Theoretical Computer Science 8 (1979) 227–237.

ARITHMETICAL HIERARCHY AND COMPLEXITY OF COMPUTATION

Petr HÁJEK
Mathematical Institute, ČSAV, 115 67 Prague, Czechoslovakia

Communicated by Jiří Bečvář
Received October 1977
Revised April 1978

Abstract. Various sets of Turing machines naturally occurring in the theory of computational complexity are shown to be complete on the respective levels of the arithmetical hierarchy. Results saying that various assertions concerning computational complexity (e.g. some relativizations of the P = NP problem) are independent of formal systems like set theory are obtained as corollaries. Provable complexity classes are also investigated.

0. Introduction

Investigation of computational complexity has led to various important properties of Turing machines, not previously studied in Recursion Theory. In this paper we are going to investigate *definitional complexity* of such properties. We shall determine the position of some natural properties (sets) of Turing machines in the Kleene's arithmetical hierarchy and show that they are complete on the respective levels.

What is the meaning of such results? First, they elucidate the nature of the investigated properties; in particular, we know the minimal number of quantifiers necessary for the definition of our property. (Compare this with the fact that, in the language of arithmetic, limit of a sequence cannot be defined by less than three quantifiers [9].) Second, we exhibit new complete sets naturally occuring in the theory of computational complexity. This gives new evidence to the following observation of Rogers [10, p. 330]: "Almost all arithmetical sets with intuitively simple definitions that have been studied by the above methods have proved to be Σ_n^0-complete or Π_n^0-complete (for some n). The reason for this (i.e., the relation between intuitive simplicity of definitions and completeness or noncompleteness) is not fully understood." Third, our results have corollaries concerning existence of Turing machines with specific properties. For example, there is a Turing machine M working in time $(n+1)$ such that the statement $\ulcorner M$ works in time $(n+1)\urcorner$ is independent of set theory. What is the reason for this? A proof-theoretical answer is:

This is because self-referential statements are available in computer science. But we can offer a recursion-theoretical answer: This is because the set $\{M : M$ works in time $(n+1)\}$ is not recursively enumerable (in fact, it is Π_1^0-complete) whereas $\{M : M$ works provably in time $(n+1)\}$ is a recursively enumerable, and hence *proper* subset of the former set. Similarly, we show that the set $\{M : M$ is total and $\mathrm{P}^{L_M} \neq \mathrm{NP}^{L_M}\}$ (where L_M is the language accepted by M) is Π_2^0-complete. Analyzing this result we obtain corollaries on the existence of various machines M such the P = NP question relativized to the oracle L_M is independent of set theory. (For some historical remarks see Section 2.)

The paper is organized as follows: In the rest of this section we survey the used framework. In Section 1, we exhibit four complete sets. Besides the completeness results mentioned above we prove that the set $\{M : M$ works in polynomial time$\}$ is Σ_2^0-complete and that the set $\{M : L_M \in \mathrm{P}\}$ is Σ_3^0-complete. (P is the class of languages accepted by deterministic Turing machines working in polynomial time). In Section 2 we relate these results to the notion of provability. First we obtain some rather immediate corollaries of the form described above. Then we investigate Hartmanis's provable complexity classes and obtain various further complete sets.

We shall use the usual notion of multi-tape deterministic and non-deterministic Turing machines as described in [1]. Some states are distinguished as accepting states; an input word x is *accepted* if there is a halting computation with the input x leading to an accepting state. (In the case of deterministic machines, each input determines uniquely the corresponding computation.) A machine *works in time* $f(n)$ if for every accepted input of length n there is an accepting computation consisting of at most $f(n)$ steps. P is the class of languages L such that there is a deterministic Turing machine which works in polynomial time and accepts L. Similarly for NP and non-deterministic machines. $M_0, M_1, M_2, \ldots$ is a fixed natural indexing of *deterministic* Turing machines. For each i, L_i is the language accepted by M_i; W_i is the set of all inputs for that M_i halts.

We shall also use the notion of a multi-tape deterministic and non-deterministic *query machine*, introduced by Cook [3]. Such a machine has a distinguished query tape and three distinguished states: the *query* state, the *yes* state and the *no* state. An *oracle* is a set X of words; if the machine enters the query state then the next state is *yes* if the word written on the query tape is in X, otherwise it is *no*. A query machine M with an oracle X works in time $f(n)$ if for every accepted input of length n there is an accepting computation consisting of at most $f(n)$ steps. P^X and NP^X have the obvious meaning.

