m, N_m)$ is true, and that k and k', the smallest and largest elements of $N_m$ respectively, are distinct.

Then if $\omega_{k,k'}$ is null-transparent, the result is proved. Otherwise, by definition there exist s,s' such that

(a) $(s,\omega_{k,k'}) \vdash (\varepsilon)\ (s',\Lambda)$,

but (b) not $(s',\omega_{k,k'}) \vdash (\varepsilon)\ (s',\Lambda)$.

Then let $N_{m+1}$ be the subset of $N_m$ indexing the most frequently occurring state, say s", in the popping sequence for (a) w.r.t. $N_m$, and let $P_{m+1} = P_m \cup \{s"\}$. No state in this popping sequence can belong to $P_m$, for that would imply, by the inductive assertion, that $s' = s" \in P_m$, which would contradict (b). Thus it follows that $P_m \subsetneq P_{m+1}$, and then also that $\overline{N}_{m+1} \geq \overline{N}_m/(q-m)$, for $m < q$. $A(P_{m+1},N_{m+1})$ is then clearly true. Also, if $\overline{N} = \overline{N}_o > q!$, the induction can continue, if necessary, until $P_m$ exhausts Q, without ever $\overline{N}_m < 2$ occurring. This completes the proof of (i).

(ii) We assume that $A(P_m,N_m)$ is true, and that k and k', now the smallest and second smallest elements of $N_m$ respectively, are distinct. Then if $\omega_{k,k'}$ is $\ell$-invisible in c, the result is proved. Otherwise, by definition, there is some $\alpha$-derivation that is an $\ell$-derivation rendering it visible, i.e.

(a) $c \vdash (\alpha)\ (s',\omega_{o,k'})$,

but (b) not $(s',\omega_{o,k'}) \vdash (\varepsilon)\ (s',\omega_{o,k})$

From the $\alpha$ derivation extract the $\varepsilon$-subderivation that pops the most segments induced by $N_m$, and let $N'_{m+1}$ be just those indices inducing these segments. Then, by definition, $\overline{N}'_{m+1} \geq (\overline{N}_m - 1)/\ell$.

No state in the popping sequence induced by $\overline{N}'_{m+1}$ can belong to $P_m$, for that would contradict (b). Let s" be the most frequently occurring state in this sequence, let $N_{m+1}$ be the set indexing these occurrences, and let $P_{m+1} = P_m \cup \{s''\}$. Then $N_{m+1} \geq N'_{m+1}/(q-m)$, Now if $\overline{N} = \overline{N}_o > \ell(\ell q)^q$ initially, the induction will continue successfully until Q is exhausted. □

Note 1

The bound for (i) of q! is the same as that of Stearns. It can be shown to be optimal by looking at segments that are ε-popped from all states thereby performing permutation operations on them, and regarding these as elements of the symmetric group on q elements.

Note 2

In the main theorem we shall be interested in $\ell$-invisible segments, where $\ell$ is exponential in q. It is here that we gain our most significant improvement, by obtaining a bound of order $(\ell q)^q$ as compared with one of $q^{\ell+q}$ given by Stearns. In general, if $\ell$ is of larger order than q, we can show that our bound is of optimal order in the following sense. A dpda can be derived from the proof of 7.2(ii) with the properties that for some configuration and index set of size $(\ell/q)^q$, no $\ell$-invisible segment can be found.

## 7.3 Main Theorem

### Theorem 7.1

If M, a dpda in normal form, has q states, t stack symbols, and stack words of length at most h in its transition rules, and if L(M) is regular, then L(M) is recognised by some finite automaton with fewer than X(q,t,h) states, where X is of order $E^2$.

### Proof

We shall prove that there is a function Y(q,t,h) of order $E^1$ such that, if any reachable configuration of M has height greater than Y, then <u>either</u> we can cut a segment out from the stack to obtain a smaller equivalent reachable configuration, <u>or else</u> there are input strings $\delta_1,\delta_2$ s.t. the configurations reached after inputs of $\delta_1\delta_2^m$ for m = 1,2,... are all pairwise inequivalent. Thus if L(M) is regular, the first possibility must always hold for configurations larger than Y, and consequently M can only be using up to $qt^Y$ pairwise distinguishable configurations in recognising the language. This gives us the required result.

To prove the existence of Y we consider an arbitrary derivation $c_s \overset{\alpha}{\rightarrow} c = (s,\omega)$, where $|\omega| = n > Y(q,t,h)$. We let N be the set of integers indexing the most frequently occurring modes in the stacking sequence of this derivation. Then clearly $\overline{N} \geq n/qth$.

