The LBA Problem and its Importance in the Theory of Computing

J. Hartmanis[†]

and

H.B. Hunt,III[††]

TR 73-171

May 1973

Department of Computer Science
Cornell University
Ithaca, New York 14850

[†] This research has been supported in part by the National Science Foundation Grant GJ-33171X.

[††] This research was supported by a National Science Foundation Graduate Fellowship in Computer Science.

The LBA Problem and its Importance
in the Theory of Computing

J. Hartmanis

and

H.B. Hunt, III

Abstract:

In this paper we study the classic problem of determining whether the deterministic and non-deterministic context-sensitive language are the same or, equivalently, whether the languages accepted by deterministic and non-deterministic linearly bounded automata are the same. We show that this problem is equivalent to several other natural problems in the theory of computing and that the techniques used on the LBA problem have several other applications in complexity theory. For example, we show that there exists a hardest tape recognizable non-deterministic context-sensitive language L_1, such that L_1 is a deterministic context-sensitive language if and only if the deterministic and non-deterministic context-sensitive languages are the same. We show furthermore, that many decision problems about sets described by regular expressions are instances of these tape-hardest recognizable context-sensitive languages. Thus, it follows that non-determinism in Turing machine computations (using at least linear tape) can not save memory over deterministic Turing machine computations if and only if the equivalence of regular expressions can be decided by a deterministic linearly bounded automaton. It also follows that the equivalence

of regular expressions can be decided by a non-deterministic linearly bounded automaton if and only if the family of context-sensitive languages is closed under complementation.

The LBA Problem and its Importance in the Theory of Computing

J. Hartmanis

and

H.B. Hunt, III

1. Introduction

In this section we sketch the history of the LBA problem and outline the results in this paper.

Linearly bounded automata were first defined and investigated by John Myhill in 1960 [13]. As Myhill points out, the definition of a linear bounded automaton was motivated by an observation made by Rabin and Scott about two-way finite automata with erasing. This remark appeared in a technical report on which the well-known Rabin and Scott paper "Finite Automata and their Decision Problems" was based. The observation was that two-way finite automata, which can erase input symbols, can accept non-regular sets and that the equivalence problem for these automata is recursively undecidable. These observations never appeared in the published paper, but the short paragraph in the original technical report sufficed to convince Myhill that this model with erasing only was artifical and that the automaton should be permitted to erase and print on the tape squares occupied by the initial input word. Thus these automata are just one-tape Turing machines which can use for computation as much tape as is needed to write down the input word. Since this definition bounds the available tape linearly to the length of the input word Myhill called them linearly bounded automata.

The importance of linearly bounded automata was further emphasized when their connection with language theory was discovered. In the late fifties and early sixties Chomsky initiated an intensive study of formal languages and defined four classes of grammars with the corresponding languages: the regular, context-free, context-sensitive and recursively enumerable languages. After it was realized in 1962 that the context-free languages were exactly the languages accepted by non-deterministic push-down automata, the regular, context-free and recursively-enumerable languages could all be defined by their grammars or equivalently by the automata which accepted them. The context-sensitive languages remained the only exception.

In 1963 Landweber [10] showed that every set or language accepted by a deterministic linearly bounded automaton was a context-sensitive language. In 1964 Kuroda [9] introduced the non-deterministic linearly bounded automaton and showed that the family of languages accepted by the non-deterministic linearly bounded automata is exactly the same as the family of languages generated by the context-sensitive grammars.

These results revealed another natural connection between families of formal languages and families of automata; but it also raised the now classic, LBA problem (or the First LBA problem):

> Are the languages accepted by deterministic and non-deterministic linearly bounded automata the same? Or equivalently, are the deterministic and non-deterministic context-sensitive languages the same? Abbreviated, DSCL=NDCSL?

If DCSL = NDCSL then the family of context-sensitive languages is closed under complementation. On the other hand, it still could happen that DCSL $\neq$ NDCSL but the family of context-sensitive languages is closed under complementation. Thus we are lead to the Second LBA problem:

> Are the context-sensitive languages closed under complementation?

Both of these problems are basically problems about the minimal amount of memory needed to perform a computation. In general, such problems are quite difficult and so far in computational complexity theory we have had little success in determining lower complexity bounds for specific computations. The above mentioned LBA problems appear to be no exception. At the same time, our inability to answer them indicates that we have not yet understood the nature of non-deterministic computations.

Considerable progress on the first LBA problem was made in 1969 by W. Savitch in his doctoral dissertation [14]. Savitch showed that every non-deterministic Turing machine using $L(n)$ tape, $L(n) \geq \log n$, can be simulated by a deterministic Turing machine using no more than $[L(n)]^2$ tape. Thus the non-deterministic context-sensitive languages can all be recognized by n^2-tape bounded deterministic Turing machines. The result was surprising since all previous simulation methods required an exponential amount of tape. Furthermore, Savitch showed that there exists one non-deterministic $L(n) = \log n$ - tape recognizable language L_0 such that if L_0

is recognizable deterministicly in ℓogn - tape, then for all tape bounds $L(n)$, $L(n) \geq \ell ogn$, the non-deterministic and deterministic recognizable languages are the same. Thus if non-determinism can be eliminated for the ℓogn - tape reconizable language L_0 then DCSL = NDCSL and we see that we have a sufficient condition for the LBA problem. Unfortunately, this was shown only to be a sufficient condition for DCSL = NDCSL.

In this paper we show that we can find necessary and sufficient conditions for DCSL = NDCSL in terms of one non-deterministic context-sensitive language by constructing a hardest deterministic tape recognizable context-sensitive language L_1. Thus we get that DCSL = NDCSL if and only if L_1 is a deterministic context-sensitive language.

Similarly, the family of context-sensitive languages is closed under complementation if and only if the complement of L_1, $\bar{L}_1$, is a non-deterministic context-sensitive language.

Actually the results are stronger in that DCSL = NDCSL implies that the deterministic and non-deterministic tape bounded computations are the same for all tape bounds $L(n) \geq n$. Furthermore, there exists a recursive translation which maps every non-deterministic Turing machine onto a deterministic one using no more tape than the non-deterministic one (provided it used at least linear tape).

Similarly, if the family of context-sensitive languages is closed under complementation then there exists a recursive translation which maps every ℓba onto another ℓba accepting exactly those sequences not accepted by the first.

Next we show that the LBA problems can be reduced to equivalent problems about very simple non-writing automata or flowchart computations. Consider finite automata with k read-only heads which can move in both directions on the input and sense when two heads are scanning the same tape square. Then, utilizing our previous results and an observation by Savitch, we show that there exists a language L_2 over a one-symbol alphabet, $L_2 \subseteq a^*$, which is recognizable by a 7-head non-deterministic finite automaton and has the property that, L_2 is recognizable by a k-head deterministic finite automaton if and only if DCSL = NDCSL.

