# INDEPENDENCE RESULTS IN COMPUTER SCIENCE

J. Hartmanis and J.E. Hopcroft

TR 76-296

Department of Computer Science
Cornell University
Ithaca, N.Y. 14853

# INDEPENDENCE RESULTS IN COMPUTER SCIENCE*

J. Hartmanis and J.E. Hopcroft

Department of Computer Science
Cornell University
Ithaca, N.Y. 14853

Abstract

In this note we show that instances of problems which appear naturally in computer science cannot be answered in formalized set theory. We show, for example, that some relativized versions of the famous P = NP problem cannot be answered in formalized set theory, that explicit algorithms can be given whose running time is independent of the axioms of set theory, and that one can exhibit a specific context-free grammar G for which it cannot be proven in set theory that $L(G) = \Sigma^*$ or $L(G) \neq \Sigma^*$.

## 1. Introduction

During the last few years research in theoretical computer science has identified several problems whose solution seems to be important for the further development of the field and on which a considerable amount of research effort has been

*This research has been supported in part by National Science Foundation Research Grant DCR 75-09433 and the Office of Naval Research under Grant N00014-76-C-0018.

expended [1,5]. In spite of all this effort these problems remain unsolved and, though we understand them much better now, no real progress seems to have been made toward their solution. Nevertheless, there remains real optimism that they will be solved and a deeply ingrained conviction that they can be solved. As a matter of fact, for the famous P = NP problem, bets have been placed whether P = NP or P $\neq$ NP with strong conviction that they will eventually be collected. Similarly, there is a strong conviction that with sufficient effort and cleverness the running time of any specific algorithm can be determined.

In this note we point out that many of these problems may not have a solution in formalized mathematical systems; more specifically, we prove that the solutions of many instances of these problems are independent of the axioms of set theory [3]. Indeed, we show that there exist relativized versions of the P = NP problem which cannot be answered in formalized set theory, that explicit algorithms can be given whose running time is independent of the axioms of set theory and show in general that many other instances of problems in computer science cannot be answered in formalized set theory.

More precisely, for a set A, A $\subseteq \Sigma^*$, let $P^A$ be the class of languages accepted in polynomial-time by deterministic Turing machines with the oracle set A and let $NP^A$ be the corresponding class of languages accepted in polynomial-time by nondeterministic Turing machines with the oracle set A.

We know [2] that there exist recursive sets B and C such that $P^B = NP^B$ and $P^C \neq NP^C$. Using this result we explicitly construct a recursive set A and show that it is independent of the axioms of set theory whether $P^A = NP^A$ or $P^A \neq NP^A$.

Unfortunately, this result shows only that there exist relativized instances of this problem which cannot be solved in the framework of formal set theory, it does not say anything directly about the classic P = NP problem. On the other hand, we can show that there are other specific algorithms whose running time is independent of the axioms of set theory. We explicitly construct an algorithm such that it is consistant with the axioms of set theory to assume that it runs in time $n^2$ or $2^n$. The algorithm can be seen (outside of formal set theory) to run in time $n^2$ but there is no proof in formalized set theory that this is the case, no bound lower than $2^n$ can be formally proven.

The same reasoning shows that many instances of questions about context-free languages and automata are independent of the axioms of set theory. For example, we construct a context-free grammar G such that there is no proof in set theory that $L(G) = \Sigma^*$ or $L(G) \neq \Sigma^*$.

Looking at problems in computer science, with these results in mind, we have to conclude that many different problems which appear naturally in computer science will have specific instances which cannot be answered by standard mathematical methods since they are independent of the axioms and formal

reasoning used in most of mathematics. It seems particularly surprising that this already happens in the analyses of running times of algorithms.

It should be pointed out that similar results are known in some parts of pure mathematics but they do not seem to be as prevalent as in computer science problems. This is caused by the difference in the problem areas: in computer science we deal with computations, with devices which perform these computations and with properties of computations. In all these cases we can embed "universal" computations and "self-referencing" in these problems and thus reach instances of problems which cannot be answered in formal mathematical systems. On the other hand, in many parts of mathematics either such embeddings and self-references do not exist or are extremely hard to find. For example, the solution of Hilbert's tenth problem [7] is recognized as a great achievement mainly because the embedding of universal computations in diophantine equations was exceedingly difficult. Once this embedding is known, one can easily show that there exist diophantine equations for which there exist (integer) solutions but that their existence is independent of the axioms of set theory.

It is to be expected that results of this type will be found in other parts of mathematics. In general though, we have to expect that this will not permiate mathematical research areas. The situation in computer science is quite different, as stated before, the central object of study in

this science is computation which unavoidably brings with it instances of problems which cannot be solved by traditional mathematical methods.

## 2. The P = NP Problem

We now turn to the problem of exhibiting a recursive set A such that $P^A = NP^A$ is independent of the axioms of set theory.