Following [2] we fix a recursive list $P_0, P_1, P_2, \ldots$ of deterministic query machines with the input alphabet $\{0, 1\}$ such that putting $p_i(n) = n^i + i$ we have the following:

(i) P_i halts in $\leqslant p_i(n)$ steps for each input of length n and for each oracle and

(ii) for each oracle X and each $L \in \mathrm{P}^X$, there is an i such that P_i with the oracle X accepts L. Such a list is easily obtained by the "clock device". Similarly for $NP_0, NP_1, NP_2, \ldots$, non-deterministic machines and NP^X.

Our results could be easily reformulated for single-tape Turing machines and query machines. (In this case, a distinguished subalphabet replaces the query tape.)

Let us recall some facts and notions concerning the arithmetical hierarchy. $\Sigma_0^0 = \Pi_0^0 =$ the class of all recursive sets of natural numbers. A set A of natural numbers is in Σ_n^0 (or is a Σ_n^0-set) if there is an $(n+1)$-place recursive relation R such that

$$A(i) \quad \text{iff} \quad (\forall k_1)(\exists k_2)\cdots R(i, k_1, k_2, \ldots)$$

for each natural number i. (Three dots stand for a block of alternating quantifiers; it ends by $\forall$ if n is even and by $\exists$ if n is odd.) Similarly, A is a Π_n^0-set if there is such an R such that

$$A(i) \quad \text{iff} \quad (\forall k_1)(\exists k_2)\cdots R(i, k_1, k_2, \ldots).$$

A set is *arithmetical* if it is a Σ_n^0-set or a Π_n^0-set for some n. Basic properties of arithmetical sets can be found in [10]. A set A is *Σ_n^0-complete* if it is in Σ_n^0 and each Σ_n^0-set B is many-one reducible to A, i.e. there is a recursive function f such that $i \in B$ iff $f(i) \in A$. If A is Σ_n^0-complete $(n > 0)$ then $A \notin \Sigma_i^0$ for each $i < n$ and $A \notin \Pi_i^0$ for each $i \leq n$. Similarly for Π_n^0-complete sets. Classical examples of complete sets:

(1) The set $K = \{i : 0^i \in W_i\}$ is a complete Σ_1^0-set.

(2) The set FIN $= \{i : W_i$ is finite$\}$ is Σ_2^0-complete.

(3) The set COFIN $= \{i : W_i$ is cofinite$\}$ is Σ_3^0-complete. (W_i is cofinite iff M_i halts for all but finitely many inputs.)

(4) A set is Π_n^0-complete iff its complement is Σ_n^0-complete.

Main results of this paper were communicated on the Logic Colloquium 1977 (August 1977, Wroclaw, Poland) and on the symposium *Mathematical Foundations of Computer Science* 1977 (September 1977, Tatranská Lomnica, Czechoslovakia). [6] served as a preliminary draft for the present paper. The author is indebted to Professor R. Solovay, M. P. Chytil and P. Pudlák for helpful discussions and comments concerning this work.

1. Some complete sets

Theorem 1. *The set $\{i : M_i$ works in time $(n+1)\}$ is Π_1^0-complete.*

Proof. Put $A = \{i : M_i$ works in time $(n+1)\}$. We first prove that A is a Π_1^0-set. Let Word (x, i, n) be the recursive predicate saying that x is the code (Gödel number) of a word in the alphabet of M_i of length n. Let Step (i, z) be a recursive function associating with each computation z of M_i its number of steps. (To be definite, a computation with the input of length n having k steps is first coded by a matrix of the format $k \times (2k + n + 1)$ where each row is an instantaneous description; this matrix is then Gödel numbered in the usual way.) Furthermore, let Halt (i, x, z) be the

recursive predicate saying that z codes a halting computation on M_i with the input x. Then $i \in A$ iff

$$(\forall n, x, z)\,(\text{Word}\,(x, i, n)\ \&\ \text{Halt}\,(i, x, z) \rightarrow \text{Step}\,(i, z) \leqslant n+1).$$

Now we prove that A is complete. (The proof is inspired by the proof of the Theorem in [8, p. 21]) For each i, let $\sigma(i)$ be the index of a Turing machine $M_{\sigma(i)}$ working as follows: Given an input of length n, the head of the input tape reads the input and at the same time, $n+1$ steps of the following computation are made: write 0^i on the first working tape and simulate the behaviour of M_i with the input 0^i. If M_i has not halted within $(n+1)$ steps than $M_{\sigma(i)}$ halts in exactly $(n+1)$ steps; otherwise $M_{\sigma(i)}$ halts in 2^n steps. Evidently, $M_{\sigma(i)}$ works in time $(n+1)$ iff $0^i \notin W_i$(i.e., M_i with the input 0^i does not halt). Thus the recursive function σ reduces the complete Σ_1^0-set K to the complement of A and hence A is Π_1^0-complete.

