Acta Informatica 6, 77—88 (1976)
© by Springer-Verlag 1976

Hierarchies of Complete Problems*

Zvi Galil

Received June 4, 1975

Summary. An attempt is made to present a framework for the diverse complete problems that have been found. A new concept—a Hierarchy of Complete Problems is defined. Several hierarchies in various domains such as graph theory, automata theory, theorem proving and games are established.

1. Introduction

In [1] S. Cook introduced the notions of polynomial time reducibility and complete languages in NP-TIME.[1] Moreover, he showed that some natural combinatorial problems[2] are complete in NP-TIME. Since then, using several efficient reducibilitiers, reasearches have found many other complete problems in DLOG, NLOG, P-TIME, NP-TIME and P-SPACE ([13, 11, 12, 9, 15], etc.). Although DLOG$\subseteq$NLOG$\subseteq$P-TIME$\subseteq$NP-TIME$\subseteq$P-SPACE we cannot find in the literature any quintuple of complete problems, one in each of these five classes of languages, such that each is a special case of the next. The goal of this paper is to present such hierarchies of complete problems.

First we recall some well-known definitions:

Definitions. Let C be a class of functions, A and B be sets and S a class of sets.

(i) $A <_c B$ if there is a function $f \in C$ such that for every w, $w \in A$ iff $f(w) \in B$.[3]

(ii) $S <_c A$ if for every $B \in S$ $B <_c A$.

(iii) A is $<_c$ complete in S if

(a) $S <_c A$ and

(b) $A \in S$.

In this paper we restrict ourselves to the set C of functions which can be computed by a Turing machine (Tm) which uses logarithmic tape. Thus we write $<$

* Some of the results in this paper were presented in the John Hopkins Conference on Information Sciences and Systems, April 1975.

1 We use the following notation: DLOG (P-TIME, P-SPACE) is the set of languages accepted by deterministic Turing machines which operate in logarithmic space (polynomial time, polynomial space respectively); NLOG (NP-TIME) is the set of languages accepted by nondeterministic Turing machines which operate in logarithmic space (polynomial time).

2 We use the terms sets, languages and problems interchangeably since all problems that we discuss have 0–1 solutions.

3 This is many-one reducibility. There are some other efficient reducibilities which are discussed in [14]. They are analogous to the various reducibilities of recursion theory (cf. [16]), but we do not consider them here.

rather than $<_c$ (this is also denoted $<_{\log}$ in the literature) and complete instead of $<$-complete. We justify our restriction by the fact that most (if not all) of the reductions in the literature which are used to describe specific complete problems are $<_{\log}$.[4] In one case, however, this restriction does not make much sense, since every $L \in$ DLOG is $<_{\log}$ complete in DLOG. To overcome this difficulty Jones has defined ([11]) the logn rudimentary functions. All the reductions in the sequel are $<_{\log}$. Since they are fairly straightforward in all cases, we omit the tedious details of the proofs that the reductions can be done using logarithmic space. In the case of DLOG our reductions can be shown also to be rudimentary.

The importance of complete problems is two-fold:

1) Often, a question about a class of languages can be translated into a seemingly simpler question about one language in the class, namely the complete language. In particular the question "Are A and B equal?" where $A \subseteq B$ can be translated into the question "Is L in A?" for L complete in B; and the question "Is A closed under complementation?" can be translated into "Is $\bar{L}$ in A?". Thus, most of the important open problems of computational complexity can be translated into questions about the corresponding complete problems.

2) In some sense the complete problem is the hardest in the class since every problem in the class can be efficiently encoded into it. Thus, studying a complete problem sheds light on the whole class that it is complete in.

We now give the definition which is of central importance to our paper.

Definition. *A hierarchy of complete problems (HCP)* is a quintuple of problems $(A_1, A_2, A_3, A_4, A_5)$ that satisfies:

(1) A_1 is complete in DLOG, A_2 is complete in NLOG, A_3 is complete in P-TIME, A_4 is complete in NP-TIME, and A_5 is complete in P-SPACE; and

(2) For $1 \leqq i \leqq 4$ A_i is a special case of A_{i+1}, i.e. A_i is obtained from A_{i+1} by adding some restrictions to A_{i+1}.

In this paper we prove the existence of several hierarchies of complete problems in varied areas such as graph theory, automata theory, theorem proving and games. Although we introduce quite a few new complete problems, this is not the purpose of this paper. Our goal is to present a framework for the diverse complete problems that have been found. So far the numerous known complete problems do not fit a unified framework. We also investigate the question of what changes make a problem easier. Since we believe that DLOG $\subset$ NLOG $\subset$ P-TIME $\subset$ NP-TIME $\subset$ P-SPACE, a restriction which is added to a problem A_{i+1} in an HCP to get A_i really makes the problem easier. Recently ([6]) a closely related question was investigated—i.e.: Which restrictions on a problem do not affect its complexity? It was shown that some strictly restricted versions of complete problems in NP-TIME are still complete in NP-TIME.

