SUCCINCTNESS OF DESCRIPTIONS OF CONTEXT-FREE, REGULAR, AND FINITE LANGUAGES

by

Erik Meineche Schmidt

DAIMI PB-84
January 1978

Institute of Mathematics University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade - 8000 Aarhus C - Denmark
Phone 06-12 83 55

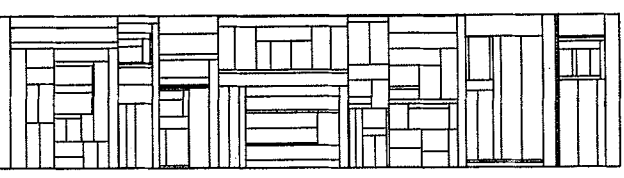


TABLE OF CONTENTS

ABSTRACT

This thesis analyzes the descriptional power of finite automata, regular expressions, pushdown automata and certain generalized models of macro grammars. Descriptional power is measured in essentially two different ways.

In the first the results are relative in the sense that the power of one class of automata or grammars is measured relative to that of another class. The emphasis in this area is on ambiguity in finite automata and pushdown automata. It is shown that ambiguous nondeterminism allows more succinct definitions than unambiguous nondeterminism which in turn allows more succinct definitions than determinism. This is true for both pushdown automata where the succinctness gain is nonrecursive, and for finite automata where the gain is nonpolynomial.

The other measuring method uses complexity theory, in particular the notion of a set being hard for a complexity class. The elements of a very hard set are considered very succinct encodings of instances of some problem. Here it is shown that the inequivalence problem for OI macro grammars generating finite languages is hard for nondeterministic double exponential time, and the complexity of the same problem for (frontiers of) term languages of higher type is analyzed.

Finally ambiguity in regular expressions is considered and it is shown that the "nonemptiness of complement" problem for unambiguous expressions is in NP and thus is easier than for ambiguous expressions (unless NP is equal to PSPACE).

ACKNOWLEDGEMENTS

First of all I want to thank my wife Else for her contribution to this "Ph.D.-project" without which it would never have succeeded.

Secondly I wish to thank my advisor Juris Hartmanis for his very encouraging and stimulating attitude towards my activities at Cornell, in particular the work reported in this thesis.

I'm also grateful for numerous inspiring discussions with Leonard Berman, Jim Donahue, Steve Fortune, John Hopcroft and Robert Constable. In particular Steve Fortune has been a very willing and able partner in discussions as well as in ping-pong.

The excellent typing of the thesis is the work of Karen Møller.

Finally, thanks to "Thanks to Scandinavia, Inc." and to Aarhus University for supporting me financially.

1. INTRODUCTION

This work deals with the descriptive power of language defining mechanisms when used in the definition of finite, regular and context-free languages. The power of a mechanism such as a grammar or an automaton will be measured in terms of how succinctly it can describe some fairly simple sets. Assume that we have a way of measuring the size of some type of automata or grammars. The problem of asserting objectively how powerful they are as descriptors is approached in two different ways.

In the first, which might be called the method of relative succinctness, the power of a class of descriptors is characterized by way of comparison with some other class of descriptors. The following example (due to Moore [28] and also to Meyer and Fisher [26]) illustrates the method. Consider the classes of nondeterministic and deterministic finite automata, and let the size of an automaton be measured by its number of states. Since a deterministic finite automaton (dfa) is by definition also a nondeterministic finite automaton (nfa), every set defined by a dfa is also defined by a nfa of the same size. The converse of this does not hold. This is because there exists for each positive integer n a regular language L_n which is accepted by a nfa with n states but not by any dfa with less than 2^n states. Thus, nondeterministic finite automata allow in general more succinct representations of regular languages than deterministic finite automata. In a case like this we will say that there is succinctness between the class of dfa's (DFA) and the class of nfa's (NFA) and if we want to be more precise we say that there is exponential succinctness (or 2^n-succinctness) between DFA and NFA. Irrespective of how we say it, the important point is that we have measured the power of nfa's relative to that of dfa's.

Now to the second way of measuring descriptive power. In this method, which might be called the method of absolute succinctness, we measure succinctness in terms of standard concepts from the theory of computational complexity. A typical way of characterizing a problem or a set in complexity theory is to show that it is hard for some complexity class. A set being hard for a sufficiently "difficult" class is then interpreted as evidence that it consists of very succinct encodings of instances of a problem. Consider as an example the problem of deciding whether a regular expression generates all words over its terminal alphabet. This problem is known to be hard (actually complete) for the class of problems which are solvable by polynomially space bounded Turing Machines (PSPACE). Since PSPACE is in practice considered a "difficult" class, the language of regular expressions provides a succinct way of defining large nontrivial sets.

This thesis contains relative as well as absolute results in the sense just mentioned. In the area of relative succinctness most of what is already known are results about the succinctness relations between determinism and nondeterminism in finite automata and pushdown automata. We have already seen that in the case of finite automata, addition of nondeterminism to the device increases its descriptional power. This is also true for pushdown automata (pda's) and in this case the succinctness gain is even more dramatic. It can be shown (see [11]) that there is no recursive function f such that size $(P_d) \leq f(\text{size}(P_n))$ where P_d and P_n are arbitrary deterministic and nondeterministic pda's, accepting the same language.
It is interesting to note that in both of these two theorems the succinct automata are not only nondeterministic but also ambiguous. Hence a natural question is whether ambiguity plays a role in this context, and the answer is affirmative, i.e. there is succinctness between deterministic and unambiguous machines as well as between unambiguous and ambiguous ones.

Ambiguity has been studied quite intensively in connection with context-free languages and it is well known that the classes of ambiguous, unambiguous and deterministic languages are all different. An example of an (inherently) ambiguous context-free language is the language $\{a^i b^j c^k \mid i, j, k \geq 1, i = j \vee j = k\}$ (see [29]) and an example of an unambiguous language which is not deterministic is $\{ww^R \mid w \in \{a,b\}^*\}$ (see [12]). Ambiguity in finite automata has not received the same attention, and the reason is probably that since the nondeterministic, unambiguous and ambiguous finite automaton languages are all the same, the concept is not interesting from the point of view of "traditional" language theory. "Succinctnesswise", however, ambiguity in finite automata or regular expressions seems to be of interest. One reason is that regular expressions are heavily used in connection with specification of the syntax of programming languages. An especially interesting application of regular expressions in this connection is their use in socalled extended context-free grammars (see [23]), where it is allowed to use regular expressions over terminals and nonterminals as righthandsides of productions. Since there are (normally) semantic actions associated with the process of recognizing the language generated by such a grammar, it is important that the expressions are unambiguous (requiring determinism seems to be too restrictive in this connection). Furthermore, if extended context-free grammars are used as inputs to a parser generator or maybe even a compiler compiler, it might be of interest to know how hard it is to determine if a regular expression is ambiguous, and if so, how big the smallest equivalent unambiguous expression is. The application of regular expressions in this connection is clearly an example where the size of the expression is a very real complexity measure.

The nonrecursive succinctness between dpda's and npda's mentioned

above was proved by Geller, Hunt III, Szymanski and Ullman in [11]. It is also a corollary of Valiant's result from [34] that there is nonrecursive succinctness between unambiguous and deterministic pda's. Here we show that the same type of succinctness is found between unambiguous pda's and arbitrary nondeterministic ones (this result has also been published by Tom Szymanski and the author in [31]). We also show that ambiguity plays the same role in finite automata as in pda's provided the word "nonrecursive" is replaces by "nonpolynomial", i.e. there is no polynomial p such that $size(M_1) \leq p(size(M_2))$ where M_1 and M_2 are arbitrary deterministic (unambiguous) and unambiguous (nondeterministic) finite automata accepting the same language. Since every n-state nondeterministic automaton has a deterministic equivalent with no more than 2^n states, nonpolynomial (or exponential) succinctness is also the best we can hope for.

In the area of absolute succinctness we shall concentrate on the problem of deciding whether a grammar or expression generates all strings over its terminal alphabet. After discussing the (known) decidability results for Σ^*-ness of context-free grammars as well as the Turing Degree of the regularity problem for pda's, the Σ^*-ness problem for finite automata and regular expressions is considered. Again emphasis is on the role played by ambiguity and it is shown that for unambiguous regular expressions, (non)Σ^*-ness is decidable in nondeterministic polynomial time (NP). Hence, this problem is easier than the general problem - unless, of course, NP happens to be equal to PSPACE.

The fact that Σ^*-ness for context-free grammars in undecidable suggests investigating restrictions on the grammar or the language under which the problem becomes decidable. It is well known that Σ^*-ness remains undecidable even under the assumption that the language is cofinite and it

is clear that the problem becomes trivial if the language is assumed to be finite. What is not so trivial, however, is to determine whether a context-free grammar, which is known to generate a finite language, generates all strings of length less than or equal to some number. In view of the connection between complexity and succinctness mentioned above, this can be reformulated as asking how succinctly context-free grammars can describe finite sets. It was proved in [20] that due to the "squaring" structure in context-free grammars (the derivation tree is a complete binary tree) this problem, which might be called the Σ^k-ness problem, is hard for nondeterministic exponential time. This result has an application in the theory of program logics ([5, 10]) in the following way. A context-free grammar corresponds in a natural way to a nondeterministic program scheme with parameterless procedures. The scheme computes a set of uninterpreted values which is equal to the language generated by the grammar (see [15]). In a program logic with the proper type of nondeterministic primitives it is easy to assert that a scheme does not compute all possible values, hence the complexity result for context-free grammars generating finite languages shows that a decision procedure for such a logic over finite interpretations must require at least nondeterministic exponential time. †

It is natural to ask what happens when more general models of program schemes are considered, and a natural extension is to allow procedures with parameters. If, as a first step, the parameters are allowed to range over sets of strings, the proper generalization of a context-free grammar generating a finite language is an OI macro grammar (M. Fisher [9]) generating a finite language. Now just as a context-free grammar can be viewed as an equation to be solved in the domain of sets of strings (see

† This was pointed out to the author by R. Constable.

[8]) an OI macro grammar can also be viewed as an equation to be solved in some domain, and the proper choice is a domain consisting of functions from sets of strings to sets of strings (where a set of strings is interpreted as a constant function) (see [6]). We show that the problem of deciding whether a "finite" macro grammar does not generate all strings of length less than or equal to some number is hard for nondeterministic double exponential time and hence that the method of defining sets as (constant) functions computed by nondeterministic programs with parameters allows very succinct representations of finite sets. If the type of parameters allowed in the procedures are generalized to functions of functions of sets and functions of functions of ... etc., the result is a hierarchy of sets which is called the OI-hierarchy in [8]. In [8] it is also shown that this family of sets can be obtained as homomorphic images of solutions to context-free like equations in an appropriate (many sorted) algebra. We use this characterization in a discussion of the complexity of determining whether a "level i" grammar does not generate all strings of length less than or equal to some number.

This thesis is organized in 5 chapters of which this is the first. Chapter 2 contains preliminary definitions from automata theory and complexity theory. The succinctness results are presented in Chapter 3, and Chapter 4 contains the material on complexity. Finally Chapter 5 consists of the complexity results for grammars generating finite languages.

2. PRELIMINARIES

The reader is assumed to be familiar with standard concepts from automata theory and complexity theory and is referred to the books by Aho and Ullman [4] and Aho, Hopcroft and Ullman [3] for definitions not explicitly presented in this thesis.

We shall use standard notation and terminology. λ is the empty word and $|x|$ is the length of the string x. We now define finite automata, pushdown automata, Turing Machines and regular expressions.

A nondeterministic finite automaton, nfa, is a system $M=(Q,\Sigma,\delta,q_0,F)$ consisting of states, input alphabet, transition function (from $Q \times \Sigma$ to subsets of Q), startstate and final states. M is in configuration $(q,x) \in Q\times \Sigma^*$ if it is in state q and x is the part of the input that remains to be read. $\vdash$ is the usual "transition relation" between configurations, i.e. $(q_1,ax) \vdash (q_2,x)$ iff $q_2 \in \delta(q_1,a)$. The language accepted by M is the set $T(M) = \{x \in \Sigma^* \mid \exists\ q \in F: (q_0,x) \vdash^* (q,\lambda)\}$ where $\vdash^*$ is the reflexive and transitive closure of $\vdash$. M is said to be an unambiguous finite automaton, ufa, if no word is accepted in more than one way, i.e. for no $x = a_1a_2\ldots a_n$ is there more than one sequence of states $q_0, q_1, \ldots, q_n$ such that $q_n \in F$ and $(q_{i-1}, a_i\ldots a_n) \vdash (q_i, a_{i+1}\ldots a_n)$ for $1 \leq i \leq n$. M is a deterministic finite automaton, dfa, if for all $q \in Q$ and $a \in \Sigma$, $\delta(q,a)$ contains at most one element.

A nondeterministic pushdown automaton, npda, is a system $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ where Q, Σ, q_0 and F are as above. Γ is the alphabet of stacksymbols, Z_0 the stackbottom, and the transition function δ maps $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$. M is in configuration $(q,x,\alpha) \in Q \times \Sigma^* \times \Gamma^*$ if it is in state q, x is the part of the input that remains to be read and α is the content of the stack. The relation $\vdash$ between

configurations is defined by $(q_1, ax, A\alpha) \vdash (q_2, x, \beta\alpha)$ if $(q_2, \beta) \in \delta(q, a, A)$. $\vdash^*$ is the reflexive and transitive closure of $\vdash$ and the language accepted by M is the set $T(M) = \{x \in \Sigma^* \mid \exists\, q \in F,\ \alpha \in \Gamma^*: (q_0, x, Z_0) \vdash^* (q, \lambda, \alpha)\}$. M is unambiguous, an upda, if there is no input x which is accepted in more than one way, i.e. there is at most one sequence of configurations $c_0, c_1, \ldots, c_n$ such that $c_0 = (q_0, x, Z_0)$, $c_i \vdash c_{i+1}$ for $0 \leq i < n$ and $c_n = (q, \lambda, \alpha)$ for some $q \in F$ and $\alpha \in \Gamma^*$. M is a deterministic pushdown automaton, dpda, if for all $q \in Q$, $a \in \Sigma \cup \{\lambda\}$ and $Z \in \Gamma$, $\delta(q, a, Z)$ contains at most one element and furthermore M never has a choice between making a λ-move and reading an input symbol.

A nondeterministic single tape Turing Machine is a system $M = (Q, \Gamma, \delta, q_0, F)$ with states, tape alphabet including the blank $\not b$, transition function, startstate and final states. δ maps $Q \times \Gamma$ to subsets of $Q \times \Gamma \times \{-1, 0, 1\}$ and is assumed to be undefined on arguments where the statecomponent is in F. A configuration of M is an element of $\Gamma^*(Q \times \Gamma)\Gamma^*$ representing in the usual way the nonblank portion of the tape, the state, and the symbol under the read-write head. Configuration c_2 follows from configuration c_1, $c_1 \vdash c_2$, if c_2 is obtained from c_1 by a single application of δ. The language accepted by M is the set $T(M) = \{a_1 \ldots a_n \in \{\Gamma - \not b\}^* \mid \exists\, q \in F,\ a \in \Gamma,\ \alpha, \beta \in \Gamma^*: (q_0, a_1)a_2 \ldots a_n \vdash^* \alpha(q, a)\beta\}$ where $\vdash^*$ is the reflexive and transitive closure of $\vdash$. A Turing Machine is deterministic if $\delta(q, a)$ never contains more than one element. Multitape Turing Machines are obtained in the usual way.

Let Σ be an alphabet. The regular expressions over Σ, $REXP_\Sigma$, is the smallest set satisfying

a) $\emptyset$, λ, and each $a \in \Sigma$ are in $REXP_\Sigma$

b) if P and Q are in $REXP_\Sigma$ then so are $(P+Q)$, $(P \cdot Q)$ and (P^*).

The language generated by a regular expression is defined in the obvious way.

Next we define some of the important concepts from complexity theory.

Let M be a multitape Turing Machine. The amount of time (amount of space) used on x is the number of steps in the shortest accepting computation (the smallest number of tape cells used in an accepting computation) if x is accepted, and the number of steps in the longest computation (the largest amount of tape cells used in any computation) if x is rejected. If the computation on x does not terminate, both amounts are undefined. M runs in time (space) t(n) for some function t(n) if for all $n \geq 0$, for every x of length n, M uses no more than t(n) time (space) on x.

Time and space classes are defined as follows.

$$\text{DTIME}(t(n)) = \{ A \mid A \text{ is accepted by a deterministic TM which runs in time } t(n) \}$$

$$\text{NTIME}(t(n)) = \{ A \mid A \text{ is accepted by a nondeterministic TM which runs in time } t(n) \}$$

$$\text{DSPACE}(t(n)) = \{ A \mid A \text{ is accepted by a deterministic TM which runs in space } t(n) \}$$

$$\text{NSPACE}(t(n)) = \{ A \mid A \text{ is accepted by a nondeterministic TM which runs in space } t(n) \}$$

Now some of the best known complexity classes are

$$P = \bigcup_{i=0}^{\infty} \text{DTIME}(n^i)$$

$$NP = \bigcup_{i=0}^{\infty} \text{NTIME}(n^i)$$

$$\text{PSPACE} = \bigcup_{i=0}^{\infty} \text{DSPACE}(n^i)$$

$$\text{EXPTIME} = \bigcup_{i=0}^{\infty} \text{DTIME}(2^{n^i})$$

$$\text{NEXPTIME} = \bigcup_{i=0}^{\infty} \text{NTIME}(2^{n^i})$$

$$\text{EXPSPACE} = \bigcup_{i=0}^{\infty} \text{DSPACE}(2^{n^i})$$

In these cases where the resource bounds are all at least linear, the amount of storage taken up by the input is also charged to the computation. If however, the space bound is sublinear we have to do something else. Then we assume that the input is placed on a special read-only input tape and only the tape cells used on the worktapes are counted. Then we have

$$\text{LOGSPACE} = \bigcup_{i=0}^{\infty} \text{DSPACE}(i \cdot \log(n))$$

$$\text{NLOGSPACE} = \bigcup_{i=0}^{\infty} \text{NSPACE}(i \cdot \log(n))$$

Let Σ and Ω be two alphabets. A function $f : \Sigma^* \to \Omega^*$ is said to be ptime (logspace) computable if there is a deterministic multitape Turing Machine with a read-only input tape and a write-only output tape which runs in polynomial time (logarithmic space) and which, given x on the input tape, halts with $f(x)$ on the output tape. A set $A \subseteq \Sigma^*$ is p-reducible (log-reducible) to a set $B \subseteq \Omega^*$, $A \leq_p B$ ($A \leq_{\log} B$), if there is a ptime (logspace) computable function $f : \Sigma^* \to \Omega^*$ such that $f(x) \in B$ iff $x \in A$. A set A is hard for a class $\mathcal{C}$ under $\leq_p$ ($\leq_{\log}$) if every set in $\mathcal{C}$ is p-reducible (log-reducible) to A. It is complete for $\mathcal{C}$ if it also belongs to $\mathcal{C}$.