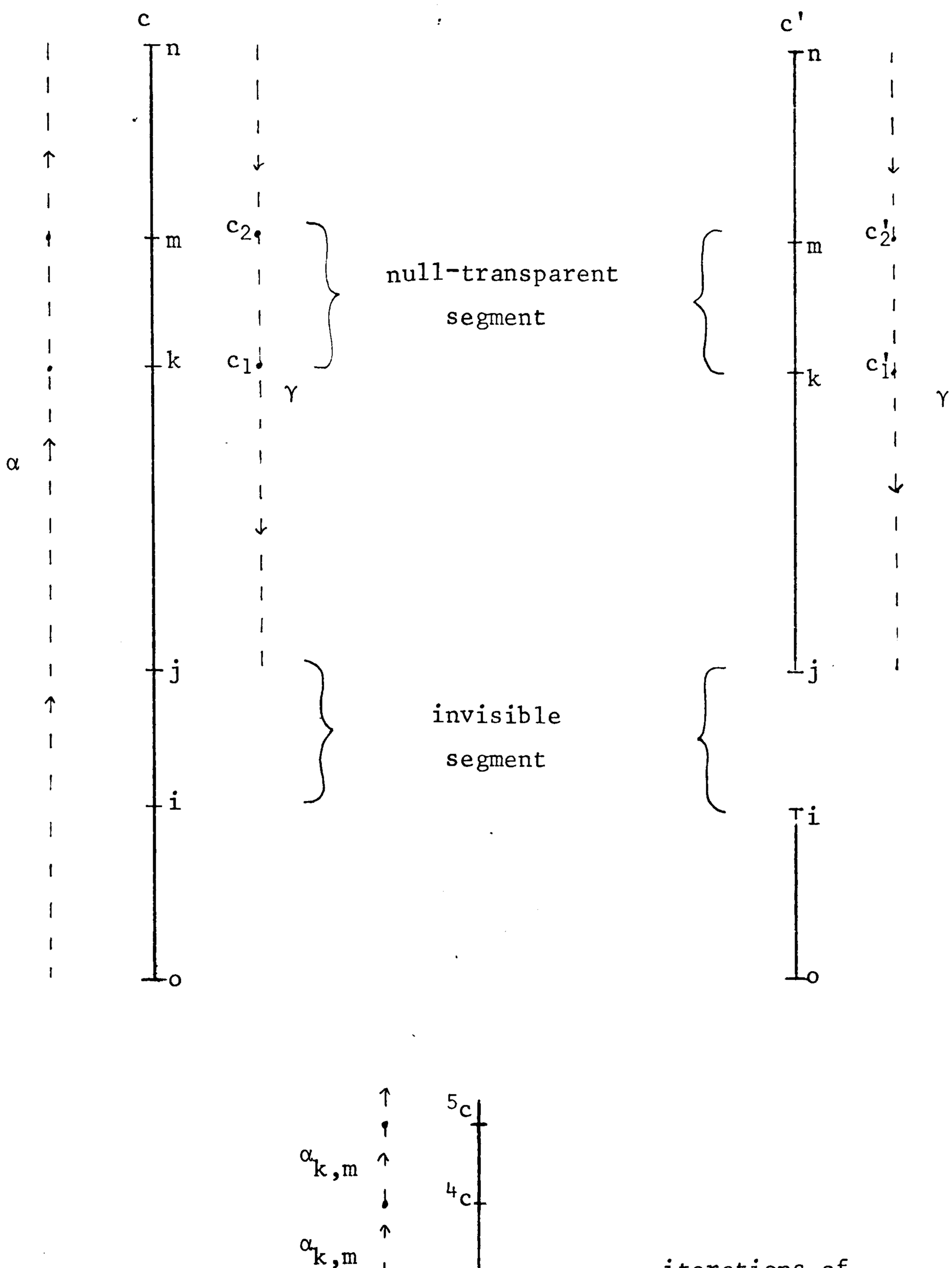



Fig. 6. Constructions in Proof of Theorem 7.1 .

If $\overline{N}$ is large enough, then by Lemma 7.2 (ii), we can find a segment $\omega_{ij}$ in it that is (qq!)-invisible in c w.r.t. N. Furthermore, by the choice of N, the configuration $c' = (s,\omega_{o,i}.\omega_{j,n})$ is reachable from $c_s$ via the input string $\alpha_{o,i}.\alpha_{j,n}$. We shall prove that if L(M) is regular, then $c \equiv c'$. For simplicity we shall not modify the indexing of the segments of the stack in the translation from c to c' (e.g we shall still refer to the top symbol of c' as $\omega_{n-1,n}$).

Suppose $c \not\equiv c'$, and let $\beta$ be the shortest string distinguishing them. Then by the construction of c', for some $\gamma,\eta$ s.t. $\beta = \gamma\eta$,

$$c \vdash (\gamma)\ (s',\omega_{o,j})$$

where this is not a (qq!)-derivation. It follows that we can pick an $N' \subset N$ of at least q! + 1 integers between j and n with the properties that

(a) no segment of $\omega$ induced by N' is popped by an $\varepsilon$-subderivation of $\gamma$, and

(b) the elements of N' all index identical states in the popping sequence for $\gamma$.

We now pick a null-transparent segment $\omega_{k,m}$ induced by N' in $\omega_{j,n}$, as guaranteed by Lemma 7.2 (i). We define some new configurations to do with the popping of this segment in c and c':

$$c \vdash (\gamma_{n,m})\ c_2,\ c_2 \vdash (\gamma_{m,k})\ c_1,\ c' \vdash (\gamma_{n,m})\ c_2',\ c_2' \vdash (\gamma_{m,k})\ c_1'.$$

Since $\gamma\eta$ is a minimal distinguishing string for c, c', $\gamma_{m,j}\eta$ must be a minimal one for $c_2$, $c_2'$, and $\gamma_{k,j}\eta$ for $c_1$, $c_1'$. Since, by the construction of N', $\gamma_{m,k}$ is non-null, $\gamma_{k,j}$ and $\gamma_{m,j}$ must be of different lengths. Hence it is impossible that both $c_1 \equiv c_2$ and $c_1' \equiv c_2'$.