Remark. (1) Let f be a recursive function such that $n+1 \leqslant f(n) < 2^n$ for all but finitely many n. Then a simple modification of the previous construction shows that the set $\{i : M_i$ works in time $f\}$ is Π_1^0-complete.

(2) If we worked with single-tape machines, $(n+1)$ would have to be replaced by $(n+1)^2$.

Theorem 2. *The set $\{i : M_i$ works in polynomial time$\}$ is Σ_2^0-complete.*

Proof. Put POL $= \{i : M_i$ works in polynomial time$\}$. Clearly, POL is in Σ_2^0. Indeed, let 'Word', 'Halt' and 'Step' have the same meaning as in the Proof of Theorem 1. Then $i \in$ POL iff

$$(\exists k)(\forall n, x, z)(\text{Word}\,(x, i, n)\ \&\ \text{Halt}\,(i, x, z) \rightarrow \text{Step}\,(i, z) \leqslant n^k + k$$

We prove that POL is complete. For each i, let $M_{\rho(i)}$ be a Turing machine working as follows: if the input word has the form $x\,\#^{k+|x|}$ (where x is a word in the alphabet of M_i and $\#$ is a symbol not in this alphabet) then $M_{\rho(i)}$ prints x on the first working tape and then simulates M_i with the input x for k steps. If M_i with the input x halts in exactly k steps then $M_{\rho(i)}$ halts in 2^n steps where $n = 2|x| + k$; in any other case $M_{\rho(i)}$ halts in $(n+1)$ steps where n is the length of the input word.

If W_i is finite then $M_{\rho(i)}$ works in polynomial time (moreover, in linear time!). If W_i is infinite then clearly $M_{\rho(i)}$ does not work in polynomial time. Thus ρ reduces FIN to POL and consequently POL is Σ_2^0-complete.

Remark. The set $\{i : M_i$ works in linear time$\}$ is obviously also Σ_2^0; the above proof shows that this set is Σ_2^0-complete. Similarly for quadratic time etc.

Theorem 3. *The set $\{i : M_i$ is total and $\mathrm{P}^{L_i} \neq \mathrm{NP}^{L_i}\}$ is Π_2^0-complete.*

Proof. For each i, let K_i be the Karp-complete language for NP^{L_i} described in [2], i.e., $K_i = \{\langle j, x, 0^n\rangle$: some computation of NP_j with the oracle L_i accepts x in fewer than n steps$\}$. By [2] we know that

$$P^{L_i} = NP^{L_i} \text{ iff } K_i \in P^{L_i}.$$

Now, $K_i \in P^{L_i}$ iff there is an e such that for each w, $[w \in K_i$ iff P_e with the oracle L_i accepts w (in short, P_e^i accepts $w)]$. If M_i is total then L_i is recursive and the relation $\{\langle e, w\rangle : P_e^i$ accepts $w\}$ is recursive. Moreover, there is a recursive relation $R(i, e, w)$ such that *if* M_i is total then $[R(i, e, w)$ holds iff P_e^i accepts $w]$.[1] Thus putting

$$NEQ = \{i : M_i \text{ total and } P^{L_i} \neq NP^{L_i}\}$$

we have $i \in NEQ$ iff M_i is total and $(\forall e)(\exists w)(w \in K_i \Leftrightarrow R(i, e, w))$. The condition "$M_i$ is total" is Π_2^0 and consequently NEQ is also Π_2^0.

We prove that NEQ is complete. Let A be a recursive set such that $P^A = NP^A$ (see [2]). We modify the construction of a set B such that $P^B \neq NP^B$ from [2] as follows: With each i we associate recursively an index $\tau(i)$ such that $M_{\tau(i)}$ is total and $[P^{L_{\tau(i)}} \neq NP^{L_{\tau(i)}}$ iff W_i is infinite.]. This will reduce the complement of FIN to NEQ.