We shall also use the following encoding of Turing Machine computations. Let $M = (Q, \Gamma, \delta, q_0, F)$ be a nondeterministic single tape machine and let $\#$ be a symbol not in $\Gamma \cup (Q \times \Gamma)$. For any input $x = a_1 \ldots a_n$, the set of

valid computations of M on x is the set

$$\text{Valcomps}(M,x) = \{\#\, z_0 \#\, z_1 \#\, \ldots \#\, z_n \# \mid \text{all } z_i\text{'s are configurations, } z_0 \text{ is the initial configuration, } z_n \text{ is a final configuration and } z_i \vdash z_{i+1} \text{ for } 0 \leq i < n\}$$

The complement of Valcomps (M, x) w.r.t. $\{\Gamma \cup (Q \times \Gamma) \cup \{\#\}\}^*$ is called Invalcomps (M, x). If M is a tape bounded TM we use a sequence of padded configurations of equal length in the definition of Valcomps. That is, if M uses space t(n) ($t(n) \geq n$) and x in an input of length n, then each z_i in Valcomps (M, x) will be of length t(n). If z_i is "not long enough" we use the blank ∅ as padding symbol. In case M uses less than linear space we do the following. A configuration of M on x will be a pair (i, z) where i is a binary number whose value is the position of the input head and z is the configuration of the worktape including the machine state. Valcomps (M, x) and Invalcomps (M, x) are then defined as before.

Finally we consider the concept of Turing Degree. Let B be a set. A Turing Machine with oracle B is a TM which in addition to its normal operations can ask the question "Is $x \in B$" for any string x. A set A is Turing reducible to a set B, $A \leq_T B$, if A is accepted by an always halting TM with oracle B. A and B are Turing equivalent, $A \equiv_T B$, if $A \leq_T B$ and $B \leq_T A$. Let A be a set. The jump of A, A^∇, is the set $\{i \mid M_i^A \text{ is defined on input } i\}$ where M_i^A is the i'th Turing Machine with oracle A. Let $[A]_T$ be A's equivalence class under $\equiv_T$. Then $[\emptyset]_T$ is called Turing Degree 0, $[\emptyset^\nabla]_T$ is Turing Degree 1, $[\emptyset^{\nabla\nabla}]_T$ is Turing Degree 2 etc. We recall that The Halting Problem is in Turing Degree 1 and that Finiteness of Turing Machines is in Turing Degree 2.

3. SUCCINCTNESS

In this chapter we consider succinctness between different classes of descriptors. We survey a number of known results and present new results mainly about ambiguity in finite automata and pushdown automata.

The notion of succinctness between two classes of descriptors is made precise in the following way. Let $\mathbb{m}_1$ and $\mathbb{m}_2$ be classes of descriptors generating the families of languages $\mathcal{L}(\mathbb{m}_1)$ and $\mathcal{L}(\mathbb{m}_2)$. Let $\underline{size}_1$ and $\underline{size}_2$ be functions mapping $\mathbb{m}_1$ and $\mathbb{m}_2$ to the nonnegative integers. By $\mathbb{m}_1 \xrightarrow{f(n)} \mathbb{m}_2$ we denote $\underline{f(n)\text{-succinctness}}$ between $\mathbb{m}_1$ and $\mathbb{m}_2$, meaning that there are languages defined by small $\mathbb{m}_1$-descriptors which require large $\mathbb{m}_2$-descriptors. The difference between small and large is determined by f(n). Formally $\mathbb{m}_1 \xrightarrow{f(n)} \mathbb{m}_2$ means that there is a family of languages $\{L(n) \mid n \in N\}$ in $\mathcal{L}(\mathbb{m}_1) \cap \mathcal{L}(\mathbb{m}_2)$ defined by a family of $\mathbb{m}_1$-descriptors $\{m_1(n) \mid n \in N\}$ such that for any family of $\mathbb{m}_2$-descriptors, $\{m_2(n) \mid n \in N\}$, defining the same languages, $\underline{size}_2(m_2(n)) \geq f(\underline{size}_1(m_1(n)))$ for almost all n.

We illustrate the use of the terminology with a result from [26]. Let NFA (DFA) be the class of nondeterministic (deterministic) finite automata and let the size of a finite automaton be its number of states.

Theorem 3.1 (Meyer and Fisher)

$$\text{NFA} \xrightarrow{2^n} \text{DFA}$$

Proof

Consider the family of regular languages $\{L(n) \mid n \in N\}$ where L(n) is the set accepted by the nondeterministic automaton $M_1(n) = (\{0, 1, \ldots, n-1\}, \{0, 1\}, \delta, 0, \{0\})$ whose transition function is defined by

$$\delta(i,1) = \{(i+1) \bmod n\} \quad \text{for } 0 \leq i \leq n-1$$

$$\delta(i,0) = \{i,0\} \quad \text{for } 1 \leq i \leq n-1$$

Consider the family of deterministic machines $\{M_2(n) \mid n \in N\}$ obtained by the usual subset construction from $\{M_1(n) \mid n \in N\}$. It is easy to show that all states in $M_2(n)$ are reachable and that no two of them can be identified, i.e. that $M_2(n)$ is reduced. But then any deterministic machine accepting $L(n)$ must have at least 2^n states and since $2^n = 2^{\text{size}(M_1(n))}$ the theorem is proved. □

Note that, by definition, f(n)-succinctness provides only a lower bound on the succinctness gap between two families of descriptors. It does not say anything about how good this lower bound is. In Theorem 3.1 the lower bound is actually optimal. This is because every nondeterministic finite automaton with n states has a deterministic equivalent with no more than 2^n states.

Theorem 3.1 shows that the addition of nondeterminism to a finite control results in a more succinct description mechanism. This is also true for pda's where as shown in [11] nondeterministic machines are so much more powerful than deterministic ones that there is no recursive function bounding the succinctness gap. Before we present the proof of this result we introduce the notation for nonrecursive succinctness and define some size measures.

Nonrecursive succinctness between the classes $\mathbb{m}_1$ and $\mathbb{m}_2$ is denoted by $\mathbb{m}_1 \xrightarrow{\text{nonrec}} \mathbb{m}_2$. It means that there is no recursive upper bound on the succinctness between $\mathbb{m}_1$ and $\mathbb{m}_2$ or, equivalently, that for any recursive function f(n) we have $\mathbb{m}_1 \xrightarrow{f(n)} \mathbb{m}_2$.

Next we determine the size measures to be used throughout the rest of this thesis. We shall in all cases choose the most natural size measure, namely the number of symbols used to specify the descriptor whose size we are measuring. This is made precise in the following definition.

Definition 3.2 Let FA, PDA, CFG, TM and REXP be the classes of finite automata, pushdown automata, context-free grammars, Turing Machines and regular expressions. The size measures on these classes, which all will be denoted by size(•)are defined as follows.

a) For M in FA, PDA and TM, size(M) is the total number of occurrences of symbols in the definition of M as an n-tuple.

b) If G is in CFG then size(G) is the total number of occurrences of symbols in G's definition as a 4-tuple.

c) If $R \in$ REXP then size(R) is the length of the expression. □

Now we present the succinctness result for dpda's and npda's.

Theorem 3.3 (Geller, Hunt III, Szymanski and Ullman)

Let DPDA (NPDA) be the classes of deterministic (nondeterministic) pda's. Then

$$\text{NPDA} \xrightarrow{\text{nonrec}} \text{DPDA}$$

Proof

Let f(n) be an arbitrary recursive function. We have to show that $\text{NPDA} \xrightarrow{f(n)} \text{DPDA}$.

Let $\{ M(n) \mid n \in N \}$ be a family of Turing Machines which all halt when started on blank tape and let $T(n)$ be the number of steps executed by $M(n)$. Consider the word $z(n) = \# z_0 \# z_1 \# z_2 \# \ldots \# z_{T(n)} \#$ where $\# z_0 \# z_1^R \# z_2 \# z_3^R \# \ldots \# z_{T(n)} \#$ is the valid computation of $M(n)$ on blank tape, i.e. $z(n)$ is the valid computation in which every second configuration is reversed. It is easy to see that the complement of $z(n)$ is a context-free language which is accepted by a npda whose size depends only on the size of $M(n)$. Let $P(n)$ be such a npda and let g be a recursive function such that for all n, $\underline{size}(P(n)) \leq g(\underline{size}(M(n)))$. Let Σ be the symbols occurring in $z(n)$. Since $\Sigma^* - \{ z(n) \}$ is cofinite it can also be accepted by a dpda $D(n)$. Now, dpda's have the following two properties:

a) given a dpda P_1 there is another dpda P_2 accepting the complement of $T(P_1)$ such that $\underline{size}(P_2) \leq (\underline{size}(P_1))^2$ (see [4]);

b) they have pumping, i.e. if a dpda accepts a "long" string (compared with its size) then it accepts infinitely many strings.

Combining these two properties we get the existence of a recursive function h such that for all dpda's P, if x is not accepted by P and $|x| \geq h(\underline{size}(P))$ then x can be "pumped" and there are infinitely many words not accepted by P. Now, since $z(n)$ is the only word not accepted by $D(n)$ it follows that $|z(n)| \leq h(\underline{size}(D(n)))$ and since $z(n)$ is at least as long as $T(n)$, which is $M(n)$'s running time, we have $T(n) \leq |z(n)| \leq h(\underline{size}(D(n)))$.

Assume that the family $\{M(n) \mid n \in N\}$ has been chosen in such a way that for almost all n, $\underline{size}(M(n)) \leq g^{-1} \circ f^{-1} \circ h^{-1}(T(n))$ - where we have assumed without loss of generality that h, g and f are all monotonically increasing . Then we have the following relationship between the nondeter-

ministic pda's $\{P(n) \mid n \in N\}$ and the deterministic ones $\{D(n) \mid n \in N\}$.

$$\begin{aligned} \underline{size}(P(n)) &\leq g(\underline{size}(M(n))) \\ &\leq g(g^{-1} \circ f^{-1} \circ h^{-1}(T(n))) \\ &= f^{-1} \circ h^{-1}(T(n)) \\ &\leq f^{-1} \circ h^{-1}(h(\underline{size}(D(n)))) \\ &= f^{-1}(\underline{size}(D(n))) \end{aligned}$$

Hence for almost all n, $f(\underline{size}(P(n))) \leq \underline{size}(D(n))$ and we have proved f(n)-succinctness on the family of languages $\{\Sigma^* - \{z(n)\} \mid n \in N\}$. All that remains to be shown is that it is possible to choose the machines M(n) such that $\underline{size}(M(n)) \leq g^{-1} \circ f^{-1} \circ h^{-1}(T(n))$. But this follows from the fact that otherwise there would be a fixed recursive relation between the size of a TM and its running time on blank tape, and that would immediately make the Halting Problem decidable. □

Next we compare two ways of extending a deterministic finite control, namely by addition of a pushdown store and by addition of nondeterminism. The first result is from [26].

Theorem 3.4 (Meyer and Fisher) There is a constant $c > 0$ such that

$$\text{DPDA} \xrightarrow{2^{2^{c \cdot \sqrt[3]{n}}}} \text{DFA}$$

Proof

In Proposition 6 in [26] a sequence of dpda's $\{P(n) \mid n \in N\}$ with the following properties is constructed. P(n) has on the order of n states, n

input symbols and n pushdown symbols and it never pushes more than two symbols in a single move. Furthermore it accepts a regular language which is not accepted by any dfa with less than 2^{2^n} states. Hence, the size of P(n) is $k \cdot n^3$ for some constant k and the size of an equivalent dfa is at least 2^{2^n}. □

Because of the exponential upper bound on the succinctness between dfa's and nfa's we get the following corollary which states that nondeterminism in a finite control is not always as powerful as the addition of a deterministic pushdown store.

Corollary 3.5 There is a constant $c > 0$ such that

$$\text{DPDA} \xrightarrow{2^{c \cdot \sqrt[3]{n}}} \text{NFA}$$ □

The next theorem shows (together with Corollary 3.5) that two classes can be incomparable from the point of view of succinctness.

Theorem 3.6 There is a constant $c > 0$ such that

$$\text{NFA} \xrightarrow{2^{c \cdot \sqrt{n}}} \text{DPDA}$$ □

Before we can prove this result we need the following theorem from [11], which is presented without proof.

Theorem 3.7 (Geller, Hunt III, Szymanski and Ullman) There is a constant $c' > 0$ such that for $n \geq 8$, any pushdown automaton accepting the language

$$Q(n) = \{x \# x \mid x \in \{0,1\}^n\}$$

has size at least $2^{c' \cdot n}$. □

Proof of Theorem 3.6

Consider for each natural number n the language

$$L(n) = \{x \# y \mid x, y \in \{0,1\}^n,\ x \neq y\}$$

It is easy to see that L(n) can be recognized by a nondeterministic finite automaton of size $O(n^2)$. The machine guesses that x and y differ on the i'th bit. Having read the value of this bit in x it skips the input until it sees # and then checks that the i'th bit in y is indeed different from the one in x. We claim that a deterministic pda recognizing L(n) must have size at least $2^{c \cdot n}$ where c is a constant. To see this observe that the language Q(n) from Theorem 3.7 is equal to $\overline{L(n)} \cap (\{0,1\}^n \# \{0,1\}^n)$. Now (as in Theorem 3.3) it is easy to show that if L(n) is accepted by a small dpda then so is $\overline{L(n)}$ and since a finite control needs no more than 2n+1 states to check that a word is in $\{0,1\}^n \# \{0,1\}^n$ it follows that if L(n) is accepted by a small dpda then so is Q(n), which is a contradiction. □

Next we consider the effect of ambiguity, first in finite automata and then in pushdown automata (or context-free grammars).

Let UFA be the class of unambiguous finite automata and recall from Chapter 2 that an ufa is allowed to use nondeterminism but not to accept a word in more than one way.

Theorem 3.8 There is a constant $c > 0$ such that

$$\text{UFA} \xrightarrow{2^{c \cdot \sqrt{n}}} \text{DFA}$$

Proof

Consider the family of languages $\{S(n) \mid n \in N\}$ where

$$S(n) = \{x\# a^m \mid x \in \{0,1\}^n,\ 1 \leq m \leq n,\ \text{the } m\text{'th bit of } x \text{ is } 1\}$$

Since a deterministic automaton must distinguish between all x-prefixes of the words $x\# a^m$, it is clear that a deterministic machine accepting $S(n)$ must have at least 2^n states. An unambiguous machine, on the other hand, can recognize $S(n)$ by guessing which bit is the m'th, skipping the input until it sees $\#$, and then check that the guess was correct. It is easy to see that such a machine needs not be any bigger than $O(n^2)$. □

Thus unambiguous nondeterminism is more powerful than determinism. Next we show that ambiguous nondeterminism is also more powerful than unambiguous nondeterminism.

Theorem 3.9 There is a constant $c > 0$ such that

$$\text{NFA} \xrightarrow{2^{c \cdot \sqrt{n}}} \text{UFA}$$

Proof

Consider again the family $\{L(n) \mid n \in N\}$ used in the proof of Theorem 3.6, where

$$L(n) = \{x\# y \mid x, y \in \{0,1\}^n,\ x \neq y\}$$

and recall that $L(n)$ is accepted by a nfa of size $O(n^2)$.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an unambiguous finite automaton accepting $L(n)$.

We show that M must have at least 2^n states, in fact we show that at least that many states are reachable from the startstate via prefixes of the form $x\#$.

Let $x \in \{0,1\}^n$ be arbitrary and assume that $K_x = \{q_1, \ldots, q_{k_x}\}$ is the set of states reachable from q_0 via $x\#$. Define for each i ($1 \leq i \leq k_x$) the set $A_x^i = \{y \in \{0,1\}^n \mid \exists\, q \in F: (q_i, y) \vdash^* (q, \lambda)\}$ consisting of the words in $\{0,1\}^n$ which lead from the state q_i (in K_x) to acceptance. Consider, for x varying over $\{0,1\}^n$, the total collection of these sets $A = \{\{A_x^i\}_{i=1}^{k_x}\}_{x \in \{0,1\}^n}$. Since the same set can occur several times in A, we let $B_1, B_2, \ldots, B_m$ be a listing of A without repetitions. Let K be the set of all states reachable from q_0 via some prefix of the form $x\#$ and consider the function which maps each state q in K to the set of words in $\{0,1\}^n$ which lead from q to a final state. It is easy to see that this function maps K onto $\{B_1, \ldots, B_m\}$. Hence in order to show that there are at least 2^n states in K it is sufficient to show that there are at least that many B_i's.

We do this by interpreting subsets of $\{0,1\}^n$ as elements of the 2^n-dimensional vector-space over the field of characteristic 2. Assume that $x_1, x_2, \ldots, x_{2^n}$ is an enumeration of the words in $\{0,1\}^n$. With each $C \subseteq \{0,1\}^n$ we associate the vector $\vec{C} = (c_1, c_2, \ldots, c_{2^n})$ where for $1 \leq j \leq 2^n$, $c_j = 1$ iff $x_j \in C$.

Now consider for each x the set $A_x = \bigcup_{i=1}^{k_x} A_x^i$ consisting of all y's such that $x\#\, y$ is in L(n). We claim that since the automaton is unambiguous, the sets $A_x^1, \ldots, A_k^{k_x}$ are mutually disjoint. This follows since if there is i, j and y such that $y \in A_x^i \cap A_x^j$ then we have

$$(q_0,\ x\#\, y) \vdash^* (q_i, y) \vdash^* (p_1, \lambda)$$

and

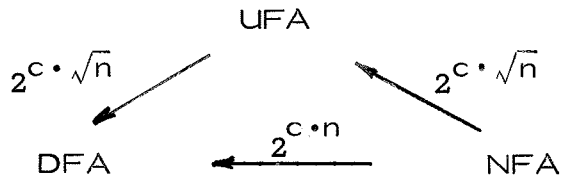
$$(q_0,\ x\#\, y) \vdash^* (q_j, y) \vdash^* (p_2, \lambda)$$

where both p_1 and p_2 are in F. But then $x\#y$ is accepted in two different ways, and that is a contradiction. Now since $A_x^1, \ldots, A_x^{k_x}$ are disjoint and furthermore all occur among the B_i's, the vector $\vec{A}_x$ can be written as a linear combination of the vectors $\{\vec{B}_i\}_{i=1}^m$, i.e. there exist coefficients $t_1, \ldots, t_m \in \{0, 1\}$ such that

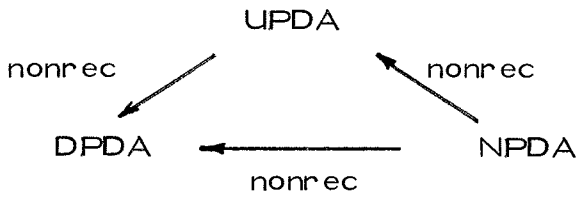
$$(*) \qquad \vec{A}_x = \sum_{j=1}^{m} t_j \vec{B}_j$$

Assume that x is the i'th element in the enumeration of $\{0, 1\}^n$, i.e. $x = x_i$. Since all words of the form $x_i \# x_j$ are in L(n), unless i = j, it follows that A_{x_i} is equal to $\{0, 1\}^n - \{x_i\}$, thus $\vec{A}_{x_i} = (1, 1, \ldots, 1, 0, 1, \ldots, 1)$, where 0 is the i'th coordinate. But it is easy to see that the vectors $\{\vec{A}_{x_i}\}_{i=1}^{2^n}$ are linearly independent and since (*) shows that they all can be written as linear combinations of $\vec{B}_1, \ldots, \vec{B}_m$ it follows that $m \geq 2^n$. Hence there are at least 2^n B_i's, consequently also at least 2^n states in K, and the theorem is proved. □

Theorems 3.1, 3.8 and 3.9 together can be represented by the following triangle.