Without loss of generality we shall assume that $c_1 \not\equiv c_2$.
We define the family of configurations $\{{}^{r}c \mid r \geq 0\}$ by $c_s \uparrow (\alpha_{o,k}\alpha^{r}_{k,m})\ {}^{r}c$.
By the choice of N,N' the top parts of the stacks of these will consist of iterations of the null-transparent word. If we let $\xi_r = \alpha_{m,n}\gamma_{n,m}\gamma^{r}_{m,k}$ for r = 0,1,..., then

$$ {}^{r+1}c \xrightarrow{\xi_r} c_2 \quad \text{and} \quad {}^{r}c \xrightarrow{\xi_r} c_1 \quad \text{for } r \geq 0. $$

Thus $c_1 \not\equiv c_2 \Rightarrow {}^{r}c \not\equiv {}^{r+1}c$ for all $r \geq 0$. From this, and the null transparency of $\omega_{k,m}$, we can now deduce that

$$ {}^{r}c \not\equiv {}^{\ell+r}c \quad \text{for all } r \geq 0,\ \ell > 0. $$

For if we assume otherwise for some r, $\ell$, and consider the effect on the configurations ${}^{x\ell+r}c$ of inputs $\alpha^{\ell}_{k,m}$ and $\alpha_{k,m}$, for successive values of x of 0,1,2..., then we are led to deduce that

$$ {}^{x\ell+r}c \equiv {}^{r}c \quad \text{and} \quad {}^{x\ell+r+1}c \equiv {}^{r+1}c \text{ for all } x \geq 0. $$

However, by Lemma 7.1, the shortest distinguishing string for ${}^{r}c$, ${}^{r+1}c$ cannot distinguish ${}^{x\ell+r}c$, ${}^{x\ell+r+1}c$ for sufficiently large x.

To summarise, the assumption $c \not\equiv c'$ has forced us to the conclusion that the configurations $\{{}^{r}c\}$ (or a corresponding set constructed from c',)are all reachable and pairwise inequivalent. Thus for regularity it must be that $c \equiv c'$, which is the result we want.

From Lemma 7.2(ii) we recall that for the above construction it is sufficient that $\bar{N}$ be greater than $(q^2 q!)^{q+1}$. Therefore

$$Y(q,t,h) = qth(q^2q!)^{q+1}$$

is the function we require. This is of order $E^1(\log t + \log h + q^2 \log q)$, and hence $X(q,t,h)$ is of order $E^2(\log t + \log h + q^2 \log q)$. □

Note 1

Meyer and Fischer[21] show that for a certain one-turn dpda with $q = t$ and $h = 2$, an equivalent finite-state automaton requires $E^2(q)$ states. The bound obtained by Stearns for X is of order $E^3(q + \log t)$ for the case of $h = 2$.

Note 2

We notice that the family of configurations $\{^r c\}$ exhibits phenomena strongly reminiscent of the notion of propriety in one-counter automata. In particular, they are either all equivalent, or all pairwise inequivalent.

## 7.4 Bounds for Subfamilies

We now give improved upper bounds for the function X for three restricted families of dpda, and show in each case that the order of the bound is achievable.

For the classes $S_0$ and R, since ε-moves no longer play a part, the above analysis becomes trivial. From the definitions in §7.2 it is immediate in such cases that $N > 1$ is sufficient to induce a null-

transparent segment, while $N > \ell + 1$ guarantees an $\ell$-invisible one. Substituting these in the above argument gives the following:

Corollary 7.1

For the class $S_0$ there is a bound X of order $E^1(ht.\log t)$.

Corollary 7.2

For the class R there is a bound X of order $E^1(hq^2t.\log t)$.

From Lemma 5.4 it can be easily deduced that for C with the restriction $h = 2$, the bound $q.(S(q) + p_3(q))$ suffices, where the dominant factor, $S(q)$, is of order $E^1(\sqrt{q \log q})$. We can obtain a similar bound more directly as follows.

Corollary 7.3

For the class C, with $h = 2$, there is a bound $X \sim q^2.S(q)$, where

$$S(q) = \max \{\ell.c.m \{n_i\} \mid \Sigma n_i = q\}.$$

Proof

Let $\{x_i\}$ be the set of net stack drops due to loops of $\varepsilon$-moves in the diagram for non-empty counter transitions of the

machine M. It follows immediately that any stack segment of length

$$x = \max\{\ell.c.m.\{x_i\}, q\}$$

will be null-transparent, where, moreover, $x \leq S(q)$ since the $\varepsilon$-loops must be disjoint.

We can then obtain the claimed bound from the construction in Theorem 7.1 if we also observe that any j-derivation (w.r.t. the positive integers) popping a segment longer than jq must finish within an $\varepsilon$-loop. □

We now show that the order of each bound is achievable in both of the senses defined in §1.6.