As we said before, our intention was not to introduce new complete problems, but our paper includes for the first time simple combinatorial problems which are complete in P-SPACE. Until now, the only natural complete problems in P-SPACE were either questions about automata (like equivalence of regular expressions) or

4 This was first observed in [15] and [11].

about logic ([15]). Our combinatorial problems are (a) to find a repeating pattern in a graph and (b) to decide whether a given person is going to die in a kind of game of life ([5]).[5]

Our hierarchies consist of complete problems in classes of feasible computations (DLOG, NLOG and P-TIME) and in classes in which computations cannot yet be proven to be unfeasible (NP-TIME and P-SPACE).

In Section 2 we construct a graph HCP, in Section 3 we give two HCP's in automata theory, in Section 4 an HCP in theorem proving and in Section 5 two HCP's about games. In all the proofs we show that the various problems are complete in the corresponding classes. By definition, to prove a problem A is complete in B we have to show (1) that it belongs to B and (2) that every problem in B can be reduced to A. In all cases except in some of the complete problems in automata theory, the first part is straightforward and we omit its proof. To prove part (2) it is enough to show that $A' < A$ when A' is a known complete problem in B. We use CNF-SATISFIABILITY ([1, 13]) for NP-TIME, PATH ([2, 12]) for P-TIME, DGAP, and GAP ([11]) for DLOG and NLOG respectively. (The latter is the graph version of the maze language of [17].) To prove that a problem A is complete in P-SPACE it will suffice to show that DLBA $< A$ where DLBA is the class of languages accepted by deterministic linear bounded automata (dlba's). This follows from the fact that there are complete problems in P-SPACE which belong to DLBA. In some cases, in order to prove that DLBA $< A$ or NP-TIME $<$ A, we will use direct encoding of a given Tm M and input x as an instance of A. This method is similar to the one we used in [4] which was inspired, in turn, by Cook's proof that SATISFIABILITY is complete in NP-TIME ([1]). We discuss this method here briefly.

First, we consider the case of NP-TIME. Given a nondeterministic Tm M with input x and the upper bound on the time $N = p(|x|)$ we use N instantaneous descriptions (i.d.'s) of length N to describe the computation. The problem will have an object $x^k_{i,j}$ (vertex in the case of graphs and literal in the case of theorem proving) corresponding to the possible fact that at time i the j-th symbol of the i.d. is σ_k. A solution to the problem A will consist of certain choices of these objects which describe a complete accepting computation of M on x as follows: It will contain objects which represent a sequence of i.d.'s of M. The structure of the problem will guarantee that (1) the first i.d. is the initial i.d. of M on $x(N-|x|$ blanks are added); (2) the final i.d. is accepting; and (3) the $(i+1)$th i.d. is one which can follow from the i-th i.d. by a move of M. The three properties will guarantee that A has a solution iff M accepts x. Actually this is a somewhat simplified description of the method, since in order to implement (3) above we will need some extra objects, however the total number of the objects will not exceed a polynomial in $|x|$. For the LBA case, we cannot use exactly the same trick since the number of i.d.'s can be exponential. We overcome this difficulty by using objects $x^k_{i,j}$ with $i \in \{0, 1\}$ only. $i = 0 (i = 1)$ will mean that the number of moves until now is even (odd). We will be able to encode the lba's action into an instance of A since the same object in A may represent many possible facts e.g. $x^k_{0,j}$ might

5 Very recently [3], another combinatorial problem complete in P-SPACE was discovered.

stand for the fact that at time 10 the j-th symbol in the i.d. is σ_k and at the same time it can stand for the same fact with time 16. The problem will allow repetition of objects in a way that the context in which some choice of $x_{i,j}^k$ is made (it might be chosen many times) determines the exact time t ($t \bmod 2 = i$) in which the j-th symbol is σ_k.

In many cases we only sketch the proofs. We hope that the reader will be able to complete the missing details.

2. A Graph HCP: Finding a Grid in a Directed Graph

We are given an arbitrary directed graph, k distinguished nodes—*the source nodes* and another distinguished node—*the goal node.* Any set of k distinct nodes is called a *layer.* A layer $(j_1, j_2, \ldots, j_k)$ *can follow* a layer $(i_1, i_2, \ldots, i_k)$ if there are directed edges $(i_1, j_1), \ldots, (i_k, j_k)$ and $(i_1, j_2), \ldots, (i_{k-1}, j_k)$ in the graph. A *grid* is a sequence of layers such that (1) the first layer consists of the source nodes; (2) the last layer contains the goal node; and the n-th layer can follow the $(n-1)$st layer. Note that a node can appear in more than one layer.

