FORMAL LANGUAGE THEORY AND THEORETICAL COMPUTER SCIENCE†

Ronald V. Book

Department of Computer Science
Yale University
10 Hillhouse Avenue
New Haven, Conn. 06520, U.S.A.

In the last fifteen years the area of formal language theory and abstract automata has enjoyed a great deal of attention from the research community in theoretical computer science. However, in the last few years the main interest in theoretical computer science has been directed elsewhere. In these notes I shall consider the role of formal language theory in theoretical computer science by discussing some areas in which formal language theory may find applications.

Behind the activities in theoretical computer science lie the major problems of understanding the nature of computation and its relationship to computing methodology. While this area is mathematical and abstract in spirit, it is not pure mathematics: indeed, theoretical computer science derives much of its motivation from the abstraction of the practical problems of computation.

Abstraction for the sake of abstraction and beauty is one of the goals of modern pure mathematics [1]. However, as part of theoretical computer science, formal language theory must speak to problems concerning the nature of computation. In Part I of these notes I point to results of language theory which stem from or lend insight to the more general study of computability and computational complexity. In Part II I discuss several problem areas where formal language theory may find interesting and fruitful application. In this context I urge the reader to consult the paper "Programming languages, natural languages, and mathematics" by Peter Naur [2].

† The preparation of this paper was supported in part by the National Science Foundation under Grant GJ-30409 and by the Department of Computer Science, Yale University.

Part I

Formal language theory is concerned with the specification and manipulation of sets of strings of symbols, i.e., languages. It is my thesis here that as an area of interest within theoretical computer science formal language theory should be closely tied to the study of computability theory and computational complexity. Computability theory is concerned with the representation of algorithms and languages, and computational complexity considers the inherent difficulty of evaluating functions and deciding predicates. Within the scope of these studies are the questions of determinism versus nondeterminism, trade-offs between measures and representations, differences in computational power stemming from different choices of atomic operations, and the existence of hierarchies.

We shall explore two topics in order to illustrate how the common features and the differences between formal language theory and computability theory can interact fruitfully. In both cases it is shown that the questions and results of formal language theory take on new significance when reconsidered in terms of computability theory and computational complexity.

A. Universal Sets

One of the most basic notions in computability theory is the notion of a "universal machine." This notion allows a representation of the concept of "self-reference" in terms of a class of algorithms. It is particularly useful when one wishes to establish that some question is undecidable by working from first principals.

In formal language theory there are concepts that are related to the notion of universal machine. Consider the specification of a class of languages by means of an algebraic definition: the smallest class containing some specific language (the generator) and closed under specified operations. One example of this is the concept of principal AFL [3]: the smallest class of languages containing the generator and closed under union, concatenation, Kleene *, intersection with regular sets, homomorphic mappings, and inverse homomorphism. Many classes of languages studied in traditional formal language theory are principal AFLs; e.g., the context-free languages, the regular sets, the context-sensitive languages. These structures have representation theorems which are generalizations of the Chomsky-Schutzenberger Theorem for context-free languages: A language L is context-free if and only if there exists a regular set R, a nonerasing homomorphism h_1, and a homomorphism h_2 such that $L = h_1(h_2^{-1}(D_2) \cap R)$, where D_2 is the Dyck set on two letters. Besides providing certain information regarding the algebraic structure of these classes, the existence of this type of generator can be identified with a characteristic of an appropriate class of abstract automata that accept the languages involved: finitely encodable abstract families of acceptors [3].

From the standpoint of computability and computational complexity, the notion of

generator with respect to some algebraic structure can be useful [31] but the more general notion of a language being "complete" for a class with respect to some type of "reducibility " is more important when considering the inherent computational complexity of the membership problem for languages in the class. However Greibach [4] has shown that in certain classes of languages the algebraic structure yields information about questions such as the complexity of the membership problem.