Example 7.1

Let M be a simple machine with $\Gamma = \Gamma_A \cup \Gamma_B$, where $\Gamma_A = \{A_i | 1 \leq i \leq m\}$, $\Gamma_B = \{B_i | 0 \leq i \leq m\}$, with $\Sigma = \{a_i | 1 \leq i \leq m\} \cup \{a_\omega | \omega \in \Gamma_A^n\}$, with $c_s = B_o$, and transitions

$$B_j \xrightarrow{a_\omega} \omega B_{j+1} \qquad 0 \leq j < m$$

$$B_m \xrightarrow{\varepsilon} \Lambda$$

$$A_j \xrightarrow{a_j} \Lambda \qquad 1 \leq j \leq m$$

Clearly the stack has to grow to height $nm + 1$ for $B_m$ to be reached and the turn to occur. But there are $m^{nm}$ distinguishable reachable configurations of this height. Since L(M) is evidently regular,

this shows that X of order $E^1$(ht.log t) is achievable. □

Example 7.2

We construct an M' $\varepsilon$ $R_0$ by generalising the previous example in the following way. We introduce a state set Q = $\{s_o,\ldots s_k\}$, augment $\Gamma$ by a symbol C, replace $c_s$ by $(s_o,B_m)$ and F by $\{(s_o,\Omega)\}$, leave the input alphabet unchanged, and replace the transition rules by

$$(s_i,B_m) \xrightarrow{a} (s_{i+1},\omega B_1) \qquad 0 \le i < k,$$

$$(s_i,B_j) \xrightarrow{a} (s_i,\omega B_{j+1}) \qquad 0 \le i \le k,\ 0 < j < m,$$

$$(s_k,B_m) \xrightarrow{a} (s_o,C\omega B_1)$$

$$(s_k,B_m) \xrightarrow{a_1} (s_k,\Lambda)$$

$$(s_i,A_j) \xrightarrow{a_j} (s_i,\Lambda) \qquad 0 \le i \le k,\ 0 < j \le m,$$

$$(s_i,C) \xrightarrow{\varepsilon} (s_{i-1},\Lambda) \qquad 0 < i \le k.$$

This again is a 1-turn machine. After the turn is made in a derivation, the remaining string is accepted iff it matches the stack contents, and there are exactly k occurrences of C in the stack. Thus the turn has to be made when there are about $k^2mn$ symbols in the stack, since each C is added after successive segments of length kmn. Thus L(M') is regular, and M' has order $E^1(hq^2t.\log t)$ distinguishable configurations. □

Example 7.3.

Let $\{x_1,\ldots x_n\}$ be the partition of q - 1 with the greatest least common multiple. Let M ε C read a string from a* and increment the counter by one for each a, while staying in the starting state. Then if a character i from {1,...n} is read, an ε-loop with $x_i$ states causing a stack drop of $x_i$, is entered. Acceptance occurs iff the contents of the counter at the turn was divisible by $x_n$. Such an M with q states clearly exists, and recognises a regular set. However, after each input of $a^m$ it needs to know $m(\text{mod } y)$ for $y = x_1,..,x_n$. This requires S(q) distinguishable configurations. □

## 7.5 Time Complexity

To test a language accepted by given dpda for regularity we can construct the candidate finite-state automaton specified in the proof of Theorem 7.1, say M', and test for their equivalence. Since this last test can be done in polynomial time (§2.5), and since M' is of $E^2$ size, to show that the regularity test itself takes no more than $E^2$ time, it remains to show that the construction of M' can be done in $E^2$ time.

The states of M' correspond to the no more than $E^2$ configurations of M of height less than Y. To construct M' the only new transitions we need are those that specify for each configuration just larger than Y, the (qq!)-invisible segments with respect to the appropriate index sets (§7.3), that can be removed from their stacks. For each segment

$\omega'$ of $\omega$, we can determine the set of pairs (s,s') with the property that $(s,\omega') \vdash (\alpha)\ (s',\Lambda)$ for $\alpha \neq \varepsilon$, and also those for which $\alpha = \varepsilon$. The possible ways of reaching each point in the stack by (qq!)-derivations can then be deduced, and hence the invisible segments found. To do all this for all the $E^2$ configurations requires only $E^2$ time.

In a similar way, the bounds for R and C give single exponential time tests. However for $R_0$, and hence also for $S_0$, regularity can be tested much more easily, and in only polynomial time. If in $M \in R_0$ an accepting derivation goes through some configuration c with height greater than $hq^2t$, then for some pair of levels repetitions must occur in both the stacking and popping sequences respectively, of the derivations before and after the occurrence of c. This would imply that accepting derivations can go through arbitrarily large configurations, and hence, by the restrictions particular to the class $R_0$, that L(M) is not regular. Since the converse of this is obvious, we conclude that we can test for regularity by testing whether, once a stack level exceeding $hq^2t$ has been reached, any further inputs lead to acceptance. This requires only an emptiness test on a polynomial size machine.

# IV AN APPLICATION TO SCHEMAS

Schemas are direct formalisations of computer programs, and closely resemble them in syntax. Their essential characterisation is that the meaning of the commands is left undefined. The theory of schemas relates the description of such a formalised program to its possible computational effects once interpretations of various kinds are given to its uninterpreted commands.

The relationship between certain schemas which have just one working register, and automata with one-way input tapes, is now well known. Rutledge[22] has established a close connection between one-register flowchart (Ianov) schemas, and finite-state automata. When such schemas are augmented by a pushdown stack, the correspondence transfers to deterministic pushdown automata. This correspondence is such that, as illustrated by Paterson[13], an equivalence test for a subfamily of this class of automata leads directly to a test for strong equivalence (in the sense of Luckham, Park and Paterson[23]) for the corresponding class of schemas. Thus, for example, our result in Chapter 5 implies that strong equivalence is decidable for Ianov schemas with an auxiliary counter. In an analogous way, the regularity problem corresponds to the question of whether a Ianov schema with a pushdown stack is strongly equivalent to one without a stack.