If we consider the same three classes of pushdown automata the situation is quite analogous provided all succinctness functions are replaced by nonrecursive functions, i.e. we have the picture



The NPDA - DPDA result was shown in Theorem 3.3. Now we prove the NPDA - UPDA result which has also appeared in [31]. The basic technique in the proof is the same as in Theorem 3.3. We show that if the NPDA - UPDA succinctness is assumed to be recursive, then the Halting Problem is decidable. Again we encode Turing Machine computations into context-free grammars but this time the encoding has to be different because UPDA is not closed under complement (see [18]).

Let M be a deterministic single tape Turing Machine which halt when started on blank tape. For technical reasons M is assumed to perform an odd number of steps and to not write the blank symbol. Thus at the end of its computation every tape cell which has been scanned will be left non-blank. Let Ψ be the class of all machines of this type.

Let Q be M's stateset and Γ its tape alphabet. Let $\{\#, a, b, c\}$ be a set of new symbols disjoint from both $Q \times \Gamma$ and Γ, and let $\Delta = (Q \times \Gamma) \cup \Gamma \cup \{\#\}$. We associate two languages $\bar{L}_M$ and $\hat{L}_M$, with M as follows:

$$\bar{L}_M = \{x_1 \# x_2^R \# x_3 \# x_4^R \# \ldots \# x_{2n}^R \; a^i b^i c^j \mid$$

a) $n \geq 1$, $i \geq 1$, $j \geq 1$

b) x_1 is the starting configuration of M when started on blank tape,

c) $x_{2p-1} \vdash x_{2p}$ for $1 \leq p \leq n\}$

$$\hat{L}_M = \{ y_1 \# y_2^R \# y_3 \# y_4^R \# \ldots \# y_{2n}^R \; a^{|y_{2n}|} b^j c^j \mid$$

a) $n \geq 1,\ j \geq 1$

b) $y_{2p} \vdash y_{2p+1}$ for $1 \leq p < n$

c) y_{2n} is a halting configuration of $M\}$

Observe that both $\bar{L}_M$ and $\hat{L}_M$ are subsets of $\Delta^+ a^+ b^+ c^+$. $\bar{L}_M$ and $\hat{L}_M$ have been chosen to generate strings consisting of pairs of M-configurations. In $\bar{L}_M$, the odd-even pairs represent single steps of M, and in $\hat{L}_M$, it is the even-odd pairs which represent single steps.

It is easy to see that $\bar{L}_M$ and $\hat{L}_M$ have unambiguous context-free grammars, $\bar{G}_M$ and $\hat{G}_M$ respectively, whose sizes are no bigger than a constant times the size of M. If we take their union, we get a grammar G_M for the language

$$L_M = \bar{L}_M \cup \hat{L}_M$$

having the property that

$$\underline{\text{size}}(G_M) \leq C \circ \underline{\text{size}}(M)$$

for some constant C independent of M. This grammar, however, is ambiguous because $\bar{L}_M \cap \hat{L}_2$ is non-empty. Indeed, this intersection contains exactly one string z which corresponds to the computation $z_1, z_2, \ldots, z_{2n}$ of M started on blank tape in the following way.

$$z = z_1 \# z_2^R \# \ldots \# z_{2n}^R a^N b^N c^N$$

where N is the amount of tape used during the computation.

Since $\bar{L}_M \cap \hat{L}_M$ is a finite set, L_M also has an unambiguous context-free grammar. One way of producing such a grammar is to first construct

unambiguous pda's for the languages $\overline{L}_M$ and $\hat{L}_M$. Modify the pda for $\hat{L}_M$ by adding appropriate states to its final control to enable it to reject z. Convert the pda's to grammars, take their union, and the result is an unambiguous grammar for L_M. Its size, however, is large due to the extra machinery necessary to avoid generating z in two different ways.

Next we prove that <u>any</u> unambiguous grammar for L_M must be huge if z is long, the reason being that since z has inherited two different structures (one from $\overline{L}_M$ and another from $\hat{L}_M$) it will have two different derivations in any small grammar generating L_M. The proof is very similar to the proof that $\{a^i b^j c^k \mid i = j \vee j = k\}$ is an ambiguous language which can be found on page 205 of the book by Aho & Ullman [4] . It uses the following result from [29].

<u>Lemma 3.10 (Ogden)</u>

Let G be a context-free grammar with m symbols (terminals and nonterminals), let h be the length of the longest righthandside of the productions and let $k = \max\{3, h^{2m+3}\}$. If $z \in L(G)$, $|z| \geq k$ and if k or more positions in z are designated as being "distinguished" then z can be written as uvwxy such that

1) w contains at least one of the distinguished positions
2) either u and v both contain distinguished positions or
   x and y both contain distinguished positions
3) vwx has at most k distinguished positions
4) there is a nonterminal A such that
   $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uv^iAx^iy \overset{*}{\Rightarrow} uv^iwx^iy$ for all $i \geq 0$ □

Using Ogden's Lemma we can prove the following lemma which is the key to the result.

Lemma 3.11

Let M be in Ψ and let N be the amount of tape used in its computation on blank tape. Let G be any context-free grammar generating L_M and let k be the constant from Ogden's Lemma,

If $N \geq k! + k$ then G is ambiguous.

Proof

Consider the language L_M and the corresponding string z as described above. Let us rewrite z as $\alpha a^N b^N c^N$ where $\alpha \in \Delta^+$ is that portion of z representing the computation of M. We will show that if $N \geq k! + k$ then z has two different derivations in G (recall that $\{z\} = \overline{L}_M \cap \hat{L}_M$).

Assume that $N = k! + j$ where $j \geq k$ and consider the word $z' = \alpha a^N b^j c^j$ which is a member of $\hat{L}_M$ and therefore of L_M. Since $j \geq k$, we may distinguish all the b's in z' and write z' as

$$z' = uvwxy$$

where u, v, w, x and y satisfy the conditions of Ogden's Lemma. We claim that v consists entirely of b's, x consists entirely of c's and $|v| = |x|$. The argument is as follows.

If x and y both have distinguished positions, then $x \in b^+$ because w has at least one distinguished position. This implies that v is λ or else v is a member of Δ^+, Δ^+a^+, $\Delta^+a^+b^+$, a^+, a^+b^+ or b^+.

If v does not contain any a's then the string uwy is of the form $\alpha' a^N b^{j-i} c^j$ for some $i > 0$. Since this string has different numbers of a's, b's and c's it can't be a member of L_M and so v must contain at least one a.

If v contains other symbols than a's, then the string uv^2wx^2y is not in $\Delta^+a^*b^*c^*$ and hence not in L_M. Thus, $v \in a^+$. Now $uwy = \alpha a^{N-|v|}b^{j-|x|}c^j$ and $uv^2wx^2y = \alpha a^{N+|v|}b^{j+|x|}c^j$ are both in L_M. This implies that $N-|v| = j-|x|$ and $N+|v| = j+|x|$ which contradicts the fact that $N \neq j$. Hence the possibilities for v are exhausted and we may conclude that x and y do not both have distinguished positions.

Accordingly, u and v must both have distinguished positions. Moreover, $v \in b^+$. The possible locations for x are $x = \lambda$, $x \in b^+$, $x \in b^+c^+$ or $x \in c^+$. The first two possibilities are eliminated by considering $uwy = \alpha a^N b^{j-|v|-|x|}c^j$ and the third possibility is eliminated by noting that uv^2wx^2y is a member of $\Delta^+a^+b^+c^+b^+c^+$. This means that $v \in b^+$ and $x \in c^+$. Moreover, if $|v| \neq |x|$, then the string $uwy = \alpha a^N b^{j-|v|}c^{j-|x|}$ is certainly not in L_M, and the proof of our claim is completed.

According to Ogden's Lemma there exists a nonterminal A such that z' can be derived as shown below:

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy = z'$$

Since v consists entirely of distinguished positions and vwx has at most k distinguished positions, $|v| < k$ and so $|v|$ divides $k!$. Let $h = k!/|v|$. By repeating the subderivation $A \overset{*}{\Rightarrow} vAx$ exactly h times, we can derive z in the following way:

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uv^hAx^hy$$
$$\overset{+}{\Rightarrow} uv^hwx^hy = \alpha a^N b^{j+h|v|}c^{j+h|v|} = z$$

(Recall that $N = k! + j = \frac{k!}{|v|} \cdot |v| + j = h|v| + j$.)

Now let us consider the string $z'' = \alpha a^j b^j c^N$ which is a member of $\overline{L}_M$ and hence L_M. By distinguishing all the b's in z'' and arguing in a

fashion similar to the above, we can write z'' as $z'' = \bar{u}\bar{v}\bar{w}\bar{x}\bar{y}$ with $|\bar{v}| = |\bar{x}|$, $\bar{v} \in a^+$ and $\bar{x} \in b^+$. As before, there exists some nonterminal B and integer m for which

$$S \overset{*}{\Rightarrow} \bar{u}B\bar{y} \overset{*}{\Rightarrow} \bar{u}\bar{v}^m B \bar{x}^m \bar{y}$$
$$\overset{*}{\Rightarrow} \bar{u}\bar{v}^m \bar{w}\bar{x}^m \bar{y} = z$$

To complete the proof we will show that the two derivations of z described above have different derivation trees. Assume the contrary. Then the derivation tree for z contains a node labeled A and a node labeled B. No A node can be a descendant of a B node in any derivation tree, for if it were, there would be a derivation of the form $B \overset{+}{\Rightarrow} t_2 A t_4$ in the grammar, and consequently L_M would contain words of the form

$$t_1 a^{|\bar{v}|}\ t_2 b^{|v|}\ t_3 c^{|v|}\ t_4 b^{|\bar{v}|}\ t_5$$

Similarly, B is not a descendant of A. This means that A and B are incomparable in the derivation tree for z and hence there are terminal strings s_1, s_2, and s_3 such that $S \overset{+}{\Rightarrow} s_1 B s_2 A s_3$ in G. By inserting the subderivations $B \overset{+}{\Rightarrow} \bar{v}B\bar{x}$ and $A \overset{+}{\Rightarrow} vAx$ we can produce, for any integer i, the string $s_1 \bar{v}^i \bar{w} \bar{x}^i s_2 v^i w x^i s_3$. By choosing i sufficiently large, we can produce strings having many more b's than either a's or c's. Such a word is clearly not in L_M, contradicting our assumption that the two derivations of z correspond to the same tree. Accordingly, G is ambiguous. □

We use Lemma 3.11 to show that the size of any unambiguous grammar for L_M must grow with the amount of tape used in M's computation.

Lemma 3.12

There exists a constant C with the following property. Let M be a machine in Ψ which uses N tape cells in its computation. Let G be an unambiguous grammar generating L_M. Then

$$\underline{\text{size}}(G) \geq C \cdot [\log \log N]^{1/2}$$

Proof

Let m and h be, respectively, the number of symbols in G and the length of the longest right side of a production in G. Let $g = \underline{\text{size}}(G)$. We know from Lemma 3.11 that

$$N < (h^{2m+3})! + h^{2m+3}$$

Since m is at least 2 (otherwise G could only generate strings of length 0 and 1) we must have $2m+3 \leq 4m$. In addition, $m \leq g$ and $h \leq g$. Hence

$$N < (g^{4g})! + g^{4g} \leq 2\,(g^{4g})!$$

which implies (since $g \geq 2$) that

$$N < (g^{4g})(g^{4g})$$

Taking logarithms twice,

$$\begin{aligned} \log \log N &< 4g \log g + \log 4 + \log g + \log \log g \\ &\leq 4g \log g + 4 \log g \\ &\leq 6g \log g \\ &\leq 6g^2 \end{aligned}$$

which is what we want. □

Theorem 3.13 (Schmidt and Szymanski)

$$NPDA \xrightarrow{nonrec} UPDA$$

Proof

Let f(n) be an arbitrary recursive function. We have to show that $NPDA \xrightarrow{f(n)} UPDA$.

So far the following has been proved about grammars generating the language L_M where M is in Ψ

a) L_M has an ambiguous grammar whose size is no more than a constant times the size of M.

b) The size of an unambiguous grammar for L_M depends monotonically on the amount of space M uses in its computation on blank tape.

Again, as in Theorem 3.3, since the amount of space used by M cannot depend recursively on M's size there is a family of TM's $\{M(n) \mid n \in N\}$ which gives the desired succinctness result. The details are similar to the proof of Theorem 3.3 and are left to the reader.

□

The last result in the "PDA succinctness triangle" is from [34]. We briefly outline the proof.

Theorem 3.14 (Valiant)

$$UPDA \xrightarrow{nonrec} DPDA$$

Proof (outline)

Let M be a TM in the class Ψ and let O_M and E_M be the two languages consisting of the odd-even pairs and even-odd pairs in M's computation on blank tape. Let a and b be new symbols and consider the language

$$L'_M = O_M a \cup E_M b$$

It is easy to see that L'_M is accepted by a small unambiguous pda. L'_M is also accepted by a dpda but since a dpda essentially has to see which marker (a or b) terminates the input before it decides which pair of steps to compare, its size will depend on the length of M's computation on blank tape. Hence we can again argue as in Theorem 3.3 □

The following table summarizes the results presented so far in this chapter. A function f(n) in row C_1 and column C_2 represents the result $C_1 \xrightarrow{f(n)} C_2$, and a "?" indicates that the nature of the succinctness is open (to the author's knowledge).

Table 3.1. Succinctness relations between finite automata and pda's.

	DFA	UFA	NFA	DPDA	UPDA	NPDA
DFA	-	-	-	-	-	-
UFA	$2^{c\cdot\sqrt{n}}$	-	-	?	-	-
NFA	$2^{c\cdot n}$	$2^{c\cdot\sqrt{n}}$	-	$2^{c\cdot\sqrt{n}}$	?	-
DPDA	$2^{2^{c\cdot\sqrt[3]{n}}}$	$2^{c\cdot\sqrt[3]{n}}$	$2^{c\cdot\sqrt[3]{n}}$	-	-	-
UPDA	?	?	?	nonrec	-	-
NPDA	nonrec	nonrec	nonrec	nonrec	nonrec	-

Most of the results in Table 3.1 have been stated explicitly in this chapter. Theorem 3.1 states that NFA $\xrightarrow{2^n}$ DFA but that was for the "number of states" measure. Since in the "length of description" measure the nfa's $M_1(n)$ in the proof of the theorem are of size $k \cdot n$ for some constant k we need the constant c in the result in the table. Of the "?"'s in the table the most interesting ones are absolutely those in the UPDA-row. Since the question of whether these functions are recursive or not is related to the complexity of deciding regularity of unambiguous context-free languages we postpone further discussion until the next chapter.

The last type of relative succinctness results we consider are results involving regular expressions. Let REXP be the class of regular expressions and recall that the size of a regular expression is its length.

Lemma 3.15

There is a constant C such that for any regular expressions R there exists a nfa, M, accepting the language generated by R such that $\underline{\text{size}}(M) \leq C[\underline{\text{size}}(R)]^2$.

Proof

This is a corollary of Theorem 9.2 in [3] where it is shown that there exists a finite automaton M' (with λ transitions) which has no more than $2 \cdot \underline{\text{size}}(R)$ states and furthermore such that no state in the state transition diagram has more than 2 successors. Since λ-transitions, which are not allowed in our definition of nfa's, can be eliminated without increasing the machine size more than quadratically, the lemma follows.

□

Thus there is at most quadratic succinctness between regular expressions and nondeterministic finite automata. The converse is different as the following result from [7] shows.

Theorem 3.16 (Ehrenfeucht and Zeiger) There is a constant $c > 0$ such that

$$\text{DFA} \xrightarrow{2^{c \cdot \sqrt{n}}} \text{REXP}$$

Proof

Consider the complete directed graph with n nodes and n^2 distinct arc-labels. It is shown in [7] that any regular expression generating the set of all paths between two specified nodes is of length at least 2^n. Since the deterministic finite automaton corresponding to the graph is obviously of size n^2, the theorem follows. □

The next result is the "inverse" to Theorem 3.16.

Theorem 3.17 There is a constant $c > 0$ such that

$$\text{REXP} \xrightarrow{2^{c \cdot \sqrt{n}}} \text{DFA}$$

Proof

The language L(n) in the proof of Theorem 3.9 is generated by the following regular expression

$$\sum_{i=0}^{n-1} (\{0,1\}^i 0 \{0,1\}^{n-i-1} \# \{0,1\}^i 1 \{0,1\}^{n-i-1} + \{0,1\}^i 1 \{0,1\}^{n-i-1} \# \{0,1\}^i 0 \{0,1\}^{n-i-1})$$

which is of length $k \cdot n^2$ for some constant k. It was shown in Theorem 3.9 that any unambiguous, hence also any deterministic, finite automaton recognizing L(n) must have size at least 2^n. □

Recalling that the functions in Fig. 3.1 are lower bounds on the succinctness gaps, we close this chapter by considering the question of how good they are.

In a very coarse classification such as polynomial, exponential, double exponential etc., all the results in the UFA- and NFA-rows are optimal for the obvious reason that we can always make a finite automaton deterministic by increasing its number of states at most exponentially. As shown in [33] the DPDA-DFA result is also optimal in this sense. (We shall come back to this upper bound in the next chapter.)