Put $n_0 = 0$; given n_s, we define n_{s+1}. Let $\bar{n} = \min_n(n > n_s$ and $p_s(n) < 2^n)$ and put $n_{s+1} = 2^{\bar{n}}$. Given i, we construct a recursive set $B_i = L_{\tau(i)}$ in countably many steps. Put $B_i(0) = \emptyset$; given $B_i(s)$ we construct $B_i(s+1)$.

Case 1. There is no z such that $n_s \leq z < n_{s+1}$ and z codes a halting computation of M_i. In this case put $A(s) = \{w \in A : n_s \leq |w| < n_{s+1}\}$ and $B_i(s+1) = B_i(s) \cup A(s)$.

Case 2. There is a z such that $n_s \leq |z| < n_{s+1}$ and z codes a halting computation of M_i. Suppose that s is the eth step for which Case 2 occurs. We investigate the behaviour of P_e with the oracle $B_i(s)$ for the input $0^{\bar{n}}$. If it accepts $0^{\bar{n}}$ (in at most $p_e(\bar{n})$ steps) then we set $B_i(s+1) = B_i(s)$ (nothing added); otherwise we set $B_i(s+1) = B_i(s) \cup \{w\}$ where w is a word of length $\bar{n}$ not used by the oracle during the computation of P_e with the oracle $B_i(s)$ and the input $0^{\bar{n}}$. Since $p_e(\bar{n}) < 2^{\bar{n}}$, w exists.

The set $B_i = \bigcup_s B_i(s)$ is recursive and an index $\tau(i)$ of a total machine $M_{\tau(i)}$ such that $B_i = L_{\tau(i)}$ is primitively recursively computable from i. Observe that if s and e are as above then the computation of P_e with the input $0^{\bar{n}}$ is the same whether we use the oracle B_i or its finite segment $B_i(s)$ (since P_e cannot query the oracle on words of length $> p_e(\bar{n})$ during this computation). If W_i is finite then B_i differs at most finitely from A and consequently $P^{B_i} = NP^{B_i}$. If W_i is infinite then each machine P_e is eventually used during the computation of B_i. Put

$$L = \{w : \text{there is an } x \in B_i \text{ such that } |x| = |w|\}.$$

Clearly, $L \in NP^{B_i}$; following [2] we prove $L \notin P^{B_i}$. Consider an arbitrary P_e; assume that in step s of the construction of B_i, P_e was investigated. Then $0^{\bar{n}} \in L$ iff P_e with the

[1] If M_i is not total then L_i may not be recursive; if L_i is not recursive then $\{\langle e, w\rangle : P_e^i$ accepts $w\}$ is *not* recursive (for a trivial machine P_e, P_e^i accepts w iff $w \in L_i$).

oracle $B_i(s)$ does not accept $0^{\bar{n}}$ iff P_e with the oracle B_i does not accept $0^{\bar{n}}$. Thus P_e with the oracle B_i does not accept L. This concludes the proof.

Theorem 4. *The set* $\{i : L_i \in \mathrm{P}\}$ *is a complete* Σ_3^0*-set.*

Proof. $L_i \in \mathrm{P}$ iff there is a machine M_e which works in polynomial time and such that $L_i = L_e$. Thus

$$L_i \in \mathrm{P} \quad \text{iff} \quad (\exists e)(e \in \mathrm{POL} \text{ and } L_i = L_e).$$

By Theorem 2, POL is in Σ_2^0; and the relation $\{\langle i, e\rangle : L_i = L_e\}$ is obviously a Π_2^0-relation. Thus $\{\langle i, e\rangle : e \in \mathrm{POL} \text{ and } L_i = L_e\}$ is a Σ_3^0-relation and hence $\{i : L_i \in \mathrm{P}\}$ is a Σ_3^0-set. We prove that it is complete.

In this proof, each natural number n is identified with the word 0^n. For each i, we define a set $C_i = L_{\pi(i)}$ such that W_i is co-finite iff $C_i \in \mathrm{P}$. C_i is constructed in countably many steps. In step n, *first* generate $\langle n, 0\rangle, \langle n, 1\rangle, \ldots, \langle n, n\rangle$ and for $j = 0, \ldots, n$ put $\langle n, j\rangle$ into $C_i(s)$ iff P_i (with the empty oracle) does not accept $\langle n, j\rangle$. *Second*, for each $j = 0, \ldots, n$, if M_i with the jth word as input halts in $\leqslant n$ steps, put $\langle j, 0\rangle, \langle j, 1\rangle, \ldots, \langle j, j\rangle$ into $C_i(s)$. (We use the natural ordering of words: shorter words precede longer ones, words of the same length are ordered lexicographically.) Put

$$C_i = \bigcup_s C_i(s) \quad \text{and} \quad \mathrm{Diag} = \{\langle n, i\rangle : n \geqslant i\}.$$

Observe that we have always $C_i \subseteq \mathrm{Diag}$.