As the translatability of such decidability results from automata to schemas is direct, and depends on a well established technique[13] we shall not pursue these further here.

In contrast, however, the complexity measures for these problems do not necessarily translate directly. For example, a canonical dpda for some such schema may require to be of exponential size in terms of the parameters of the schema description. For this reason we shall investigate the complexity of just one of these problems, for a case which is of special schema theoretic interest.

Monadic functional (deBakker-Scott) schemas[13,24,25] are a formalisation of recursive programs with a single working register. Paterson[26] has shown that for some such schema there does not exist an equivalent flowchart schema with any finite number of registers. This can be interpreted as corroborating our intuitions about the increased power of recursive notation. Other examples are known which can be flowcharted, but not with a single register. We shall now elaborate on these relationships by considering those monadic functional schemas that do have equivalent Ianov schemas, and giving a measure to the substantial succinctness with which some large Ianov schemas can be re-expressed by equivalent small functional ones. In particular, we shall show that some functional schemas require an equivalent flowchart to be of a double exponential ($E^2$) size in terms of its original parameters. Further, we shall show that they may not require larger flowcharts than this order.

Thus we shall be giving a theoretical result to correspond to our intuitive knowledge about the considerable economy of description that can sometimes be gained by recursive notation.

# Chapter 8 MONADIC FUNCTIONAL SCHEMAS

## 8.1 Introduction

In our definitions we shall in the main follow Ashcroft, Manna and Pnueli[24], who give illustrative examples and rather more details.

A dB-S schema has a finite set F, of $t_v$ monadic function variables $\{F_i\}$ (with a distinguished initial function $F_o$), a finite set f, of $t_c$ monadic function constants $\{f_i\}$, a finite set of g monadic predicates $\{p_i\}$, and an individual variable x. A term is a composition of functions applied to x, e.g. $f_1(F_2(f_3(x)))$ which is written as $f_1F_2f_3$ for short, omitting the brackets and the x. A conditional term is any finite expression of the form

$$\text{if } p_i \text{ then } \tau_1 \text{ else } \tau_2$$

where $\tau_1$, $\tau_2$ are terms or conditional terms. The schema itself is specified by a set of function definitions, one for each $F_i$, of the form

$$F_i \Leftarrow \tau$$

where $\tau$ is some term or conditional term. It is useful to reserve the name I for the identity function ($I(x) = x$), and $F_\infty$ for the undefined function ($F_\infty \Leftarrow F_\infty$).

The schema is evaluated in the expected way by starting from $F_o(x)$, and applying the rightmost function variable each time to the argument (i.e. to the string from f*x to its right). The only

further information we need to define completely the course of such an evaluation is to specify the values taken by the predicates each time a conditicnal term has to be evaluated. Thus an interpretation specifies the values taken by the predicates for each argument, and we assume that it does so uniquely. Thus in evaluation steps in which no function constant is applied to the argument, the truth values of the predicates cannot change. In other words, we can regard an interpretation as a function from f*x to $\delta = \{-, +\}^g$, where - and + indicate falsity and truth respectively.

Since each interpretation for a schema uniquely specifies an evaluation, it defines a string (possibly infinite) of the form:

$$\delta_o \; f_{i_o} \; \delta_1 \; f_{i_1} \; \ldots.$$

where $\delta_j$ is the truth vector defined by the interpretation for the argument

$$f_{i_{j-1}} \ldots. \; f_{i_1} \; f_{i_o} \; x.$$

Conversely, each such string describes the step by step evaluation of a function, as well as a set of interpretations.

The schema for a particular interpretation is said to have a defined value if and only if the evaluation terminates producing a term containing no more function variables. This term, consisting of only function constants applied to x, is then the defined value.

## 8.2 Evaluating Pushdown Automata

A dpda M can be easily constructed to evaluate such a functional schema. At each step M keeps all the unevaluated functions (i.e. everything to the left of the argument) in its pushdown store, with the rightmost function variable at the top. It reads input words from $(\delta f)^*$ i.e. strings of alternating truth vectors and function constants.

When there is a function variable at the top of the stack and a truth vector has just been read, the variable is replaced in the stack by the term specified by the corresponding function definition for the truth values read. The vector is remembered in the finite state control, and these replacements continue in accordance with it, until a function constant first appears at the top of the stack. This is checked against the next symbol on the input tape, and rejected via a $F_\infty$ replacement if it does not match. Otherwise, the function constant is popped (and can be regarded as being output and applied to the previous argument,) and the next $\delta_j$ is read. If further function constants then appear at the top of the stack, they are similarly checked against the input tape, with the intermediate truth vectors on the tape being ignored.

The dpda M starts with $F_o$ in its stack, and accepts strings by empty stack. Then clearly any tape accepted by it will specify the value taken by the schema for all interpretations consistent with the tape. Also, the language recognised from an <u>arbitrary</u> configuration

of M will relate in the same way to the values taken by the corresponding term in the schema for different interpretations. From this it follows that two configurations of M are equivalent in the automaton sense if and only if, for any interpretation, the two corresponding terms in the schema either evaluate to the same value, or are both undefined. Thus if M has only a finite number X, of pairwise distinguishable reachable configurations, then the schema can be rewritten as a Ianov schema with X boxes.