There exists a "hardest" context-free language [4]: a language with the property that the inherent time complexity of the membership problem for this language yields the order of the achievable least upper bound for the time necessary for the general context-free membership problem. Formally, there is a context-free language L_0 with the property that for any context-free language L there exists a homomorphism h such that for all strings w, $w \in L$ if and only if $h(w) \in L_0$.† Thus if one has an algorithm Θ for deciding membership in L_0, then for any context-free language L one can obtain an algorithm Θ_L for deciding membership in L and the order of the running time of Θ_L is the order of the running time of Θ. In this sense L_0 is a "hardest" context-free language. Similar considerations hold for space bounds.

Greibach's result says something further regarding the algebraic structure of the class $\underline{CF}$ of context-free languages, in particular that $\underline{CF} = \{h^{-1}(L_0) \mid h \text{ a homomorphism}\}$. Following [5], let us call a class C a <u>principal cylinder</u> if there is some $L_0 \in C$ such that $C = \{h^{-1}(L_0) \mid h \text{ a homomorphism}\}$. Thus if C is a principal cylinder with generator L_0, then for every $L \in C$ there exists a function f such that for all w, $w \in L$ if and only if $f(w) \in L_0$, and f is very simple to compute (being a homomorphism). In this way a class of languages is a principal cylinder if and only if it has a language which is complete with respect to homomorphic mappings (viewed as a type of many-one reducibility) and closed under inverse homomorphism.

There are a number of classes of languages which are principal cylinders, and viewing them as such is useful when one considers these classes as complexity classes [6]. For example, each of the following classes is a principal cylinder:

(i) For any $\varepsilon > 0$, the class of languages accepted in space n^{ε} by deterministic (nondeterministic) Turing machines;

(ii) For any $k \geq 1$, the class of languages accepted in time n^k by nondeterministic Turing machines;

(iii) The class of languages accepted in exponential time by deterministic (nondeterministic) Turing machines.

If we consider the notion of a principal cylinder, then we have an example of the algebraic structure of a class of languages yielding insight into the computational complexity of the class. However we can discuss the notion of a class having a hardest language without reference to the algebraic structure.

Let C be a class of languages and let $B \in C$. Then B is a <u>hardest</u> language

† For the purpose of this discussion, the empty word is ignored.

for C (B is C-hardest) if the following condition is satisfied: if there exists an algorithm Θ for deciding membership in L_0 and Θ has running time $t(n)$, then for every $L \in C$ there exists a modification Θ_L of Θ and constants c_1, c_2 such that Θ_L is an algorithm for deciding membership in L and Θ_L has running time $c_1 t(c_2 n)$.

In this way the complexity of recognition of a C-hardest language is at least as great as the least upper bound of the complexity of recognition of all languages in C and thus the inherent time complexity of the recognition problem for this language yields the order of the achievable least upper bound for the inherent time complexity of the general recognition problem for the class C.

If C is a principal cylinder with generator L_0, then L_0 is a C-hardest language -- this is in the spirit of Greibach's result on the context-free languages [4]. However there are classes which are not principal cylinders but do have hardest languages [7]: the deterministic context-free languages, the languages accepted in real time by deterministic multitape Turing machines, the intersection closure of the class of context-free languages. Thus the notion of a class having a hardest language is distinct from the notion of a class being a principal cylinder.

While the definition of hardest language is fairly broad, one can use elementary techniques to show that certain well-studied classes do not have hardest languages. For example, the class P of languages accepted in polynomial time by deterministic Turing machines (or random-access machines or string algorithms) does not have a hardest language. The proof of this fact depends on a simple property of algebraic structures. The class P can be decomposed into an infinite hierarchy of classes of languages: if for each $k \geq 1$, $\mathrm{DTIME}(n^k)$ denotes the class of languages accepted in time n^k by deterministic Turing machines, then for every $j \geq 1$ $\mathrm{DTIME}(n^j) \subsetneq \mathrm{DTIME}(n^{j+1})$ and $P = \bigcup_j \mathrm{DTIME}(n^j)$. If P has a hardest language L_0, then there is some least k such that $L_0 \in \mathrm{DTIME}(n^k)$. From the properties of the class $\mathrm{DTIME}(n^k)$ and the definition of hardest language, this implies that $P \subseteq \mathrm{DTIME}(n^k)$ so that the hierarchy $\mathrm{DTIME}(n) \subsetneq \mathrm{DTIME}(n^2) \subsetneq \ldots$ is only finite ending with $\mathrm{DTIME}(n^k)$, a contradiction.