Again, $\bar{L}_2$ is recognizable by a k-head non-deterministic finite automaton if and only if the family of CSL's is closed under complementation which happens, iff the family of languages over a one-symbol alphabet recognizable by multi-head automata is closed under complementation.

Thus we will show that if non-determinism can be eliminated in one specific 7-head finite automaton by using more heads then it can be eliminated in all Turing machine computations using no less than linear tape. A similar result holds for flowcharts where we must eliminate non-determinism by using more variables.

To relate the LBA problems to a different problem area we show that the complexity of the LBA problems is equivalent to many decision problems about sets described by regular expressions. In this case the proofs exploit an observation due to Meyer and Stockmeyer [12] about the descriptive power of regular expressions. It turns out that for any non-deterministic ℓba M_i there exists a deterministic ℓba which for any input y to M_i can write down a regular expression R(y) describing the set of all invalid computations of M_i on input y. Therefore the input y is accepted by M_i if and only if there is a valid computation by M_i on y, which happens if and only if $L[R(y)] \neq \Sigma^*$, where L(R) denotes the language described by R. Thus we see that if a deterministic ℓba can check whether a regular expression describes a set not equal to Σ^*, every non-deterministic ℓba M_i can be replaced by a deterministic ℓba, using the above procedure. Furthermore, since the set of all regular expressions R not describing Σ^* is, easily seen to be, a non-deterministic $cs\ell$, we get the following result:

DCSL = NDCSL if and only if

$$L_3 = \{R \mid R \text{ regular expression, } L(R) \neq \Sigma^*\}$$

is a deterministic context-sensitive language.

Similarly one proves that the family of context-sensitive languages is closed under complementation if and only if $\bar{L}_3$ is a $cs\ell$.

A generalization of this result leads to a metatheorem about properties of regular expressions which link the LBA problems to the tape complexity of many other decision problems about regular sets.

Let P be any property on the regular sets over $\Sigma = \{0,1\}$ such that

1) $P(\Sigma^*)$ = True, and

2) The set of languages

$$\bigcup_{x \in \Sigma^*} \{x\backslash L \mid P(L) = \text{True}\}$$

is properly contained in the family of regular sets over Σ where $x\backslash L = \{w \mid x\,w \in L\}$

Let

$L = \{R \mid R$ is a regular expression over $\{0,1\}$ and $P[L(R)] = \text{False}\}$ be a non-deterministic csℓ. Then L is a deterministic csℓ if and only if DCSL = NDCSL.

Similarly,

$\tilde{L} = \{R \mid R$ is a regular expression over $\{0,1\}$ and $P[L(R)] = \text{True}\}$ is a non-deterministic csℓ if and only if the family of non-deterministic csℓ's are closed under complementation.

To illustrate the power of this result we list five other decision problems about regular sets such that any one of them can be recognized by a det. ℓba if and only if NCSL = DCSL, and furthermore if the complement of any one of these languages is a csℓ then the context-sensitive languages are closed under complementation. In all examples R and S are restricted regular expressions over $\{0,1\}$:

$\{(R,S) \mid L(R) \neq L(S)\}$

$\{R \mid L(R) \neq \Sigma^*\}$

$\{R \mid L(R)$ is coinfinite$\}$

$\{R \mid L(R) \neq \text{REVERSAL } L(R)\}$

$\{R \mid L(R) \neq L(R^*)\}$.

2. Hardest Tape and Time Recognizable CSL.

In this section we give the first of two proofs that there exists a hardest tape and time recognizable context-sensitive language and show, furthermore, that the LBA problem is equivalent to the problem of eliminating non-determinism in non-writing automata or flowchart computations.

For the sake of completness we recall that a linearly bounded automaton is a one-tape Turing machine whose input is placed between end markers and the TM cannot go past these end markers. Thus all the computations of the ℓba are performed on as many tape squares as are needed to write down the input and since the ℓba can have arbitrary large (but fixed) tape alphabet, we see that the amount of tape for any given ℓba (measured as length of equivalent binary tape) is linearly bounded by the length of the input word. If the TM defining the ℓba operates deterministicly we refer to the automaton as a deterministic ℓba, otherwise as a non-deterministic ℓba or simply an ℓba.

Since the connection between linearly bounded automata and context-sensitive languages is well-known we will also refer to the languages accepted by non-deterministic and deterministic ℓba's as non-deterministic and deterministic context-sensitive languages, respectively.

The essence of the first proof is to write down a universal context-sensitive language so that no other csℓ can be more difficult to recognize. The surprising thing is that this can be done very easily. Below we give a "universal" $cs\ell$.

$L_1 = \{\#M_i\#\ \mathrm{CODE}(x_1x_2...x_n)\# \mid x_1x_2...x_n$ is accepted by ℓba $M_i\}$

Thus the sequences in L_1 consist of a simple encoding of an ℓba, M_i, followed by an encoded form of an input accepted by M_i. The input encoding $\mathrm{CODE}(x_1x_2...x_n)$ is any straight-foward, symbol by symbol encoding of sequences over alphabets of arbitrary cardinality (the input and tape alphabet of M_i) into a fixed alphabet, say $\{0,1\ \#\}$; with the provision that $|\mathrm{CODE}(x_j)| \geq$ the cardinality of the tape alphabet of M_i. Clearly, by inspecting the description of M_i it can be determined what encoding is used.

It is easily seen that L_1 is a $cs\ell$ since it can be accepted by a non-deterministic ℓba, M, which simulates M_i on input $x_1 \ldots x_n$. Since M_i uses no more tape than required to write down the input, the encoded input $\mathrm{CODE}(x_1 \ldots x_n)$ gives enough tape for M to simulate M_i. Thus L_1 is a context-sensitive language and we get the next result in terms of L_1.

<u>Theorem 1</u>: 1. $L_1 \in$ NDCSL.

2. $L_1 \in$ DCSL iff NDCSL = DCSL.

3. $\bar{L}_1 \in$ NDCSL iff the family of context-sensitive languages is closed under complementation.

<u>Proof</u>: From the construction of L_1 we know that L_1 is a $cs\ell$. This follows, as mentioned above, since the codes for the input symbols x_j of M_i are long enough to encode all tape symbols of M_i. Thus NDCSL = DCSL implies that L_1 is recognized by a deterministic ℓba.

On the other hand, if L_1 is recognizable by a deterministic ℓba, M_D, then DCSL = NDCSL since for every ndℓba M_i we can recursively construct an equivalent deterministic ℓba $M_{D(i)}$. The dℓba $M_{D(i)}$ operates as follows: for input $x_1 \ldots x_n$ $M_{D(i)}$ writes $\#M_i\#CODE(x_1x_2\ldots x_n)\#$ on its tape and starts the dℓba M_D on this input and accepts the input iff M_D accepts its input. Because of the definition of M_D the input $\#M_i\#CODE(x_1x_2\ldots x_n)\#$ is accepted by M_D if and only if the input $x_1\ldots x_n$ is accepted by M_i and therefore $M_{D(i)}$ and M_i accept the same set. Furthermore, since the length of $\#M_i\#CODE(x_1x_2\ldots x_n)\#$ is linearly bounded by the length of $x_1x_2\ldots x_n$ (for any fixed i) we see that $M_{D(i)}$ is a deterministic ℓba. Thus NDCSL = DCSL, as was to be shown.