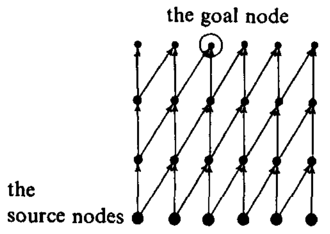
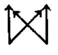


Fig. 1 A grid ($k = 6$)

We now define five problems:

A_5: Input (1) a directed graph (2) an integer k (3) a list of k source nodes (4) a goal node. Does the graph contain a grid?

A_4: Same as A_5 except that we do not allow repetition of nodes in a grid.

A_3: Same as A_4 except that we allow as inputs only *simple graphs* (by simple graph we mean a directed graph without the subgraph $C_{2,2}$: ).

A_2: Same as A_3 except that $k = 1$.

A_1: Same as A_2 except that input graphs are further restricted to have nodes with outdegree 1.

Theorem 1. $(A_1, A_2, A_3, A_4, A_5)$ is an HCP.

Proof. First we show that A_4 is complete in NP-TIME. Given a nondeterministic Tm M which operates in polynomial time and an input x we can encode M and x into an instance of A_4 in such a way that the latter contains a grid iff M accepts x.

Assume that Σ is an alphabet which suffices to describe i.d.'s of M and that $N = p(|x|)$ is the time bound for M. We define the graph G with nodes V, $V =$

$\{v^{a}_{i,j}, v^{cd}_{k,l}, v^{efg}_{m,n} \mid 0 \leqq i, j \leqq N, 0 \leqq k, l, m \leqq N-1, 1 \leqq n \leqq N-1$ and $a, b, c, d, e, f \in \Sigma\}$ $= V_1 \cup V_2 \cup V_3$.[6]

The meaning of $v^{a}_{i,j}$ is that at time i the j-th symbol in the i.d. is a. The meaning of $v^{cd}_{k,l}$ is that at time k the l-th and $(l+1)$st symbols of the i.d. are c and d. Similarly the meaning of $v^{efg}_{m,n}$ is that at time m the $(n-1)$st, n-th and $(n+1)$st symbols of the i.d. are e, f and g respectively. The source nodes (the initial layer) will be those describing the initial i.d. of M on x and the goal node will be a node representing the accepting state (without loss of generality we can assume that M accepts in a unique state q, only at time $=N$ while its head scans some given symbol, $\#$, at the leftmost tape cell, thus the goal node is $v^{(\#,q)}_{N,0}$).

A grid (if exists) will consist of a layer from V_1 followed by a layer from V_2 followed by a layer from V_3 followed by a layer from V_1 and so forth. The set of edges defined below will assure that if the first V_1 layer represents an i.d. then (i) the next layer must be in V_2 and will represent the same i.d. in which every two symbols are packed together; (ii) the next layer must be in V_3 and will represent the same i.d. in which every three symbols are packed together; and (iii) the next layer must be in V_1 and will represent an i.d. which can follow from the previous one by one move of M. This will guarantee that M accepts x iff there is a grid (which represents the computation of M on x).

We now define E to be the class of all edges which appear in Fig. 2.

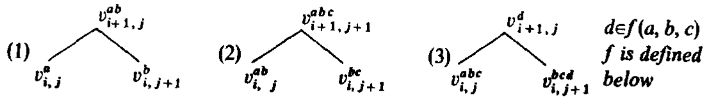


Fig. 2

(i) and (ii) above can be easily checked. For $a, b, c \in \Sigma$, $d \in f(a, b, c) \subseteq \Sigma$ if d is a possible j-th symbol in an i.d. whenever a, b and c are the $(j-1)$st j-th and $(j+1)$st symbols in the previous i.d. (note that d depends only on a, b and c). iii) is not yet satisfied since we must force the next layer to have nodes (from V_1) that correspond to the same choice of a next move of M. Without loss of generality M has exactly two choices in each step, thus $f(a, b, c) = \{d_1, d_2\}$. We duplicate the sets V_1 and V_2 and create new sets U_1 and U_2. The edges will be of the form (1) and (2) of Fig. 2 and (1)′ (2)′ (3)* and (3)′ of Fig. 3.

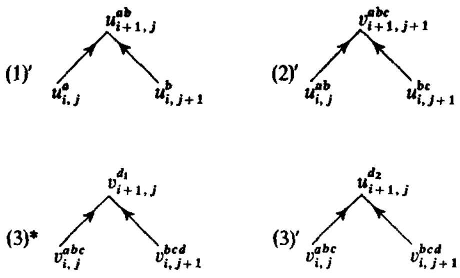


Fig. 3

6 $V_1 = \{v^{a}_{i,j}\}$, $V_2 = \{v^{cd}_{k,l}\}$ and $V_3 = \{v^{efg}_{m,n}\}$.

The u's have the same meaning as the v's except that the v's (u's) represent first (second) choice for next move. One can verify that if we have a layer in V_3 which is followed by a layer that contains elements from V_1 and from U_1, then there cannot be a next layer. Thus if a grid exists the next layer is either in V_1 or in U_1 and the next one is either in V_2 or in U_2 respectively. With this addition (iii) above is satisfied and the proof is completed.