Let F be any formal mathematical system for proving theorems. We assume that F is axiomatizable (i.e. that the set of provable theorems is recursively enumerable), that F is consistant, that the provable theorems are intuitively true and that F is of sufficient power to prove the basic theorems of set theory. Let $\{\phi_1, \phi_2, \ldots\}$ be an acceptable enumeration of all one-tape Turing machines. Thus we know that the Smn and recursion theorems hold for this enumeration [8]. Furthermore, for the sake of brevity, we will write $P^{\phi_i}$ and $NP^{\phi_i}$, respectively, for $P^A$ and $NP^A$ provided $\phi_i$ accepts the set A. Let $\Downarrow$ denote convergence and $\Uparrow$ divergence of algorithms.

Theorem: For every F we can effectively construct an i such that $\phi_i$ is recursive and the relativized $P^{\phi_i} = NP^{\phi_i}$ is independent of F. That is $P^{\phi_i} = NP^{\phi_i}$ can neither be proved nor disproved in F.

Proof: Let B and C be recursive sets such that $P^B = NP^B$ and $P^C \neq NP^C$. From [2] we know that such sets exist and are effectively constructable. Define

$$\phi(x,j) = \begin{cases} \Downarrow \text{if there exists a proof among the first } x \text{ proofs in F that } P^{\phi_j} = NP^{\phi_j} \text{ and } x \in C \text{ or if there exists a proof among the first } x \text{ proofs in F that } P^{\phi_j} \neq NP^{\phi_j} \text{ and } x \in B \\ \Uparrow \text{otherwise} \end{cases}$$

Now by the Smn theorem there exists a recursive $\sigma$ such that

$$\phi_{\sigma(j)}(x) = \phi(x,j).$$

By the recursion theorem there exists $i_0$ such that

$$\phi_{i_0}(x) = \phi_{\sigma(i_0)}(x).$$

Thus

$$\phi_{i_0}(x) = \phi(x,i_0).$$

If there is a proof in F that $P^{\phi_{i_0}} = NP^{\phi_{i_0}}$, then the set accepted by $\phi_{i_0}$ differs at most finitely from C and hence we know $P^{\phi_{i_0}} \neq NP^{\phi_{i_0}}$. Similarly if there is a proof in F that $P^{\phi_{i_0}} \neq NP^{\phi_{i_0}}$, then the set accepted by $\phi_{i_0}$ differs at most finitely from B and hence $P^{\phi_{i_0}} = NP^{\phi_{i_0}}$. Since F is such that any theorem proved in F is intuitively true we conclude that there can be no proof of either $P^{\phi_{i_0}} = NP^{\phi_{i_0}}$ or $P^{\phi_{i_0}} \neq NP^{\phi_{i_0}}$ in F. Thus whether or not $P^{\phi_{i_0}} = NP^{\phi_{i_0}}$ is independent of the formal mathematical system F. ◘

Intuitively we know that $\phi_i$ in the above construction accepts the empty set and $P^{\phi_i} = NP^{\phi_i}$ is true if and only if P = NP in the classical version. Still it does not follow from

this proof that the unrelativized version is not provable in set theory. What we have just proven suggests the possibility that the unrelativized P = NP problem could be independent of the axioms of set theory. It may be possible to have two completely consistent theories of computation, one in which P = NP is an axiom and the other in which P $\neq$ NP is an axiom.

## 3. Analysis of Algorithms and Other Problems

Next we exhibit an algorithm which cannot be analyzed in the sense that its running time is independent of the axioms of F. Blum's speed up theorem [4] shows that there are functions with no best algorithm. What we are saying is something entirely different. There are algorithms with definite running times which are not provable in set theory. In particular we will exhibit an algorithm which runs in time $n^2$, but for which there is no proof in F of an upper bound less than $2^n$ nor is there a proof in F of a lower bound greater than $n^2$. Consequently formal mathematics is not powerful enough to analyze all algorithms.

To do this we first prove a lemma.

Lemma: Given the formal system F we can exhibit an i such that the halting of the $i^{\underline{th}}$ Turing machine when started on blank tape cannot be proved or disproved in F.

Proof: Let $\phi_i(-)$ denote the $i^{\underline{th}}$ Turing machine with blank input tape. Define

$$\phi(x,j) = \begin{cases} \Downarrow \text{if there exists a proof in F} \\ \quad \text{that } \phi_j(-)\Uparrow \\ \Uparrow \text{otherwise} \end{cases}$$

Again by the Smn and recursion theorems there exist $\sigma$ and $i_0$ such that

$$\phi(x,j) = \phi_{\sigma(j)}(x)$$

and

$$\phi_{i_0}(x) = \phi_{\sigma(i_0)}(x).$$

By a meta argument we conclude that $\phi_{i_0}(-)$ does not halt, since if $\phi_{i_0}(-)$ halts then there cannot be a proof in F that $\phi_{i_0}\Uparrow$, which is the only way $\phi_{i_0}(-)$ can halt. Thus there is no proof in F that $\phi_{i_0}(-)$ does not halt nor can there be a proof in F that it does halt. ■

Clearly we already knew of the existence of the i satisfying the last theorem. A proof in F for each i that $\phi_i$ did or did not halt would imply that $\{i|\phi_i(-)\Downarrow\}$ was recursive. What the lemma shows is that we can effectively exhibit a specific Turing machine which does not halt on blank tape but for which there is no proof of this fact in F.