The third part of this theorem follows by a similar argument.

It is interesting to note that if L_1 can be recognized on a deterministic ℓba then all non-deterministic tape computations using $L_i(n) \geq n$ tape can be replaced by equivalent deterministic computations using no more tape. Furthermore, there is a recursive translation which maps the non-deterministic Turing machines onto the equivalent deterministic Turing machines.

<u>Corollary 2</u>: DCSL = NDCSL if and only if there exists a recursive translation σ such that for every non-deterministic TM M_i, which uses $L_i(n) \geq n$ tape, $M_{\sigma(i)}$ is an equivalent deterministic TM using no more than $L_i(n)$ tape.

<u>Proof</u>: The "if" part of the theorem is obvious.

To show the "only if" part, let M_i be any non-deterministic TM accepting the set $A_i \subseteq \Sigma^*$ and using $L_i(n) \geq n$ tape.

We first define two auxillary languages used in the proof. Let

$$A_i' = \{\#w\#^t \mid M_i \text{ on input } w \text{ uses more than } (t + |w| - 1) \text{ tape squares}\}.$$

Clearly, A_i' is a non-deterministic csℓ, since we can run M_i non-deterministicly on input w and see whether for some choice of moves more than $t + |w| - 1$ tape is required. But if DCSL = NDCSL then A_i' is accepted by a deterministic ℓba M_i'.

Next, we define

$$A_i'' = \{\#w\#^t \mid M_i \text{ accepts } w \text{ using no more than } (|w| + t) \text{ tape squares}\}$$

Again, A_i'' is accepted by a non-deterministic ℓba and therefore, by our assumption, A_i'' is accepted by a deterministic ℓba M_i''.

We now show that from M_i' and M_i'', which can be obtained recursively from M_i by Theorem 1, we can recursively obtain $M_{\sigma(i)}$ which accepts A_i using no more than $L_i(n)$ deterministic tape.

$M_{\sigma(i)}$ operates as follows:

1. for input $w = x_1 \ldots x_n$ $M_{\sigma(i)}$ finds the largest t (if it exists) such that

   $$\#w\#^t \text{ is in } A_i'$$

   by successively checking

   $$\#w\#,\ \#w\#^2,\ \#w\#^3, \ldots.$$

   with the deterministic ℓba M_i' .

2. On $\#w\#^t$ $M_{\sigma(i)}$ simulates the deterministic ℓba M_i'' and accepts the input w iff M_i'' accepts $\#w\#^t$.

Clearly, $M_{\sigma(i)}$ accepts A_i on deterministic tape $L_i(n)$, as was to be shown.

From the above results we see that if DCSL = NDCSL then all other deterministic and non-deterministic tape-bounded computations using more than a linear amount of tape are the same. On the other hand, we have not been able to force the equality downward. For example, we have not been able to show that if all deterministic and non-deterministic tape-bounded computations using $L_i(n) \geq 2^n$ tape are the same, that then DCSL = NDCSL.

Similarly, it could happen that DCSL = NDCSL but that the ℓogn - bounded deterministic languages are properly contained in the non-deterministic ℓogn - bounded computations.

Our next result shows that the previous theorem can be generalized to hold for a wide class of tape-bounded languages. Similar results have also been obtained by R. V. Book [1] using AFL Theoretic techniques.

We say that $f:N \to N$ is a semihomogeneous function if for all $c > 0$ there exists a k_c such

$$f(cn) \leq k_c f(n).$$

Thus $f(n) = n^5$ is a semihomogeneous function but $f(n) = 2^n$

is not. We say that f(n) is <u>non-deterministic</u> <u>tape</u> <u>constructable</u> iff there exists a non-deterministic TM which for input a^n computes f(n) using no more than f(n) tape squares.

Let

$$L_f = \left\{ \begin{array}{l} \#CODE(x_1x_2\ldots x_n)\# \\ \#M_i\ \#\ldots\qquad\qquad\# \end{array} \middle| \; |M_i| \leq |CODE(x_1x_2\ldots x_n)|, \right.$$

$x_1x_2\ldots x_n$ is accepted by M_i using no more than f(n) tape, and $|CODE(x_j)| \geq$ cardinality of tape alphabet of $M_i\}$.

We assume that all codes of input and tape alphabet symbols of M_i have the same length.

<u>Theorem 3</u>: Let f be a non-deterministic tape constructible, semihomogeneous function such that for all k, $k \geq 1$, $f(kn) \geq kf(n) > 0$. Then L_f is non-deterministic f(n) - tape recognizable. Furthermore, L_f is deterministic f(n) - tape recognizable iff the deterministic and non-deterministic f(n) - tape recognizable languages are the same.

Proof: Let L_f be defined as above and note that $f(kn) \geq k f(n)$ implies for all n $f(n) \geq c \cdot n$, for a fixed constant $c > 0$.

Then the following algorithm describes a f-tape bounded non-deterministic TM which recognizes L_f.

1. Check $|M_i| \leq |\mathrm{CODE}(x_1 x_2 \ldots x_n)|$.
2. Verify that the format is correct and that the proper coding is used.
3. On a work track of the tape mark off $|\Sigma| f(n)$ squares for scratch space, Σ is the tape alphabet of M_i.
4. Simulate M_i on $x = x_1 x_2 \ldots x_n$ using the scratch space from 3. Accept the input iff M_i accepts x. Note: we have enough tape since the simulation needs to encode no more than f(n) tape symbols of M_i.

The space required to execute (1) and (2) is linear in n. To execute steps (3) and (4) we need $|\Sigma|$ f(n) tape squares. But $|\Sigma| f(n) \leq f(|\Sigma| n) \leq k \cdot f(n)$, thus L_f is non-deterministic f(n) - tape acceptable.

On the other hand, if L_f is deterministic f(n) - tape acceptable, then there exists a deterministic f(n) - tape bounded TM M' such that $L(M') = L_f$. We use M' to find for every non-deterministic f(n) - tape bounded TM an equivalent deterministic f(n) - tape bounded machine. For any TM M_i construct $M_{\sigma(i)}$ as follows:

1. Short inputs are accepted by table look-up.

For input $x_1x_2\ldots x_n$, such that

$$|\mathrm{CODE}(x_1x_2\ldots x_n)| \geq |M_i|$$

$M_{\sigma(i)}$ writes out

$$\#\mathrm{CODE}(x_1x_2\ldots x_n)\#$$
$$\#\ M_i\ \#\ \ldots\qquad \#\ .$$

2. $M_{\sigma(i)}$ applies M' to the new input from (1).