The proof that A_5 is complete in P-SPACE is similar. The changes are: (1) We can delete the U's; (2) i can take the values 0 or 1 only; and (3) the edges are of the forms (1), (2) and (3) in Fig. 1 with $(i+1) \bmod 2$ instead of $i+1$. The meaning of $v^a_{i,j}$ is that at some time k with $k \bmod 2 = i$ the j-th symbol of the i.d. is a. The time k is determined by the number of the layer in which $v^a_{i,j}$ appears (it can appear in many layers).

The proof that A_3 is complete in P-TIME is also similar. Here too we do not need the U's and the computation of M on x can be encoded into a simple graph since M is deterministic. (One can show that subgraphs of type $C_{2,2}$ are possible only due to edges of types (3) and only when there is more than one choice for next move.)

A_2 and A_1 are simply GAP and DGAP ([11]) for simple graph. These are still complete in NLOG and DLOG since every graph G can be converted to a simple graph G' by replacing every edge $e\ \begin{array}{c} v \\ \circ \\ \uparrow \\ \circ \\ u \end{array}$ by two edges and a new node $\begin{array}{c} \\ e_2 \\ \\ e_1 \\ \\ \end{array} \begin{array}{c} v \\ \circ \\ \uparrow \\ \circ \\ \uparrow \\ \circ \\ u \end{array}\ w.$ Obviously there exists a path in G iff there exists a path between the corresponding nodes in G'.

3. Two HCP's in Automata Theory

We consider five automata:

1. dfa — deterministic finite automaton[7]
2. nfa — nondeterministic finite automaton
3. npda — nondeterministic pushdown automaton
4. rcsa — restricted nondeterministic checking stack automaton
5. csa — nondeterministic checking stack automaton.

Except rcsa all the others are well known. Their definition can be found in the literature ([10, 8]). *rcsa* is a csa with the following restriction: once the automaton enters its stack it does not distinguish between stack symbols, i.e. it interprets the contents of the stack as a unary number.

It is obvious that the automata above are ordered in increasing power: The csa has finite control and a stack and after it enters into its stack it cannot change the contents of the stack. The rcsa is csa with the restriction above. By not allowing the rcsa to enter the stack we obtain the npda, by deleting the stack we get nfa and by restricting the automata further to deterministic moves we get the dfa.

Let B_i be the class of problems: "Does A accept the empty word (ε)?" where A varies over machines of type i, $1 \leqq i \leqq 5$.

7 We only allow the dfa to have ε-moves in a way that if some state has an ε-move it cannot have a read move nor can it have more than one ε-move per state.

The discussion above implies that B_i is a special case of B_{i+1} since the i-th machine is a special case of the $(i+1)$th machine $(1 \leqq i \leqq 4)$. Moreover, we have

Theorem 2. $(B_1, B_2, B_3, B_4, B_5)$ is an HCP. Note that for $1 \leqq i \leqq 5$ B_i is of the type 'Is $\varepsilon \in L(A)$?' ($L(A)$ is the language accepted by A). The complexity of this problem characterizes in some sense the computational power of the corresponding automaton. It ignores several factors such as (1) number of heads (2) the difference between the one way and two way device and (3) the 'help' the automaton can get from specific input formats. Thus it mainly characterizes the power of the control of the corresponding machine.

Proof of the theorem. A slight variation of B_1 was shown to be complete in DLOG in [11]. In [11] it was also shown that B_2 is complete in NLOG. In [12] it was shown that B_3 is complete in P-TIME. (npda can easily accept an encoding of admissible path systems.) We show that (1) B_5 is complete in P-SPACE and (2) B_4 is complete in NP-TIME.

(1) To prove that P-SPACE $< B_5$ we show that DLBA $< B_5$: Given a dlba M and an input x, $|x| = n$ we can easily construct a csa A which guesses a string which it pushes onto its stack and then checks whether it is an accepting computation of M on x. A accepts its input iff the check succeeds. It needs no more than $4n$ states ($\sim n$ to check whether the initial i.d. is $\# q_0 x \#$ and $\sim 2n$ states to go back and forth and check whether each i.d. can follow from its predecessor). Thus $\varepsilon \in L(A)$ iff M accepts x.