By examining the dpda M we have just described, we find that it can be constructed to have the following parameters:

$$t = t_c + t_v,$$

$h = h'$ where $h'$ is the length of the longest term in the function definitions,

$$p = 2^g + t_c,$$

$$q = 2^g + 1.$$

We notice that the only kind of memory capacity that M needs is that of being able to remember the last input character read. The subclass of $D_0$ with this property we shall denote by dB-S. We note that the evaluating machine M is a special form of a dB-S machine, since a subset, f, of its input alphabet need not be distinguished in memory, and also no element of f, regarded now as stack symbols, can be $\varepsilon$-popped.

## 8.3 dB-S Automata

The above mentioned restriction on memory implies directly that for dB-S machines, states cannot change in the course of ε-derivations. A consequence of this is that a simplified analogue of Lemma 7.2 can be derived.

Lemma 8.1

For a configuration c of a dB-S machine, if c has stack $\omega$, and N is an index set of $\overline{N}$ elements all less than $|\omega|$, then

(i) Any segment of $\omega$ is null transparent, and

(ii) If $\overline{N} > 2\ell^t$, where $\ell \geq 2$, then some segment of $\omega$ induced by N is $\ell$-invisible in c.

Proof

(i) Trivial.

(ii) Using the same notation as in the proof of Lemma 7.1,

We apply the induction principle there stated. However, now we choose $P = \Gamma$ and the inductive assertion $A(P_m,N_m)$ to be

$$i,j \in N_m,\ A \in P_m \Rightarrow \text{ A does not occur in } \omega_{i,j}.$$

Then $A(P_o,N_o)$ is trivially true if $P_o = \emptyset$ and $N_o = N$. We then assume that $A(P_m,N_m)$ is true, and that r,r', the smallest and second smallest elements of N, are distinct. If $\omega_{r,r'}$ is

$\ell$-invisible, then the desired result is proved. Otherwise there must be some $\alpha$-derivation that is an $\ell$-derivation rendering it visible i.e.

(a) $c \vdash (\alpha) \quad (s', \omega_{o,r'})$

but (b) not $(s', \omega_{o,r'}) \vdash (\varepsilon) \; (s', \omega_{o,r})$.

We note that this second condition now necessarily implies that $\omega_{r,r'}$ cannot be $\varepsilon$-popped at all from the configuration $(s', \omega_{o,r'})$. Let $N_{m+1}$ be the largest subset of $N_m$ induced by the popping sequence of an $\varepsilon$-subderivation of the $\alpha$ derivation. Then $\overline{N}_{m+1} \geq (N_m - 1)/\ell$. This $\varepsilon$-derivation must eventually terminate when some A' in the stack is reached. Further, this occurrence must be at a higher level in $\omega$ than r, which means that $A' \notin P_m$. Also, A' cannot occur in any segment induced by $N_{m+1}$, for then that occurrence could not have been popped in the same $\varepsilon$-derivation. Thus, if $P_{m+1} = P_m \cup \{A'\}$, then $A(P_{m+1}, N_{m+1})$ must also be true. To ensure that the induction can proceed, if necessary, until $\Gamma$ is exhausted, it is now sufficient that $\overline{N} > 2\ell^t$. □

In the light of this we can rework Theorem 7.1 to obtain the following.

Lemma 8.2

A dB-S dpda recognising a regular set can have no more than $X = E^2(t.\log q + \log h)$ pairwise inequivalent reachable configurations.

Proof

We use the same notation as in the proof of Theorem 7.1. Having chosen N, we now require only a q-invisible segment induced by N. For then, using the shortest distinguishing string $\beta$, assuming that it exists, we can find a null-transparent segment induced by N such that the $\beta$-subderivation that pops it reads some input, but does not cause a net change in state. This is the construction we require, and, from Lemma 8.1, it clearly works for any configuration of height greater than Y, where Y is of order $2hqtq^t$. This gives the required bound for X. □

Note

This shows that the dB-S restriction reduces the q-dependence of the bound X from a double to a single exponential expression.

## 8.4 Bounds on Succinctness

From Lemma 8.2 and the observations of §8.2 we immediately derive

Theorem 8.1

For a dB-S schema with t function symbols, g predicates, and terms of length no more than h in its function definitions, if a Ianov schema strongly equivalent to it (i.e. under all interpretations) exists, then the latter need have no more than order $E^2(tg + \log h)$ boxes. □

It remains to show that an $E^2$ order of size may indeed be necessary.

Theorem 8.2

For each positive integer n, there is a dB-S schema with 3(n+1) function symbols, n + 1 predicate symbols, h = 2, and with total description linear in n, that has strongly equivalent Ianov schemas, but only of size at least $E^2(n)$.

Proof

We use the same idea as Meyer and Fischer[21] use for their corresponding result for dpda, and show that it can be made to work even in this more restricted framework.

For each n we construct the following schema with function variables $\{F_o, F_1^+,\ldots F_n^+, F_1^-,\ldots F_n^-, \underline{F}_1,\ldots \underline{F}_n,\}$, function constants $\{f^+, f^-\}$ and predicates $\{p,p_1,\ldots p_n\}$:

$F_o \Leftarrow$ if $p_1$ then (if p then $F_1^+ F_o f^+$ else $F_1^- F_o f^+$) else if $p_2$...
else if $p_n$ then (if p then $F_n^+ F_o f^+$ else $F_n^- F_o f^+$)
else $F_1^+ f^+$.