The tape required by $M_{\sigma(i)}$ is less than

$$k_1n + f(k_1n)$$

which is less than

$$k_1n + k_2\ f(n)\ ,$$

since f is semihomogeneous. But then the required tape can be bounded by c f(n) and we see that $M_{\sigma(i)}$ is a deterministic f(n) - tape bounded TM, as was to be shown.

Note that in Theorem 3 we could replace the condition

$$f(kn) \geq kf(n)$$

by the weakened condition

$$f(kn) \geq (\ell ogk)\ f(n)\ ,$$

and still carry through the proof. Thus we know, for example, that there exists hardest tape recognizable languages for functions such as: $n^{1/2}$, $n^{1/3}$, $n^{2/3}$, etc. Combining this observation with our previous result we get:

Corollary 4: For any positive rational number r the language L_{n^r} is $f(n) = n^r$ - non-deterministic tape reconizable. Furthermore L_{n^r} is deterministic n^r-tape reconizable iff all n^r non-deterministic tape bounded computations can be so recognized.

So far all considerations have involved tape as our computational complexity measure. It turns out that the hardest tape recognizable language L_1 is also a hardest time recognizable context-sensitive language. We cast our result in terms of polynomial time computable languages.

Theorem 5: All context-sensitive languages can be recognized in deterministic polynomial time (non-deterministic polynomial time) if and only if the csℓ L_1 can be recognized in deterministic polynomial time (non-deterministic polynomial time).

Proof: Recall that

$$L_1 = \{\#M_i\#\text{CODE}(x_1x_2\ldots x_n)\# \mid x_1x_2\ldots x_n \text{ is accepted by } \ell\text{ba } M_i\}$$

Clearly, if csℓ's are accepted in polynomial time then so is the csℓ L_1.

If L_1 is accepted in polynomial time by a multi-tape Turing machine M then for any ℓba M_i we can recursively obtain a TM $M_{\rho(i)}$ accepting the same language in polynomial time. $M_{\rho(i)}$ operates as follows: for input $x_1x_2\ldots x_n$ $M_{\rho(i)}$ writes down

$$\#M_i\#\text{CODE}(x_1x_2\ldots x_n)\#$$

and then simulates M on this input. Clearly, if M operates in polynomial time then so does $M_{\rho(i)}$, as was to be shown., It is worth mentioning that Greibach [4] has recently exhibited a context-free language which plays the same role among context-free languages as L_1 does for context-sensitive languages. Namely, this context-free language is the hardest time and tape recognizable cfℓ and there also exist two recursive translations mapping context-free grammars onto Turing machines recognizing the language generated by the grammer in the minimal time and on the minimal amount of tape, respectively. Though at this time we do not know what is the minimal time or tape required for the recognition of context-free languages.

Before proceeding with the study of context-sensitive languages we will state two conjectures about tape requirements for the recognition of context-free languages.

<u>Conjecture 1</u>: There exists a context-free language which cannot be recognized non-deterministically on ℓogn - tape. Though we know that all context-free languages are deterministically recognizable on $[\ell og\ n]^2$ tape [11].

<u>Conjecture 2</u>: If L is a non-regular context-free language which can be recognized deterministically on ℓog ℓog n - tape, then $\bar{L}$ is not a context-free language. We know that there exist ℓog ℓogn - tape recognizable context-free languages [11], but in all such cases the complement is not a context-free language and its recognition does not require counting (i.e. ℓog n - tape). On the other hand, intuitively it seems that if L and $\bar{L}$ are non-regular context-free languages then the recognition process must involve counting and therefore must require at least ℓog n - tape.

Finally we note that the methods used to construct the "universal" csℓ L_1 can be used to construct other "universal" languages. We illustrate this by constructing the language $\tilde{L}_1$, which plays the same role for non-deterministic polynomial time-bounded computations as L_1 does for the context-sensitive languages.

Let DPTIME and NDPTIME designate the families of languages accepted by deterministic and non-deterministic polynomial time-bounded Turing machines, respectively.

We will say that a language L is <u>p-complete</u> iff L is in NDPTIME and for all L_i in NDPTIME there exists a deterministic polynomial time-bounded function f_i such that

$$x \text{ is in } L_i \text{ iff } f_i(x) \text{ is in } L .$$

Let

$$\tilde{L}_1 = \{\#M_i\#\mathrm{CODE}(x_1x_2 \ldots x_n)\#^{3|M_i|t} \mid x_1x_2 \ldots x_n \text{ is}$$

accepted by the one-tape, non-deterministic TM M_i in time t}

<u>Theorem 6</u>: The language $\tilde{L}_1$ is accepted in non-deterministic linear time by a four tape TM. Furthermore,

$$\tilde{L}_1 \text{ is in DPTIME iff NDPTIME = DPTIME.}$$

<u>Proof</u>: It is easily seen that a four-tape TM M' can accept $\tilde{L}_1$ in linear time. We indicate how M' uses its tapes: on the first sweep of the input M' checks the format of the input, copies M_i from the input on the first working tape and $\#^{3|M_i|t}$ on the second working tape. The third working tape is used to record the present state of M_i (in a tally notation) during the step-by-step simulation of M_i. It is seen that with the available information

on its working tapes M' can simulate M_i on the input in time $2|M_i|t$ (for an appropriate, agreed upon representation of M_i). Thus M' operates in non-deterministic linear time and accepts $\tilde{L}_1$. Therefore, $\tilde{L}_1$ is in NDPTIME and the assumption

$$\text{NDPTIME} = \text{DPTIME}$$

implies that $\tilde{L}_1$ is in DPTIME.

To prove that $\tilde{L}_1$ in DPTIME implies that DPTIME = NDPTIME, assume that $\tilde{L}_1$ is accepted by a deterministic TM M" operating in deterministic time n^p. Then for any non-deterministic TM M_i working in time n^q we can recursively construct a TM $M_{\sigma(i)}$ operating in polynomial time as follows:

1. for input $x_1x_2\ldots x_n$ $M_{\sigma(i)}$ writes down

$$\#M_i\#\text{CODE}(x_1x_2\ldots x_n)\#^{3|M_i|n^q}$$

2. $M_{\sigma(i)}$ starts the deterministic machine M' on the sequence in (1) and accepts the input $x_1x_2\ldots x_n$ iff M" accepts its input.

Clearly, M_i and $M_{\sigma(i)}$ are equivalent, furthermore $M_{\sigma(i)}$ operates in time less than

$$2[3|M_i|n^q + |\#M_i\#\text{CODE}(x_1x_2\ldots x_n)|]^p \leq Cn^{pq}$$

Thus $M_{\sigma(i)}$ operates in polynomial time, as was to be shown.