Now we show that $B_5 \in$ P-SPACE. Given a csa A we first construct a nonerasing csa A' such that $\varepsilon \in L(A)$ iff $\varepsilon \in L(A')$. Note that before A enters the stack it behaves as an npda. Therefore we can use the notation for npda's ([10] chapter 5). Without loss of generality A pushes at most one symbol onto its stack in one move. We change A as follows: if (i) $(q, Z_1 Z_2) \in \delta(p, \varepsilon, Z)$ and (ii) $(p, Z_2) \overset{*}{\underset{A}{\vdash}} (r, \varepsilon)$ we add the rule $(r, Z_1) \in \delta(p, \varepsilon, Z)$. Note that (ii) can be found in polynomial time (it can be transformed to emptiness problems for npda). The new rule prevents A from pushing Z_2 if it is possible that it is going to erase it later, and to move into the state into which it would move if it went through the computation which erases Z_2. Adding the new rules and deleting all the rules which erase a stack symbol we obtain A' such that $\varepsilon \in L(A)$ iff $\varepsilon \in L(A')$. Now, by [10] acceptance for nonerasing stack automaton can be checked in space proportional to the square of the number of the states of A'. Thus "Does A accept ε?" can be checked in polynomial space.

(2) We first show that NP-TIME $< B_4$ by showing that CNF-SATISFIABILITY $< B_4$. Let α be a boolean formula in CNF with l variables. Let A be the following rcsa: On every input A first pushes Z^k on its stack for some k chosen nondeterministically. Then it checks whether α is satisfied by the assignment $x_i = \begin{cases} 0 & \text{if } k \bmod p_i = 0 \\ 1 & \text{otherwise} \end{cases}$ where p_i is the i-th prime, $1 \leqq i \leqq l$. If α is satisfied by this assignment A accepts the input. Obviously $\varepsilon \in L(A)$ iff α is satisfiable. A needs $\sim N = \sum_{i=1}^{l} p_i$ states and by the Prime Number Theorem $N < l^3$.

To show that $B_4 \in$ NP-TIME we show that B_4 can be reduced to the nonemptiness problem for two way nondeterministic finite automata (2 nfa) over a one symbol alphabet $\Sigma=\{a\}$. This will complete the proof since the latter is in NP-TIME by the next lemma. As we saw above, given an rcsa A we may assume that A is non-erasing. We construct a 2nfa over one symbol alphabet B such that $L(B)\neq\phi$ iff $\varepsilon\in L(A)$. B will have two modes of operation: in mode 2 it will behave exactly as A behaves after it enters its stack and will accept if A does. (By definition of rcsa it behaves exactly as 2nfa over one symbol alphabet at this stage. Also remember that we are looking at acceptance of ε only.) In mode 1 B will behave as a one way nfa. It will have additional states $\{[q,Z]\,|\,q$ is a state of A and Z a stack symbol of $A\}$ and will start at $[q_0Z_0]$ (q_0 is the initial state of A and Z_0 is its bottom of stack symbol). It will move as follows: if q is a state of A in which it enters its stack, then B passes to mode 2; otherwise $[q'Z']\in\delta^B([qZ],a)$ if $(q,Z)\overset{*}{\underset{A}{\vdash}}(q',\bar{Z}Z')$. Note that this last condition can be checked in polynomial time (as above it can be reduced to emptiness problem for npda). Note that during the first stage B can scan a^k iff the length of the stack at the end of the first stage can be k. Thus $L(B)\neq\phi$ iff $\varepsilon\in L(A)$. This completes the proof.

In the proof above we used the following Lemma:

Lemma. The nonemptiness problem for two way nondeterministic finite automaton (2nfa) languages over one symbol is in NP (in fact it is complete in NP).

Proof. Given a 2nfa A with n states, by [10] there is an equivalent nfa B with $N\leqq n^{n+1}$ states. Thus $L(A)\neq\phi$ iff there is $k\leqq n^{n+1}$ such that $a^k\in L(A)$ (a is the single input symbol). To check if $L(A)\neq\phi$ we first guess $k\leqq n^{n+1}$ and then check if $a^k\in L(A)$. Note that we cannot simulate A in polynomial time since k can be exponential in n. Let a configuration of A be a pair (s,i) where $s\in S$ is a state and $0\leqq i\leqq k+1$ is the head position. If $a^k\in L(A)$, then there is a sequence of moves which accepts it without repeating a configuration. Without loss of generality A accepts when its head scans the left endmarker. Thus if $a^k\in L(A)$ there is a non-repeating sequence of configurations $c_0,c_1,c_2,\dots,c_u$ such that $c_0=(q_0,0)$, $c_u=(p,0)$, p accepting, and c_{l+1} follows from c_l by one move of A for $0\leqq l\leqq u-1$. Unfortunately u can be very large. We consider only the configurations in which A scans an endmarker ($i=0$ or $i=k+1$) $d_0,d_1,\dots,d_v$ $v\leqq 2|S|$ (the sequence is without repetition). Obviously $d_0=c_0$, $d_v=c_u$ and d_{l+1} can follow d_l after a sequence of moves in which A does not hit any endmarker. Thus we guess the sequence $\{d_l\}_{l=0}^{v}$ and check if it satisfies the above. We show how to check if $d_{l+1}=(q,k+1)$ follows from $d_l=(p,0)$ without hitting any endmarker. The other three cases $d_{l+1}=(q,0)$ $d_l=(p,k+1)$, $d_{l+1}=(q,0)$ $d_l=(p,0)$ and $d_{l+1}=(q,k+1)$ $d_l=(p,k+1)$ are similar.