Case 1. W_i is co-finite. Then C_i differs from Diag only by finitely many elements, thus $C_i \in \mathrm{P}$.

Case 2. W_i is co-infinite. Then $C_i \notin \mathrm{P}$: given e, we prove that P_e (with the empty oracle) does not accept C_i. Let $n > e$ be such that the nth word is not in W_i. Then $\langle n, i\rangle \in C_i$ iff $\langle n, i\rangle$ is not accepted by P_e.

An index $\pi(i)$ such that $C_i = \mathrm{L}_{\pi(i)}$ can be obtained primitively recursively from i. Thus π reduces COFIN to $\{i : L_i \in \mathrm{P}\}$ and consequently the last set is Σ_3^0-complete.

Remark. The same proof shows that the set of all i such that L_i is recognizable by a deterministic Turing machine working in linear time is Σ_3^0-complete. See also the next section.

2. Provability and unprovability

If a Turing machine M has a property $\boldsymbol{P}$ we can ask whether the statement $\ulcorner M$ has the property $\boldsymbol{P}\urcorner$ is provable in a given formal theory like Peano arithmetic or Zermelo–Fraenkel set theory (provided this property is formally expressible in the theory). If it is the case than we shall say that M has *provably* the property $\boldsymbol{P}$.

Hartmanis and Hopcroft [8] exhibited some simple properties $\boldsymbol{P}$ such that there is a Turing machine having $\boldsymbol{P}$ but not provably. In particular, they constructed a (single-tape) machine M working in quadratic time such that no time bound lower than 2^n can be formally proved for M in set theory. Furthermore, they used the construction of recursive oracles A, B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$ (Baker, Gill and Solovay [2]) to construct a machine M_i such that the formula $\ulcorner \mathrm{P}^{L_i} = \mathrm{NP}^{L_i} \urcorner$ is independent of set theory (neither provable nor refutable). The result of [2] shows that the P = NP problem cannot be solved by methods that easily relativize; and the mentioned result of [8] indicates that the formula $\ulcorner \mathrm{P} = \mathrm{NP} \urcorner$ might be independent of set theory. (Note that various results similar to those of [2] were obtained by Dekhtyar [4].) We shall relate results of this form to our investigation.

In the sequel, F is a fixed formal theory with a recursive set of axioms satisfying the following:

(i) F contains the Peano arithmetic PA (i.e., each formula of PA is naturally identified with a formula of F and whenever a formula is provable in PA then it is provable in F) and

(ii) F is sound for the language of PA, i.e., whenever a formula of PA is provable in F then it is true.

Formulas of PA are built from variables, the constant $\bar{0}$, the equality predicate and function symbols S (successor), + and · using logical connectives and quantifiers. Δ_0 is the class of bounded formulas, i.e., formulas in which quantifiers occur only in contexts of the form $(\exists x)(x < t \,\&\, \cdots)$, $(\forall x)(x < t \to \cdots)$ where t is a term built from variables, the constants $\bar{0}$ and function symbols $S, +$ and $\cdot$. Σ is the least class of formulas containing all bounded formulas and closed under conjunction, disjunction, bounded universal quantification and arbitrary existential quantification. For each natural number n, we have the numeral $\bar{n}$ defined inductively as follows: $\bar{0}$ is the constant as above, $\overline{(n+1)}$ is $S(\bar{n})$. It can be shown that if φ is a Σ-formula and if $\varphi(\overline{n_1}, \ldots, \overline{n_k})$ is true then the formula $\varphi(\overline{n_1}, \ldots, \overline{n_k})$ is provable in F (see [5])[2]. We first present some corollaries of results of Section 1.

Corollary 1. *There is a Turing machine M_i such that M_i works in time $(n+1)$ but the statement $\ulcorner M_i$ works in time $(n+1)\urcorner$ is not provable.*

Proof. Let A be as in Theorem 1 and let $A' = \{i : M_i \text{ works provably in time } (n+1)\}$. Then A' is Σ_1^0 (recursively enumerable) and $A' \subseteq A$; thus A' is a proper subset of A. Take an $i \in A - A'$.