One should note the difference between the fact that P does not have a hardest language and the results in [8] showing that there are certain languages which are complete for P with respect to the class of reducibilities computed in log space (that is, there are languages $L_0 \in P$ with the property that for every $L \in P$ there is a function f which can be computed by a deterministic Turing machine using only $\log n$ space such that for all w, $w \in L$ if and only if $f(w) \in L_0$). A function computed in log space can be computed in polynomial time. However the result presented above shows that not all of the reducibilities used in [8] can be computed in linear time. The argument used above shows even more: the reducibilities used in [8] cannot all be computed in time n^k for any fixed k.

The concept of a universal element is central to computability theory. Studies of generators and complete sets play important roles in formal language theory and in

computational complexity. These closely related notions each represent some aspects of the self-referential properties of a class. The results described here suggest that attention be given to the comparative power of the basis operations needed for these representations, as part of the development of a conceptual framework to provide a unified view of results of this type.

B. Nondeterminism

The mathematical construct of nondeterminism has surfaced periodically in the forefront of activity in formal language theory. It arises when acceptance of languages by automata and their generation by grammars, and closure properties of classes of languages are considered. While nondeterminism is essential for presentation of certain classes of languages, e.g., the context-free languages, it is not clear what role this construct plays in computability theory and computational complexity - indeed, the underlying notion of nondeterminism is not well understood.

Abstract automata are said to operate nondeterministically if the transition function (which specifies what to do at each step) is multi-valued -- there is not a unique move to be performed at each step but rather a finite set of possible next moves. The automaton arbitrarily chooses (guesses) which move to perform and any choice is valid. In this way computations of a nondeterministic automaton on a given input may be represented as a tree in which each node has branches corresponding to the possible next steps. Such a device is said to accept an input if there exists some path through the tree ending in a node that indicates an accepting configuration of the automaton.

A nondeterministic automaton may be considered as a representation of a nondeterministic algorithm [32], an algorithm in which the notion of choice is allowed. However in this case all the branches of the tree must be explored and all choices followed, so that the entire tree must be searched. All configurations resulting from the data underlying the algorithm must be generated until either a solution is encountered or all possibilities are exhausted. The implementation of such algorithms is usually carried out by means of backtracking, and these nondeterministic algorithms are a useful design tool for parsing [33] and for programs to solve combinatorial search problems, and nondeterministic algorithms provide an efficient specification for algorithms which deterministically simulate nondeterministic processes.

It was once thought that nondeterminism was a mathematical construct that arose in the study of automata theory and formal languages but had nothing to do with other aspects of computer science. This view has changed since the observation by Cook [34] of the importance of the class NP of languages accepted in polynomial time by nondeterministic Turing machines (or random-access machines or programs written in some general purpose language). It has been shown by Cook, Karp [35], and others (see [36]) that a wide variety of important computational problems in combinatorial mathematics, mathematical programming, and logic can be naturally represented so as to be in NP. It is not known whether these problems are in the class P of languages accepted in polynomial time by deterministic algorithms. While every language in NP can be accepted by some

deterministic algorithm, the known results force the deterministic algorithm to operate in exponential instead of polynomial time. In computational complexity, it is widely accepted that a computational problem is tractable if and only if there is an algorithm for its solution whose running time is bounded by a polynomial in the size of the input. Thus it is not known if some of the problems in NP are tractable. Any proof of their tractability is likely to yield some fundamental insights into the nature of computing since it would show that the results of backtracking and of combinatorial searching can be obtained efficiently. See [37, 38] for results in this area and for some interpretations of the role of the question "does P equal NP?" plays in theoretical computer science.