We call a sequence of configurations $[m_1,m_2]$ *bounded* if each configuration in the sequence (p,i) satisfies $m_1\leqq i\leqq m_2$.

Let $p\in S$ and m be an integer. We define

$$R_{p,m}=\{q\,|\,(p,1)\overset{*}{\underset{A}{\vdash}}(q,m+1)\quad\text{in a } [1,m+1] \text{ bounded sequence}\},$$

$$L_{p,m}=\{q\,|\,(p,m+1)\overset{*}{\underset{A}{\vdash}}(q,1)\quad\text{in a } [1,m+1] \text{ bounded sequence}\},$$

$$LR_{p,m}=\{q\,|\,(p,m+1)\overset{*}{\underset{A}{\vdash}}(q,m+1)\quad\text{in a } [1,m+1] \text{ bounded sequence}\},$$

$$RL_{p,m}=\{q\,|\,(p,1)\overset{*}{\underset{A}{\vdash}}(q,1)\quad\text{in a } [1,m+1] \text{ bounded sequence}\}.$$

Now we can build R_{p,m_1+m_2} recursively by the following program:

```
$R \leftarrow R_{p,m_1}$;
do while $R$ does not increase;
    $R \leftarrow R \cup \{q \mid \exists q_1 \in R$ and $q \in RL_{q_1,m_2}\}$;
    $R \leftarrow R \cup \{q \mid \exists q_1 \in R$ and $q \in LR_{q_1,m_1}\}$
end;
$R_{p,m_1+m_2} \leftarrow \{q \mid \exists q_1 \in R$ and $q \in R_{q_1,m_2}\}$
```

Note that the final value of R is $\{q \mid (p, 1) \overset{*}{\underset{A}{\vdash}} (q, m_1+1)$ in a $[1, m_1+m_2+1]$ bounded sequence$\}$. Fig. 4 indicates why the program above is correct.

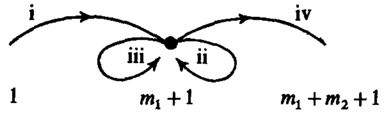


Fig. 4. Note that ii and iii can repeat at most $|S|$ times

L_{p,m_1+m_2}, LR_{p,m_1+m_2}, RL_{p,m_1+m_2} can be computed similarly. Thus we construct $R_{p,2^m}$, $L_{p,2^m}$, $LR_{p,2^m}$, $RL_{p,2^m}$ for all $0 \leqq m \leqq [\log_2(k-1)]$ and $p \in S$. Then we compute $R_{p,k-1}$ using the binary representation of $k-1$. Now $d_{l+1} = (q, k+1)$ can follow $d_l = (p, 0)$ without hitting the endmarkers iff there are $p_1, p_2 \in S$ such that A can move from $(p, 0)$ to $(p_1, 1)$ and from (p_2, k) to $(p, k+1)$ in one step and $p_2 \in R_{p_1,k-1}$. This completes the proof of the lemma.

It is interesting to note that although quite a few complete problems about automata were found they do not fit into an HCP. In particular, we could not construct any HCP from all the complete problems dealing with regular expressions ([9], [15]) or finite automata. The best we could do was to construct the following incomplete HCP:

Let C_5: Nonemptiness of 2nfa languages over binary alphabet.
C_4: Nonemptiness of 2nfa languages over unary alphabet.
C_3: ?
C_2: Nonemptiness of (one way) nfa languages over unary alphabet.
C_1: Nonemptiness of (one way) dfa languages over unary alphabet.

It was shown in [12] that C_1 and C_2 are complete in DLOG and NLOG. We can show that C_4 and C_5 are complete in NP-TIME and P-SPACE. (The most difficult part is the lemma proved above.) However we were unable to find some C_3 lying between C_2 and C_4 which is complete in P-TIME.

4. An HCP in Theorem Proving

Definition. A *proof system* consists of (1) A set of *literals* $X = \{x_1, x_2, \dots, x_n\}$ which contains three distinct subsets: AX—the set of *axioms*, TH—the set of *theorems* and AS—the set of *assumptions* $= \{x_{i_k}, x_{j_k} \mid i_k \neq j_k, k = 1, 2 \dots\}$. The assumptions are arranged in *conjugate pairs* $(x_{i_k}, x_{j_k})\, x_{j_k}$ is the conjugate of x_{i_k} and vice versa; and (2) A set of *rules* of the form $y_1, \ldots, y_r \to z$ such that $z \in X\text{-}AS$ and $y_i \in X$, $1 \leqq i \leqq r$. We will say that z *can be derived* from $y_1, \ldots, y_r$.