[2] Note in passing a mistaken remark in [8, p. 16]: The authors claim that there exist diophantine equations for which there exist (integer) solutions but that their existence is independent of the axioms of set theory. This is false: if a diophantine equation has a solution then this fact is provable (since it can be expressed by a Σ-formula). But there are diophantine equations having *no* solutions but such that this fact is independent of the axioms of set theory.

Remark. Observe that for each i the following is provable: ⌜For each x of length n, $M_{\sigma(i)}$ with input x halts either after $\leqslant (n+1)$ steps or after $\geqslant 2^n$ steps.⌝ The range of σ is recursive. Put $A_0 = A \cap \text{range}(\sigma)$ and $A_0' = A' \cap \text{range}(\sigma)$. It follows that A_0 is Π_1^0-complete and A_0' is a proper Σ_1^0-subset of A_0. If $i \in A_0 - A_0'$ then M_i works in time $(n+1)$ but no bound less than 2^n can be formally proved.

Lemma. (1) *The set* $\{i : W_i \text{ is infinite}\}$ *is many-one reducible to the set* $A_1 = \{i : M_i$ *is provably total and* $\mathrm{P}^{L_i} \neq \mathrm{NP}^{L_i}$. *Thus* A_1 *is* Π_2^0*-complete.*

(2) *The set* $\mathrm{FIN} = \{i : W_i \text{ is finite}\}$ *is many-one reducible to the set* $A_2 = \{i : M_i$ *is provably total and* $\mathrm{P}^{L_i} = \mathrm{NP}^{L_i}$. *Thus* A_2 *is* Σ_2^0*-complete.*

Proof. By inspection of the proof of Theorem 3; our construction of the set B_i is such that the machine $M_{\tau(i)}$ accepting B_i is provably total. Observe that the condition "M_i is provably total" is Σ_1^0; this is used in showing that A_2 is a Σ_2^0-set.

Corollary 2. (1) *There is a total Turing machine* M_i *such that* $\mathrm{P}^{L_i} \neq \mathrm{NP}^{L_i}$ *and the assertion* $\ulcorner \mathrm{P}^{L_i} = \mathrm{NP}^{L_i} \urcorner$ *is independent (neither provable nor refutable).*

(2) *There is a total Turing machine* M_i *such that* $\mathrm{P}^{L_i} = \mathrm{NP}^{L_i}$ *and the assertion* $\ulcorner \mathrm{P}^{L_i} = \mathrm{NP}^{L_i} \urcorner$ *is independent.*

Remark. The preceding result differs from [8] where one has an i such that $\ulcorner \mathrm{P}^{L_i} = \mathrm{NP}^{L_i} \urcorner$ is independent but the truth-value of the last formula is unknown and equal to the truth-value of the formula $\ulcorner \mathrm{P} = \mathrm{NP} \urcorner$.[3]

Corollary 3. *There is a Turing machine* M_i *such that* $L_i \in \mathrm{P}$ *(i.e.,* L_i *is accepted by a Turing machine working in polynomial time) but the formula* $\ulcorner L_i \in \mathrm{P} \urcorner$ *is independent.*

Proof. Immediate from Theorem 4. Note that M_i can be chosen provably total.

In the rest of the paper, we investigate provable complexity classes introduced by Hartmanis. For each recursive function f (specified, e.g. by a particular Turing machine), TIME(f) is the class $\{L_i : M_i$ works in time $f\}$ and FTIME(f) is the class $\{L_i : M_i$ works provably in time $f\}$. We make a slight generalization and replace f by a class C of functions: TIME(C) is the class of all L_i such that M_i works in time C (i.e., in time f for some $f \in C$). We shall assume the following on C:

(i) The set $\{i : m_i$ works in time $C\}$ is a Σ_2^0-set.

(ii) there is a recursive enumeration $A_0, A_1, \ldots,$ of all sets in TIME(C) and

(iii) there is an $f \in C$ such that $f(n) \geqslant (n+1)$ for all n.

[3] This does *not* imply that the formula $\ulcorner \mathrm{P} = \mathrm{NP} \urcorner$ is independent, since for this i the sentence $\ulcorner L_i = \emptyset \urcorner$ is true but *unprovable*. Consequently, if we had a proof of $\mathrm{P} \neq \mathrm{NP}$ formalizable in the formal theory F in question, we would *know* that $\mathrm{P}^{L_i} \neq \mathrm{NP}^{L_i}$ but we still could not *prove* this in F.