Let us consider the known results regarding nondeterminism in resource bounded or restricted access automata. For the two extreme cases of finite-state acceptors and arbitrary Turing machines, nondeterminism adds nothing to the power of acceptance; both deterministic and nondeterministic finite-state acceptors characterize the class of all regular sets, and both deterministic and nondeterministic Turing acceptors characterize the class of all recursively enumerable sets. Between these extremes there is just one case for which it is known that the deterministic and nondeterministic modes of operation are equivalent: auxiliary pushdown acceptors. An auxiliary pushdown acceptor [39] has a two-way read-only input tape, and auxiliary storage tape which is bounded in size (as a function of the length of the input), and an auxiliary storage tape which is a pushdown store and is unbounded. If f is a well-behaved space bound, then a language is accepted by a deterministic auxiliary pushdown acceptor which uses at most $f(n)$ space on its auxiliary storage tape if and only if that language is accepted by a nondeterministic auxiliary pushdown acceptor which operates in time $2^{cf(n)}$ for some $c > 0$.

There are a few cases where it is known that a language can be accepted by a nondeterministic device with certain characteristics but cannot be accepted by any deterministic device with those characteristics. Some examples of this include the following:

(i) pushdown store acceptors (nondeterministic pushdown store acceptors accept all and only context-free languages while deterministic pushdown store acceptors accept only deterministic context-free languages, the languages generated by L R(k) grammars);

(ii) on-line one counter acceptors (the language $\{w\ c\ y \mid w,y \in \{a,b\}^*, w \neq y\}$ is accepted in real time by a nondeterministic on-line one counter acceptor but is not accepted by any deterministic on-line one counter acceptor);

(iii) multitape Turing acceptors which operate in real time (the class of languages accepted in real time by nondeterministic multitape Turing acceptors are the quasi-realtime languages of [40] while the class of languages accepted in real time by deterministic multitape Turing acceptors are the real-time definable languages [41, 42]; the former is closed under linear erasing and can be decomposed

into only a finite hierarchy based on the number of storage tapes while the latter is not closed under linear erasing and can be decomposed into an infinite hierarchy based on the number of tapes);

(iv) multitape Turing acceptors which operate in real time and are reversal-bounded (the class of languages accepted in real time by deterministic multitape Turing acceptors which are reversal-bounded have the property that if a language L is accepted by such a device and for every $x,y \in L$, either x is a prefix of y or y is a prefix of x, then L is regular; the class of languages accepted by the nondeterministic devices operating in this way do not have this property [43]);

(v) on-line multitape Turing acceptors which are reversal-bounded (the class of languages accepted by deterministic reversal-bounded Turing acceptors is a subclass of the recursive sets while every recursively enumerable set is accepted by a nondeterministic reversal-bounded acceptor [44]).

In each of the cases (i) - (v), each language L_1 accepted by a nondeterministic device of the class in question can be represented as the homomorphic image of a language L_2 accepted by a deterministic device of that class. When the nondeterministic device is forced to operate in real time, then the corresponding homomorphic mappings are nonerasing (i.e., $h(w) = e$ if and only if $w = e$), and the class of languages accepted by the nondeterministic devices is closed under nonerasing homomorphism while the class corresponding to the deterministic devices is not.

Generally, a class of languages accepted by nondeterministic devices which are on-line and have finite-state control are closed under nonerasing homomorphism. In such a setting the operation of nonerasing homomorphism corresponds to the notion of bounded existential quantification. Thus in the cases above, the homomorphism (or existential quantifier) chooses (a representation of) an accepting computation of the device on the input given. The deterministic device is able to check if indeed the choice is a representation of an accepting computation. In this way an accepting computation of an on-line nondeterministic device can be viewed as having two distinct phases, one which allows a choice or "guess" and one which checks to see if the choice has the appropriate qualities. This view of nondeterministic operation is brought forth strongly in the proofs in [40, 45] where a machine first "guesses" what the input will be and performs a computation on this guess before checking whether the guess is correct.