The previous proof shows that if $\tilde{L}_1$ is in DPTIME, then we can recursively obtain for every M_i running in time n^q an equivalent deterministic TM running in time $O[n^{pq}]$. Unfortunately, for a given TM we cannot recursively determine the running time and thus we do not know whether M_i runs in polynomial time or not. Even if we know that M_i runs in polynomial time we can still not recursively determine the degree of the polynomial.

Our next result shows that, nevertheless, we can get a general translation result. For a related result see [3].

Theorem 7: DPTIME = NDPTIME iff there exists a recursive translation σ and a positive integer k, such that for every non-deterministic TM M_i, which uses time $T_i(n) \geq n$, $M_{\sigma(i)}$ is an equivalent deterministic TM working in time $O[T_i(n)^k]$.

Proof: The "if" part of the proof is obvious. To prove the "only if" part assume that DPTIME = NDPTIME. We will outline a proof that we can recursively construct for any M_i, running time $T_i(n) \geq n$, an equivalent deterministic TM $M_{\sigma(i)}$ operating in time $O[T_i(n)^k]$, for a fixed k.

In our construction we use two auxillary languages:

$$B_i' = \{\#w\#^t \mid M_i \text{ accepts } w \text{ in less than } t \text{ time}\}$$

$$B_i'' = \{\#w\#^t \mid M_i \text{ on input } w \text{ takes more than } t \text{ time}\}.$$

Clearly, both languages can be accepted in non-deterministic linear time. Therefore, by our previous result, we can recursively construct two deterministic machines M_i' and M_i'' which accept B_i' and B_i'', respectively, and operate in time $O[n^p]$. From M_i' and M_i'' we can recursively construct the deterministic TM $M_{\sigma(i)}$, which operates as follows:

1. For input w $M_{\sigma(i)}$ finds the smallest t_o such that $\#w\#^{t_o}$ is not in B_i''. This is done by checking with M_i'' successively $\#w\#$, $\#w\#^2$, $\#w\#^3$, ...

2. $M_{\sigma(i)}$ starts M_i' on input $\#w\#^{t_o}$ and accepts w iff M_i' accepts $\#w\#^{t_o}$.

Clearly, $M_{\sigma(i)}$ is equivalent to M_i and $M_{\sigma(i)}$ operates in time

$$O[\sum_{1}^{T_i(n)} n^p] = O[\ T_i(n)^{p+1}] \quad .$$

By setting k = p+1, we have completed the proof.

We conclude by observing that $\tilde{L}_1$ is a p-complete problem, as defined above.

3. Non-writing Devices and Flowcharts

Next we will show that the LBA problem is equivalent to problems about eliminating non-determinism in some very simple non-writing automata. Then we will use this result to show that the LBA problem is also equivalent to eliminating non-determinism from a single 10-variable elementary flowchart by using more variables.

A k-head finite automaton (or a multi-head finite automaton) is a one-tape Turing machine with k read-only heads, $k = 1,2,3,\ldots$. The input string is written on the tape with special end markers at both ends of the input, and the finite automaton is so designed that the read-heads cannot leave the input. The automaton is an accepting device and an input is accepted if, after starting the automaton in its starting state with all heads on the left end marker, the automaton enters an accepting state and halts. We assume that the automaton is capable of sensing when two heads are on the same tape square. We distinguish between deterministic and non-deterministic multi-head automata.

We first establish a relationship between linearly bounded languages and ℓogn - tape bounded languages over one-letter alphabets, due to Savitch [15].

For any language A over an alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$, $A \subseteq \Sigma^*$, let

$$\mathrm{TALLY}(A) = \{1^{n(w)} | w \text{ in } A\} ,$$

where n maps each word w in Σ^* onto the number n(w) which w denotes in k-adic notation, that is

$$n(a_{i_0} a_{i_1} \ldots a_{i_t}) = \sum_{j=0}^{t} a_{i_j} k^j .$$

(where we interpret a_i as i).

Clearly, this mapping establishes a one-one correspondence between strings over Σ and non-negative integers; zero is denoted by the null string.

Lemma 8: The language A, $A \subseteq \Sigma^*$ with $|\Sigma| = k$, is accepted by a deterministic (non-deterministic) linearly bounded automaton if and only if TALLY(A) is accepted by a deterministic (non-deterministic) ℓogn - tape bounded Turing machine.

Proof: Since going from A to TALLY(A) the length of every string is increased exponentially, for input 1^{n_i} the ℓogn_i-tape bounded Turing machine has as much tape available as the ℓba has for input n_i. Thus a ℓogn - tape bounded Turing machine can accept TALLY(A) if an ℓba can accept A. Conversely, if $A = \{1^{n_i}\}$ is accepted by a ℓogn - tape bounded TM, then $\{n_i\}$, where n_i is written in k-adic notation with $k \geq 2$, can be accepted by an ℓba which simulates the ℓogn - tape bounded TM. Since this ℓba, has enough tape to carry out the simulation we have the desired result.

Thus we immediately obtain the following result.

Corollary 9: The deterministic and non-deterministic context-sensitive languages are the same if and only if the deterministic and non-deterministic ℓogn - bounded languages over one-letter alphabets are the same.

At the same time it is known that:

Lemma 10: The language A, $A \subseteq \Sigma^*$, is accepted by a deterministic (non-deterministic) multi-head finite automaton if and only if A is accepted by a deterministic (non-deterministic) ℓogn - tape bounded Turing machine.

Proof: (For a more complete proof see [5]). The basic idea of the proof is that a ℓogn - tape bounded TM can count up to n (k-times) and thus can encode the k-head positions of a k-head automaton, say in binary form, on the ℓogn - tape and use this encoding for a stepwise simulation of the k-head finite automaton. Thus every set accepted by a k-head automaton is also accepted by a ℓogn - tape bounded TM.

Conversely , every ℓogn - tape bounded Turing Machine can be simulated by a k-head finite automaton which encodes the tape content of the ℓogn - tape bounded Turing machine by its head positions on the input tape. Since a ℓogn - tape we can record no more than n^p different patterns (for some p), we see that on input of length n , p heads can encode all these patterns. With a few additional bookkeeping heads, utilizing the encoded ℓogn - tape bounded TM tape content, the k-head automaton can simulate the ℓogn - tape bounded TM. Thus every ℓogn - tape bounded language can be accepted by a multi-head automaton. Since these considerations hold for deterministic as well as non-deterministic automata, we have completed the outline of the proof.

From this we get Savitch's result.

Corollary 11: The deterministic and non-deterministic context-sensitive languages are the same if and only if the languages over a one-letter alphabet accepted by the deterministic and non-deterministic multi-head finite automata are the same.

Next we show that we can strengthen this result by using the language

$$\mathrm{TALLY}(L_1) ,$$

where L_1 is the "universal" $cs\ell$ defined before.

Theorem 12: 1. The language TALLY(L_1) is recognizable by a k_0-head non-deterministic automaton.

