An Algorithmic Proof that NP $\not\subseteq$ P

J.E. Rogan [*]

Abstract

We give an algorithmic proof that NP $\not\subseteq$ P, based on two intermediate results: first, we give an example of a nondeterministic, polynomial-time Turing machine, denoted M_{np}, with the property that any deterministic Turing machine which recognizes the same language as M_{np} must employ an algorithm of Savitch type, that is, an algorithm which references an encoding of some nondeterministic Turing machine; second, we show that there can be no algorithm of Savitch type which recognizes the same language as M_{np} in polynomial time.

1. **Introduction**

The relative inclusion of P and NP has been of considerable interest in computer science ([Tra84], [Sip92],[Coo03],[All09], [For09],[Joh12], [Mul12]), logic ([Bus12]) and related areas of mathematics ([Sma98], [Wig06], [Lan10]). A broad perspective on the question, including the role of impossibility theorems, is found in [Aar16]. In particular, the relativization impossibility results of Baker, Gill and Solovay [BGS75] place restrictions on the techniques which may be used to separate P from NP, while the theorems of Razborov and Rudich [RR97] place restrictions on the size of function classes which may be used in separation proofs.

The first of two results in the present paper complements this tradition. Rather than place further barriers in the way of proving NP $\not\subseteq$ P, we construct a counterexample which removes some incorrect proofs of NP = P from consideration. Specifically, we give an algorithm for a nondeterministic polynomial-time Turing machine, denoted M_{np}, and we show that any deterministic Turing machine which recognizes the same language as M_{np}, denoted $L(M_{np})$, must reference an encoding of some nondeterministic Turing machine. Savitch [Sav70] considered deterministic Turing machines which explore the computational paths of a nondeterministic Turing machine and demonstrated that the languages recognized by such Turing machines are in PSPACE. In recognition

*Amateur complexity theorist. Email `EffectiveAlgorithms@gmail.com`

of the central role of this result in the present work, we refer to the general class of deterministic algorithms which reference the encoding of a nondeterministic algorithm as algorithms of Savitch type. We will establish that any proposed proof of $\mathsf{NP} = \mathsf{P}$ must imply the existence of a polynomial-time algorithm of Savitch type for the specific counterexample M_{np}.

This restriction allows us to obtain our second result and complete the proof of $\mathsf{NP} \not\subseteq \mathsf{P}$. We demonstrate that no algorithm of Savitch type exists that recognizes $L(M_{np})$ in polynomial time. To establish this result, we show that all encodings of any Turing machine implementing such an algorithm must have unbounded length. Since a Turing machine must have some finite-length encoding, we have shown that no such machine can exist. Thus $L(M_{np})$ is in NP, but not in P.

2. **Preliminaries: notation, definitions, related work and contributions**

In this section, we give references for our notation and definitions, none of which are original. We then go through each of the techniques and ideas used in the paper and discuss the sources for them. Our contribution has been to synthesize the work of others: only two of the techniques or ideas used in the paper have any original elements. These are discussed in this section in the context of related work. We close the section with an outline of the paper, including a summary of our results.

For concepts related to complexity classes, including time bounds, we adopt the notation and definitions found in [Sto87] and [Joh90].

For algorithms and Turing machine computations, we follow the notation of [AB09], with an extension: we extend the notation for an encoded n-tuple of strings to the case $n = 1$, that is, we denote an encoding of a single string x as $\langle x \rangle$.

We follow Hopcroft and Ullman [HU79] in defining a deterministic Turing machine by the property that its state transition relation is a function. We also follow their definition for a nondeterministic Turing machine, which reduces to the statement that multiple elements in the range of the state transition relation may correspond to a given element in the domain of the state transition relation. We choose a universal Turing machine, denoted U, with the property that verifying whether a given string may contain a U-encoding of a nondeterministic Turing machine can be accomplished by counting the range elements corresponding to each domain element of the state transition relation.

We use the letter Q to denote a variable ranging over formulas expressing some condition on strings. We distinguish between the results of evaluating formulas denoted by Q and the validity of proofs involving these formulas by using the notations $\top$ and $\bot$, denoting True and False, respectively, for the result of evaluating Q.

M_{np} and M_{nexp} denote specific non-deterministic Turing machines. M_{nd} denotes a Turing machine that is nondeterministic, but otherwise unspecified. M_d denotes a deterministic Turing machine, and M_n denotes a nondeterministic Turing machine; these Turing machines are specified once the condition Q is fixed. The notations M_π, M_σ denote deterministic Turing machines corresponding to algorithms of Savitch type, that is, algorithms which take an encoding of a nondeterministic Turing machine as input and search for an accepting path.

We use the notation $L(M)$ to denote the language recognized by the Turing machine M.

We next outline the work of others that we have synthesized to obtain our results and, where appropriate, we indicate those contributions thought to be original. The references cited here are not intended to be exhaustive or complete. We invite the reader to consult the works cited in the Introduction, Section 1 and in this Section, below, for authoritative bibliographies.

We investigate the role of nondeterminism in complexity classes, using the idea of nondeterministic algorithms articulated in [Flo67] and notions of complexity classes found in [Sto87] and [Joh90].

We consider algorithms of the type considered by Savitch [Sav70]. These are deterministic algorithms which, given a nondeterministic algorithm, explore the paths computed by a Turing machine implementing the nondeterministic algorithm. The deterministic algorithm accepts if it succeeds in finding a computational path on which the nondeterministic algorithm accepts. Such algorithms were not discovered by Savitch, and he cites earlier sources for them in [Sav70]. We call them algorithms of Savitch type in recognition of the importance to the present work of the deterministic storage bounds established in [Sav70]; these bounds imply that any nondeterministic polynomial-time algorithm must have a deterministic version that corresponds to a language in PSPACE.