In a finer classfication of functions, where the degrees of the arguments to the exponentials are considered, it is doubtful that any of the results in Fig. 3.1 (except the $2^{c \cdot n}$ in the NFA-DFA entry) are optimal. Indeed, Kozen [22] has pointed out that both the UFA-DFA and the NFA-UFA result can be strengthened to $2^{c \cdot n}$ by considering instead of the finite languages S(n) and L(n) in Theorems 3.8 and 3.9 the following infinite languages

$$S'(n) = \{x\#a^m \mid 1 \leq m \leq n,\ |x| \geq m,\ \text{the } m'\text{th bit of } x \text{ is } 1\}$$

$$L'(n) = \{x\#y \mid \exists\ i,\ 1 \leq i \leq m : |x| \geq i,\ |y| \geq i,\ x \text{ and } y \text{ differ on the } i'\text{th bit}\}.$$

We conjecture that also the functions in the two last entries in the DPDA row can be improved, the DPDA-UFA result probably to a double exponential.

4. COMPLEXITY

In this section we consider complexity results from the point of view of succinctness. We specifically look at the complexity of the problem of deciding whether a descriptor of some type does not generate all strings over its terminal alphabet. This problem being hard for a sufficiently difficult complexity class is taken as evidence that the descriptors being considered can represent information very succinctly. Again we shall emphasize the role played by ambiguity and demonstrate that its presence or absence does indeed influence the complexity of this problem. As before we are mainly interested in finite automata and pda's.

To make a connection to the last section we begin by discussing the regularity problem for deterministic, unambiguous and ambiguous context-free grammars. The recursive upper bound on the succinctness gap between the classes DPDA and DFA was first proved by Stearns in [32] and later the bound was improved ([33]) to the following.

Theorem 4.1 (Valiant) Let P be a dpda with q states and t pushdown symbols and assume that P pushes no more than h symbols on the stack in one move. If P accepts a regular language then this language is accepted by a dfa with no more than

$$2_{2^{q^2}\log q + \log t + \log h}$$

states. □

Hence the succinctness result for DPDA and DFA corresponds to a decidability result in the following way. To decide whether a dpda P accepts a regular language, just enumerate all finite automata with fewer

states than the bound in Theorem 4.1 and check for equivalence with P (which is clearly decidable). The nonrecursive lower bound in the NPDA-DFA entry in Table 3.1 also has an (un)decidability counterpart. We recall that Turing Degrees were defined in Chapter 2.

Theorem 4.2 (Hartmanis & Hopcroft, [16]). The problem of deciding whether a nondeterministic pda accepts a regular language is of Turing Degree 2. □

Thus, not only is the NPDA-DFA succinctness not recursive, but it is also not bounded by any function computed by a Turing Machine with a Halting Oracle. This follows since the equivalence problem between pda's and finite automata obviously is of Turing Degree 1.

Finally, the "?" in the NPDA-DFA entry in Table 3.1 corresponds to the

Open Problem: Is regularity decidable for unambiguous pushdown automata?

It is interesting to note that, irrespective of the answer to the question, regularity of unambiguous pda's is easier than regularity of arbitrary pda's.

Theorem 4.3 The regularity problem for unambiguous pda's is at most of Turing Degree 1.

Proof

The theorem is a straightforward consequence of the interesting result in the book of Salomaa and Soittola ([30]), originally due to A.L. Semenov,

that the equivalence problem for unambiguous context-free languages and regular languages is decidable. If G is a context-free grammar we have

$$L(G) \text{ is regular} \Leftrightarrow \exists M,\ M \in FA \wedge T(M) = L(G)$$

Now if G is unambiguous the predicate "$M \in FA \wedge T(M) = L(G)$" is recursive, hence the whole thing is at most of Turing Degree 1. □

Although the complexity of the regularity problem for unambiguous pda's is open, Theorems 4.1 to 4.3 is supporting evidence that, just as in the "succinctness triangles" in Chapter 3, unambiguous nondeterminism falls between determinism and (ambiguous) nondeterminism. This pattern reappears when we in the following consider the complexity of the predicate "M does not accept every word over its input alphabet".

The following table is a summary of the results in Theorems 4.1 to 4.3.

Table 4.1. Complexity of "Regularity for pda's".

Type of pda	Complexity of "L(P) is regular"
deterministic	decidable
unambiguous	at most Turing Degree 1
ambiguous	Turing Degree 2

Next we consider "$L(M) \neq \Sigma^*$" where M varies over the three classes of finite automata. The complexity of the problem is well known for both nondeterministic [27] and deterministic machines [21] (recall that $\leq_{log}$ ($\leq_p$) denotes log-space (p-time) reductions).

Theorem 4.4 (Meyer & Stockmeyer, Jones)

a) The set $\{M \in NFA \mid T(M) \neq \Sigma^*\}$ is complete for PSPACE under $\leq_p$.

b) The set $\{M \in DFA \mid T(M) \neq \Sigma^*\}$ is complete for NLOGSPACE under $\leq_{log}$. □

The reason "$T(M) \neq \Sigma^*$" is so difficult in general is that the length of the shortest string not accepted by a nondeterministic finite automaton can be exponential in the size of the machine. In the next lemma we use an argument similar to the proof of Theorem 3.9 to show that ambiguity plays an essential role in this respect.

Lemma 4.5 Let $M = (Q, \Sigma, \delta, q_0, F)$ be an unambiguous finite automaton with m states. The shortest word not accepted by M is no longer than m+1.

Proof

Let $w = a_1 \ldots a_n$ be one of the shortest words not accepted by M and let $K_0, K_1, K_2, \ldots, K_n$ be the set of states reachable from q_0 via λ, $a_1, a_1a_2, \ldots, a_1 \ldots a_{n-1}$ and $a_1 \ldots a_n$. We will show that the set of states $K = \bigcup_{i=0}^{n-1} K_i$ contains at least n-1 elements. As in Theorem 3.9 we do this by associating with each state in K a set of words which gets interpreted as a vector in an appropriate vector-space in such a way that n-1 of the vectors become linearly independent.

Here the proper choice of words is the set of suffixes of w. Let for $1 \leq i \leq n$ x_i denote $a_i a_{i+1} \ldots a_n$ and let $X = \{x_1, x_2, \ldots, x_n\}$. We associate subsets of X with the states in K_i in the following way. Assume $K_i =$

$\{q_1,\ldots,q_{k_i}\}$ and let for $1 \le j \le k_i$, A_i^j be the set of words in X leading from q_j to acceptance, i.e. $A_i^j = \{x \in X \mid \exists\, q \in F\colon (q_j,x) \vdash^* (q,\lambda)\}$. Consider the union $A_i = \bigcup_{j=1}^{k_i} A_i^j$ of these sets. Since $a_1\ldots a_{i-1}$ leads to the states in K_i and since A_i consists of the words leading from there to acceptance, $x_i = a_i\ldots a_n$ cannot be in A_i because then the automaton would accept $w = a_1\ldots a_{i-1}a_i\ldots a_n$. Furthermore, since w is the shortest string rejected by M the i+1 words $a_1\ldots a_{i-1}x_{i+1},\ldots,a_1\ldots a_{i-1}x_n$, which are all shorter than w, are accepted. But that means that $x_{i+1},\ldots,x_n$ all are in A_i.

Now consider the n-dimensional vector-space over the field of characteristic 2 and interpret subsets of X as vectors in the following way. If $C \subseteq X$ then $\vec{C} = (c_1,\ldots,c_n)$ where for $1 \le j \le n$, $c_j = 1$ iff $x_j \in C$. By the above argument we know that $x_i \notin A_i$ and that $x_j \in A_i$ for $i < j \le n$. Hence the vector $\vec{A}_i$ is of the form

$$\vec{A}_i = (b_i^1,\ldots,b_i^{i-1},0,1,\ldots,1)$$

where the first i-1 coordinates are determined as follows:

$$b_i^j = \begin{cases} 1 & \text{if } a_1\ldots a_{i-1}x_j \text{ is accepted by M} \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } 1 \le j \le i-1$$

Let $B_1,B_2,\ldots,B_k$ be a listing, without repetitions, of the sets appearing in the total collection of sets $\{\{A_i^j\}_{j=1}^{k_i}\}_{i=1}^{n}$. Again, as in Theorem 3.9, since the automaton is unambiguous each A_i is a disjoint union of $A_i^1,\ldots,A_i^{k_i}$, hence $\vec{A}_i$ can be written as a linear combination of the vectors $\vec{B}_1,\ldots,\vec{B}_k$. Also, the number of states in K is greater than or equal to the number of sets in the listing $B_1,B_2,\ldots,B_k$, so all that remains is

to show that sufficiently many of the vectors $\vec{A}_1,\ldots,\vec{A}_n$ are linearly independent.

Consider the matrix

$$A = \begin{Bmatrix} 0 & 1 & \ldots\ldots\ldots\ldots & 1 \\ b_2^1 & 0 & 1 & \ldots\ldots\ldots & 1 \\ b_3^1 & b_3^2 & 0 & 1 & \ldots\ldots\ldots & 1 \\ \vdots & & & & & \\ b_i^1 & \ldots\ldots\ldots & 0 & 1 \ldots & 1 \\ \vdots & & & & \\ b_n^1 & \ldots\ldots\ldots & b_n^{n-1} & 0 \end{Bmatrix}$$

whose i'th row is $\vec{A}_i$. If we disregard the first column and the last row we get the $(n-1) \times (n-1)$ submatrix

$$A' = \begin{Bmatrix} 1 & 1 & \ldots\ldots\ldots\ldots & 1 \\ 0 & 1 & \ldots\ldots\ldots\ldots & 1 \\ b_3^2 & 0 & 1 & \ldots\ldots\ldots & 1 \\ \vdots & & & & \\ b_{n-1}^2 & \ldots\ldots\ldots & b_{n-1}^{n-1} & 1 \end{Bmatrix}$$

which is easily seen to reduce, by Gaussian elimination over the field of characteristic 2, to the $(n-1) \times (n-1)$ unit matrix. From this we conclude that the matrix A has rank n-1 and hence that n-1 of the vectors $\vec{A}_1,\ldots,\vec{A}_n$ are linearly independent. Now the same argument as in Theorem 3.9 applies, and we conclude that the automaton has at least n-1 states. □

Lemma 4.5 has the immediate corollary that deciding whether an automaton accepts all words over its input alphabet is probably easier for unambiguous machines than for ambiguous ones.

Corollary 4.6 The set

$$\{ M \in UFA \mid T(M) \neq \Sigma^* \}$$

is in NP.

Proof

The method is exactly the same as the one used to show that the problem is in PSPACE for arbitrary finite automata (see Lemma 10.3 in [3]). But since, by Lemma 4.5, the shortest string not accepted by an unambiguous machine is no longer than the number of states in the machine, the algorithm stops after a polynomial amount of time in this case. □

We get analogous results if regular expressions are considered rather than finite automata. Thus the set $\{R \in REXP \mid L(R) \neq \Sigma^*\}$ is complete for PSPACE and $\{R \in UREXP \mid L(R) \neq \Sigma^*\}$ is in NP, where UREXP is the class of unambiguous regular expressions, i.e. expressions that generate each string over its alphabet at most once. The notion of determinism seems unnatural for regular expressions and we shall not attempt to give any definitions.

Next we consider the complexity of determining whether a regular expression is ambiguous.

Theorem 4.7 The set

$$A = \{R \in REXP \mid R \text{ is ambiguous}\}$$

is complete for NLOGSPACE.

Proof

First we show that A is in NLOGSPACE. Let R be a regular expression of length n (over the alphabet Σ). We show that if there is a word which is generated in more than one way, then there is one which is no longer than $2n^2$. It is easy to see that there is a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ with no more than n states accepting the language generated by R such that if a word is generated in two different ways by R then it is accepted in two different ways by M. Now, let w be the shortest word accepted in two different ways by M and assume that $|w| \geq 2n^2+1$. Let the state-sequences corresponding to the two accepting computations be $s_1 = q_0 q_1 \cdots q_m$ and $s_2 = p_0 p_1 \cdots p_m$ where $q_m, p_m \in F$, $p_0 = q_0$ and $m = |w|$. Since $m \geq 2n^2 + 1$, some pair of states occurs at least three times in the sequence $(q_0, p_0)(q_1, p_1), \ldots, (q_m, p_m)$. Assume that $(q_i, p_i) = (q_j, p_j) = (q_k, p_k)$ for some i, j and k with $0 \leq i < j < k \leq m$. Since the sequences s_1 and s_2 are different, either $q_0 \cdots q_i q_{j+1} \cdots q_m$ is different from $p_0 \cdots p_i p_{j+1} \cdots p_m$ or $q_0 \cdots q_j q_{k+1} \cdots q_m$ is different from $p_0 \cdots p_j p_{k+1} \cdots p_m$. In either case we can, by cutting out the proper piece of the input, obtain a shorter word which is also accepted in two different ways, and that contradicts the assumption that the shortest word is longer than $2n^2$. This shows that a log-space bounded Turing Machine is powerful enough to guess (symbol by symbol) a word generated in more than one way if there is one.

To see that the machine is also capable of checking that such a word

is indeed generated in two ways by the expression, we first note that this would be straightforward if the input had been the automaton M, rather than the expression R. Then we would just guess two sequences of states and accept in case they both end in final states and are different. Now, since states in M correspond to positions in R, we construct the nondeterministic Turing Machine such that it guesses two sequences of positions, each corresponding to a parse of the word. The reason this is not difficult is that R is assumed to be syntactically correct, hence we can always, by counting parentheses, find the subexpression beginning at or ending at a certain position. The details of the construction are left to the reader, but it should be clear that the algorithm works correctly and runs in nondeterministic logarithmic space.

Next we show that the set A is hard for NLOGSPACE. Given a nondeterministic log-space bounded Turing Machine M and an input x we construct two unambiguous regular expressions $\bar{R}_x$ and $\hat{R}_x$, such that $\bar{R}_x \cup \hat{R}_x$ is ambiguous if and only if M accepts x. The technique is the same as in Theorem 3.13, $\bar{R}_x$ represents all the odd-even pairs of consecutive configurations of M on x and $\hat{R}_x$ all the even-odd pairs. Then $\bar{R}_x \cap \hat{R}_x$ is nonempty exactly in case there is a valid computation of M on x.

We assume without loss of generality that the machine M always performs an odd number of steps. Consider the following two sets of words

$$\overline{W}_x = \{\#i_1\$z_1\#i_2\$z_2\#i_3\$z_3\#\ldots\#i_{2n}\$z_{2n}\#|$$

a) $n \geq 1$

b) (i_1, z_1) is the starting configuration of M on x

c) (i_{2j}, z_{2j}) follows from (i_{2j-1}, z_{2j-1}) by M's transition function (for $1 \leq j \leq n$)$\}$

$$\hat{W}_x = \{\#i_1\$z_1\#i_2\$z_2\#i_3\$z_3\#\ldots\#i_{2n}\$z_{2n}\#|$$

a) $n \geq 1$

b) (i_{2n}, z_{2n}) is an accepting configuration of M on x

c) (i_{2j+1}, z_{2j+1}) follows from (i_{2j}, z_{2j}) by M's transition function (for $1 \leq j < n$)$\}$

$\overline{W}_x$ and $\hat{W}_x$ are generated by the following two unambiguous regular expressions

$$\overline{R}_x = \#\ 1\ \$\ z_1\ \# \cdot N \cdot (P^*)$$
$$\hat{R}_x = B \cdot (P^*) \cdot F$$

where

a) z_1 is the starting worktape configuration

b) N is the sum of all expressions of the form i $ z# such that (i, z) follows from $(1, z_1)$

c) P is the sum of all expressions of the form i $ z# i' $ z'# such that (i', z') follows from (i, z)

d) B is the sum of all expressions of the form # i $ z#

e) F is the sum of all expressions of the form i $ z# where the state in z is a final state.

It is clear that $\overline{R}_x$ and $\hat{R}_x$ are unambiguous and also that the expressions N, P, B, F - and therefore also $\overline{R}_x$ and $\hat{R}_x$ - can be computed from M and x by a deterministic log-space bounded Turing Machine. But $\overline{W}_x \cap \hat{W}_x$ is nonempty if and only if it contains a word representing a valid computation of M on x. Hence the expression $R_x = \overline{R}_x + \hat{R}_x$ is ambiguous if and only if M accepts x. This shows that the set A is hard for NLOGSPACE. □

In this chapter we have shown that ambiguity in finite automata makes the set $\{M \mid T(M) \neq \Sigma^*\}$ harder than the corresponding "unambiguous" set (unless PSPACE $\neq$ NP). The situation is similar for pda's where it is well known that the set $\{P \in NPDA \mid T(P) \neq \Sigma^*\}$ is nonrecursive whereas, as mentioned in Theorem 4.3, the set $\{P \in UPDA \mid T(P) \neq \Sigma^*\}$ is recursive. Hence, the "unambiguous" sets are easier than the "ambiguous" ones. However we do not know exactly how easy they are because we have not been able to obtain any nontrivial results about their hardness. The standard way of generating invalid computations (which will be used intensively in the next chapter) does not seem to work because the resulting grammars and expressions appear to become (very) ambiguous.

5. MACRO GRAMMARS AND THE OI-HIERARCHY

In this chapter we consider the complexity of regular term grammars of higher type generating finite languages. We specifically consider what Engelfriet and the author called the OI-hierarchy in [8]. The first two families in this hierarchy are the context-free languages and the OI macro languages [9], which are also equal to Aho's indexed languages [1]. It is known that the problem "Is L(G) different from the set of all strings of length less than or equal to some number" is hard for nondeterministic exponential time when G is context-free and we show that it is hard for nondeterministic double exponential time when G is an OI macro grammar. We also discuss the complexity of the problem when G varies over grammars at level i in the hierarchy.

The proof that the problem is hard for NEXPTIME (under p-time reductions) when G is a context-free grammar generating a finite language was essentially given in [20]. The reason the problem is this hard is the "squaring power" of context-free grammars, which is illustrated by the following grammar of size O(n) which generates the language $\{w \in \{a,b\}^* \mid |w| = 2^n\}$

$$
\begin{aligned}
S &\rightarrow A_1A_1 \\
A_1 &\rightarrow A_2A_2 \\
&\vdots \\
A_i &\rightarrow A_{i+1}A_{i+1} \\
&\vdots \\
A_{n-1} &\rightarrow A_nA_n \\
A_n &\rightarrow a \mid b
\end{aligned}
$$

It is shown in [20] that this ability to generate exponentially long strings is sufficient to produce invalid computations of a Turing Machine running in nondeterministic exponential time. It was not mentioned in [20] that the problem is actually complete for NEXPTIME so we shall give the (easy) proof here.

First some notation. An integer expression, E, is a normal arithmetic expression with (nonnegative) integers, the normal operators +, -, *, / and also exponentiation which is denoted by "↑". ↑ takes precedence over the other operators, thus $5 * 2 \uparrow 3$ means $5 \cdot 2^3$. From [20] we borrow the notation $\Sigma^{**}(E)$ which for an alphabet Σ and integer expression E denotes the set of strings over Σ of length E.