$F_i^+ \Leftarrow$ if $p_1$ then I else if $p_2$ . . . . . else if $p_{i-1}$ then I
else if $p_i$ then (if p then $\underline{F}_i f^+$ else I)
else if $p_{i+1}$ then $F_\infty$ . . . . . else if $p_n$ then $F_\infty$
else I.

$F_i^- \Leftarrow$ if $p_1$ then I else if $p_2$ . . . . . else if $p_{i-1}$ then I

else if $p_i$ then (if p then I else $\underline{F}_i$ $f^-$)

else if $p_{i+1}$ then $F_\infty$ . . . . . else if $p_n$ then $F_\infty$

else I.

$\underline{F}_i \Leftarrow$ if $p_1$ then $F_\infty$ else if $p_2$ . . . . . else if $p_i$ then $F_\infty$

else I.

Let $m = \min \{ j \mid p_i([f^+]^j)$ is false for $1 \leq i \leq n \}$ for a particular interpretation. Then the corresponding evaluation will consist of two parts, the first one consisting of m applications of the $F_o$ definition, and the second only of applications of the others. The first part produces an unevaluated term which is a string of length m over the function variables $\{F_1^+,\ldots F_n^+, F_1^-,\ldots F_n^-\}$, while the second monotonically shortens this, unless an $F_\infty$ occurs.

Changing to our dpda terminology, we notice that we have a 1-turn machine which, depending on the input, can read any word from $\{F_1^+,\ldots F_n, F_1^-,\ldots F_n^-\}^*$ into the stack on the upstroke. Denoting by $\underline{k}$ any truth vector such that $p_k$ = true, but $p_j$ = false for $j < k$, we observe that each $F_i^+$ or $F_i^-$ is $\varepsilon$-popped by $\underline{k}$ if $k < i$, and leads directly to rejection if $k > i$. If $k = i$ but the sign of $F_i$ does not match the truth value of p, then the symbol is again $\varepsilon$-popped. However, if there is matching, then the appropriate one of $f^+$ of $f^-$ is "evaluated" and the next truth vector is read from the input tape (i.e. the "interpretation") leaving $\underline{F}_i$ at the top of the stack. The role of $\underline{F}_i$ is to insist that the next vector be $\underline{k}$ for $k > i$. Thus in the downstroke no more than n truth vectors and n function constants can be read, from which we deduce that

there are only a finite number of inequivalent configurations in such a machine.

To see that this number is, however, of order $E^2(n)$, we show that for any of the $2^{2^n}$ subsets of $\{f^+, f^-\}^n$, there is some configuration which can evaluate, depending on the interpretation, to everything in the subset but to nothing in its complement. We observe that any such subset can be represented by a binary tree of depth n with branches marked by a sign from {+, -}. Furthermore any such tree can be represented by a string from $\{F_1^+,\ldots F_n^+, F_1^-,\ldots F_n^-\}^*$ in the following polish notation: Each $F_i^-$ represents a left branch from the (i-1)th to the i-th level of nodes, and $F_i^+$ a similar right branch. The string itself is the expansion of the tree from the 0-th level (the root) by the following recursive process:

$$\langle tree_i \rangle \rightarrow \langle branch_i^- \rangle \langle branch_i^+ \rangle , \quad 0 \leq i < n$$

$$\langle tree_n \rangle \rightarrow \Lambda$$

$$\langle branch_i^- \rangle \rightarrow \Lambda \mid F_{i+1}^+ \langle tree_{i+1} \rangle, \quad 0 \leq i < n$$

$$\langle branch_i^+ \rangle \rightarrow \Lambda \mid F_{i+1}^- \langle tree_{i+1} \rangle, \quad 0 \leq i < n$$

Any string generated from $\langle tree_o \rangle$ defines the set of paths in the tree that go from the root to n-th level nodes. It is easy to verify that the evaluation of such a string by our schema corresponds exactly to tracing a path in the tree it specifies, and that the possible final values such a string can take correspond to the paths in this tree. □

We have therefore established that the succinctness measure we are investigating involves two levels of exponentiation. We note, however, that in the above example g and t cannot be varied independently. Thus to obtain more detailed results for the leading exponent for the different combinations of parameter values, further analysis is necessary. For example, in the schema above, we could have economised on the number of predicates used, at the expense of greater complications in description, to obtain a slightly better result for one particular such case.

# CONCLUSION

We have shown that the various deterministic families of pushdown automata are rich in decidable and potentially decidable properties. In doing so we have also indicated areas outside automata theory to which our results relate.

Several well motivated decision problems have been left unresolved. Moreover, even for those shown to be decidable, the derived procedures usually require at least exponential time. As we do not know of any arguments to show that polynomial time algorithms do not exist for these, important gaps remain here also.

We can, however, single out from among these open questions the ones which appear the most immediate.

Finding an equivalence test for the unrestricted class of deterministic pushdown automata was the primary unachieved goal of our work. Although the existence of one can perhaps be now conjectured with considerable confidence in the light of our results, a proof of this would still be of great interest, for the additional insights it may provide. It appears plausible that techniques related to our parallel and alternate stacking constructions, and our simulations by nondeterministic pushdown automata, will play a part in settling the problem. Our work suggests that the resolution of our conjecture about alternate stacking for the class $R_0$, and the finding of an equivalence test for the class S, may be significant next steps to that end. A positive solution to the latter problem, which we have not investigated,

also appears to be a prerequisite for finding a test of strong equivalence for deBakker-Scott schemas.