This result leaves open two questions which we address in this paper. First, can nondeterministic polynomial-time algorithms have corresponding deterministic versions that are not of Savitch type? We answer this by giving an example of a nondeterministic polynomial-time algorithm whose only deterministic versions must be of Savitch type. Second, the question of time bounds is left open in Savitch's work. We know of algorithms of Savitch type that explore the computational paths of a nondeterministic polynomial-time algorithm in exponential time. However, we do not know whether an algorithm of Savitch type exists that halts in polynomial time on all inputs. Our results address this question as discussed below.

The state transitions of a nondeterministic algorithm may be encoded into the instructions for an algorithm of Savitch type, or given as input to the algorithm. We allow either possibility and distinguish which case is under consideration in the appropriate context.

We approach the $\mathsf{NP} \not\subseteq \mathsf{P}$ *problem by proving an impossibility theorem for* $\mathsf{NP} = \mathsf{P}$. This approach is one of two principal contributions of the paper. It is a slight modification of the approach taken by [BGS75]; instead of establishing barriers to proving $\mathsf{NP} \not\subseteq \mathsf{P}$, we establish a barrier to proving $\mathsf{NP} = \mathsf{P}$. To do this, we construct a counterexample, a language that is in NP that is not recognized by any deterministic Turing machine whose algorithm is not of Savitch type. This establishes the following barrier to proving $\mathsf{NP} = \mathsf{P}$: *any general method for recognizing languages in* NP *in deterministic polynomial time must be of Savitch type.*

We use the fact that any Turing machine has an encoding which has finite length. The relevance of this finite encoding condition to $\mathsf{NP} \not\subseteq \mathsf{P}$ derives from the theorem of Karp and Lipton (with Kannan) [KL82], that $\mathsf{P} = \mathsf{NP}$ if, and only if, $\mathsf{NP} \subseteq \mathsf{P}/log$, and their related result (with Sipser) concerning $\mathsf{P}/poly$. These results are the origin of the method used in the present paper, namely, to attack the relative containment of P and NP by investigating conditions that would force a deterministic polynomial time machine to access an infinite string (in our case one of its own encodings).

We use counterexamples to show that certain sets of natural numbers have the properties we require. Since our results concern a counterexample, our proofs do not relativize [BGS75], arithmetize or algebrize ([AW09],[IKK09]), and our proofs are not natural in the sense of Razborov and Rudich [RR97] (see also [Wil16]) .

We construct a Turing machine or algorithm with specific properties required to obtain the results we seek. This method is used in Turing's original paper [Tur36] and in Post's paper on Thue systems [Pos47]. A survey of algorithmic proof techniques is found in [HO02].

We base the construction of our counterexample, M_{np}, *on the* NP-complete *language* SHORTEST COMMON SUPERSTRING. In Section 4, we construct a nondeterministic Turing machine, M_{np}, and give a polynomial time bound for its operation. We then consider deterministic Turing machines which recognize the same language, $L(M_{np})$. In order for this to make sense, we must first show that $L(M_{np})$ is nonempty. In Lemma 4.1, we prove that M_{np} recognizes encoded (instance, certificate, verifier) triples for the SHORTEST COMMON SUPERSTRING decision problem, shown in [GMA80] to be NP-complete (see [Joh12] for references on NP-completeness).

We use a time complexity result of Knuth, Morris and Pratt [KMP77]. In Section 3, we construct a nondeterministic Turing machine, denoted M_n, which forces any deterministic Turing machine recognizing the same language $L(M_n)$ to satisfy a given condition, denoted by Q, defined on the encoded strings input to M_n. In the algorithm for M_n, we will need to compare any state transition rules encoded in the input with the rest of the input string. (We use this comparison procedure to determine if there are multiple state transition rules with different range elements corresponding to the same domain element. This allows us to check whether the input may encode a nondeterministic Turing machine.) In establishing an overall computational time bound for M_n, for this step we use the time bound given in [KMP77], which in our case reduces to $\mathcal{O}(|\langle l, s, t\rangle|)$, where $\langle l, s, t\rangle$ is the string input to M_n.

We pad the input with explicit time bounds encoded in unary. Several examples of padding are given in [AB09], including references to [Lad75] and [BFT98]; additional references to padding, including [Boo74], [Boo76], and [Iba72] are found in [Pap95]. Savitch [Sav70] also uses the technique, as pointed out in bibliographic remarks found in [BDG88].

Time-bounded computations are studied in [HS65], [CR72] and [Sip83]. Padding the input with explicit time bounds is used in the first step of the proof of the Cook-Levin theorem ([Coo71],[Lev73]), as outlined in [Aar16]. We note that this technique converts any algorithm that halts on all inputs into a polynomial-time algorithm.

We use ideas related to self-reference [Bol15]. We accomplish the computational step which forces any deterministic version of M_n to satisfy the condition Q on its input strings using a technique, related to the liar paradox [BG14], which might reasonably be called the partial liar: if an encoding of a deterministic version of M_n halts and rejects when simulated, M_n accepts. We use a second technique related to self-reference to force M_d itself to satisfy the condition Q, where M_d denotes a deterministic version of M_n. This is accomplished by simulating M_d with its own encoding as input. Borrowing a term used in λ-calculus [HS08], type theory [NG14], and proof theory [Str03], this could be called *self-application*. Both techniques can be viewed as weakened forms of self-reference.