Let CFGFIN be the class of context-free grammars generating finite languages, and let IE be the class of integer expressions.

Theorem 5.1 The set

$$F_0 = \{(G,E) \in CFGFIN \times IE \mid L(G) \neq (\Sigma \cup \{\lambda\})^{**}(E)\}$$

is complete for NEXPTIME under $\leq_p$.

Proof

It was shown in [20] that F_0 is hard for the class $\bigcup_{j=0}^{\infty} NTIME(2^{j \cdot n})$ under log-space reductions, and it is an easy exercise to extend that construction to show hardness for NEXPTIME $(= \bigcup_{j=0}^{\infty} NTIME(2^{n^j}))$ under p-time reductions.

To see that F_0 is in NEXPTIME we note that the pumping lemma for context-free languages yields a constant k_G bounding the length of any string generated by G (recall that G generates a finite language). Let

member(G, x) be a Boolean procedure with the property that for all context-free grammars G and all terminal strings x, member(G, x) is true if $x \in L(G)$ and false otherwise. It is well known that there exist such procedures which run in deterministic polynomial time (see [3]). Now the following nondeterministic algorithm accepts the set F_0;

Input: $(G, E) \in CFGFIN \times IE$

Output: "Yes" if $L(G) \neq (\Sigma \cup \{\lambda\})^{**}(E)$

Method: Compute k_G (the constant from the pumping lemma).

```
if k_G < E then answer "Yes" and halt
else
  begin
    comment check whether L(G) -(Σ ∪ {λ})** (E) ≠ ∅;
    guess x = a_1 a_2 ... a_m for some m with E ≤ m ≤ k_G;
    Output ("Yes") if member(G, x)
  or
    comment check whether (Σ ∪ {λ})** (E) - L(G) ≠ ∅;
    guess x = a_1 a_2 ... a_m for some m ≤ E;
    Output ("Yes") if not member(G, x)
  end
fi;
```

The algorithm runs in nondeterministic exponential time because of the well known fact that there exists $c > 0$ such that $k_G \leq 2^{c \cdot \text{size}(G)}$. Hence it doesn't take too much time to guess the string x. □

Next we show that the set F_0 is complete for nondeterministic double exponential time when we consider OI macro grammars generating finite languages. Macro grammars were introduced by M. Fisher in [9] as a generalization of context-free grammars. If context-free grammars are viewed as nondeterministic macroes without parameters, then macro grammars are nondeterministic macroes with parameters. Before giving the formal definition we introduce some auxiliary concepts.

A ranked alphabet Ω is an indexed family $\langle \Omega_n \rangle_{n \in N}$ of disjoint sets. A symbol f in Ω_n has rank n. Let Ω be a ranked alphabet. The set of macro-terms over Ω is the smallest set of strings satisfying

a) λ and $f \in \Omega_0$ are macro-terms

b) If t_1 and t_2 are macro-terms then $t_1 \cdot t_2$ is a macro-term

c) If $f \in \Omega_n$ and $t_1, \ldots, t_n$ are macro-terms then $f(t_1, \ldots, t_n)$ is a macro-term.

Let $X = \{x_1, x_2, \ldots, x_n, \ldots\}$ be a denumerable set (of variables) and let for each $k \geq 0$, $X_k = \{x_1, \ldots, x_k\}$. For any ranked alphabet Ω we denote by $\Omega(X_k)$ the ranked alphabet where $\Omega(X_k)_0 = \Omega_0 \cup X_k$ and $\Omega(X_k)_i = \Omega_i$ for $i \geq 1$.

Definition 5.2 A macro grammar is a system $G=(\Sigma, \mathfrak{F}, P, S)$ where

Σ is a finite alphabet of terminals

$\mathfrak{F}$ is a finite ranked alphabet of nonterminals

P is a finite set of productions of the form $F(x_1, \ldots, x_k) \rightarrow \tau$ where F is a nonterminal of rank k and τ is a macro-term over $\mathfrak{F}(\Sigma \cup X_k)$ $(k \geq 0)$

S is a nonterminal of rank 0 (the startsymbol).

Note that a macro-term over $\mathfrak{F}(\Sigma \cup X_k)$ contains variables from $\{x_1,\ldots,x_k\}$. □

Example 5.3 The following is a macro grammar. $G = (\{a,b\},\mathfrak{F},P,S)$ where $\mathfrak{F}_0 = \{S,A\}$, $\mathfrak{F}_1 = \{F,H\}$, $\mathfrak{F}_i = \emptyset$ for $i \geq 2$ and P is the following set of productions

$$
\begin{aligned}
S &\rightarrow F(A)\\
F(x_1) &\rightarrow F(H(x_1))\\
F(x_1) &\rightarrow x_1\\
H(x_1) &\rightarrow x_1x_1\\
A &\rightarrow a\\
A &\rightarrow b
\end{aligned}
$$

□

When several productions have the same lefthandside we shall write them as "one production" with "|" separating the righthandsides.

The language generated by a macro grammar G is the set of terminal strings derived by macro expansion from the startsymbol. Since macroes can be nested within macroes, the mode of derivation has to specify where in a macro-term expansion can take place. We shall be concerned with the socalled OI-mode, where only outermost macroes can be expanded.

Let τ be a macro-term with variables $x_1,\ldots,x_k$ and let $t_1,\ldots,t_k$ be macro-terms. Then $\tau[t_1,\ldots,t_k]$ is the macro-term obtained from τ by replacing each occurrence of x_i in τ by t_i. OI-derivation is defined formally as follows.

Definition 5.4 Let $G = (\Sigma, \mathfrak{F}, P, S)$ be a macro grammar and let σ_1 and σ_2 be macro-terms. Then σ_1 OI-derives σ_2, $\sigma_1 \underset{OI}{\Rightarrow} \sigma_2$, if and only if

i) For some $F \in \mathfrak{F}$, σ_1 contains a subterm of the form $F(t_1, \ldots, t_n)$ which is not itself a subterm of any other term of the form $G(s_1, \ldots, s_m)$

ii) There is a production $F(x_1, \ldots, x_n) \rightarrow \tau$ in P

iii) σ_2 is obtained by replacing the occurrence of $F(t_1, \ldots, t_n)$ in σ_1 by $\tau[t_1, \ldots, t_n]$

The OI-language generated by G is the set $\{w \in \Sigma^* \mid S \overset{*}{\underset{OI}{\Rightarrow}} w\}$ where $\overset{*}{\underset{OI}{\Rightarrow}}$ is the reflexive and transitive closure of $\underset{OI}{\Rightarrow}$. □

Example 5.5 Consider the grammar G from Example 5.3. It is left to the reader to show that G OI-generates the language

$$L_{OI}(G) = \{w \in \{a,b\}^* \mid |w| = 2^n \text{ for } n \geq 0\}. \quad \square$$

Note that if we consider IO-derivation, where innermost macroes are expanded first, then the grammar G generates the language

$$L_{IO}(G) = \{a^{2^n}, b^{2^n}\}$$

because then the derivation has to start with

$$S \underset{IO}{\Rightarrow} F(A) \Rightarrow F(a), \text{ or}$$
$$S \underset{IO}{\Rightarrow} F(A) \Rightarrow F(b).$$

From now on we shall talk about OI-generation only. Let MACFIN be the class of macro grammars which (OI-) generate a finite language and let F_1 be the analogue of F_0, i.e.

$$F_1 = \{(G,E) \in \text{MACFIN} \times \text{IE} \mid L(G) \neq (\Sigma \cup \{\lambda\})^{**}(E)\}.$$

Also define the complexity class NEXP(1) by

$$\text{NEXP}(1) = \bigcup_{j=0}^{\infty} \text{NTIME}(2^{2^{n^j}})$$

We shall show that F_1 is hard for NEXP(1) under $\leq_p$. Just as the key in (the omitted part of the proof of) Theorem 5.1 was the ability of context-free grammars to generate exponentially long strings, the key here will be the ability of macro grammars to generate strings of double exponential length. The following grammar G_n of size O(n) generates the set $\{w \in \{a,b\}^* \mid |w| = 2^{2^n}\}$.

$$\begin{aligned}
S &\rightarrow H_0(B)\\
H_0(x) &\rightarrow H_1(H_1(x))\\
&\vdots\\
H_i(x) &\rightarrow H_{i+1}(H_{i+1}(x))\\
&\vdots\\
H_{n-1}(x) &\rightarrow H_n(H_n(x))\\
H_n(x) &\rightarrow xx\\
B &\rightarrow a \mid b
\end{aligned}$$

Theorem 5.6

The set F_1 is hard for NEXP(1) under $\leq_p$.

Proof

Let $M = (Q, \Gamma, \delta, q_0, F)$ be a nondeterministic Turing Machine which runs in time $T(n) = 2^{2^{n^j}}$ and let $x = a_1 \ldots a_n$ be an input of length n. We assume without loss of generality that M is a singletape machine. We shall construct a pair (G_x, E) where G_x is a macro grammar generating a finite language and E is an integer expression, such that $L(G_x) \neq (\Sigma - \{\lambda\})^{**}(E)$ exactly in case M accepts x. The terminal alphabet, Σ, of G_x will be $\Sigma = \Gamma \cup (Q \times \Gamma) \cup \{\#\}$.

According to the definitions in Chapter 2, a valid computation of M on x is of the form $\#[(\Sigma - \{\#\})^{**}(T(n))\#]^h$ where $1 \leq h \leq T(n)+1$. Hence each valid computation has length at most $k_x = 2^{2^{n^j+1}} + 2^{2^{n^j+1}} + 2$. G_x will generate all invalid computations of length $\leq k_x$, and E will be the expression $2 \uparrow (2 \uparrow (n \uparrow j + 1)) + 2 \uparrow (2 \uparrow (n \uparrow j) + 1) + 2$. We now follow [20] in characterizing the set of invalid computations of length $\leq k_x$.

A string w, $|w| \leq k_x$ is an invalid computation if and only if

a) $|w|$ is less than the length of the initial configuration, i.e.

$$w \in (\Sigma \cup \{\lambda\})^{**}(T(n) + 1)$$

b) $|w| \leq k_x$ but w does not end in "#", i.e.

$$w \in [(\Sigma \cup \{\lambda\})^{**}(k_x - 1)] \cdot (\Sigma - \{\#\})$$

c) w contains an error between two consecutive configurations

d) w does not contain a symbol of the form (q, a) where $q \in F$ and $a \in \Gamma$, i.e.

$$w \in ((\Sigma - \bigcup_{\substack{q \in F \\ a \in \Gamma}} (q, a)) \cup \{\lambda\})^{**}(k_x)$$

e) w does not begin with

$$\#(q_0, a_1)a_2 \ldots a_n[(b)^{**}(T(n)-n)] \cdot \#$$

Since we have already shown how to construct a macro grammar of size O(n) generating the language $\{w \in \{a,b\}^* \mid w_0 = 2^{2^n}\}$ it is easy to see how to construct grammars of size $O(n^j)$ generating the sets of w's satisfying a), b) and d). Now we show how to obtain the sets in c) and e).

Let $p(n) = n^j$ and consider the following grammar $G_{p(n)}$

$$
\begin{aligned}
S &\rightarrow F_1(H_1(A), F_1(A,B)) \\
&\vdots \\
F_i(x,y) &\rightarrow F_{i+1}(H_{i+1}(x), F_{i+1}(x,y)) \\
H_i(x) &\rightarrow H_{i+1}(H_{i+1}(x)) \\
&\vdots \\
F_{p(n)}(x,y) &\rightarrow xy \mid yx \\
H_{p(n)}(x) &\rightarrow xx
\end{aligned}
$$

$G_{p(n)}$ generates (as we shall show in Lemma 5.7) the following language over $\{A,B\}$, $S_{p(n)} = \{A^jBA^{T(n)-j-1} \mid 0 \leq j < T(n)\}$ (recall that $T(n) = 2^{2^{p(n)}}$). We generate the set in c) as follows. If w contains an error between two consecutive configurations, then w is of the form $z_1 = u_1u_2\ldots u_juu_{j+1}\ldots u_{T(n)}$ where each of the u_j's is an arbitrary "configuration" and where u represents the two consecutive configurations containing the error. We can generate most of z_1 by first generating (in $G_{p(n)}$) $\alpha_1 = \#A^jBA^{T(n)-j-1}$ and then let each of the A's generate the set $(\Sigma \cup \{\lambda\})^{**}(T(n)+1)$. Now, u, which will be generated by B, is of the form $z_2 = v_1a_1b_1c_1(\Sigma^{**}(T(n)-1))a_2b_2c_2v_2$ where the triple $a_2b_2c_2$ does not follow from $a_1b_1c_1$ by a single application of M's transition function, and v_1 and v_2 is the part of u that we "don't care about". z_2 is at most of length $2T(n)+2$, hence $|v_1| + |v_2|$ is at most $T(n)-3$. To gene-

rate z_2 we construct another grammar of size O(p(n)) which from the B in α_1 generates an arbitrary element of $R_{p(n)} = \{C^j DC^{T(n)-3-j} \mid 0 \leq j \leq T(n)-3\}$, then we let the C's generate $\Sigma \cup \{\lambda\}$ and from D we generate all elements in $a_1b_1c_1(\Sigma^{**}(T(n)-1))a_2b_2c_2$ where the pair $(a_1b_1c_1, a_2b_2c_2)$ represents a transition error. That we can generate $R_{p(n)}$ in a small grammar follows from the fact that macro languages are sizeclosed under intersection with regular sets (see the remark following the proof).

To obtain the invalid computations with a wrong initial configuration we note (again following [20]) that if w does not begin with $\#(q_0, a_1)a_2 \ldots a_n [(b)^{**}(T(n) - n)]\#$ then it is because

i) it does not begin with $\#(q_0, a_1)a_2 \ldots a_n$ or

ii) it does not contain two occurrences of # or

iii) the two first occurrences of # are not separated by T(n) characters or

iv) it has a nonblank character in its j'th position where $n+2 \leq j \leq T(n) + 1$

It is easy to construct grammars of size O(p(n)) generating the sets in i), ii) and iii), so we only have to worry about iv). Again we use closure of macro languages under intersection with regular sets to show that we can obtain the language $\{C^{n+1}A^jBA^{T(n)-n-j} \mid 0 \leq j \leq T(n)-n-1\}$ from which it is easy to generate the set in iv).

Putting all the pieces together we have shown that there is a grammar G_x of size O(p(n)), which can be constructed in polynomial time from M and x such that $L(G_x) \subseteq (\Sigma \cup \{\lambda\})^{**}(k_x)$ and $L(G_x) \neq (\Sigma \cup \{\lambda\})^{**}(k_x)$ if and only if M accepts x. Hence the set F_1 is hard for NEXP(1) under $\leq_p$. □

Remark By macro languages being sizeclosed under intersection with regular sets we mean that there exists a polynomial $p(n,m)$ such that for all macro grammars G and all finite automata M, 1) $L = L(G) \cap T(M)$ is a macro language and 2) L is generated by a macro grammar G' such that $size(G') \leq p(size(G), size(M))$.

In [1], [2] and [9] it is shown that the classes of languages generated by (OI) macro grammars, indexed languages and nested stack automata are all equal. Furthermore [1], [2] and [9] contain the following sequence of (language preserving) transformations

Macro → Indexed → Nested Stack → Indexed → Macro

Since in each transformation the size of the output descriptor is polynomially bounded by the size of the input descriptor, and since nested stack automata have a normal finite control, the usual state-product construction between finite automata shows that macro languages are sizeclosed under intersection with regular sets. □

The next lemma was used in the proof of Theorem 5.6.

Lemma 5.7

The macro grammar G_n with productions

$$
\begin{aligned}
S &\rightarrow F_1(H_1(a), F_1(a,b)) \\
&\vdots \\
F_i(x,y) &\rightarrow F_{i+1}(H_{i+1}(x), F_{i+1}(x,y)) \\
H_i(x) &\rightarrow H_{i+1}(H_{i+1}(x)) \\
&\vdots \\
F_n(x,y) &\rightarrow xy \mid yx \\
H_n(x) &\rightarrow xx
\end{aligned}
$$

generates the language $L(G_n) = \{a^j b a^{2^{2^n}-j-1} \mid 0 \leq j < 2^{2^n}\}$.

Proof

We prove the following by (backwards) induction on i.

For all terms t_1, t_2, t_3

a) $G_i(t_1) \overset{*}{\underset{OI}{\Rightarrow}} (t_1)^{2^{2^{n-i}}}$

b) $F_i(t_1, t_2) \overset{*}{\underset{OI}{\Rightarrow}} t_1^j t_2 t_1^{2^{2^{n-i}} - 1 - j}$ for $0 \leq j < 2^{2^{n-i}}$

a) and b) are obviously true for i = n. Assume that a) and b) are true for i + 1. Then

a) $G_i(t_1) \underset{OI}{\Rightarrow} G_{i+1}(G_{i+1}(t_1))$

$\overset{*}{\underset{OI}{\Rightarrow}} (G_{i+1}(t_1))^{2^{2^{n-(i+1)}}}$ (by ind. hyp.)

$\overset{*}{\underset{OI}{\Rightarrow}} [(t_1)^{2^{2^{n-(i+1)}}}]^{2^{2^{n-(i+1)}}}$ (by ind. hyp)

$= (t_1)^{2^{2^{n-i}}}$

b) $F_i(t_1, t_2) \underset{OI}{\Rightarrow} F_{i+1}(G_{i+1}(t_1), F_{i+1}(t_1, t_2))$

$\overset{*}{\underset{OI}{\Rightarrow}} (G_{i+1}(t_1))^j F_{i+1}(t_1, t_2) (G_{i+1}(t_1))^{2^{2^{n-(i+1)}} - j - 1}$

for all j $0 \leq j < 2^{2^{n-(i+1)}}$ (by ind. hyp.)

$\overset{*}{\underset{OI}{\Rightarrow}} [(t_1)^{2^{2^{n-(i+1)}}}]^j t_1^k t_2 t_1^{2^{2^{n-(i+1)}} - 1 - k}$

$[(t_1)^{2^{2^{n-(i+1)}}}] \uparrow (2^{2^{n-(i+1)}} - j - 1)$

for all $0 \leq j, k < 2^{2^{n-(i+1)}}$ (by ind. hyp.)

but this last set of strings is easily seen to be equal to $\{(t_1)^j\, t_2\, (t_1) \uparrow (2^{2^{n-i}} - 1 - j) \mid 0 \leq j < 2^{2^{n-i}}\}$ which is what we want.

Now the language generated from S is the set of strings generated from $F_1(G_1(a), F_1(a,b))$ which by a) and b) contains all strings of the form $(a^{2^{2^{n-1}}})^j\, [a^k\, b\, a^{2^{2^{n-1}} - k - 1}].\ (a^{2^{2^{n-1}}}) \uparrow (2^{2^{n-1}} - j - 1)$ for all $0 \leq k, j < 2^{2^{n-1}}$. Again it is easy to see that this set is equal to $L(G_n)$.