2. TALLY(L_1) is recognizable by a deterministic $(k_0 + p)$-head automaton iff DCSL = NDCSL.

Proof: Since L_1 is a $ndcs\ell$ we know, from our previous results, that TALLY(L_1) is accepted by a k_0-head non-deterministic automaton. (k_0 can be explicitly computed from L_1).

Similarly, if TALLY(L_1) can be accepted by a deterministic (k_0+p)-head automaton then we know that L_1 can be accepted by a $d\ell ba$, and vice versa. But then, using Theorem 1 , we get that TALLY(L_1) is deterministically recognizable on some (k_0+p) head automaton iff DCSL = NDCSL, as was to be shown.

The next result shows that the number of heads k_0 in the previous result can be reduced to 7 heads.

For $L \subseteq a^*$ define

$$L^{[k]} = \{a^{n^k} \mid a^n \text{ in } L\}.$$

Corollary 13: 1. The language

$$[\mathrm{TALLY}(L_1)]^{[k_0]}$$

is accepted by a 7-head non-deterministic finite automaton.

2. $[\mathrm{TALLY}(L_1)]^{[k_0]}$ is accepted by a determininistc multi-head automaton iff DCSL = NDCSL.

Proof: Follows from the next lemma.

Lemma 14: Let A, $A \subseteq a^*$, be a set accepted by a non-deterministic k-head finite automaton. Then

$$A^{[k]} = \{a^{n^k} \mid a^n \text{ in } A\}$$

is accepted by a 7-head non-deterministic finite automaton and A is accepted by a deterministic multi-head finite automaton if and only if $A^{[k]}$ is accepted by a deterministic multi-head finite automaton.

Proof: The main tool in this proof is the method of encoding the position of the k-heads of a finite automaton M on the input a^n by one head of an automaton M_1 an input a^{n^k} and then, using six additional read-only heads, to carry out a simulation of M by M_1. The essential steps in the simulation are described below. First we note that with five read heads a deterministic finite automaton can check whether the input a^t is such that $t = n^k$, for some n. Thus the format of the input can be checked and a head can be placed on the n-th tape square if $t = n^k$.

To encode the k heads of M on input a^n as one head position of M_1 on input a^{n^k}, order the k-heads arbitrarily and place the "encoding" head of M_1 on the r-th tape square $1 \leq r \leq n^k$ with

$$r-1 = (d_1-1)+(d_2-1)n+(d_3-1)n^2+\ldots+(d_n-L)n^{k-1}$$

iff the i-th head of M, $1 \leq i \leq n$, is on the d_i-th tape square. After this by a lengthy but straightforward argument one can show that M_1 can carry out a step by step simulation of M and thus M accepts input a^n if and only if M_1 accepts input a^{n^k}.

Clearly, if A can be accepted by a deterministic multi-head finite automaton then so can $A^{[k]}$ for any k. If $A^{[k]}$ can be accepted by a deterministic p-head finite automaton M_2 then we can design a p(k+1)-head deterministic automaton M_3 which accepts A.

For input a^n the automaton M_3 will simulate M_2 on input a^{n^k} as follows: M_3 uses the first p heads to mimick the p heads of M_2 , as long as these heads stay on the first n tape squares. If a head of M_2 goes further than the first n tape squares (recall, the simulated input is n^k long) then k heads are used on the input of length n to count how far the head has moved. Since we can count up to n^k with k-heads on an input of length n, the (k+1)p heads suffice for M_3 on input a^n to simulate M_2 on input a^{n^k}. Thus M_3 accepts a^n if and only if M_2 accepts a^{n^k}, but then M_3 accepts A. Thus $A^{[k]}$ is a deterministic language if and only if A is. This completes the proof.

Next we show that the previous results have a natural interpretation for flowchart computations, thus relating the classic non-determinism problem for context-sensitive languages to a somewhat more programming oriented problem.

We say that a flowchart is <u>elementary</u> (or an <u>E-flowchart</u>) if and only if it is a flowchart made up of the assignment statements

$$x := x \dot{-} 1$$

$$x := y$$

and the tests

$$x = 0$$

$$x = y \ .$$

An E-flowchart is <u>deterministic</u> if and only if every assignment statement and every test branch leads to exactly one assignment statement or test. If some assignment statement or test branch leads to more than one assignment or test, or leads to one or more assignments and tests, then the flowchart is <u>non-deterministic</u>.

The flowcharts are used as accepting devices of sets of integers. The integer n is accepted if and only if the flowchart computation with the first variable set equal to n ends at some exit labelled with "accept". Otherwise the input is rejected. (Note that the accepting condition can be handled in many different ways. For example, we could have demanded that the computation halts and that a specified variable is set to one for accepting and zero for rejecting).

__Theorem 15:__ There exists a set of integers L accepted by a non-deterministic E-flowchart with 10 variables such that the following two statements are equivalent.

1. L is accepted by a deterministic E-flowchart.
2. DCSL = NDCSL.

__Proof:__ The proof consists of a reasonably straightforward simulation of multi-head automata by E-flowcharts and vice versa.

In simulating the flowcharts on k-head automata the head positions on the input tape encode the contents of the variables of the flowchart and vice versa. The three additional variables are needed to obtain a subflowchart which performs the assignment

$$x := x+1$$

for x less than the input variable and to permanently store the input. This completes the outline of the proof.

For related results see Warkentin and Fischer [16].

We do not know whether the number of heads or the number of flowchart variables can be reduced further in the two previous results. We conjecture, however, that this is the case. We believe that it would be worthwhile to investigate the non-deterministic k-head automata languages over a one symbol alphabet for k = 2 and 3. The case of 2 heads seems simple and it would be very interesting to determine whether all 2-head non-deterministic finite automata can be replaced by equivalent deterministic multi-head finite automata. It is our hope that these k-head automata with small values of k may provide a place where some further insights can be gained into the LBA problem and more generally, in the nature of non-determinism in computing.

We also believe that the linearly bounded automata with oracles deserve further investigation. The main problem here is to determine whether there exist recursive oracles such that the deterministic and non-deterministic ℓba language accepted with these oracles are different.

It is interesting to note that T. Baker [2] has shown that there are recursive oracles for which the deterministic and non-deterministic polynomial time-bounded TM computations are the same and that there are other oracles for which they are different.

4. Decision Problems about Regular Expressions

In this section we show that the LBA problem can be related to several natural decision problems about regular expressions. This approach yields many hardest tape recognizable languages which appear more natural than the "universal" context-sensitive language constructed in the second section.

The main tool in this work is the observation made by Meyer and Stockmeyer [12] that restricted regular expressions can be used to describe invalid ℓba computations very economically. Thus these results, as well as many other results about the complexity of various decision problems [6,7,12], should also be viewed as results about the descriptive power of regular expressions.

A restricted regular expression or simply a regular expression is any valid expression over the alphabet consisting of 0,1,·,+,* and the delineation (,). The operators ·, + and * have their well-known meaning of concatenation, set union and Kleene closure. For a regular expression R the set of sequences described by R is designated by L(R).