We apply a theorem of Hennie and Stearns to construct time bounds for our algorithms. Arora and Barak [AB09] show how to use a theorem of Hennie and Stearns [HS66] to establish bounds on the time required for a universal Turing machine to simulate another Turing machine.

We apply this theorem to construct bounds for Turing machines built up from special purpose Turing machines, as illustrated by the following example.

Say a Turing machine M reads a string $\langle s\rangle$, copies $\langle s\rangle$ onto the input tape for a universal Turing machine U_1, runs U_1 on input $\langle s\rangle$ for time bounded by $|\langle s\rangle|^2$, and, if U_1 has halted with $\langle REJECT\rangle$ on its output tape, M accepts. Otherwise M rejects.

We can establish upper bounds on the time required for M to perform this computation by building M up from special-purpose Turing machines, using the Hennie-Stearns theorem as follows.

1. A special-purpose Turing machine that only reads strings can read $\langle s\rangle$ in time bounded by $\mathcal{O}(|\langle s\rangle|)$. Thus, by the Hennie-Stearns theorem, a universal Turing machine (call it U_2) can simulate this special-purpose machine in time bounded by $\mathcal{O}(|\langle s\rangle| log(|\langle s\rangle|))$. Note that we also have the time bound $\mathcal{O}(|\langle s\rangle|^2))$ for U_2 to simulate this computation, and in general, a computation time bounded by $\mathcal{O}(n^k))$ can be simulated by another universal Turing machine in time bounded by $\mathcal{O}(n^{k+1}))$. We will use the simpler polynomial form of the time bound.

2. A special-purpose Turing machine that copies strings can copy $\langle s\rangle$ onto U_1's input tape in time bounded by $\mathcal{O}(|\langle s\rangle|)$, thus U_2 can simulate this special-purpose machine also in time bounded by $\mathcal{O}(|\langle s\rangle|^2))$.

3. Since the universal Turing machine U_1 halts in time bounded by $|\langle s\rangle|^2$, U_2 can simulate this computation in time bounded by $\mathcal{O}(|\langle s\rangle|^3)$.

4. The control logic to examine the string on U_1's tape and accept or reject can be simulated by U_2 in time bounded by $\mathcal{O}(1))$.

We can encode the U_2-instructions for all four of these special-purpose Turing machines, including control logic for each special purpose Turing machine to start and halt, as instructions for a third universal Turing machine, U_3. Note that it is not necessary to pass any variable length strings, representing results of intermediate computations, from one special-purpose machine to another. Then U_3 can simulate the combined instructions in time bounded by $\mathcal{O}(|\langle s\rangle|^4))$.

We give an algorithm for a Turing machine whose computations are used to prove one of our results. This technique is used in Section 5. The application of the Curry-Howard-de Bruijn correspondence [SU06] to extract programs from proofs is developed in [PMW93], [CS93], [BC04], [PCW05], and [PCG$^+$10]. Conversely, here we apply details of the computational steps performed by a Turing machine to establish inferences in our proofs. From this point of view, the role of Lemmas 5.2 through 5.4 is to provide a verification that a Turing machine implementing Algorithm 5.1 has the correct properties required to prove Lemma 5.5. This overall approach is well-known, having been used in at least two different ways in the proof of the four-color map theorem [Gon08].

We close this Section with a brief outline of the paper. In Section 3, we develop the algorithmic proof techniques which will be used in Section 4 and modified for use in Section 5. Section 3 begins with an algorithm implemented by a Turing machine M_n which forms the basis for the counterexample developed in Section 4. While the techniques used to construct this algorithm are well-known, as described above, the details of the algorithm are thought to contribute some original elements. The main result of Section 3 is Corollary 3.6, which establishes that the computation performed by M_n forces a condition, denoted by Q, to hold for an infinite number of input strings.

The two principal results of the paper are proved in Sections 4 and 5. In Section 4, we develop the algorithm given in Section 3 further, giving a specific condition Q on the length of the input strings. We denote a Turing machine implementing this algorithm, with Q as in Section 4, by M_{np}. We prove Theorem 4.3, which asserts that *any deterministic version of M_{np} must be of Savitch type.* This immediately provides a result which eliminates some false proofs of P = NP from consideration, which we state as Corollary 4.4: *any correct proof that* P = NP *must imply the existence of a polynomial-time algorithm of Savitch type for M_{np}.* In Section 5, we complete our proof that NP $\not\subseteq$ P by proving Theorem 5.7: *no algorithm exists which recognizes the language $L(M_{np})$ in polynomial time.* Thus $L(M_{np})$ is an example of a language in NP which is not in P, and we have shown that NP $\not\subseteq$ P.

3. **An example of the algorithmic proof technique used in the paper**

Here we develop, by means of an example, the principal proof techniques used in the paper. We consider a nondeterministic Turing machine M_n, and a deterministic Turing machine, M_d, which recognizes the same language, $L(M_n)$. We prove Corollary 3.6, which states that, unless M_d is of Savitch type, we can force any valid encoding of M_d to satisfy a condition, denoted by Q, for an infinite number of input strings. We prepare for that result by first proving, in Lemma 3.2, that M_n is nondeterministic for any valid choice of the condition Q, and then, in Lemma 3.3, that M_n halts on all inputs. We next consider the computational paths available to M_n when an encoding $\langle M_d \rangle$ of M_d is an appropriately coded substring of the input string, and where M_d is not of Savitch type. M_n is structured so that, as in Lemmas 3.4 and 3.5, the partial liar technique can be used along with the fact that M_d must recognize $L(M_n)$, to show that we can always find input substrings for which M_n has such a computational path on which Q is satisfied.