It is left to the reader to verify (by induction) that the grammar only generates strings which have length 2^{2^n} and contain exactly one b. □

Having shown that macro grammars provide a very succinct way of describing finite sets we now generalize the notion of a macro grammar by allowing the parameters to be more complicated. As mentioned in the introduction we shall consider parameters which are functions (of higher type) and we shall view the nonterminals of the grammar as operators mapping functions to functions. In this terminology, a nonterminal in an ordinary macro grammar is a function mapping sets of strings to sets of strings and the language generated by the grammar is the minimal fixed point of the function which corresponds to the startsymbol of the grammar. The purpose of the generalization is to study the succinctness of descriptions of finite languages when these generalized grammars are used.

In order to show how grammars can be viewed as functions, we first present the characterization of OI macro languages as socalled OI(1)-equational subsets of the algebra of strings. This characterization, which is from [8], uses concepts from universal algebra, which we now introduce briefly. The reader is referred to [8] and [14] for missing definitions as well as greater detail.

Let Σ be a ranked alphabet. A Σ-algebra, A, consists of a set A called the carrier of the Σ-algebra and for each nonnegative integer n and $f \in \Sigma_n$ an operation $f_A : A^n \to A$. If $n = 0$ then f_A is a constant, i.e. $f_A \in A$. If A and B are Σ-algebras, a Σ-homomorphism $h : A \to B$ is a mapping such that 1) for each $f \in \Sigma_0$, $h(f_A) = f_B$ and 2) if $f \in \Sigma_n$ and $a_1, \ldots, a_n \in A$ then $h(f_A(a_1, \ldots, a_n)) = f_B(h(a_1), \ldots, h(a_n))$. The set of Σ-trees, T_Σ, is the smallest set of strings such that

a) each $f \in \Sigma_0$ is in T_Σ

b) if $f \in \Sigma_n$, $t_1, \ldots, t_n \in T_\Sigma$ then $f(t_1 \ldots t_n) \in T_\Sigma$

It is well known that if T_Σ is interpreted as a Σ-algebra in the obvious way (i.e. $f(t_1, \ldots, t_n) = f(t_1 \ldots t_n)$) then T_Σ is free in the class of Σ-algebras i.e. given a Σ-algebra A, there is a unique Σ-homomorphism from T_Σ to A.

Let X be a set of variables and recall that the ranked alphabet $\Sigma(X)$ is defined as follows. $\Sigma(X)_0 = \Sigma_0 \cup X$, $\Sigma(X)_i = \Sigma_i$ for $i \geq 1$. $T_\Sigma(X)$ denotes the algebra $T_{\Sigma(X)}$ which is the free Σ-algebra on generators X i.e. given a Σ-algebra A and a mapping $h : X \to A$ then there is a unique Σ-homomorphism $\bar{h}$ from $T_\Sigma(X)$ to A which extends h (i.e. $\bar{h}(x) = h(x)$ for all $x \in X$). Let t be an element of $T_\Sigma(X)$ and assume that $x_1, \ldots, x_n$ are all the variables occurring in t. The derived operation $t_A : A^n \to A$ is defined as follows (cf. [14]). For each $a_1, \ldots, a_n \in A$, $t_A(a_1, \ldots, a_n) = \bar{a}(t)$ where $\bar{a}$ is the unique homomorphism from $T_\Sigma(X)$ to A with $\bar{a}(x_i) = a_i$ for $1 \leq i \leq n$.

Example 5.8 Let Ω be the ranked alphabet with $\Omega_0 = \{a_1, \ldots, a_n, e\}$, $\Omega_2 = \{m\}$ and Ω_i empty otherwise. The string concatenation algebra W_Ω, or just W when Ω is understood, is the Ω-algebra whose carrier is the set of strings $\{a_1, \ldots, a_n\}^*$, where $a_W = a$ for each of the a's in Ω_0, $e_W = \lambda$, and where m is concatenation of strings. The unique Ω-homomorphism from T_Σ to W, which we call FRONT, maps Ω-trees to strings in the obvious way. For example $h(m(a_1 m(a_2 a_1))) = \{a_1 a_2 a_1\}$. □

From now on Ω will always be the ranked alphabet from example 5.8. Let $G = (N, \{a_1, \ldots, a_n\}, P, S)$ be a context-free grammar with nonterminals $N = \{A_1, \ldots, A_k\}$. We associate a set of equations with G in the following way. Replace in P each righthandside of the form $c_1 c_2 \ldots c_k$ by i) e if $k = 0$, ii) c_1 if $k = 1$ and iii) $m(c_1 m(c_2 \ldots m(c_{k-1} c_k) \ldots))$ when $k \geq 2$. Now the righthandsides are elements of $T_\Omega(N)$ i.e. they are Ω-trees with nonterminals as variables. Hence we can interpret the productions of the grammar as a system of equations of the form

$$(*) \qquad \begin{aligned} A_1 &= R_1 \\ &\vdots \\ A_k &= R_k \end{aligned}$$

where each R_i is a finite subset of $T_\Omega(N)$. In general a set of equations like (*) where all variables have rank 0 is called a regular set of equations. The proper domain in which to solve such a set of equations is a subset algebra, which we introduce next.

Let Σ be a ranked alphabet and let A be a Σ-algebra. The subset algebra of A, denoted by $\mathcal{P}(A)$, is the Σ-algebra whose carrier is the set of subsets of A and whose operations are defined as follows. For $f \in \Sigma_0$, $f_{\mathcal{P}(A)} = \{f_A\}$ and for $f \in \Sigma_n$ $(n > 0)$ $f_{\mathcal{P}(A)}$ is defined by $f_{\mathcal{P}(A)}(L_1,\ldots,L_n) = \{f_A(a_1,\ldots,a_n) \mid a_i \in L_i \text{ for } 1 \leq i \leq n\}$. A derived operation in A, t_A, extends to a derived operation in $\mathcal{P}(A)$ in the same way (i.e. $t_{\mathcal{P}(A)}(L_1,\ldots,L_n) = \{t_A(a_1,\ldots,a_n) \mid a_i \in L_i\}$) and a set, R, of derived operations in A define a derived operation, t^R, in $\mathcal{P}(A)$ as follows: $t^R(L_1,\ldots,L_n) = \bigcup_{t_A \in R} t_A(L_1,\ldots,L_n)$. Now any regular system of equations such as (*) can be interpreted as defining the following mapping M_G: $\mathcal{P}(A)^k \to \mathcal{P}(A)^k$

$$M_G(L_1,\ldots,L_n) = (t^{R_1}(L_1,\ldots,L_k),\ldots,t^{R_k}(L_1,\ldots,L_n))$$

Since we want to talk about M_G's minimal fixed point we need the following continuity concepts. Let S be a partially ordered set (poset) with ordering $\leq$ and minimal element $\perp$. A nonempty subset S_1 of S is called directed if any two elements of S_1 have an upper bound in S_1. S is called $\sqcup$-complete (Δ-complete) if every subset (every directed subset) S_1 has a least upper bound $\text{lub}(S_1)$ in S. If T is another poset and $f: S \to T$ then f is called $\sqcup$-continuous (Δ-continuous) if $f(\text{lub}(S_1)) = \text{lub}(f(S_1))$ for all subsets (all directed subsets) S_1 of S for which $\text{lub}(S_1)$ exists.

If the carrier of the subset algebra is ordered (partially) by normal set inclusion then it can be shown (see [8]) that the mapping M_G defined above is $\sqcup$-continuous and hence that it has a minimal fixed point $(L_1^0,\ldots,L_k^0)$. A component of this fixed point, L_i^0, is called an equational element of $\mathcal{P}(A)$ (equational subset of A).

Example 5.9 Consider the algebra W_Ω from Example 5.8. The subset algebra of W_Ω, $\mathcal{P}(W_\Omega)$ will be called the string language concatenation algebra. Its carrier consists of subsets of $\{a_1,\ldots,a_n\}^*$ and m denotes usual concatenation of sets of strings. If the set of equations (*) is solved in $\mathcal{P}(W_\Omega)$ then the equational subset of W_Ω corresponding to the i'th component in the minimal fixed point of M_G is equal to the set of strings generated by G from nonterminal A_i. Thus the first component in the fixed point is equal to L(G) (recall that A_1 is the startsymbol). □

Let Σ be a ranked alphabet and let F be a system of regular equations like (*). There is a subset algebra in which it is particularly interesting to solve E and that is the subset algebra of the free algebra T_Σ. Let A be any Σ-algebra and let h be the unique homomorphism from T_Σ to A. It is easy to see that h extends to a unique ($\sqcup$-continuous) homomorphism from $\mathcal{P}(T_\Sigma)$ to $\mathcal{P}(A)$; but then it follows (again see [8]) that we can obtain the solution to E in $\mathcal{P}(A)$ as the image under h of the solution in $\mathcal{P}(T_\Sigma)$.

Example 5.10 Let $\mathcal{P}(W_\Omega)$ be the string language concatenation algebra. The solution to (*) in $\mathcal{P}(W_\Omega)$ is the FRONT-image (cf. example 5.8) of the solution in $\mathcal{P}(T_\Omega)$. Note that the set of Ω-trees, which is the solution in $\mathcal{P}(T_\Omega)$ looks very much like the set of (normal) derivation trees of the context-free grammar G. □

Now consider macro grammars as defined in Definition 5.2 Again we want to view a grammar as a set of equations, but this time we have the additional complication that the unknowns (the nonterminals) are allowed to have rank different from 0, i.e. they occur in "the middle" of terms which

are righthandsides of productions. The way to handle this is, as shown by Downey in [6] , to view the unknowns as ranging over functions from sets of strings to sets of strings rather than sets of strings as in the case of regular equations. The meaning of an occurrence of a nonterminal with rank different from 0 is then that the function associated with this nonterminal has to be composed with the function(s) occurring as argument(s). Hence the solution (or the minimal fixed point) of the equation obtained from a macro grammar is a tuple of functions mapping sets of strings to sets of strings. In particular some of the components of this tuple can be just a set of strings which is then viewed as a constant function.

Example 5.11 Consider the following macro grammar

$$
\begin{aligned}
S &\rightarrow F(A) \mid G(\lambda, B)\\
F(x_1) &\rightarrow x_1 x_1\\
A &\rightarrow a \mid b\\
G(x_1, x_2) &\rightarrow x_1 x_2\\
B &\rightarrow B
\end{aligned}
$$

The solution to the set of equations we intend to associate with this grammar will be the following five tuple $(S^\circ, F^\circ, G^\circ, A^\circ, B^\circ)$ where $S^\circ = \{ab, ba, aa, bb\}$, $A^\circ = \{a, b\}$, $B^\circ = \emptyset$ and F° and G° are the following functions

$$
\begin{aligned}
\forall L_1, L_2 \subseteq \{a,b\}^* : \quad & F^\circ(L_1, L_2) = L_1 \cdot L_1\\
- \text{ " } - \quad : \quad & G^\circ(L_1, L_2) = L_1 \cdot L_2
\end{aligned}
$$

□

As we want the solutions to macro equations to be elements of appropriate algebras we have to take care of the problem that the nonterminals have different rank. This is handled with the use of heterogeneous (or many sorted) algebras where it is possible to have more than one carrier. We briefly introduce the necessary concepts.

Let S be a set (of sorts). An S-sorted alphabet Σ is an indexed family $<\Sigma_{w,s}>_{<w,s>\in S^*\times S}$ of disjoint sets. A symbol $f \in \Sigma_{w,s}$ is called an operator of type $<w,s>$. If $w = \lambda$ then f is a constant of sort s. Let Σ be an S-sorted alphabet. A Σ-algebra A consists of a family $<A_s>_{s\in S}$ of sets called the carriers of A (A_s is the carrier of sort s) and for each $<w,s> \in S^* \times S$ and each $f \in \Sigma_{w,s}$ an operation $f_A: A_{s_1} \times \ldots \times A_{s_n} \to A_s$ where $s_1 \ldots s_n = w$. If A and B are Σ-algebras, a Σ-homomorphism is an indexed family $<h_s>_{s\in S}$ of mappings $h_s: A_s \to B_s$ such that 1) if $f \in \Sigma_{\lambda,s}$ then $h(f_A) = f_B$ and 2) is $f \in \Sigma_{s_1 \ldots s_n, s}$ and $a_i \in A_{s_i}$ for $1 \leq i \leq n$, then $h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_s(a_n))$.

The set of Σ-trees, T_Σ, is the smallest family $<T_{\Sigma,s}>_{s\in S}$ which satisfies

i) for $s \in S$: $\Sigma_{\lambda,s} \subseteq T_{\Sigma,s}$

ii) for $n \geq 1$ and $s, s_1, \ldots, s_n \in S$ if $f \in \Sigma_{s_1 \ldots s_n, s}$ and $t_i \in T_{\Sigma,s_i}$ for $1 \leq i \leq n$, then $f(t_1 \ldots t_n) \in T_{\Sigma,s}$

Variables are introduced in the following way. Let $X = <X_s>_{s\in S}$ be a family of disjoint sets and let $\Sigma(X)$ be the S-sorted alphabet where for each $s \in S$, $\Sigma(X)_{\lambda,s} = \Sigma_{\lambda,s} \cup X_s$ and for each $w \neq \lambda$ and $s \in S$, $\Sigma(X)_{w,s} = \Sigma_{w,s}$. Now, $T_\Sigma(T_\Sigma(X))$ is the Σ-algebra with $T_{\Sigma,s}$ ($T_{\Sigma(X),s}$) as carrier

of sort s and operations defined in the obvious way. As in the ranked case, T_Σ ($T_\Sigma(X)$) is the free algebra in the class of Σ-algebras (the class of Σ-algebras with generators X). Finally for a Σ-algebra A, the subset algebra $\mathcal{P}(A)$ is defined in a way completely analogous to the ranked case.

Now we can start building the heterogeneous algebra we are interested in. First we introduce the socalled derived alphabet (Maibaum, [25]) of a ranked alphabet. This alphabet will be used extensively in the following.

Definition 5.12 Let Σ be a ranked alphabet and let N be the set of non-negative integers. The derived alphabet of Σ, $D(\Sigma)$, is the following N-sorted alphabet.

Let, for each $n \geq 0$, $\overline{\Sigma}_n = \{\overline{f} \mid f \in \Sigma_n\}$ be a new set of symbols; let for each $n \geq 1$ and $1 \leq i \leq n$, π_i^n be a new symbol; and let, for each $n \geq 0$, $k \geq 0$ $c_{n,k}$ be a new symbol. Then

i) $D_{\lambda,0} = \overline{\Sigma}_0$

ii) for $n \geq 1$, $D_{\lambda,n} = \overline{\Sigma}_n \cup \{\pi_i^n \mid 1 \leq i \leq n\}$

iii) for $n,k \geq 0$, $D_{\underbrace{nkk\ldots k}_{n \text{ times}},k} = \{c_{n,k}\}$

iv) $D_{w,s} = \emptyset$ otherwise □

The symbols of $D(\Sigma)$ will be used as operators in $D(\Sigma)$-algebras whose carriers are functions. The π_i^n's will denote projections and the $c_{n,k}$'s will be used as composition symbols.

First we define the appropriate algebra for solving macro equations. The carrier(s) will consist of functions from sets of strings to sets of strings, and the $c_{n,k}$'s will denote composition of such functions. Since we want the solutions to be the same as the functions computed by the grammar under OI-derivation, our algebra has to reflect the fact that the "OI-composition" of functions is not $\sqcup$-continuous but only Δ-continuous (see [8] for a more detailed discussion of this).

Definition 5.13 Let $D(\Omega)$ be the derived alphabet of Ω. $\mathfrak{F}_\Delta(\mathcal{P}(W))$, the $D(\Omega)$ algebra of Δ-continuous functions over $\mathcal{P}(W)$ is defined as follows

i) for $i \geq 0$, $\mathfrak{F}_\Delta(\mathcal{P}(W))_i$ is the set of all Δ-continuous functions $\mathcal{P}(W)^i \to \mathcal{P}(W)$, in particular $\mathfrak{F}_\Delta(\mathcal{P}(W))_0 = \mathcal{P}(W)$

ii) for $1 \leq i \leq n$, $\bar{a}_i = \{a_i\}$

iii) $\bar{m}$ = concatenation in $\mathcal{P}(W)$

iv) for $i \geq 1$, $1 \leq j \leq i$ and $L_1, \ldots, L_i$ in $\mathcal{P}(W)$
$\pi^i_j(L_1, \ldots, L_i) = L_j$

v) for $i, j \geq 0$, $f \in \mathfrak{F}_\Delta(\mathcal{P}(W))_i$, $g_1, \ldots, g_i \in \mathfrak{F}_\Delta(\mathcal{P}(W))_j$
$c_{i,j}(f, g_1, \ldots, g_i) = f \circ (g_1, \ldots, g_i)$ □

Now recall the way in which we solved regular Ω-equations in the Ω-algebra $\mathcal{P}(W)$ and recall also that the solutions were called equational elements of $\mathcal{P}(W)$. It is easy to show that we can use exactly the same approach here, i.e. we can solve regular $D(\Omega)$-equations in the $D(\Omega)$-algebra $\mathfrak{F}_\Delta(\mathcal{P}(W))$, thus obtaining equational elements of $\mathfrak{F}_\Delta(\mathcal{P}(W))$. Note that in particular elements of $\mathfrak{F}_\Delta(\mathcal{P}(W))_0$ $(= \mathcal{P}(W))$ can be equational.

Definition 5.14 An element of $\mathfrak{P}(W)$ is said to be OI(1)-equational if it is equational as an element of $\mathfrak{F}_{\Delta}(\mathfrak{P}(W))$. □

The following theorem is one of the main results in [8].

Theorem 5.15 (Engelfriet and Schmidt) The OI(1)-equational subsets of $\mathfrak{P}(W_{\Omega})$ are exactly the OI macro languages over the alphabet $\{a_1, \ldots, a_n\}$. □

We shall not attempt to prove this theorem, rather we show in the following example how the language generated by the grammar in Example 5.11 is obtained as an equational element of $\mathfrak{P}(W)$.