Representing languages accepted by on-line nondeterministic devices in terms of homomorphisms or existential quantifiers and languages accepted by deterministic devices is consistent with Karp's presentation of NP-complete problems (languages) [35]. Further, the strategy of "guessing" and then "checking" is fundamental to the structure of the specific "hardest context-free language" presented by Greibach in [4].

When one considers devices which have two-way input tapes, then the above representation does not seem applicable. For example, we see no way of representing the

languages accepted by nondeterministic linear bounded automata in terms of the languages accepted by deterministic linear bounded automata. While the question of determinism vs. nondeterminism can be treated in terms of complete sets and reducibilities in this case as well as in the case of on-line devices [46], the role of nondeterminism is even less understood in the case of two-way devices.

One area where nondeterminism appears to be a useful tool but where it has previously played little role is in the representation and analysis of logical predicates. Recently the class of rudimentary relations of Smullyan -- the smallest class of string relations containing the concatenation relations and closed under the Boolean operations, explicit transformation, and a form of bounded (existential and universal) quantification -- has been studied as a class RUD of formal languages [47]. This class contains all context-free languages, all languages accepted in real time by nondeterministic multitape Turing machines, and all languages accepted in log(n) space by nondeterministic Turing machines. On the other hand, every language in RUD is accepted in linear space by a deterministic Turing machine.

This class is of interest for various reasons. It is closed under most of the operations associated with classes specified by nondeterministic devices -- union, concatenation, Kleene *, nonerasing homomorphism, inverse homomorphism, linear erasing, reversal -- as well as under complementation, a property usually associated with a class specified by deterministic devices. Wrathall [47] has shown that RUD can be represented in terms of nondeterministic devices: RUD is the smallest class of languages containing the empty set and closed under the operation of relative acceptance by nondeterministic linear time-bounded oracle machines. This class can be decomposed into a hierarchy based on the number of applications of the oracle machine. It is not known whether this hierarchy is infinite and this question is closely tied to the questions of determinism vs. nondeterminism in automata based on computational complexity. The results presented in [47] show that the study of closure operations usually associated with classes of formal language theory can yield insight into the questions arising in computational complexity.

The study of nondeterminism takes on an extremely important role in theoretical computer science when it is removed from the context of formal language theory and considered in terms of computability and computational complexity. The techniques, results, and insights provided by the previous (and ongoing) work in language theory are extremely valuable when considering this problem. It is hoped that a healthy interaction between formal language theory and computational complexity will lead to a complete understanding of this important construct.

C. Reprise

In this section I have discussed two examples of notions that can be studied in formal language theory and in computability and computational complexity. I have attempted to show that studying these notions in terms of the interplay between these areas will lead to a better understanding of the problems they reflect than studying

them from either area exclusively. It is in this spirit that the role of formal language theory in theoretical computer science may find renewal.†

† In this light the reader should find [48] of interest.

Part II

In this portion of the paper, several problem areas are discussed. These areas are not part of the "classical" theory of formal languages and abstract automata. However each may prove to be an area where formal language theory may find interesting and fruitful application.

A. L Systems

In the last few years a new area of activity has excited many researchers in automata and formal language theory. This is the study of L (for Lindenmayer) systems. Originally L systems were put forth as an attempt to model certain phenomena in developmental biology, and this motivation continues to provide a robust quality to the work in this field. However current activity has drawn researchers from cellular automata and tessellation structures, self-reproducing automata and systems, graph generating and graph recognition systems, as well as from classical automata and formal language theory.

From the biological point of view, L systems provide mathematical tools for building models of actual individuals or species in order to investigate and predict their behavior. Further, it is possible to use the machinery of L systems to express general statements about non-specific organisms and to precisely express hypotheses concerning the mechanisms behind certain phenomena. To what extent this area will affect the study of cellular behavior in development depends largely on how fully the analogy between L systems and the developing organisms they model can be exploited [10].