A special case of this example will be studied in Section 4, and a modified version of this example will be developed in Section 5.

A partial liar algorithm: Fix a universal Turing machine U, let Q be a condition on strings, and consider a Turing machine, denoted M_n, which implements the following algorithm:

Algorithm 3.1. *On input string $\langle l, s, t \rangle$, M_n checks whether the string $\langle l, s, t \rangle$ contains U-encodings of any nondeterministic state transitions. If so, M_n writes $\langle REJECT \rangle$ on its output tape and halts.*

If not, M_n selects strings l_ϵ, s_ϵ and t_ϵ of length $|\langle l_\epsilon\rangle| = |\langle l\rangle|$, $|\langle s_\epsilon\rangle| = |\langle s\rangle|$ and $|\langle t_\epsilon\rangle| \leq |\langle t\rangle|$, respectively.

M_n places U in a halt state. If $|\langle t\rangle| > 0$, M_n runs U on input $\langle s_\epsilon, l_\epsilon, s_\epsilon, t_\epsilon\rangle$ for time bounded by $|\langle t\rangle|$ and suspends any computation still running on U.

If $\langle l_\epsilon\rangle = \langle l\rangle$ and $\langle s_\epsilon\rangle = \langle s\rangle$, M_n performs the following checks:

M_n checks whether U has entered a halt state.

if U has not halted, M_n writes $\langle REJECT\rangle$ its output tape and halts;

if U has halted with $\langle ACCEPT\rangle$ on its output tape:

M_n evaluates the condition Q;

if Q evaluates to $\top$, M_n writes $\langle ACCEPT\rangle$ its output tape and halts;

if Q evaluates to $\bot$, M_n writes $\langle REJECT\rangle$ its output tape and halts.

If U has halted with $\langle REJECT\rangle$ on its output tape, M_n writes $\langle ACCEPT\rangle$ its output tape and halts.

Otherwise, M_n rejects and halts.

We may assume that M_n is able to recognize elements of any state transition relation encoded in $\langle l, s, t\rangle$. For each element r of any state transition relation encoded in $\langle l, s, t\rangle$, M_n scans $\langle l, s, t\rangle$, looking for encoded state transition rules mapping the domain element of r to some range element or elements different from those encoded in r. If any such rules are found, M_n rejects; if no such rules are found, $\langle l, s, t\rangle$ cannot encode any nondeterministic Turing machine. Of course, M_n will also reject encodings of Turing machines that happen to have some ineffective nondeterministic state transition rules but are otherwise deterministic; this, however, has no consequences for the results we obtain here.

On input $\langle l, \langle M_d\rangle, t\rangle$, for infinitely many strings l, M_n ensures that condition Q is satisfied. To establish this, we examine the computation performed by M_n on input $\langle l, \langle M_d\rangle, t\rangle$.

We use the fact that M_d recognizes the same language as M_n to show that, for any l, we can find t such that M_n accepts on input $\langle l, \langle M_d\rangle, t\rangle$ and U halts on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_\epsilon\rangle$ (i.e., self-application of M_d) within time bounded by $|\langle t\rangle|$, with $\langle ACCEPT\rangle$ on U's output tape and with $|\langle t_\epsilon\rangle| \leq |\langle t\rangle|$. This is accomplished in the proof of Lemma 3.4, using the fact that M_n halts on all inputs (established in Lemma 3.3) and the partial liar technique. But then Q must evaluate to $\top$ (Lemma 3.5), and we arrive at Corollary 3.6.

We first establish that M_n rejects input strings that contain one of its own encodings, $\langle M_n\rangle$. This will follow from the proof that M_n is nondeterministic. We restrict consideration to conditions which are valid in the sense that M_n is always able to evaluate Q when required.

Lemma 3.2. *M_n is nondeterministic for any valid choice of the condition Q.*

Proof. The selection of l_ϵ, s_ϵ and t_ϵ results in different computational paths for M_n. This selection does not depend on the choice of Q. □

We next establish that M_n halts on all input strings having the correct form; this result will be used in the proof of Lemma 5.3 below.

Lemma 3.3. *Let l, s, and t be any strings. Then M_n halts on all inputs of the form $\langle l, s, t\rangle$, with precisely one of the strings $\langle ACCEPT\rangle$ or $\langle REJECT\rangle$ on its output tape.*

Proof. Given any input of the form $\langle l, s, t\rangle$, M_n checks whether $\langle l, s, t\rangle$ contains U-encodings of any nondeterministic state transitions. If so, M_n halts with $\langle REJECT\rangle$ on its output tape. If not, M_n selects l_ϵ, s_ϵ, and t_ϵ and invokes U on input $\langle s_\epsilon, l_\epsilon, s_\epsilon, t_\epsilon\rangle$, forcing U to suspend after time $|\langle t\rangle|$. M_n then proceeds to check whether U has entered a halt state; either U has halted or it has not. If U has not halted, M_n rejects and halts. If U has halted, U will have some string $\langle x\rangle$, possibly the empty string, on its output tape. If $\langle x\rangle = \langle REJECT\rangle$, M_n accepts and halts. If $\langle x\rangle = \langle ACCEPT\rangle$, M_n evaluates the condition Q; M_n has the power to compute this evaluation by assumption. If Q evaluates to $\top$, M_n accepts and halts; if Q evaluates to $\perp$, M_n rejects and halts. In all other cases, including any case where U halts with some string other than $\langle ACCEPT\rangle$ or $\langle REJECT\rangle$ on its output tape, M_n rejects and halts. □

Now assume that there exists a deterministic Turing machine, denoted M_d, which recognizes $L(M_n)$, and let $\langle M_d\rangle$ be a fixed encoding of M_d, otherwise arbitrary. Note that, if M_d is of Savitch type, M_d explores the computational paths available to the nondeterministic Turing machine M_n. In order to do this, M_d must have access to some encoding of the nondeterministic state transitions executed on those paths. Thus, some nondeterministic state transitions must be encoded into $\langle l, \langle M_d\rangle, t\rangle$, and so M_n will reject on any input which includes this encoding.