Example 5.16 Let Ω' be the ranked alphabet where $\Omega'_0 = \{a, b, e\}$, $\Omega'_2 = \{m\}$ and $\Omega'_i = \emptyset$ otherwise. The derived alphabet $D(\Omega')$ consists of the symbols $\bar{a}$, $\bar{b}$, $\bar{e}$, $\bar{m}$ together with projection- and composition symbols. Consider the following set, E, of regular $D(\Omega')$ equations.

$$S = \begin{array}{ccc} & c_{1,0} & \\ \swarrow & & \searrow \\ F & & A \end{array} \;\Big|\; \begin{array}{ccc} & c_{2,0} & \\ \swarrow & | & \searrow \\ G & \bar{e} & B \end{array}$$

$$F = \begin{array}{ccc} & c_{2,1} & \\ \swarrow & | & \searrow \\ \bar{m} & \pi^1_1 & \pi^1_1 \end{array}$$

$$A = \bar{a} \mid \bar{b}$$

$$G = \begin{array}{ccc} & c_{2,2} & \\ \swarrow & | & \searrow \\ \bar{m} & \pi^2_1 & \pi^2_2 \end{array}$$

$$B = \begin{array}{ccc} & c_{2,0} & \\ \swarrow & | & \searrow \\ \bar{m} & \bar{a} & B \end{array}$$

If E is solved in $\mathfrak{F}_{\Delta}(\mathcal{P}(W_{\Omega'}))$ then we obtain exactly the solution $(S^{\circ}, F^{\circ}, G^{\circ}, A^{\circ}, B^{\circ})$ from Example 5.11. The righthandsides are derived operations whose meaning follow from Definition 5.13. Thus, for example, the righthandside for G, $c_{2,2}(\bar{m}\,\pi_1^2\,\pi_2^2)$ is interpreted in the following way. The function of two variables $\bar{m}$ (which, by the way, is concatenation) is composed with the two projection functions π_1^2 and π_2^2. c's indices 2,2 express that $\bar{m}$ is a function of two arguments and that the result of the whole thing also has two arguments. Take as another example the second righthandside of S, $c_{2,0}(G\ \bar{e}\ B)$. Here $c_{2,0}$ expresses that G ranges over functions of two variables and that we compose with two constants, thereby getting a constant back. □

In view of Theorem 5.15 it is hardly surprising that there is a systematic way of transforming any macro grammar into an equivalent set of regular $D(\Omega)$ equations. Again the reader is referred to [8] for details.

The characterization of macro languages in Theorem 5.15 is not strong enough to be really useful, because it does not specify what the solutions to systems of regular equations look like. In the case of regular Ω-equations we saw how to solve in $\mathcal{P}(W)$ by solving in $\mathcal{P}(T_{\Omega})$ and then taking the FRONT-image to get a set of strings in $\mathcal{P}(W)$. We would like to do something similar in this case, i.e. we would like to get the solution to a regular $D(\Omega)$ equation in $\mathfrak{F}_{\Delta}(\mathcal{P}(W))$ as the homomorphic image of its solution in an algebra whose elements are some kind of $D(\Omega)$-trees. First we need the following definition.

Definition 5.17 Let Σ be a ranked alphabet and let $D(\Sigma)$ be the derived alphabet. The tree-substitution $D(\Sigma)$-algebra $DT_{\Sigma}(X)$ is defined as follows.

The domain of sort n is $T_\Sigma(X_n)$. For $f \in \Sigma_n$, $\bar{f}$ is the tree $f(x_1 \ldots x_n)$ (for $f \in \Sigma_0$, $\bar{f} = f$). For $n \geq 1$ and $1 \leq i \leq n$, $\pi_i^n = x_i$; for $n \geq 0$, $k \geq 0$, $t \in T_\Sigma(X_n)$ and $t_1, \ldots, t_n \in T_\Sigma(X_k)$, $c_{n,k}(t, t_1, \ldots, t_n) = t[t_1, \ldots, t_n]$ i.e. the result of substituting t_i for x_i in t. □

If we denote by YIELD the unique homomorphism from the (free) $D(\Sigma)$-algebra $T_{D(\Omega)}$ to $DT_\Omega(X)$ then an intuitively appealing way of obtaining equational subsets of $\mathfrak{F}_\Delta(\mathcal{P}(W))$ is the following. Solve the regular $D(\Omega)$-equations in the subset algebra $\mathcal{P}(T_{D(\Omega)})$ thereby obtaining a regular set of $D(\Omega)$-trees, and then take the YIELD-image of this set of trees. This results in a set of Ω-trees from which we get a set of strings by application of FRONT. Unfortunately, this does not work because, as shown in [8], we end up with the IO-language rather than the OI-language which we are interested in. The mathematical reason for this is that the subset algebra $\mathcal{P}(T_{D(\Omega)})$ is free in the "wrong" class of $D(\Omega)$-algebras, namely in the class of algebras where all operations and homomorphisms are $\sqcup$-continuous. As we have already mentioned, the OI-mode of derivation leads to an operation of function composition which is Δ-continuous but not $\sqcup$-continuous. Hence, instead of $\mathcal{P}(T_{D(\Omega)})$ we want an algebra which is free in the class of $D(\Omega)$-algebras with Δ-continuous homomorphisms. Such an algebra was defined in [14] and we now briefly introduce the necessary concepts. The reader is referred to [14] for further details.

Let Σ be an S-sorted alphabet and assume that the symbol $\perp$ is in $\Sigma_{\lambda,s}$ for all s. CT_Σ is the set of all finite and infinite trees labeled by Σ such that a node labeled with a symbol in $\Sigma_{s_1 \ldots s_n, s}$ has exactly n

successors which are roots of subtrees of type $s_1, s_2, \ldots, s_n$. CT_Σ is ordered (partially) by a "subtree-ordering" in which $\perp$ is the minimal element (see [14]). When viewed as a Σ-algebra CT_Σ is free in the class of Σ-algebras with Δ-continuous and $\perp$-preserving homomorphisms. For an S-sorted set of variables X, $CT_\Sigma(X)$ is the free Σ-algebra on generators X. We can view $CT_\Sigma(X)$ as a $D(\Sigma)$-algebra in the same way as $DT_\Sigma(X)$ above, the only difference being that $c_{n,k}$ is now substitution of infinite trees. The unique homomorphism from $CT_{D(\Sigma)}$ to $CT_\Sigma(X)$ will again be called YIELD.

The relevance of infinite trees to our problem is the following. Given a regular set of $D(\Omega)$ equations like in Example 5.16 we view it as a "deterministic" set of equations by introducing the binary operator + and replacing righthandsides like $r_1 | r_2 \ldots | r_k$ by $+(r_1 + (r_2 \ldots + (r_{k-1} r_k) \ldots))$. Formally we add + to the derived alphabet $D(\Omega)$ as a new element of sort 2 and denote the resulting alphabet by $D(\Omega)^+$. Then the solution in $CT_{D(\Omega)^+}$ to a regular system of $D(\Omega)^+$-equations of the form

$$\{ A_i = t_i \}_{i=1}^{k}$$

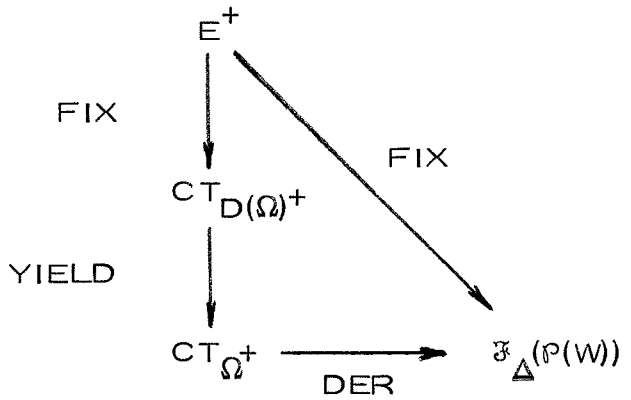
is the k-tuple of infinite trees obtained as follows. Start with $(A_1, \ldots, A_k)$ and replace the A_i's by their righthandsides, then replace all A_i's by their righthandsides, and so on. For obvious reasons a tree of this type is called a <u>regular infinite tree</u>. If we apply YIELD to such an infinite tree we obtain as result another infinite tree in the $D(\Omega)^+$-algebra $CT_\Omega+(X)$. In this last infinite tree the internal nodes are labeled by m or + and the leaves by $\perp$, $a_1, \ldots, a_n$, e, or variables from X. If + is interpreted as union and m as concatenation of string languages, the tree defines a derived operation in $\mathfrak{F}_\Delta(\mathcal{P}(W))$, which is equal to (a component of) the solution in $\mathfrak{F}_\Delta(\mathcal{P}(W))$ to the original set of regular equations. Formally, let DER be

the unique Δ-continuous, $\perp$-preserving $D(\Omega)^+$-homomorphism which maps trees in $CT_{\Omega^+}(X)$ to derived operations in $\mathfrak{F}_\Delta(\mathcal{P}(W))$. We have the following theorem from [8].

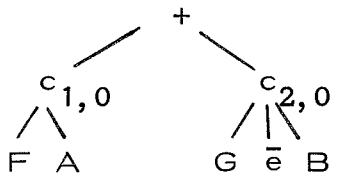
Theorem 5.18 Let $E = \{A_i = R_i\}_{i=1}^k$ be a system of regular $D(\Omega)$-equations and let $E^+ = \{A_i = r_i\}_{i=1}^k$ be the corresponding "deterministic" $D(\Omega)^+$ equations. If $(F^\circ_1, \ldots, F^\circ_k)$ is the solution of E in $\mathfrak{F}_\Delta(\mathcal{P}(W_\Omega))$ and $t^\infty_1, \ldots, t^\infty_k$ are the infinite trees obtained from $A_1, \ldots, A_k$ by successively replacing A_i's with r_i's then we have for $1 \leq i \leq k$,

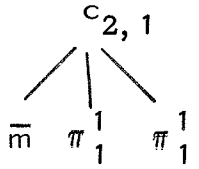
$$F^\circ_i = DER \circ YIELD\ (t^\infty_i)$$ □

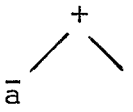
This can be summarized in the statement that the following diagram is commutative. FIX denotes the operation of solving the equation E^+ (finding its minimal fixed point).

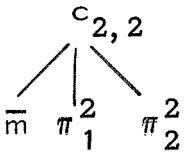


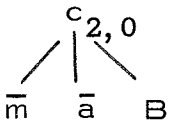
Example 5.19 Consider the regular set of equations E in Example 5.16. The corresponding "deterministic" $D(\Omega)^+$ system is

$S =$ 

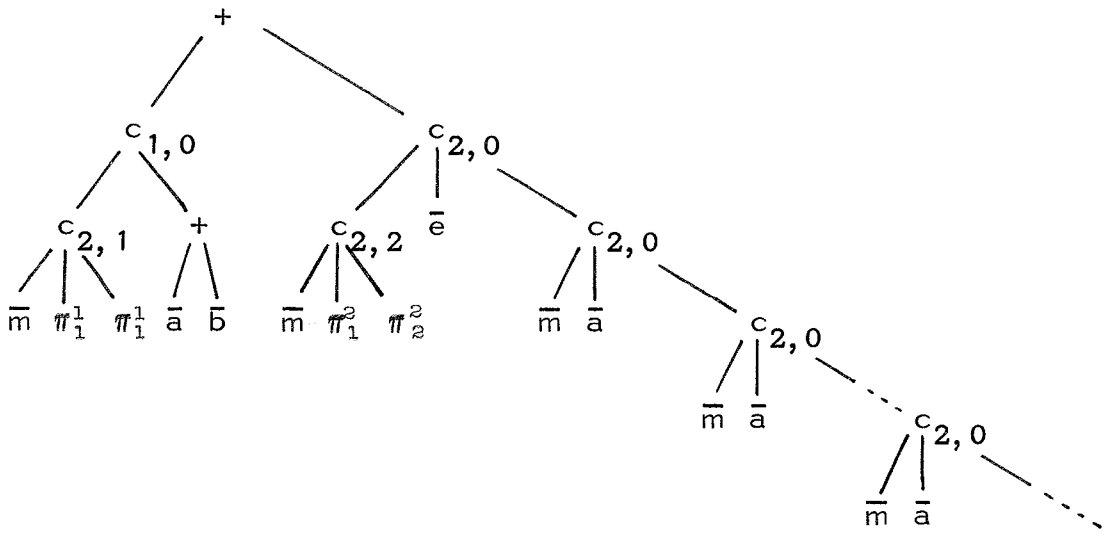
$F =$ 

$A =$ 

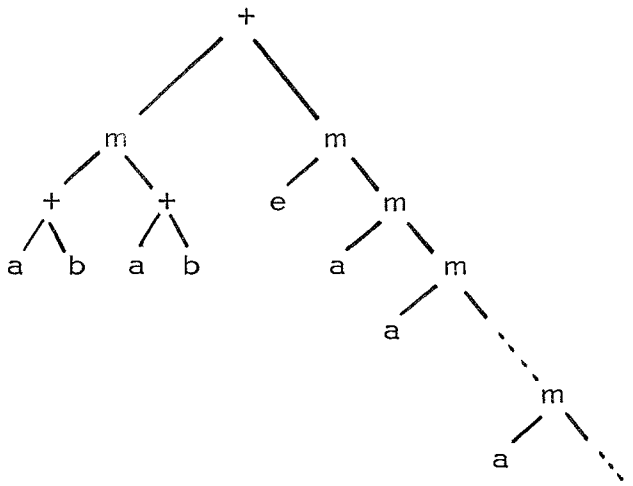
$G =$ 

$B =$ 

and the infinite tree corresponding to S is



If YIELD is applied to this tree we obtain the following tree in $CT_{\Omega^+}(X)$



Finally application of DER results in the set of strings $\{aa, ab, ba, bb\}$.

□

Having seen how to obtain OI macro languages as equational subsets of $\mathfrak{F}_\Delta(\mathcal{P}(W))$ or, equivalently, as images under DER$\circ$ YIELD of infinite regular trees, we are now ready to define the OI-hierarchy. The first family of languages in the hierarchy (after the OI-languages) is obtained by solving sets of regular equations in an algebra whose elements are operators. Recall the definition of $\mathfrak{F}_\Delta(\mathcal{P}(W))$, where the π's and c's in $D(\Omega)$ were associated with projection and composition of functions. If we could derive the alphabet once more, then we would obtain some new π's and c's which could be interpreted as projection and composition of operators. This is done as follows (note that so far we have only defined derivation of ranked alphabets and that we now define it for arbitrary many sorted alphabets).

Definition 5.20 Let Σ be an S-sorted alphabet. The derived $(S^* \times S)$-sorted alphabet $D(\Sigma)$ is obtained as follows. Let, for each $\langle w, s\rangle \in S^* \times S$ and each $f \in \Sigma_{w,s}$, $\overline{f}$ be a new symbol; let for each $w \in S^*$ $(w \neq \lambda)$ and each i, $1 \leq i \leq |w|$, π_i^w be a new symbol; and let for each $w, v \in S^*$, $s \in S$, $c_{w,v,s}$ be a new symbol. Then $D(\Sigma)$ consists of these symbols with their types (elements of $(S^* \times S)^* \times (S^* \times S)$) specified as follows

i) for $f \in \Sigma_{w,s}$, $\overline{f}$ has type $\langle \lambda, \langle w, s\rangle\rangle$

ii) π_i^w has type $\langle \lambda, \langle w, w_i\rangle\rangle$

iii) $c_{w,v,s}$ has type $\langle\langle w, s\rangle\langle v, w_1\rangle \ldots \langle v, w_n\rangle, \langle v, s\rangle\rangle$. □

We also need the following generalization of Definition 5.13.

Definition 5.21 Let Σ be an S-sorted alphabet and B a Δ-continuous Σ-algebra with $\sqcup$-complete carriers. The $D(\Sigma)$-algebra of Δ-continuous functions over B, denoted by $\mathfrak{F}_\Delta(B)$, is defined as follows.

i) the domain of sort $\langle s_1 \ldots s_n, s\rangle$ is the set of all Δ-continuous functions from $B_{s_1} \times \ldots \times B_{s_n} \to B_s$

ii) for $f \in \Sigma_{w,s}$, $\overline{f} = f_B$

iii) for $w = w_1 \ldots w_n$ and $b_1 \in B_{w_1}, \ldots, b_n \in B_{w_n}$, $\pi_i^w(b_1, \ldots, b_n) = b_i$

iv) for $w = w_1 \ldots w_n$, $f \in \mathfrak{F}_\Delta(B)_{\langle w, s\rangle}$, $g_1 \in \mathfrak{F}_\Delta(B)_{(v, w_1)} \ldots$ $g_n \in \mathfrak{F}_\Delta(B)_{\langle v, w_n\rangle}$, $c_{w,v,s}(f, g_1, \ldots, g_n) = f \circ (g_1, \ldots, g_n)$ □

Since it is easy to see that for any Δ-continuous Σ-algebra with $\sqcup$-complete carriers B, $\mathfrak{F}_\Delta(B)$ is a Δ-continuous $D(\Sigma)$-algebra with $\sqcup$-complete carriers, we can iterate both the process of deriving alphabets and the process of constructing function algebras. We define the n'th derived alphabet $D^n(\Sigma)$ inductively as follows, 1) $D^0(\Sigma) = \Sigma$, and 2) for $n \geq 1$, $D^n(\Sigma) = D(D^{n-1}(\Sigma))$. The n'th iterate function algebra of the Σ-algebra B is the $D^n(\Sigma)$-algebra $\mathfrak{F}^n_\Delta(B)$ defined by 1) $\mathfrak{F}^0_\Delta(B) = B$, and 2) for $n \geq 1$, $\mathfrak{F}^n_\Delta(B) = \mathfrak{F}_\Delta(\mathfrak{F}^{n-1}_\Delta(B))$.

Now the formal definition of the OI-hierarchy is the following.

Definition 5.22 A set of strings in $\mathcal{P}(W)$ is OI(n)-equational if it is (a component of) the solution of a system of regular $D^n(\Omega)$ equations in $\mathfrak{F}^n_\Delta(\mathcal{P}(W))$. □

Again we have a "useful" characterization of these sets. Recall that $CT_{D^n(\Omega)^+}$ consists of infinite $D^n(\Omega)^+$-trees and let for each $n \geq 1$, YIELD denote the unique Δ-continuous $\perp$-preserving homomorphism from $CT_{D^n(\Omega)^+}$ to $CT_{D^{n-1}(\Omega)^+}$. The following theorem was proved in [8].

Theorem 5.23 A set of strings in $\mathcal{P}(W)$ is OI(n)-equational if and only if it is equal to DER $\circ$ YIELD$^n(t^\infty)$ where t^∞ is an infinite regular $D^n(\Omega)^+$-tree of appropriate sort. □

Example 5.24 Let Σ be the ranked alphabet with $\Sigma_0 = \{a, b\}$, $\Sigma_2 = \{m\}$ and $\Sigma_n = \emptyset$ otherwise. Σ can be viewed as an S-sorted alphabet where S is a singleton ($S = \{s\}$). Then a, b and m have the following types

a, b has type $\langle \lambda, s \rangle$
m " " $\langle ss, s \rangle$

Let $D(\Sigma)$ be the derived alphabet of Σ according to Definition 5.12. $D(\Sigma)$ can be viewed as sorted by $(S^* \times S)^* \times (S^* \times S)$. The elements then have the following types.