We will say that $x \in X$ *can be proven* if there is a subset of $X\text{-}AS$, $\{z_1, \ldots, z_p\}$ such that (1) $z_p = x$; (2) There is $k \leqq p$ with $\{z_1, \ldots, z_k\} \subseteq AX$; (3) z_{l+1} can be derived from $\{y_1, \ldots, y_m\} \subseteq AS \cup \{z_1, \ldots, z_l\}$ for $k \leqq l \leqq p-1$ and (4) Throughout this process, if an assumption x_{i_t} is used in the derivation of some z_l, then its conjugate x_{j_t} is not used in any derivation.

We can view AS as a source for additional axioms, i.e. for any proof we can add to AX any subset of AS which does not contain a pair of conjugate literals.

A proof system is *valid* if every $x \in TH$ can be proven.

Our HCP will consist of various restricted forms of valid proof systems.

We restrict all our proof systems to have $|TH| = 1$. This restriction is not necessary but it simplifies the proofs.

Define D_4: The set of all valid proof systems.
D_3: D_4 except that $AS = \phi$.
D_2: D_3 with rules restricted to have $r = 1$.
D_1: D_2 with the restriction that no two literals can be derived from the same literal.

In order to construct an HCP we define D_5 for which D_4 is a special case. We extend the definition of a proof system as follows: A *generalized proof system* is a proof system with the addition that $x \in X\text{-}AX$ has a superscript (i.e. $x^{(i)}$) and the rules should be interpreted as $y_1^{(i)}, \ldots, y_r^{(i)} \to z^{(i+1)}$. Thus

D_5: The set of all valid generalized proof systems.

Theorem 3. $(D_1, D_2, D_3, D_4, D_5)$ is an HCP.

Proof. D_1, D_2 and D_3 are complete in DLOG, NLOG and P-TIME respectively since one can easily reduce DGAP, GAP and PATH ([11], [12], [2]) to D_1, D_2 and D_3 respectively. D_4 is complete in NP-TIME since a computation of a nondeterministic Tm which operates in polynomial time can be encoded into an instance of D_4 (the encoding is much simpler than that of Section 2): The axioms play the role of the initial configuration, the theorem stands for the fact that the input is accepted, each assumption stands for one of the two choices for next move and the rules represent the transformation of an i.d. to the one which follows from it. Similarly D_5 is complete in P-SPACE (superscript i corresponds to time $= i$).

5. HCP's in Games[8]

Let G be a directed graph with set of edges E and set of vertices V (V represents a given population). For $v \in V$ let $N_v = \{w \mid (w, v) \in E\}$ be the set of *neighbours* of v (note that being a neighbour is not necessarily commutative). Let $k > 0$ be a given integer, and for each $v \in V$ and $t \geqq 0$ let $a_v^{(t)}$ be an integer between 0 and k. $a_v^{(t)} = i$ means that at time t the person v has i-th degree of sickness ($i = 0$ means healthy, $i = k$ means dead). The rules for the spreading of the disease are given as follows: for every assignment to any N_v at time t there are

8 The problems below are called games since they resemble the game of life ([5]).

probabilities $p_i \geqq 0$, $0 \leqq i \leqq k$, $\sum_{i=0}^{k} p_i = 1$ where p_i is the probability that $a_v^{(t+1)} = i$ (the probability does not depend on t).

Define E_5: Given G, $\{a_v^{(0)}\}_{v \in V}$ and the set of rules. Is there a positive probability that $v_0^{(t)} = k$ for some $t \geqq 0$, where v_0 is a distinguished vertex?

E_4: Same as E_5 except that $a_v^{(t+1)} \geqq a_v^{(t)}$ for all $v \in V$ and $t \geqq 0$.

E_3': Same as E_4 without probability, i.e. $a_v^{(t+1)}$ is uniquely determined by $\{a_w^{(t)} \mid w \in N_v\}$.

E_3: Same as E_3 with $k = 1$.

E_2: Same as E_3 with restricted rules: $a_v^{(t+1)} = 1$ if for some $w \in N_v$ $a_w^{(t)} = 1$.

E_1: Same as E_2 except $|N_v| = 1$ for all $v \in V$.

Theorem 4. $(E_1, E_2, E_3, E_4, E_5)$ is an HCP.

Proof. E_5 is complete in NP-SPACE and hence in P-SPACE since every lba computation can be encoded into an instance of E_5: The initial assignment $\{a_v^{(0)}\}_{v \in V}$ represents the initial i.d., the rule encode the move of the machine and the distinguished person—the accepting state. The probabilities encode the nondeterminism. Before moving to the next step a small local disease can simulate the check that exactly one move is chosen. If this does not happen, the disease stops spreading. Similarly E_4 can encode a nondeterministic Tm which operates in polynomial time. E_3' and E_3 can easily encode PATH ([2]). E_2 and E_1 can encode GAP and DGAP ([11]).

It is interesting to consider E_6 for which E_5 is a special case:

E_6: The same as E_5 except that the question is: Is the probability that $a_{v_0}^{(t)} = k$ for some t is greater than half.