From the mathematical point of view, L systems have provided an entirely new perspective on languages, grammars, and automata. While the definition of application of rewriting rules varies from the Chomsky-type model discussed in the classical theory, many of the questions pursued are the standard ones: characterization of weak and strong generative capacity, decidability of properties of systems and languages, closure properties, etc. However a number of questions which are not particularly meaningful in the classical theory play a prominent role in the study of this new model. For example, while the notion of derivational complexity is of interest when studying context-sensitive and general phrase-structure grammars, it is of little interest when studying context-free grammars; however, the study of growth functions (closely related to the "time functions" of [12]) plays an important role in the study of L systems.

Some of the questions arising in the study of L systems are closely related to the questions regarding extensions of context-free grammars. It appears that there are many possibilities for these two activities to affect one another, both in terms of techniques (such as the study of recursion, fixed-point theorems, and tree-manipulating systems) and actual results. Further the techniques used in the study of L systems can provide interesting tools for studying problems such as synchronization in general algorithmic systems and discrete pattern recognition.

Just as much of the work in classical automata and formal language theory has

strayed from the central questions (of computer science) regarding models for computation and should be regarded as a part of applied mathematics, much of the research in L systems should be viewed as an exciting new aspect of modern applied mathematics.

The literature in the area of L systems appears to grow without bound (or at least at the same rate that cells interact and grow). The interested reader can find a brief introduction in [9] and there is an excellent book [13] by G.T. Herman and G. Rozenberg which will introduce the mathematically sophisticated reader to most problem areas currently studied. There are three recent conference proceedings [11, 14, 15] which will help the reader to learn about relatively current research.

B. Program Schemes

Many results and techniques from automata and formal language theory have found application in the study of the mathematical theory of computation. This has been particularly true in the study of the comparative power of different types and features of programming languages when explicit functions are banished and abstract uninterpreted programs are considered. This is the realm of program schemes.

Program schemes can be viewed as abstract models of programs where assignment statements use unspecified functions and the test statements that control the direction of flow through a program during a computation make use of unspecified predicates. Thus, a single program scheme is an abstract model of a family of different programs, each program arising from the program scheme through a specific interpretation of the function and predicate symbols used to define the scheme. In this way the scheme represents the control structure of the program and the interpretation represents the meaning of its instructions.

Many of the results concerning flowchart schemes and recursion schemes depend on the type of "syntactic" arguments used in the study of acceptance of formal languages by abstract automata. Also, many results in language theory can be applied to problems after some rephrasing. This is particularly true when considering problems regarding the decidability of questions such as weak and strong equivalence, inclusion, and translation.

Here we point out only a few of the results that directly connect the study of schemes to formal language theory. Friedman [23] has shown that the inclusion problem for simple deterministic languages is undecidable and has applied this result to show that the inclusion problem for monadic recursion schemes is undecidable. Further, Friedman [24] has reduced the question of equivalence of deterministic pushdown store acceptors to the question of strong equivalence of monadic recursion schemes, a result that is quite surprising in that it shows the two questions to be reciprocally reducible.

Nivat [21] and his students and colleagues in Paris have made some interesting connections among questions of semantics, polyadic recursion schemes, and general rewriting systems. Downey [19] has also used general rewriting systems to study recursion in language definition. Rosen [18] has used techniques involving context-free

grammars to study the structure of computations by recursion schemes. Engelfriet [20] has made an explicit attempt to exploit the connections between language theory and schemes.

Finally, it should be noted that the fundamental paper [17] of Luckham, Park, and Paterson on flowchart schemes makes frequent use of concepts from automata and language theory as do others [16, 20].