We note further that, if M_d is a deterministic Turing machine that is not of Savitch type, there must exist encodings of M_d that do not reference any nondeterministic state transitions. Any other encodings of M_d must be obtained from one of these by adding ineffective nondeterministic state transitions, and these can be eliminated to obtain a valid encoding. Thus, unless M_d is of Savitch type, we may assume that there exists a computational path for M_n on which M_n will simulate M_d on U.

Algorithm 3.1 uses the partial liar technique to force Q to hold. The partial liar is implemented in the statement:

"If U has halted with $\langle REJECT\rangle$ on its output tape, M_n writes $\langle ACCEPT\rangle$ its output tape and halts."

Computational paths evaluating this statement will be important in the proof of the next lemma. Unless M_d is of Savitch type, Q will be forced to evaluate to $\top$ on any computational path where M_n accepts on input $\langle l, \langle M_d\rangle, t\rangle$ and U halts on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_\epsilon\rangle$ with $\langle ACCEPT\rangle$ on its output tape in time bounded by $|\langle t\rangle|$, with $|\langle t_\epsilon\rangle| \leq |\langle t\rangle|$. We show that such computational paths exist for any string l.

Lemma 3.4. *For any l, there exist strings t_1 and t_2 such that the following conditions hold simultaneously*:

a) *M_n accepts on input $\langle l, \langle M_d\rangle, t_2\rangle$.*

b) *Unless M_d is of Savitch type, on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_1\rangle$, U halts in time bounded by $|\langle t_2\rangle|$ with $\langle ACCEPT\rangle$ on its output tape.*

c) $|\langle t_1\rangle| \leq |\langle t_2\rangle|$.

Proof. For any strings l, s, and t, we know that M_d halts on all inputs of the form $\langle l, s, t\rangle$, since, by Lemma 3.3, M_n does. In particular, M_d halts on input $\langle l, \langle M_d\rangle, t\rangle$ for arbitrary l and t. Then U must halt on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t\rangle$ since U simulates M_d in this case. Thus we can choose some string t_0 and let t_h be any string with $|\langle t_h\rangle| \geq |\langle t_0\rangle|$ and $|\langle t_h\rangle|$ greater than or equal to the time required for U to halt on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_0\rangle$.

We assume M_d is not of Savitch type and consider the following cases.

Case 1

If M_n accepts on input $\langle l, \langle M_d\rangle, t_h\rangle$ and U halts on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_0\rangle$ with $\langle ACCEPT\rangle$ on its output tape, set $t_1 = t_0$, $t_2 = t_h$ and we are done.

Case 2

On the other hand, if M_n accepts on input $\langle l, \langle M_d\rangle, t_h\rangle$ but U halts on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_0\rangle$ with $\langle REJECT\rangle$ on its output tape, let t' be any string with $|\langle t'\rangle| \geq |\langle t_h\rangle|$. Then, on input $\langle l, \langle M_d\rangle, t'\rangle$, M_n has an accepting computational path, since M_n may select $t_\epsilon = t_0$ and run U on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_0\rangle$ for time bounded by $|\langle t'\rangle|$. On this input, U halts with $\langle REJECT\rangle$ on its output tape, and so M_n must accept. Since M_d recognizes $L(M_n)$, M_d must also accept on input $\langle l, \langle M_d\rangle, t'\rangle$. Now consider the result of simulating M_d on this input: U must halt on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t'\rangle$ with $\langle ACCEPT\rangle$ on its output tape, for any $|\langle t'\rangle| \geq |\langle t_h\rangle|$. So we set $t_1 = t'$ and let t_k be a string whose length is greater than or equal to the time required for U to halt on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_1\rangle$. Choose any t_2 satisfying $|\langle t_2\rangle| \geq \max\{|\langle t_k\rangle|, |\langle t_1\rangle|\}$. Then M_n has an accepting path as follows: M_n may choose $t_\epsilon = t_1$ and run U on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_1\rangle$ for time bounded by $|\langle t_2\rangle|$. U will then halt with $\langle ACCEPT\rangle$ on its output tape. We already know that M_n accepts on input

$\langle l, \langle M_d, t_2\rangle$ since $|\langle t_2\rangle| \geq |\langle t_1\rangle|$, $t_1 = t'$, $|\langle t'\rangle| \geq |\langle t_h\rangle|$, and M_n accepts for any $\langle l, \langle M_d\rangle, t\rangle$ with $|\langle t\rangle| \geq |\langle t_h\rangle|$.

Case 3

There remains the possibility that M_n rejects on input $\langle l, \langle M_d\rangle, t_h\rangle$. In this event, set $t_0 = t_h$, and determine a new t_h with $|\langle t_h\rangle| \geq |\langle t_0\rangle|$ and$|\langle t_h\rangle|$ greater than or equal to the time required for U to halt on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_0\rangle$. For this, t_h, M_n has an accepting path on input $\langle l, \langle M_d\rangle, t_h\rangle$, since M_n may select $t_\epsilon = t_0$ and run U on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_0\rangle$ for time bounded by $|\langle t_h\rangle|$. On this input, U halts with $\langle REJECT\rangle$ on its output tape, and so M_n must accept. Then Case 2 above holds, and this completes the proof. □

Thus, for any l, we can find t such that M_n accepts on input $\langle l, \langle M_d\rangle, t\rangle$ and U halts on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_\epsilon\rangle$ with $\langle ACCEPT\rangle$ on its output tape, within time bounded by $|\langle t\rangle|$: we obtain t_1 and t_2 from Lemma 3.4 and set $t = t_2$, $t_\epsilon = t_1$. This is all we need to force condition Q to hold, as we establish next.