Next we look at valid ℓba computations which will be used to link the LBA problem to the complexity of several decision problems about regular expressions. Consider an ℓba M with tape alphabet T and state set Q working on an input $y = x_1 \ldots x_n$. At each discrete time interval during the computation we can describe the state of the computation by giving the tape content, the head position of M and its state. If the computation is deterministic then after k steps of computing there will be a unique configuration describing the situation and for a non-deterministic ℓba there will be a set of possible configurations. To make these ideas more precise we will refer to a sequence

$$x_1x_2 \ldots x_{j-1}(q_1x_j)x_{j+1} \ldots x_n$$

as an <u>instantaneous</u> <u>description</u>. This sequence means that the tape content is $x_1 \ldots x_n$ the ℓba is in state q and the reading head is scanning the j-th tape symbol x_j. Thus an instantaneous description is any string in $[T + (Q \times T)]^*$, which contains exactly one symbol in $Q \times T$. If the start state is q_0 then

$$(q_0,x_1)x_2 \ldots x_n$$

is an <u>initial</u> <u>configuration</u> and any configuration containing a halting state is a <u>final</u> <u>configuration</u>. One instantaneous description ID_{i+1} <u>follows</u> ID_i if and only if there exists a move of M which changes ID_i in one operation to ID_{i+1}. A <u>valid</u> <u>computation</u> of M an input $y = x_1 \ldots x_n$ is a sequence of instantaneous descriptions

$$\#ID_0\#ID_1\#ID_2\# \ldots ID_t\#$$

where ID_0 is the initial configuration on the input $x_1 \ldots x_n$, i.e.,

$$ID_0 = (q_0,x_0)x_2x_3 \ldots x_n ,$$

ID_t is a final configuration and for all i, $0 \leq i \leq t$, ID_{i+1}

follows ID_i. We denote the set of all valid computations of M on input y by $R_M(y)$. Thus we see that y is accepted by M if and only if

$$R_M(y) \neq \phi ,$$

or equivalently, the set of <u>invalid</u> <u>computations</u> of M and y $\bar{R}_M(y)$, must not contain all sequences, i.e.

$$I_M(y) = \bar{R}_M(y) \neq \Sigma^* .$$

The main observation [due to Meyer and Stockmeyer] is that for every ℓba M and input y the set of invalid computations of M on y is a regular set and that it can be described by a restricted regular expression such that

$$|I_M(y)| \leq c_M|y|,$$

and furthermore that a deterministic ℓba can map y onto $I_M(y)$. This is the critical step in the argument which links ℓba computations to regular expressions.

Thus we have the following result:

<u>Theorem 16</u> (Meyer and Stockmeyer): Let M be a non-deterministic LBA with tape symbol set T, state set S, and set of designated accepting states F, with $F \subseteq S$. Let all accepting states be final. Let q_0 be the unique start state of M. Let $y = x_1x_2...x_n$ be an input to M.

Then there is a deterministic ℓba M' such that M', started with $\#(q_0,x_1)x_2...x_n\#$ on its tape, halts with a regular expression

β_y over Σ on its tape such that

$$L(\beta_y) = \overline{R_M(y)} = I_M(y) \ .$$

Proof: We only sketch the idea of the proof.
For a complete proof see [6].

β_y is the union of:

β_1 , the set of strings that do not begin with $\#(q_0,x_1)x_2\ldots x_n\#$;

β_2 , the set of strings that do not contain a symbol (q_f,t), where $q_f \in F$.

β_3 , the set of words that make a mistake between one i.d. and the next (i.e., ID_{j+1} doesnot follow from ID_j by one application of a move rule of M.)

But, $\beta_1 = [(\Sigma-\#) \cup \#\cdot[(\Sigma-(q_0,x_1)) \cup (q_0,x_1)\cdot$
$[(\Sigma-x_2) \cup x_2[\ldots\cup x_n[(\Sigma-\#)]\ldots] \ ] \ ] \ ]\cdot \Sigma^*$

The reader should note the similarity of the above to Horner's method for evaluating polynomials, i.e.,

$$a_0+a_1x+a_2x^2+\ldots+a_nx^n = a_0+x[a_1+x[a_2+\ldots+x[a_n]\ldots]] \ .$$

$$\beta_2 = [\Sigma-(\bigcup_{q_f\in F} \{q_f\} \times T)]^* \ .$$

$$\beta_3 = \bigcup_{\sigma_1,\sigma_2,\sigma_3\in\Sigma} \Sigma^*\cdot\sigma_1\cdot\sigma_2\cdot\sigma_3\cdot \Sigma^{|y|-2}\cdot [\Sigma^3-f_M(\sigma_1,\sigma_2,\sigma_3)]\cdot \Sigma^*,$$

where $f_M\colon \Sigma^3 \to 2^{\Sigma^3}$, which essentially maps correct triples of symbols into correct triples. Essentially β_3 says that mistakes occur n symbols apart.

The remainder of the proof consists in noting that $|\beta_y| \leq C_M \cdot |y|$ and that given y, the time required to deterministicly write out β_y is bounded by a polynomial in $|\beta_y|$.

Using Theorem 16 a simple coding argument yields

<u>Theorem 17</u>: Let

$$L_4 = \{(R_1, R_2) | R_1 \text{ and } R_2 \text{ are regular expressions over } \{0,1\} \text{ and } L(R_1) \neq L(R_2)\}$$

Then $L_4 \in$ NDCSL and $L_4 \in$ DCSL iff NDCSL = DCSL.

<u>Proof</u>: If L_4 is in DCSL then we can check $L(\beta_y) \neq \Sigma^*$ on a deterministic ℓba and from Theorem 16 it follows that DCSL = NDCSL.

To see that L_4 is in NDCSL, we note that to verify

$$L(R_1) \neq L(R_2)$$

we need only to give a string x one symbol at a time and verify that

$$x \in [L(R_1) \cap L(\bar{R}_2)] \cup [L(\bar{R}_1) \cap L(R_2)] .$$

This can be done on a non-deterministic ℓba in a straightforward way, which completes the proof.

We next extend Theorem 17 to prove a metatheorem about the deterministic tape complexity of many decision problems about the regular sets. Define

$x\backslash L = \{w \mid x\,w \in L\}$ and $L/x = \{w \mid w\,x \in L\}$.

Theorem 18: Let P be any predicate on the regular sets over $\{0,1\}$ such that

1) $P(\{0,1\}^*)$ is true and

2) $\mathscr{P}_L = \bigcup_{x \in \{0,1\}^*} \{x\backslash L \mid P(L) = \text{True}\}$

[or $\mathscr{P}_R = \bigcup_{x \in \{0,1\}^*} \{L/x \mid P(L) = \text{True}\}$]

is not the set of all regular sets over $\{0,1\}$.