C. *Pattern Recognition*

Many constructs from formal language theory have been used in the study of automatic recognition and generation of patterns, particularly in the case of picture processing techniques. This application of language theoretic notions is particularly fruitful when the approach to pattern recognition is syntactic or structural as opposed to decision-theoretic. Some of this work is surveyed in [25].

Recently it has been shown that techniques from formal language theory are extremely useful in certain problems of automatic speech recognition. Lipton and Snyder [26] have treated the parsing of speech as a problem of parsing probabilistic input and have presented an efficient optimal algorithm for this problem. The algorithm they use is a generalization of a well known parsing algorithm for context-free grammars. Levinson [27] has taken a similar approach and his work indicates that the area of automatic speech recognition may be an extremely fruitful area for the application of notions of automata and formal language theory.

D. *Parallel Computation*

One area in which notions from automata and formal language theory may find application is that of parallel computation. A wide variety of models of parallel computation have been presented in the literature. Most models have been motivated by problems arising in the study of operating systems and these models have properties that reflect the characteristics of certain systems. Some of these models have been explored by researchers in theoretical computer science because of their ability to express general problems of sequencing, synchronization, and communication and interaction between processes. In particular, Petri nets and vector addition systems have received a great deal of attention.

Some of the questions modeled in these systems (e.g., deadlock, liveness) have been represented in terms of abstract automata in order to determine whether or not they are decidable and, if so, to determine the inherent complexity of the decision problem. In this area language theory may be useful since many of the properties studied can be represented by means of language-theoretic notions. Examples of this may be found in [28, 29, 30].

References

[1] G.H. Hardy. *A Mathematician's Apology*. Cambridge Univ. Press, 1940, reprinted 1967.

[2] P. Naur. Programming languages, natural languages, and mathematics. Conference Record, *2nd ACM Symp. Principles of Programming Languages*. Palo Alto, Calif., 1975, 137–148.

[3] S. Ginsburg and S.A. Greibach. Principal AFL. *J. Computer System Sci.* 4 (1970), 308–338.

[4] S.A. Greibach. The hardest context-free language. *SIAM J. Computing* 2 (1973), 304–310.

[5] L. Boasson and M. Nivat. Le cylindre des langages lineaires n'est pas principal. *Proc. 2nd GI - Profession Conf. Automata Theory and Formal Languages*. Springer Verlag, to appear.

[6] R. Book. Comparing complexity classes. *J. Computer System Sci.* 9 (1974), 213–229.

[7] R. Book. On hardest sets. In preparation.

[8] N. Jones and W. Laaser. Complete problems for deterministic polynomial time recognizable languages. *Proc. 6th ACM Symp. Theory of Computing*. Seattle, Wash., 1974, 40–46.

[9] A. Salomaa. *Formal Languages*. Academic Press. New York, 1973.

[10] P.G. Doucet. On the applicability of L-systems in developmental biology. In [11].

[11] A. Lindenmayer and G. Rozenberg (eds.). Abstract of papers presented at a Conference on Formal Languages, Automata, and Development. Univ. Utrecht, Utrecht, The Netherlands, 1975.

[12] R. Book. Time-bounded grammars and their languages. *J. Computer System Sci.* 4 (1971), 397–429.

[13] G.T. Herman and G. Rozenberg. *Developmental Systems and Languages*. North-Holland Publ. Co. Amsterdam, 1974.

[14] G. Rozenberg and A. Salomaa (eds.). *L Systems*. Lecture Notes in Computer Science, Vol. 15. Springer-Verlag, 1974.

[15] *Proceedings of the 1974 Conference on Biologically Motivated Automata Theory*. McLean, Va. Published by the IEEE Computer Society.

[16] S. Garland and D. Luckham. Program schemes, recursion schemes, and formal languages. *J. Computer System Sci.* 7 (1973), 119–160.

[17] D. Luckham, D. Park, and M. Paterson. On formalized computer programs. *J. Computer System Sci.* 4 (1970), 220–249.

[18] B.K. Rosen. Program equivalence and context-free grammars. *Proc. 13th IEEE Symposium on Switching and Automata Theory*. College Park, Md., 1972, 7–18.