Lemma 3.5. *Let t be such that M_n accepts on input $\langle l, \langle M_d\rangle, t\rangle$ and let there exist t_ϵ, with $|\langle t_\epsilon\rangle| \leq |\langle t\rangle|$, such that U halts on input $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_\epsilon\rangle$, with $\langle ACCEPT\rangle$ on U's output tape and in time bounded by $|\langle t\rangle|$. Then condition Q evaluates to $\top$.*

Proof. We work backwards from the unique point in Algorithm 3.1 where M_n accepts. In order to reach this point, Q must evaluate to $\top$. Continuing to work backwards, we see that, in order for M_n to evaluate Q, U must have halted with $\langle ACCEPT\rangle$ on its output tape. In addition, M_n must have run U with $\langle\langle M_d\rangle, l, \langle M_d\rangle, t_\epsilon\rangle$ as input, in order to satisfy $\langle l_\epsilon\rangle = \langle l\rangle$ and $\langle s_\epsilon\rangle = \langle M_d\rangle$, and U must have halted in time bounded by $|\langle t\rangle|$. Thus all computational paths accessible to M_n for which the conditions in the hypothesis of the Lemma hold must incorporate a state transition in which Q evaluates to $\top$. □

By Lemma 3.4, for any string l, we can always find t and t_ϵ satisfying the hypotheses of Lemma 3.5; the main result of this section is now an easy corollary.

Corollary 3.6. *If M_n has a deterministic version M_d that is not of Savitch type, then for any string l and any encoding $\langle M_d\rangle$ of M_d there exists a string t such that, on input $\langle l, \langle M_d\rangle, t\rangle$, M_n forces Q to evaluate to $\top$.*

4. M_{np} is a nondeterministic polynomial-time Turing machine whose deterministic counterpart must be of Savitch type

Consider a Turing machine, which we denote M_{np}, and which executes Algorithm 3.1 with condition Q specified as $(|\langle l\rangle| \leq |\langle s\rangle|)$. In this section, M_d

will denote a deterministic version of M_{np}. We know that, if $L(M_{np})$ is in NP, a deterministic Turing machine of Savitch type with $L(M_{np}) = L(M_d)$ must exist (by Savitch's theorem). We will eliminate all other possibilities, that is, we will show that no deterministic Turing machine M_d that is not of Savitch type can exist with $L(M_{np}) = L(M_d)$.

We establish this result as follows: first we show that $L(M_{np})$ is in the complexity class NP. We demonstrate this by showing that $L(M_{np})$ is nonempty (Lemma 4.1) and that M_{np} recognizes $L(M_{np})$ in time bounded by $\mathcal{O}(|\langle l, s, t\rangle|)^4$ (Lemma 4.2). Second, we know that for any Turing machine M, there is a finite string that is a U-encoding of M, where U is a universal Turing machine. In Lemma 4.5, we then apply Corollary 3.6 to the condition Q, with $\langle s\rangle = \langle M_d\rangle$, so that Q is given by $(|\langle l\rangle| \leq |\langle M_d\rangle|)$. Since l is an arbitrary string, l may have any finite length. Further $\langle M_d\rangle$ can be any encoding of M_d, so we conclude that M_d does not have a finite encoding. As a consequence, M_d cannot be a Turing machine. M_{np} thus provides an example of a Turing machine which has no deterministic versions except those of Savitch type.

Of course, this result does not establish that $\mathsf{NP} \not\subseteq \mathsf{P}$; there could still exist a deterministic algorithm of Savitch type that runs in polynomial time for M_{np}. In Section 5, we will show that we can eliminate that possibility as well.

We start by proving that $L(M_{np})$ *is in* NP. M_{np} is nondeterministic by Lemma 3.2. We will demonstrate that $L(M_{np})$ is in NP by first showing that $L(M_{np})$ is nonempty, in fact infinite, and then showing M_{np} runs in time bounded by a fixed polynomial which is $\mathcal{O}(|\langle l, s, t\rangle|)^4$ in the length of the input string $\langle l, s, t\rangle$. We demonstrate that $L(M_{np})$ is nonempty by showing that M_{np} recognizes instances of the SHORTEST COMMON SUPERSTRING decision problem.

LEMMA 4.1. *M_{np} accepts all strings of the form $\langle l, s, t\rangle$ where*

$\langle s\rangle$ contains U-encodings of an instance of, and a verifier for, the NP-complete *decision problem* SHORTEST COMMON SUPERSTRING,

$\langle l\rangle$ contains a U-encoding of a certificate for the instance encoded in $\langle s\rangle$, that is, a shortest common superstring, and

$|\langle t\rangle|$ is a time bound for U to read the input, unpack the instance, verifier and certificate, run the verifier encoded in $\langle s\rangle$ on the instance encoded in $\langle s\rangle$ and the certificate encoded in $\langle l\rangle$, and write $\langle ACCEPT\rangle$ on its output tape.