Then

$\{R \mid R$ is a regular expression over $\{0,1\}$ and $P[L(R)] = \text{False}\}$

in DCSL implies NDCSL = DCSL.

Similarly,

$\{R \mid R$ is a regular expression over $\{0,1\}$ and $P[L(R)] = \text{True}\}$

in NDCSL implies that NDCSL is closed under complementation.

Proof: Let L_0 be a regular set over $\{0,1\}$ not in $\mathscr{P}_L$. Let $h_0(0) = 00$ and $h_0(1) = 01$. Then given R_i a regular expression over $\{0,1\}$ we can effectively find in linear space and deterministic polynomial time in $|R_i|$ a regular expression R_j such that

$$L(R_j) = h_0(L(R_i))\ 10\ (0 + 1)^* +$$
$$(00 + 01)^*\ 10\ L_0 + \overline{(00 + 01)^* 10 (0 + 1)^*}$$

$$= h_0(L(R_i))\ 10\ (0 + 1)^* + (00 + 01)^*10\ L_0 + (00 + 01)^*\ [\Lambda + 0 + 1 + 11(0 + 1)^*]$$

Case 1:

$$L(R_i) = (0 + 1)^* .$$

Then

$$h_0(L(R_i)) = (00 + 01)^*$$

and

$$L(R_j) = (0 + 1)^* .$$

Hence,

$$P(L(R_j)) = \text{True}.$$

Case 2:

$$L(R_i) \neq (0 + 1)^* .$$

Then

$$\exists\ x \in (0 + 1)^* - L(R_i) .$$

Hence

$$h_0(x) \in (00 + 01)^* - h_0(L(R_i)).$$

But

$$P(L(R_j)) = \text{True}$$

implies

$$h_0(x)\ 10 \backslash L(R_j) = L_0 \in \mathscr{P}_L .$$

Hence

$$P(L(R_j)) \text{ is FALSE.}$$

Therefore,

$$P(L(R_j)) = \text{TRUE}$$

if and only if

$$L(R_i) = (0 + 1)^* ,$$

Thus if $P(L(R_j)) = \text{False}$ is decidable by a dℓba then so is $L(R_i) \neq (0+1)^*$, and therefore, by our previous results it follows

<u>Corollary 19</u>: DCSL = NDCSL iff any one of the following languages is in DCSL. Similarly, NDCSL is closed under complementation iff the complement of any one of the following languages is in NDCSL:

1. $\{R|R$ is a regular expression and $L(R) \neq \{0,1\}^*\}$;
2. $\{R|R$ is a regular expression and $L(R) \neq L(R^*)\}$;
3. $\{R|R$ is a regular expression and $L(R) = L(R)^{REV.}\}$;
4. $\{R|R$ is a regular expression and $L(R)$ is cofinite$\}$;

5.$(\forall k \geq 1)$ $\{R|R$ is a regular expression and $L(R)$ is not k definite$\}$.

It is interesting to note that in the proofs of Theorems 16 and 17 we only used regular expressions of star-height 1 (i.e., no nested *'s). Thus if there exists a regular expression R_0 of star-height 1 not in $\mathscr{P}_L$, then Theorem 18 can be changed to read

"$\{R|R$ is a regular expression over $\{0,1\}$ of star-height 1 and $P[L(R)] = \text{False}\}$ in DCSL implies DCSL = NDCSL".

We finally note that all the languages in Corollary 19 can be choosen to be of star-height 1. Thus we get, for example,

<u>Corollary 20</u>: The language

$\{R|R$ is a regular expression of star-height 1 and $L(R) \neq L(R^*)\}$

is a tape and time hardest recognizable csℓ.

We conclude by stating a result obtained by Hunt which indicates further similarities between the LBA problem and the NDPTIME problem. From the above observations we know that even if we restrict ourselves to regular expressions of star-height 1, then the language

$$\{(R_i, R_j) \mid L(R_i) \neq L(R_j)\}$$

is a hardest tape recognizable CSL. The next result shows that if we drop the Kleene star completely then we get a p-complete problem.

<u>Theorem 21</u>: Let R_i, R_j be regular expressions over 0, 1, $\cdot$, $+$. Then

$$L = \{(R_i, R_j) \mid L(R_i) \neq L(R_j)\}$$

is a p-complete problem. Thus L is in DPTIME iff

NDPTIME = DPTIME.

<u>Proof</u>: See [6].

5. Bibliography

[1] Book, R.V., "Comparing Complexity Classes", Technical Report No. 1-73, Harvard University, Cambridge, Mass.

[2] Baker, T.P., "Computational Complexity and Non-determinism in Flowchart Programs", Ph.D. Dissertation 1973, Cornell University, Ithaca, N.Y.

[3] Fagin, R., "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets". These proceedings.

[4] Greibach, S., "Jump PDA's, Deterministic Context-Free Languages , Principal AFDL's and Polynomial Time Recognition". Proceedings of Fifth Annual ACM Symposium on Theory of Computing (1973), 20-28.

[5] Hartmanis, J., "On Non-Determinacy in Simple Computing Devices", Acta Informatica, (1972), 334-336.

[6] Hunt, H.B. III., "On Time and Tape Complexity of Languages", Ph.D. Dissertation 1973, Cornell University, Ithaca, N.Y.

[7] Hunt, H.B. III., "On Time and Tape Complexity of Languages I", Proceedings of Fifth Annual ACM Symposium on Theory of Computing", (1973), 10-19.

[8] Karp, R., "Reducibilities Among Combinatorial Problems", in R. Miller and J. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, (1972), 85-104.

[9] Kuroda, S.Y., "Classes of Languages and Linearly-Bounded Automata", Information and Control 2 (1964), 207-223.

[10] Landweber, P.S., "Three Theorems on Phrase Structure Grammars of Type 1", Information and Control 2 (1963), 131-136.

[11] Lewis, P.M., R.E. Stearns and J. Hartmanis, "Memory Bounds for Recognition of Context-Free and Context-Sensitive Languages", IEEE Conference Record on Switching Circuit Theory and Logical Design (1965) 191-202.

[12] Meyer, A. and L. Stockmeyer, "The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space", Conf. Record IEEE Thirteenth Annual Symposium on Switching and Automata Theory, (1972), 125-129.

[13] Myhill, J., "Linearly Bounded Automata", WADD Technical Note 60-165, June 1960.

[14] Savitch, W.J., "Relations Between Non-deterministic and Deterministic Tape Complexities", JCSS 4 (1970), 177-192.

[15] Savitch, W.J., "Multi-head Automata and Context-Sensitive Languages", Technical Report, July 1972, Dept. of Applied Physics and Information Science, University of California, San Diego, California.

[16] Warkentin, J.C. and P.C. Fischer, "Predecessor Machines and Regression Functions", Proceedings of Fourth Annual ACM Symposium on Theory of Computing (1972), 81-87.