We now show that some algorithms with well defined running times cannot be analyzed in F.

To reveal the simplicity of this proof we make an additional assumption about the formal system F, as described below. This assumption is not essential for the next result, but without it our proofs become considerable longer, since we have to

show that the application of the Smn and recursion theorems in this proof do not change the running times of algorithms drastically. This can be done, but for the sake of brevity, we are omitting this longer proof.

For any j let $\phi_{\rho(j)}$ be a Turing machine which for input n simulates $\phi_j(-)$ for n steps and if $\phi_j(-)$ has not halted in n steps $\phi_{\rho(j)}(n)$ halts in exactly $n^2$ steps; if $\phi_j(-)$ does halt in n steps then $\phi_{\rho(j)}(n)$ halts in exactly $2^n$ steps.

We assume that there is a construction $\rho$, as described above, such that we can prove in F for all j that:

$\phi_{\rho(j)}$ runs in time $< 2^n$ iff $\phi_j(-)$ does not halt.

Theorem: There exists an algorithm (which can be explicitly given) whose running time is $n^2$, but there is no proof in F that it runs in time $< 2^n$.

Proof: Let $i_0$ be an index such that $\phi_{i_0}(-)$ does not halt and for which there is no proof in F that $\phi_{i_0}(-)$ does not halt; our previous corollary guarantees that we can effectively obtain such $i_0$. Then, from our assumptions about F, it follows that there is no proof in F that $\phi_{\rho(i_0)}$ runs in time $< 2^n$ since this would prove in F that $\phi_{i_0}(-)$ does not halt. Thus $\phi_{\rho(i)}$ is an algorithm running in time $n^2$ for which there is no proof in F that it runs in time less than $2^n$. □

A similar proof yields the following result.

Corollary: There exists an algorithm $\phi_{j_0}$ which runs in time

$n^2$ but for which there is no proof in F that it is a total function, thus no running time bound can be proven for $\phi_{j_0}$ in F.

Next we show that simple problems about context-free languages are independent of the axioms of F, provided we can prove in F elementary facts about context-free languages.

Theorem: One can exhibit a context-free grammar G such that $L(G) = \Sigma^*$ but for which there is no proof in F that $L(G) = \Sigma^*$ or $L(G) \neq \Sigma^*$.

Proof: Using standard techniques [6] given i one can construct $G_i$ such that $L(G_i) = \Sigma^*$ iff $\phi_i(-)\uparrow$. Furthermore, if F is sufficiently rich, a proof in F that $L(G_i)$ does or does not equal $\Sigma^*$ can be extended to a proof in F that $\phi_i$ does or does not converge. But then using an $i_0$ for which there is no proof that $\phi_{i_0}(-)\uparrow$, we get a $G_{i_0} = \Sigma^*$ but there is no proof in F that $L(G_{i_0}) = \Sigma^*$ or $L(G_{i_0}) \neq \Sigma^*$. ∎

Corollary: One can exhibit a specific recursive function t such that the equality of the time complexity classes $C_t$ and $C_{t^2}$ is independent of F.

Proof: Similar to the previous proof. ∎

## 4. Conclusion

These results are presented not as something new or profound but only as a caution to computer scientists working in

complexity, lower bounds, analysis of algorithms, and related topics. One should recognize that due to the self-referencing ability of our formalisms we can reformulate Gödel's incompleteness theorem in computer science problems. Thus given set theory or any other formal theory there are many specific instances of problems with which we are concerned but which are independent of the theory. What this suggests is that our inability to settle questions like the P = NP problem or prove lower bounds may be a consequence of the power (or weakness) of formal systems such as set theory. Clearly an exciting result would be to discover a natural instance of such a problem.

References

1. Aho, A.V., J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms," Addison-Wesley Publishing Co., Reading, Mass., 1974.

2. Baker, T., J. Gill and R. Solovay, "Relativizations of the P = ?NP Question," SIAM J. on Comp. 4:4 (1975), pp. 431-442.

3. Bernays, P., and A.A. Fraenkel, "Axiomatic Set Theory," North Holland Publ., Amsterdam, 1958.

4. Blum, M., "A machine-independent theory of recursive functions," JACM 14:2 (1964), pp. 322-336.

5. Hartmanis, J. and J. Simon, "On the structure of feasible computations" in Advances in Computers Vol. 14 (Edits. Morris Rubinoff and Marshall C. Yovits), Academic Press, New York, N.Y., 1976. pp. 1-43.

6. Hopcroft, J. and J. Ullman, Formal languages and their relation to automata, Addison-Wesley, Reading, Mass., 1969.

7. Matijasevic, Y. "Enumerable sets are Diophantine" (Russian), Dokl. Acad. Nank. SSSR 191 (1970), pp. 279-282.

8. Rogers, Hartly, Jr., "Theory of Recursive Functions and Effective Computability," McGraw-Hill, New York, N.Y., 1976.