For the regularity problem, since we have improved Stearns' test to a near optimal level, to achieve further improvements a new approach is necessary. The relationship we have established between this and a number of containment problems which are currently open, can be interpreted as a two sided challenge. In the absence of a more efficient regularity test, it hints that if these other problems are decidable, then it may be difficult to prove them to be so.

We have not given much attention to the inclusion problem because of the well-known negative results concerning it. However, since we have proved its undecidability for even a very restricted case of $R_0$, a resolution of this problem for the simple machines $S_0$ seems most timely.

Thus our work suggests that there remain numerous distinct features of the structure of these classes of automata yet to be uncovered, and indicates some specific directions along which these might be sought. The rewards of this search will be, we believe, to increase our understanding of these computations, and to render particular instances of them more susceptible to practical analysis.

# REFERENCES


[1] GINSBURG, S. and GREIBACH, S. A.
Deterministic Context-free Languages.
Inf. and Control, 9, 620-648, (1966).

[2] STEARNS, R. E.
A Regularity Test for Pushdown Machines.
Inf. and Control, 11, 323-340, (1967).

[3] HOPCROFT, J. E. and ULLMAN, J. D.
Formal Languages and their Relation to Automata.
Addison-Wesley, Reading, Mass., (1969).

[4] GINSBURG, S. and SPANIER, E. H.
Bounded Algol-like Languages.
Trans. Amer. Math Soc., 113, 333-368, (1964)

[5] McNAUGHTON, R.
Parenthesis Grammars.
JACM, 14, 490-500, (1967).

[6] RABIN, M.O. and SCOTT, D.
Finite Automata and their Decision Problems.
IBM J. Res. 3 : 2, 115-125, (1959).

[7] BIRD, M.R.
The Equivalence Problem for Deterministic Two-tape Automata.
JCSS 7, 218-236, (1973).

[8] ROSENKRANTZ, D. J. and STEARNS, R. E.
Properties of Deterministic Top-Down Grammars.
Inf. and Control. 17, 226-255, (1970).

[9] KORENJAK, A. J. and HOPCROFT, J. E.
Simple Deterministic Languages.
IEEE 7th Symp. on Switching and Automata Theory,
Berkeley, California, (1966).

[10] KARP, R.
Reducibility Among Combinatorial Problems,
in Complexity of Computer Computations (R. E. Miller
and J. W. Thatcher, eds.), Plenum Press, N.Y. (1972).

[11] KNUTH, R.
On the translation of Languages from left to right.
Inf. and Control, 8, 607-639, (1965).

[12] HARRISON, M. A. and HAVEL, I. M.
On a Family of Deterministic Grammars,
in Automata, Languages and Programming (M. Nivat, ed.),
North-Holland, (1973).
[Also three more detailed reports; Department of
Computer Science, University of California, Berkeley.]

[13] PATERSON, M. S.
Decision Problems in Computational Models.
Proc. of ACM Symp. on Proving Assertions about Programs,
Las Cruces, New Mexico, (1972).

[14] CHOMSKY, N.
Context-free Grammars and Pushdown Storage.
Quart. Prog. Rept. No. 65, MIT Res. Lab. Elect., 187-194,
(1962).

[15] GREIBACH, S. A.
A New Normal Form Theorem for Context-free Phrase Structure
Grammars.
JACM, 12, 42-52, (1965).

[16] MINSKY, M. L.
Computation: Finite and Infinite Machines.
Prentice-Hall, New Jersey, (1967).

[17] BAR-HILLEL, Y., PERLES, M. and SHAMIR, R.
On Formal Properties of Simple Phrase Structure Grammars,
in Y. Bar-Hillel, Language and Information,
Addison-Wesley, Reading, Mass., (1964).

[18] FRIEDMAN, E. R.
The Inclusion Problem for Monadic Recursion Schemes.
Report, Center for Research in Computing Technology,
Harvard University, (1973).

[19] GINSBURG, S. and SPANIER, E.
Finite-turn Pushdown Automata.
SIAM J. on Control 4, 423-434, (1966).

[20] VALIANT, L. G. and PATERSON, M. S.
Deterministic One-Counter Automata.
Proc. GI Conf. on Automata Theory and Formal Languages,
Bonn, Germany, (1973).

[21] MEYER, A. R. and FISCHER, M. J.
Economy of Description by Automata, Grammars, and Formal Systems.
IEEE 12th Symp. on Switching and Automata Theory, (1971).

[22] RUTLEDGE, J. D.
On Ianov's Program Schemata.
JACM 11, 1 - 9, (1964).

[23] LUCKHAM, D. C., PARK, D. M. R. and PATERSON, M. S.
On Formalised Computer Programs.
JCSS 4, 220-249, (1970).

[24] ASHCROFT, E., MANNA, Z. and PNUELI, A.
Decidable Properties of Monadic Functional Schemas.
Int. Symp. on Theory of Machines and Computation, Haifa, Israel, (1971).

[25] DE BAKKER, J. W. and SCOTT, D.
A Theory of Programs.
Memo., 1969.

[26] PATERSON, M. S.
A Simple Monadic Recursive Schema which is not Equivalent to any Program Schema.
Memo., 1970.