$\bar{a}, \bar{b}$ has type $\langle \lambda, \langle \lambda, s \rangle \rangle$

$\bar{m}$ " " $\langle \lambda, \langle ss, s \rangle \rangle$

$\bar{\pi}_i^n$ " " $\langle \lambda, \langle s^n, s \rangle \rangle$

$c_{n,k}$ " " $\langle \langle s^n, s \rangle \langle s^k, s \rangle \ldots \langle s^k, s \rangle, \langle s^k, s \rangle \rangle$

The derived alphabet of the manysorted alphabet $D(\Sigma)$, $D(D(\Sigma))$, obtained by Definition 5.20 consists of the following elements with types as specified.

$\bar{\bar{a}}, \bar{\bar{b}}$ with type $\langle \lambda, \langle \lambda, \langle \lambda, s \rangle \rangle \rangle$

$\bar{\bar{m}}$ " " $\langle \lambda, \langle \lambda, \langle ss, s \rangle \rangle \rangle$

$\bar{\bar{\pi}}_i^n$ " " $\langle \lambda, \langle \lambda, \langle s^n, s \rangle \rangle \rangle$

$\bar{c}_{n,k}$ " " $\langle \lambda, \langle \langle s^n, s \rangle \langle s^k, s \rangle \ldots \langle s^k, s \rangle, \langle s^k, s \rangle \rangle \rangle$

$\pi_i^{n_1 \ldots n_k}$ " " $\langle \lambda, \langle \langle s^{n_1}, s \rangle \ldots \langle s^{n_k}, s \rangle, \langle s^{n_i}, s \rangle \rangle \rangle$

$c_{n_1 \ldots n_k, m_1 \ldots m_h, n_0}$ " "
$\langle \langle \langle s^{n_1}, s \rangle \ldots \langle s^{n_k}, s \rangle, \langle s^{n_0}, s \rangle \rangle$
$\langle \langle s^{m_1}, s \rangle \ldots \langle s^{m_h}, s \rangle, \langle s^{n_1}, s \rangle \rangle$
$\vdots$
$\langle \langle s^{m_1}, s \rangle \ldots \langle s^{m_h}, s \rangle, \langle s^{n_k}, s \rangle \rangle,$
$\langle \langle s^{m_1}, s \rangle \ldots \langle s^{m_h}, s \rangle, \langle s^{n_0}, s \rangle \rangle \rangle.$

□

Consider the $D(D(\Omega)$-algebra $\mathfrak{F}_\Delta(\mathfrak{F}_\Delta(\mathcal{P}(W)))$. Here $\pi_i^{n_1 \ldots n_k}$ is interpreted as the operator which takes as argument k functions (with $n_1, n_2, \ldots, n_k$ arguments) and returns as result a function with n_i arguments. $c_{n_1 \ldots n_k, m_1 \ldots m_h, n_0}$ is interpreted in a similar fashion.

Now we want to study how succinctly the regular equations "in the OI-hierarchy" can describe finite sets. First we present in the following example three systems of regular equations of similar size (over Σ, $D(\Sigma)$ and $D(D(\Sigma))$) whose solutions in $\mathcal{P}(W_\Sigma)$, $\mathfrak{F}_\Delta(\mathcal{P}(W_\Sigma))$ and $\mathfrak{F}_\Delta(\mathfrak{F}_\Delta(\mathcal{P}(W_\Sigma)))$ are of different magnitudes. The intuition behind the examples is as follows. Consider the set of regular Σ-equations E_n.

E_n:

$S = m(A_1, A_1)$

$\vdots$

$A_i = m(A_{i+1}, A_{i+1})$

$\vdots$

$A_n = +(a, b)$.

It is clear that the S-component of E_n's solution in $\mathcal{P}(W_\Sigma)$ is equal to $L_0 = \{w \in \{a,b\}^* \mid |w| = 2^n\}$. Now, let d be the one-argument function which "squares" its argument (one representation of d is $d(x) = m(xx)$). Intuitively, if in E_n we replace m by some appropriate composition symbol from $D(\Sigma)$, and $+(a, b)$ by the function d, then we should be able to generate the function d^{2^n}. Hence the following system of equations (where the sort of the c's has been left unspecified) should generate the language $L_1 = \{w \in \{a,b\}^* \mid |w| = 2^{2^n}\}$ $(= d^{2^n}(\{a,b\}))$.

$D(E_n)$:

$S = c(c(A_1, A_1), +(\bar{a}, \bar{b}))$

$\vdots$

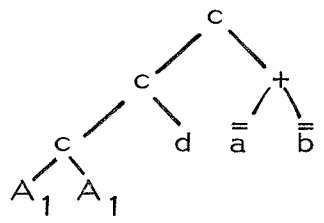
$A_i = c(A_{i+1}, A_{i+1})$

$\vdots$

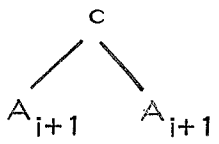
$A_n = d$

Similarly, if in $D(E_n)$ c is replaced by composition of operators and d by the operator D, which composes its argument with itself ($D(x) = x \circ x$) then we should be able to obtain the set $L_2 = \{w \in \{a,b\}^* \mid |w| = 2^{2^{2^n}}\}$ in the following way

$D(D(E_n))$: $S =$



$A_i =$



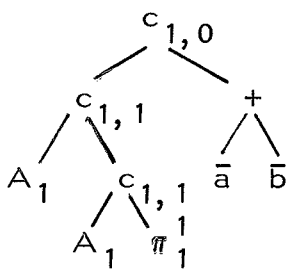
$A_n = D$

The next example shows how this is done formally.

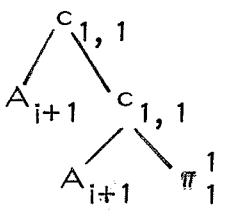
Example 5.25

a) L_1 is the S-component of the solution in $\mathfrak{F}_\Delta(\mathcal{P}(W_\Sigma))$ to the following system of regular $D(\Sigma)$-equations

$S =$

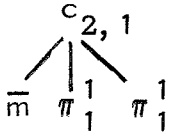


$A_i =$



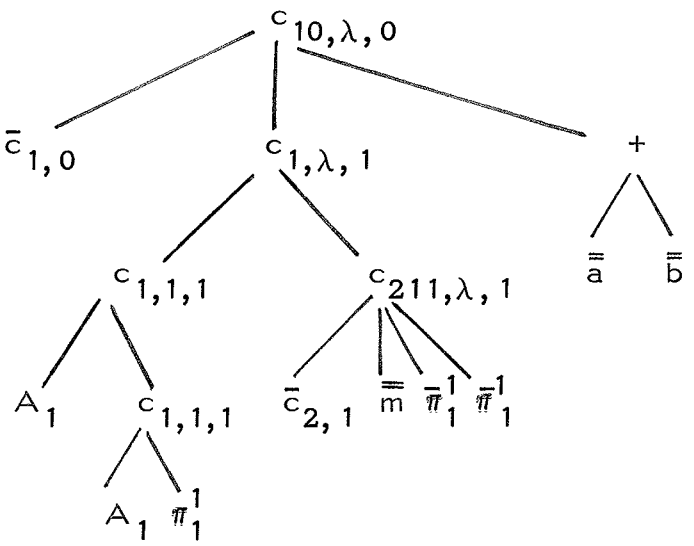
.
.

A_n =



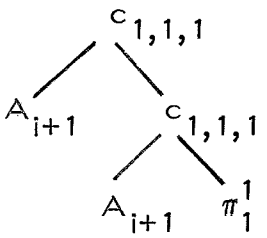
b) L_2 is obtained as the S-component of the solution in $\mathfrak{F}_\Delta(\mathfrak{F}_\Delta(\mathcal{P}(W_\Sigma)))$ to the following system of regular $D(D(\Sigma))$ equations

S =



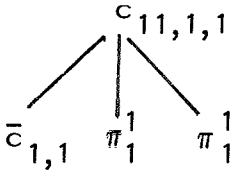
.
.
.

A_i =



.
.
.

A_n =



□

By a straightforward generalization of the technique used in Example 5.25 we can prove the following lemma.

<u>Lemma 5.26</u> For each $k \geq 1$, the set $\{w \in \{a,b\}^* \mid |w| = 2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}\}$ (k) is OI(k)-equational. Furthermore, it is equal to $DER \circ YIELD^k(t)$ where t is a component of the solution in $CT_{D^k(\Omega)+}$ to a system of regular $D^k(\Omega)$-equations of size O(n). □

Now recall that the reason the set F_0 (F_1) in the beginning of this section was hard for nondeterministic exponential (double exponential) time was the ability to generate $\{w \in \{a,b\}^* \mid |w| = 2^n\}$ ($\{w \in \{a,b\}^* \mid |w| = 2^{2^n}\}$) by a context-free (macro) grammar of size O(n). Lemma 5.26 suggests that these results might be generalized and in order to do that we need the following notation.

Let $E_0(n) = 2^n$ and let for each $k \geq 1$, $E_k(n) = 2^{E_{k-1}(n)}$. Let R(k) be the class of regular $D^k(\Omega)$-equations and assume that each equation has a designated unknown S which is of type $\overbrace{\langle\lambda,\langle\lambda,\ldots,\lambda\rangle,\langle\lambda,s\rangle\ldots\rangle}^{k}$. Let L(G) be the component of the solution to G in $\mathfrak{F}^k_\Delta(P(W))$ which corresponds to S. Let R(k)FIN be the class of equations G, for which L(G) is finite, and consider for each $k \geq 1$, the set F_k and the complexity class NEXP(k) defined as follows

$$F_k = \{(D,E) \in R(k)FIN \times IE \mid L(G) \neq \Omega_0^{**}(E)\}$$

$$NEXP(k) = \bigcup_{i=0}^{\infty} NTIME(E_k(n^i))$$

It was shown in Theorem 5.6 that F_1 is hard for NEXP(1) under $\leq_p$, and we would like to use the same method to show that F_k is hard for NEXP(k), also under $\leq_p$. The generalization of the proof works well

except that here we can't use sizeclosure † under intersection with regular sets. It is an open problem whether OI(k)-equational sets are closed under intersection with regular languages. However we can prove the following theorem.

Theorem 5.27 If the OI(k)-equational sets are sizeclosed under intersection with regular sets then F_k is hard for NEXP(k) under $\leq_p$.

Proof

The proof is exactly the same as the proof of Theorem 5.6 with the exception that now the running time of the Turing Machine for which we are generating invalid computations is $T(n) = E_k(p(n))$ for some polynomial $p(n)$ (rather than $E_1(p(n))$). It follows from the proof of Theorem 5.6 that the essential point is the ability to obtain the following sets as solutions to equations of size $O(n)$

$$S_{p(n)} = \{A^jBA^{T(n)-1-j} \mid 0 \leq j \leq T(n)-1\}$$
$$R_{p(n)} = \{A^jBA^{T(n)-3-j} \mid 0 \leq j \leq T(n)-3\}$$
$$U_{p(n)} = \{C^{n+1}A^jBA^{T(n)-(n+1)-j} \mid 0 \leq j \leq T(n)-(n+1)\}$$

Now from Lemma 5.26 and the assumption that OI(k)-equational sets are sizeclosed under intersections with regular sets it follows immediately that $S_{p(n)}, R_{p(n)}$ and $U_{p(n)}$ are indeed obtainable as solutions to "small" $D^k(\Omega)$-equations.

We can also simulate the use of these sets as sentential forms in the following way. Recall that in Theorem 5.6 the words in (for example) $S_{p(n)}$

---

† See the remark following Theorem 5.6.

were used to generate invalid computations containing a transition error. First $A^jBA^{T(n)-j-1}$ was generated and then the A's generated $(\Sigma \cup \{\lambda\})^{**}$ $(T(n)+1)$ and B generated the pair of consecutive configurations containing the error. We can obtain the same effect here by adding a new set of equations in which the A-component of the solution is $(\Sigma \cup \{\lambda\})^{**}$ $(T(n)+1)$ and the B-component is the set of consecutive configurations in error. That this works is a consequence of the characterization of the OI(k)-equational sets as DER $\circ$ YIELD^k of solutions in $CT_{D^k(\Omega)+}$. Applying DER last in the "evaluation" corresponds to first generating the sentential forms and then expanding the remaining nonterminals. □

We note in passing that we can obtain the set $S_{p(n)}$ in a way similar to what we did for macro grammars, thus the assumption about sizeclosure with regular sets is needed only to obtain $R_{p(n)}$ and $U_{p(n)}$ and it is entirely possible that they also can be defined "directly", i.e. without use of the assumption. Note also that if the OI(k)-equational sets can be defined by some class of natural machines with a finite control, then the assumption of the theorem is satisfied. The author conjectures that there is such a machine model for the OI(k)-equational sets and hence that the sets F_k are hard for the classes NEXP(k). Finally we note that if the macro grammars of level k from [24] are equivalent to regular $D^k(\Omega)$-equations then results from [24] imply that the assumption in Theorem 5.27 is satisfied.

We close this chapter by pointing out that for $k \geq 1$ it is an open question whether the sets F_k are in the classes NEXP(k). Recall that we showed (in Theorem 5.1) that F_0 is complete for NEXP(0). The proof was based on the pumping lemma for context-free languages and on the

existence of polynomial time algorithms for parsing context-free grammars. Although there exists a pumping lemma for OI macro languages (see [17]) there are no efficient parsing method for the grammars and higher up in the hierarchy neither pumping lemmas nor efficient parsing methods are available.

References

1. Aho, A.V. [1968]. "Indexed grammars - an extension of context-free grammars", J.ACM 15:4, 647-671.

2. Aho, A.V. [1969]. "Nested Stack Automata", J.ACM 16:3, 383-406.

3. Aho, A.V., J.E. Hopcroft and J.D. Ullman [1974]. The design and analysis of computer algorithms, Addison-Wesley, Reading, Massachusetts.

4. Aho, A.V. and J.D. Ullman [1972]. The Theory of Parsing, Translation and Compiling, Volume 1: Parsing, Prentice-Hall, Englewood Cliffs, N.J.

5. Constable, R.L. [1977]. "On the theory of programming logics", Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, 269-285.

6. Downey, P.J. [1974]. "Formal languages and recursion schemes", Ph.D. Thesis, TR 16-74, Center for Research in Computing Technology, Harvard University.

7. Ehrenfeucht, A. and P. Zeiger [1976]. "Complexity Measures for Regular Expressions", JCSS 12:2, 134-146.

8. Engelfriet, J. and E.M. Schmidt [1975]. "IO and OI", DAIMI PB-47, Dept. of Computer Science, Aarhus University. To appear in JCSS.

9. Fisher, M.J. [1968]. "Grammars with macro-like productions", Ph.D. Thesis, Harvard University.

10. Fisher, M.J. and R.E. Ladner [1977]. "Propositional modal logic of programs", Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, 286-294.

11. Geller, M.M., H.B. Hunt III, T.G. Szymanski and J.D. Ullman [1975]. "Economy of description by parsers, DPDA's and PDA's", TR # 209, Computer Science Lab., Princeton University.

12. Ginsburg, S. and S. Greibach [1966]. "Deterministic context-free languages", Information and Control 9:6, 620-648.

13. Ginsburg, S. and H.G. Rice [1962]. "Two families of languages related to ALGOL", J. ACM 9, 350-371.

14. Goguen, J.A., J.W. Thatcher, E.G. Wagner and J.B. Wright [1977]. "Initial Algebra Semantics and Continuous Algebras", J. ACM 24:1, 68-95.

15. Greibach, S.A. [1975]. "Theory of Program Structures: Schemes, Semantics, Verification", Lecture Notes in Computer Science 36, Springer-Verlag, New York.

16. Hartmanis, J. and J.E. Hopcroft [1970]. "What makes Some Language Theory Problems Undecidable", JCSS 4:4, 368-376.

17. Hayashi, T. [1972]. "On derivation trees of indexed grammars - an extension of the uvwxy-theorem", RIMS-122, Res. Inst. for Mathematical Sciences, Kyoto University.

18. Hibbard, T. and J. Ullian [1966]. "The independence of inherent ambiguity from complementedness among context-free languages", J. ACM 13:4, 588-593.

19. Hopcroft, J.E. and J.D. Ullman [1969]. Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Massachusetts.

20. Hunt III, H.B., D.J. Rosenkrantz and T.G. Szymanski [1976]. "On the Equivalence, Containment, and Covering Problems for the Regular and Context-free Languages", JCSS 12:2, 222-268.

21. Jones, N.D. [1975]. "Space-Bounded Reducibility among Combinatorial Problems", JCSS 11:1, 68-85.

22. Kozen, D. [1977]. Personal communication.

23. Madsen, O.L. and B.B. Kristensen [1976]. "LR-Parsing of Extended Context-Free Grammars", Acta Informatica 7:1, 61-73.

24. Maslov, A.N. [1974]. "The Hierarchy of Indexed Languages of an Arbitrary Level", Soviet. Math. Dokl. 15:14, 1170-1174.

25. Maibaum, T.S.E. [1974]. "A generalized approach to formal languages", JCSS 8:3, 409-439.

26. Meyer, A.R. and M.J. Fisher [1971]. "Economy of Description by Automata, Grammars and Formal Systems", Conference Record, IEEE 12th Annual Symposium on Switching and Automata Theory, 188-190.

27. Meyer, A.R. and L. Stockmeyer [1972]. "The equivalence problem for regular expressions with squaring requires exponential space", Conference Record, IEEE 13th Annual Symposium on Switching and Automata Theory, 125-129.

28. Moore, Frank R. [1971]. "On the Bounds for State-Set Size in the Proofs of Equivalence Between Deterministic, Nondeterministic and Two-Way Finite Automata", IEEE Transactions on Computing C-20:10, 1211-1214.

29. Ogden, W. [1968]. "A helpful result for proving inherent ambiguity", Mathematical Systems Theory 2:3, 191-194.

30. Salomaa, A. and M. Soittola. Automata-Theoretic Aspects of Formal Power Series, Springer-Verlag. To appear in 1978.

31. Schmidt, E.M. and T.G. Szymanski [1977]. "Succinctness of Descriptions of Unambiguous Context-Free Languages", SIAM J. Computing 6:3, 547-553.

32. Stearns, R.E. [1967]. "A regularity test for pushdown machines", Information and Control 11:3, 323-340.

33. Valiant, L.G. [1975]. "Regularity and Related Problems for Deterministic Pushdown Automata", J. ACM 22:1, 1-10.

34. Valiant, L.G. [1976]. "A Note on the Succinctness of Descriptions of Deterministic Languages", Information and Control 32:2, 139-145.