Proof. The verifier, instance, and unpacking instructions can be encoded in $\langle s\rangle$ without any nondeterministic state transitions, so a computational path exists on which M_{np} runs U on input $\langle s, l, s, t\rangle$ for time $|\langle t\rangle|$. We encode U–instructions in $\langle s\rangle$ to unpack the verifier and instance of shortest common superstring and then execute the verifier on that instance, along with the certificate encoded in $\langle l \rangle$. We can also arrange the encoding of the certificate, instance and verifier to ensure that $|\langle l \rangle| \leq |\langle s \rangle|$, that is, Q evaluates to $\top$. Then there exist computational paths with $\langle l_\epsilon \rangle = \langle l \rangle$ and $|\langle t_\epsilon \rangle| = |\langle t \rangle|$, and for any of these computational paths, U will halt within time $|\langle t \rangle|$ and write $\langle ACCEPT \rangle$ on its output tape. M_{np} will read that output, find that $|\langle l \rangle| \leq |\langle s \rangle|$, accept and halt. $\square$

Note that it is not necessary for s to read the input t; in fact, s can simply ignore t.

Since M_{np} is nondeterministic and $L(M_{np})$ is nonempty, the language recognized by M_{np} will be in NP if we can find a fixed polynomial in the length of the input string that bounds the running time for M_{np} on all inputs. We establish this in the following lemma.

Lemma 4.2. *M_{np} runs in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|^4)$.*

Proof. We will analyze M_{np}'s operation into steps, estimate each step, and then assemble the estimates using the Hennie-Sterns theorem. M_{np} can (i) read the input in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|)$, (ii) check whether $\langle l, s, t \rangle$ encodes any nondeterministic state transitions in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|^2)$, (iii) select and check the lengths of l_ϵ, s_ϵ, and t_ϵ in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|)$, (iv) place U in a halt state in time bounded by $\mathcal{O}(1)$, (v) run U for time bounded by $|\langle t \rangle|$ in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|)$, (vi) check $\langle l_\epsilon \rangle = \langle l \rangle$, and $\langle s_\epsilon \rangle = \langle s \rangle$ in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|)$, (vii) check whether U has halted, in time $\mathcal{O}(1)$, (viii) evaluate Q in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|)$, (ix) write $\langle ACCEPT \rangle$ or $\langle REJECT \rangle$ and enter a halt state in time bounded by $\mathcal{O}(1)$.

We apply the Hennie-Stearns bound twice, first to each of the above steps individually, and second to a simulated version of M_{np} constructed by assembling all of the above steps together, along with suitable control logic.

(First application of Hennie-Stearns). Each of these steps can be individually simulated on a universal Turing machine in the time required to simulate the step with the highest time complexity, i.e. step (ii), which requires time bounded by $\mathcal{O}(|\langle l, s, t \rangle|^3)$ to simulate.

(Second application of Hennie-Stearns). Finally, each of the steps can be combined and simulated together with control logic, itself requiring time $\mathcal{O}(1)$ to simulate, to create an implementation of M_{np}. This implementation of M_{np} will run in time bounded by $\mathcal{O}(|\langle l, s, t \rangle|^4)$. $\square$

We now state the main theorem of this section:

Theorem 4.3. *Any deterministic algorithm that recognizes $L(M_{np})$ must be of Savitch type.*

Before proving Theorem 4.3, we take note of an important consequence: unless an algorithm of Savitch type exists which recognizes $L(M_{np})$ in polynomial time, the language accepted by M_{np} is in NP but not in P, and M_{np} provides a counterexample disproving $\mathsf{P} = \mathsf{NP}$. Theorem 4.3 thus yields the following barrier to proving $\mathsf{P} = \mathsf{NP}$:

Corollary 4.4. *Any correct proof that* $\mathsf{P} = \mathsf{NP}$ *must imply the existence of a polynomial-time algorithm of Savitch type for* M_{np}.

To establish Theorem 4.3, we assume the existence of a deterministic Turing machine, M_d, which recognizes $L(M_{np})$ but is not of Savitch type, and show that this leads to a contradiction.

Lemma 4.5. *Unless* M_d *is of Savitch type, any encoding* $\langle M_d \rangle$ *of* M_d *must have infinite length.*

Proof. By Corollary 3.6, we have $|\langle l \rangle| \leq |\langle M_d \rangle|$. But l can be any string, and can thus have any finite length. Thus the length of $\langle M_d \rangle$ is larger than any finite number, and hence is infinite. But $\langle M_d \rangle$ was arbitrary, so all encodings of M_d must be of infinite length. $\square$

Proof. (of Theorem 4.3)
Assume there exists a deterministic version M_d of M_{np} that is not of Savitch type. Then, by Lemma 4.5, any encoding of M_d must have infinite length. But any Turing machine must have some finite length encoding, so this assumption leads to a contradiction, and we conclude that any deterministic version of M_{np} must be of Savitch type. $\square$

We end this section with a Lemma establishing a property of M_{np} which we will require in Section 5.

Lemma 4.6. *The number of computational paths for* M_{np} *is bounded below by* $2^{|\langle l,s,t \rangle|}$.