[19] P.J. Downey. Formal languages and recursion schemes. *Proc. 8th Princeton Conference on Information Science and Systems*. Princeton, N.J., 1974.

[20] J. Engelfriet. *Simple Program Schemes and Formal Languages*. Lecture Notes in Computer Science, Vol. 20. Springer-Verlag, 1974.

[21] M Nivat. On the interpretation of recursive program schemes. IRIA Technical Report, 1974.

[22] E. Ashcroft, Z. Manna, and A. Pnueli. Decidable properties of monadic functional schemes. *J. Assoc. Comput. Mach.* 20 (1973), 489-499.

[23] E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science* 1 (1975). To appear.

[24] E.P. Friedman. Relationships between monadic recursion schemes and deterministic context-free languages. *Proc. 15th IEEE Symposium on Switching and Automata Theory.* New Orleans, La., 1974, 43-51.

[25] K. Fu. *Syntactic Methods in Pattern Recognition.* Academic Press. New York, 1974.

[26] R. Lipton and L. Snyder. On the parsing of speech. Technical Report Number 37, Department of Computer Science, Yale University, 1975.

[27] S. Levinson. An Artificial Intelligence Approach to Automatic Speech Recognition. Doctoral Dissertation, U. Rhode Island, 1974.

[28] J.L. Peterson. Computation sequence sets. Unpublished manuscript.

[29] W.H. Byrn. Sequential Processes, Deadlocks, and Semaphore Primitives. Doctoral Dissertation, Harvard University, 1974.

[30] W.E. Riddle. Modeling and Analysis of Supervisory Systems. Doctoral Dissertation, Stanford University, 1972.

[31] R. Book. On languages accepted in polynomial time. *SIAM J. Computing* 1 (1972), 281-287.

[32] R. Floyd. Nondeterministic algorithms. *J. Assoc. Comput. Mach.* 14 (1967), 636-644.

[33] A. Aho and J. Ullman. *The Theory of Parsing, Translating, and Compiling, Vol. I.* Prentice-Hall Publ. Co., 1972.

[34] S. Cook. The complexity of theorem-proving procedures. *Proc. 3rd ACM Symp. Theory of Computing.* Shaker Hts., Ohio, 1973, 343-353.

[35] R. Karp. Reducibilities among combinatorial problems. In *Complexity of Computer Computation* (R. Miller and J. Thatcher, eds.). Plenum, N.Y., 1972, 85-104.

[36] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[37] R. Karp (ed.). *Complexity of Computation.* SIAM-AMS Proc. VII, Amer. Math. Soc. Providence, R.I., 1974.

[38] J. Hartmanis and J. Simon. Feasible computations. *Proc. GI-Jahrestagung 74.* Springer-Verlag, to appear.

[39] S. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. Assoc. Comput. Mach.* 18 (1971), 4-18.

[40] R. Book and S.A. Greibach. Quasi-realtime languages. *Math. Systems Theory* 4 (1970), 97-111.

[41] A. Rosenberg. Real-time definable languages. *J. Assoc. Comput. Mach.* 14 (1967), 645-662.

[42] S. Aanderaa. On k-tape versus (k+1)-tape real time computation. In [37].

[43] R. Book and M. Nivat. On linear languages and intersections of classes of languages. In preparation.

[44] B. Baker and R. Book. Reversal-bounded multi-pushdown machines. *J. Computer System Sci.* 8 (1974), 315-332.

[45] R. Book, M. Nivat, and M. Paterson. Reversal-bounded acceptors and intersections of linear languages. *SIAM J.Computing* 3 (1974), 283-295.

[46] J. Hartmanis and H. Hunt. The LBA problem and its importance in the theory of computing. In [37].

[47] C. Wrathall. Rudimentary predicates and relative computation. In preparation.

[48] H. Hunt, D. Rosenkrantz, and T. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Computer System Sci.*, to appear.