E_6 is complete in probabilistic polynomial space ([7]), a class which contains P-SPACE. (It is not known whether the inclusion is proper.)

A similar hierarchy which does not use probability (i.e. $a_v^{(t+1)}$ is determined uniquely by $\{a_w^{(t)} \mid w \in N_v\}$) is the following:

F_5: Given a graph and rules and a subset U of the vertices (the source of the plague). Is there an initial assignment $\{a_v^{(0)}\}_{v \in V}$ such that $a_v^{(0)} = 0$ for $v \notin U$ such that for some t $a_{v_0}^{(t)} = k$.

F_4: Same as E_5 except that $a_v^{(k+1)} \geqq a_v^{(t)}$ for all $v \in V$ and $t \geqq 0$.

F_3: Same as F_4 but $|U| = 1$.

F_2: Same as F_3 except the rules are restricted as in E_2.

F_1: Same as F_2 except $|N_v| = 1$ for all $v \in V$.

Theorem 5. $(F_1, F_2, F_3, F_4, F_5)$ is an HCP.

Proof. The part of the proof concerning F_1, F_2 and F_3 is similar to that concerning E_1, E_2 and E_3. F_4 and F_5 are different. In the case of F_5 we have rules and U such that only one initial assignment which describes the first i.d. of the given lba M and its input x can result in an expansion of the disease, and then it does so simulating the lba. In the case of F_4 the initial assignment contains two parts. One is responsible for the initial i.d. as in F_5 and the other is responsible for the nondeterministic choices that the machine can make during the computation.

Acknowledgments. I am indebted to Professors J. Hopcroft and J. Hartmanis for their encouragement and advice. I would also like to thank J. Simon and M. Solomon for useful discussions and help with the presentation.

References

1. Cook, S. A.: The complexity of theorem proving procedures. Proceedings of 3rd Annual ACM Symposium of Theory of Computing, Sheiker Heights (Ohio) 1971, p. 151–158
2. Cook, S. A.: An observation on time-storage trade off. Proceedings of 5th Annual ACM Symposium on Theory of Computing, Austin (Tex.) 1973, p. 29–33
3. Even, S., Tarjan, R. E.: A combinational problem which is complete in polynomial space. Proceedings of 7th Annual ACM Symposium on Theory of Computing, Albuquerque (New Mexico) 1975, p. 66–71
4. Galil, Z.: On some direct encodings of nondeterministic Turing machines operating in polynomial time into P-complete problems. SIGACT News **6**, 19–24 January 1974
5. Gardner, M.: The fantastic combinations of John Conway's new solitaire game 'Life'. Scientific American **223**, 120–123 (1970)
6. Garey, M. R., Johnson, D. S., Stockmeyer, L. J.: Some simplified NP-complete problems. Proceedings of 6th ACM Symposium on Theory of Computing, Seattle (Wash.) 1974, p. 91–95
7. Gill, J. T.: Computational complexity of probabilistic Turing machines. Proceedings 6th Annual ACM Symposium on Theory of Computing, Seattle (Wash.) 1974, p. 91–95
8. Greibach, S. A.: Checking automata and one-way stack languages. J. Computer and System Sciences **3**, 196–217 (1969)
9. Hartmanis, J., Hunt, III., H. B.: The LBA problem and its importance in the theory of computing. Cornell University, Technical Report 73–171, 1973
10. Hopcroft, J. E., Ullman, J. D.: Formal languages and their relation to automata. Reading (Mass.): Addison-Wesley 1969
11. Jones, N. D.: Reducibility among combinatorial problems in logn space. Proceedings of 7th Annual Princeton Conference on Information Sciences and Systems, 1973, p. 547–551
12. Jones, N. D., Laaser, W. T.: Complete problems for deterministic polynomial time. Proceedings of 6th Annual ACM Symposium on Theory of Computing, Seattle (Wash.) 1974, p. 40–46
13. Karp, R. M.: Reducibilities among combinatorial problems. In: Miller, R., Thatcher, J. (eds): Complexity of computer computations. New York: Plenum Press 1972, p. 85–104
14. Ladner, R., Lynch, N., Selman, A.: Comparison of polynomial time reducibilities. Proceedings of 6th Annual ACM Symposium on Theory of Computing, Seattle (Wash.) 1974, p. 110–121
15. Meyer, A. R., Stockmeyer, L. J.: Word problems requiring exponential time. Proceedings of 5th Annual ACM Symposium on Theory of Computing, Austin (Tex.) 1973, p. 1–9
16. Rogers, J., Jr.: Theory of recursive functions and effective computability. New York: McGraw-Hill 1967
17. Savitch, W. J.: Relationship between nondeterministic and deterministic tape complexities. J. Computer and System Sciences **4**, 177–192 (1970)

Zvi Galil
Computer Sciences Department
IBM T. J. Watson Research Center
Yorktown Heights, New York 10598
U.S.A.