Examples: (i) the class of all polynomials, (ii) one-element class $C = \{f\}$ where $f(n) = n+1$; $f(n) = n^2+1$; $f(n) = 2^n$ etc. We fix a particular Σ_2^0 definition of the set $\{i : M_i \text{ works in time } C\}$. Then, for each i, we may express the condition "M_i works in time C" formally in F and ask whether the sentence $\ulcorner M_i$ works in time $C\urcorner$ is provable. We define FTIME(C) to be the class $\{L_i : M_i \text{ works provably in time } C\}$. (Of course, F refers to the formal theory F fixed at the beginning of this section.)

TIME(C) and FTIME(C) are families of languages; obviously, FTIME(C) $\subseteq$ TIME(C) thanks to our assumption of soundness. Given a language L, what does it mean that L is in TIME(C) or in FTIME(C)? To answer this, we must be explicit about *how* a language is given. Let us agree that we think of languages as given by Turing machines. Such a Turing machine M_i is thought as a *description* of a language and need not be optimalized with respect to computational complexity. Asking whether the language L_i accepted by M_i is in TIME(C) we ask whether there is a Turing machine M_e working in time C (say "a fast machine" for a moment) accepting L_i, i.e. such that $L_i = L_e$. M_e is thought not as a description but as a fast computing device accepting L_i. Similarly for FTIME(C): we ask whether there is a *provably fast* machine M_e that accepts our language. If M_e is such a machine then $L_i = L_e$ and the sentence saying that M_e works in time C is not only true but provable. But observe that in both cases the equality $L_i = L_e$ is assumed only to be true. Thus we may investigate at least the following four index sets:

$I_1 = \{i : (\exists e)(L_i \text{ provably equals to } L_e \text{ and } M_e \text{ works provably in time } C)\}$,

$I_2 = \{i : (\exists e)(L_i \text{ provably equals to } L_e \text{ and } M_e \text{ works in time } C)\}$,

$I_3 = \{i : (\exists e)(L_i = L_e \text{ and } M_e \text{ works provably in time } C)\}$,

$I_4 = \{i : (\exists e)(L_i = L_e \text{ and } M_e \text{ works in time } C)\}$.

Using once more the term "fast" we may say that

$i \in I_1$ iff L_i is provably accepted by a provably fast machine.

$i \in I_2$ iff L_i is provably accepted by a fast machine,

$i \in I_3$ iff L_i is accepted by a provably fast machine and

$i \in I_4$ iff L_i is accepted by a fast machine.

It is immediate that

$$\text{TIME}(C) = \{L_i : i \in I_4\} = \{L_i : i \in I_2\}$$

and

$$\text{FTIME}(C) = \{L_i : i \in I_3\} = \{L_i : i \in I_1\}.$$

(If $i \in I_3$ then, for some e, $L_i = L_e$ and M_e provably works in time C, hence $e \in I_1$ and $L_i = L_e$; thus $L_i \in \{L_j; j \in I_1\}$.)

Hartmanis gives an example of a recursive function f such that TIME(f) $\neq$ FTIME(f), which gives $I_3 \neq I_4$; but observe that e.g. if C is the class of all polynomials than we can easily prove TIME(C) = FTIME(C) and $I_3 = I_4$ using the "clock technique". We shall show that I_1, I_2, I_3 are pairwise distinct for *each* C by investigating the arithmetical complexity of $I_1, \ldots, I_4$. Note the obvious inclusions

$I_1 \subseteq I_2 \subseteq I_4$ and $I_1 \subseteq I_3 \subseteq I_4$. Using the techniques of Section 1, it is easy to show that I_1 is a Σ_1^0-set, I_2 is a Σ_2^0-set and both I_3 and I_4 are Σ_3^0-sets. We shall prove that all four sets are complete; then it follows immediately that all the above inclusions are proper.

Theorem 5. *I_3 and I_4 are Σ_3^0-complete.*

Proof. Observe that Theorem 4 asserts Σ_3^0-completeness of I_4 for C being the class of all polynomials. The same proof functions for each C satisfying our assumptions. To prove that I_3 is complete observe that if W_i is cofinite then the set C_i from the proof of Theorem 4 differs from Diag by only finitely many elements so that it is evidently accepted by a machine M_e which works provably in time $(n+1)$. Thus π reduces COFIN also to I_3.

Theorem 6. *The set I_2 is Σ_2^0-complete.*

Proof. For each i, let $M_{\vartheta(i)}$ be a machine working as follows: For an input x, $M_{\vartheta(i)}$ checks whether it has the form $y \#^k$ where y is a word in the alphabet $\{0, 1\}$ and $\#$ is a new symbol; at the same time, y is printed on the first working tape and k steps of the computation of M_i with the input y are simulated. If the input word is not of the desired form or else if the simulated computation does not halt in exactly k steps then the input is rejected. If x has the desired form, if the simulated computation halts in the desired number of steps and if x is the nth input for which this case occurs then $M_{\vartheta(i)}$ accepts x iff x is not accepted by the machine P_n.

If W_i is finite then $M_{\vartheta(i)}$ itself works in polynomial time; obviously, $\ulcorner L_{\vartheta(i)} = L_{\vartheta(i)} \urcorner$ by a trivial proof, and consequently $\vartheta(i) \in I_2$. If W_i is infinite then $L_{\vartheta(i)} \notin \mathrm{P}$; thus $\vartheta(i) \notin I_2$. Hence ϑ reduces FIN to I_2.

Theorem 7. *The set I_1 is Σ_1^0-complete.*

Proof. For each i, let $\xi(i)$ be a machine that for an input x of length n simulates n steps of the computation of M_i with the input 0^i; if M_i with this input has not halted in at most n steps then $M_{\xi(i)}$ operates on x exactly as a fixed machine M^* accepting a language L^* not in TIME(C). If M_i has halted in at most n steps then $M_{\xi(i)}$ rejects x.

Then $L_{\xi(i)} = L^*$ iff M_i with the input 0^i diverges. If M_i with the input x halts in k steps then the sentence $\ulcorner M_i$ with the input x halts in k steps$\urcorner$ is provable (being a Σ-formula) and so its consequence saying that each word accepted by $M_{\xi(i)}$ has length at most k is also provable. Thus in this case $L_{\xi(i)}$ is provably accepted by a machine provably working in time $(n+1)$. We have shown that ξ reduces K to I_1.

If $i \in I_1$ then there is a proof of the sentence saying that L_i is accepted by a machine working in time C. Can we conclude that there is a short (feasible) proof of this fact?

Fix an arbitrary recursive function, say 2^{2^n}, and define

$$I_0 = \{i : \text{there is a proof of length } \leq f(i) \text{ of } \ulcorner L_i \text{ is accepted by a machine working in time } C^{\urcorner}\}.$$

Obviously, I_0 is recursive, which implies $I_0 \neq I_1$.

We can summarize our results concerning $I_0, \ldots, I_4$ into the following diagram:

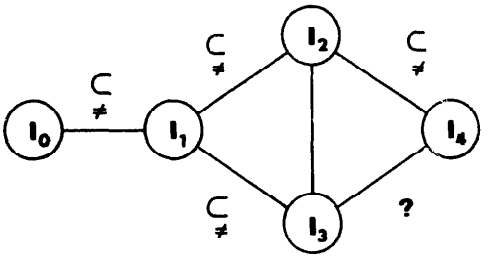

Diagram 1. ? means that $I_3 = I_4$ for some C and $I_3 \neq I_4$ for some other C.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] T. Baker, J. Gill and R. Solovay, Relativizations of the P = ? NP question, *SIAM J. Comput.* **4** (1975) 431–442.

[3] S.A. Cook, The complexity of theorem proving procedures, Proc. Third Annual ACM Symposium on Theory of Computing (1971) 151–158.

[4] M.I. Dekhtyar, On the relation of deterministic and nondeterministic complexity classes, in: A. Mazurkiewicz, ed., *Mathematical Foundations of Computer Science* 1976, Lecture Notes in Computer Science 45 (Springer-Verlag, Berlin, 1976) 255–259.

[5] S. Feferman, Arithmetization of metamathematics in a general setting, *Fund. Math.* **49** (1960) 35–92.

[6] P. Hájek, Arithmetical complexity of some problems in computer science, in: J. Gruska, ed., *Mathematical Foundations of Computer Science* 1977, Lecture Notes in Computer Science 53 (Springer-Verlag, Berlin, 1977) 282–287.

[7] J. Hartmanis, Relations between diagonalization, proof systems and complexity gaps (preliminary version), in: Proceedings 9th ACM Symp. on Theory of Computing (Boulder 1977) 223–227.

[8] J. Hartmanis and J.E. Hopcroft, Independence results in computer science, *SIGACT News* **8** (4) (1976) 13–24.

[9] A. Mostowski, Examples of sets definable by means of two and three quantifiers, *Fund. Math.* **42** (1955) 259–270.

[10] H. Rogers Jr., *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).