Proof. On input $\langle l, s, t \rangle$ M_{np} selects strings s_ϵ, l_ϵ and t_ϵ of length $|\langle s_\epsilon \rangle| = |\langle s \rangle|$, $|\langle l_\epsilon \rangle| = |\langle l \rangle|$ and $|\langle t_\epsilon \rangle| \leq |\langle t \rangle|$, respectively. For encodings over $\{0, 1\}$, there are $(2^{|\langle l \rangle|})(2^{|\langle t \rangle|})(\sum_{k=0}^{|\langle t \rangle|} 2^k)$ distinct ways to make this selection, each of which corresponds to a computational path for M_{np}, and we have

$$(2^{|\langle l \rangle|})(2^{|\langle t \rangle|})(\sum_{k=0}^{|\langle t \rangle|} 2^k) \geq (2^{|\langle s \rangle|})(2^{|\langle l \rangle|})(2^{|\langle t \rangle|}) = 2^{(|\langle s \rangle| + |\langle l \rangle| + |\langle t \rangle|)} \geq (2^{|\langle l,s,t \rangle|}).$$

$\square$

5. No algorithm exists which recognizes the language $L(M_{np})$ in polynomial time

In Sections 3 and 4, we applied intermediate results on algorithms for Turing machines M_n and M_{np} to establish a result concerning $L(M_{np})$ (Theorem 4.3). As seen in the proofs of Lemmas 3.4 and 3.5, specific computational steps performed by the algorithms lead to specific inferences in those proofs. In this section, we take this approach a step further, and construct an algorithm for the sole purpose of proving certain intermediate results, namely Lemmas 5.2 through 5.4 below, which are then applied to prove Lemma 5.5.

We introduce a nondeterministic exponential time Turing machine, M_{nexp}, which executes Algorithm 5.1.

Algorithm 5.1. *On the input string $\langle q_1, q_2, r, l, s, t_1, t_2\rangle$, M_{nexp} first checks to see if either of the input substrings $\langle q_1\rangle$ or $\langle q_2\rangle$ may encode any nondeterministic state transitions. If so, M_{nexp} rejects and halts.*

If not, M_{nexp} places U in a halt state. M_{nexp} then selects a string $\langle t_\epsilon\rangle$ of length $|\langle t_\epsilon\rangle| \le |\langle t_2\rangle|$ and runs U on input $\langle r, q_1, q_2, r, l, s, t_1, t_\epsilon\rangle$ for time bounded by $2^{|\langle t_2\rangle|}$, suspends any further computation by U, and checks whether U has halted or not. If U has not halted, M_{nexp} writes $\langle REJECT\rangle$ on its output tape and halts.

If U has halted, M_{nexp} examines the string on U's output tape. Denote this string by $\langle x\rangle$. If $\langle x\rangle = \langle REJECT\rangle$, M_{nexp} writes $\langle ACCEPT\rangle$ on its output tape and halts.

If $\langle x\rangle = \langle ACCEPT\rangle$, M_{nexp} evaluates condition Q as follows: M_{nexp} runs U on input $\langle q_1, \langle M_{np}\rangle, l, s, t_1\rangle$ for time bounded by $2^{|\langle t_2\rangle|}$. M_{nexp} then computes the actual time used by U and stores this positive integer, which we denote as $K_1(\langle l, s, t_1\rangle)$. M_{nexp} also runs U on input $\langle q_2, \langle M_{np}\rangle, l, s, t_1\rangle$ for time bounded by $2^{|\langle t_2\rangle|}$. M_{nexp}, computes the actual time used by U and stores this positive integer, which we denote as $K_2(\langle l, s, t_1\rangle)$. If $|\langle q_1\rangle|K_1(\langle l, s, t_1\rangle) \le |\langle q_2\rangle|K_2(\langle l, s, t_1\rangle)$, M_{nexp} accepts and halts.

Otherwise, M_{nexp} rejects and halts.

Note that Algorithm 5.1 has been obtained from Algorithm 3.1 by modifying the input string and the condition Q. We will use Algorithm 5.1 differently, however; our aim in the present Section is not to prove results about M_{nexp}, but to apply steps of the computation performed by M_{nexp} to prove results about M_{np}.

Let M_σ denote an algorithm of Savitch type that, on input $\langle\langle M_{nd}\rangle, l, s, t\rangle$ explores all possible computational paths encoded in M_{nd}, where M_{nd} denotes a nondeterministic Turing machine, and writes $\langle ACCEPT\rangle$ on its output tape if M_{nd} accepts on any of its computational paths; if no such path is found, M_σ writes $\langle REJECT \rangle$ on its output tape.

We assume the existence of a polynomial-time deterministic version of M_{np} (otherwise the results of Section 4 provide a counterexample which proves that $\mathsf{NP} \not\subseteq \mathsf{P}$). Let the Turing machine M_π implement this algorithm, which must necessarily be of Savitch type, again by the results of Section 4. By assumption, then, on input $\langle \langle M_{np} \rangle, l, s, t \rangle$, M_π recognizes $L(M_{np})$ in time bounded by some fixed polynomial in the length of the input string.

Consider the input string $\langle \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_2 \rangle$, where $\langle M_\pi \rangle$ is an arbitrary encoding of M_π. We examine the computation performed by M_{nexp} on this input, and show that the details of this computation imply that all such encodings have infinite length. This shows that M_π cannot be a Turing machine, since a Turing machine must have some encoding of finite length. This result and the results of the previous subsection imply that $\mathsf{NP} \not\subseteq \mathsf{P}$.

The condition in Algorithm 5.1 corresponding to Q in Algorithm 3.1 is, in this case: $|\langle M_\sigma \rangle| K_1(\langle l, s, t_1 \rangle) \leq |\langle M_\pi \rangle| K_2(\langle l, s, t_1 \rangle)$. We will show that the computation performed by M_{nexp} forces this condition, which we also denote by Q, to evaluate to $\top$.

Note that we apply this condition Q to the inputs q_1 and q_2, so these are now the inputs which must be checked to make sure they do not encode any nondeterministic state transitions.

It is possible to prove that $L(M_{nexp})$ is in NEXP; however, we will not require this result.

We are now ready to prove that *no polynomial-time algorithm of Savitch type exists corresponding to* M_{np}. Fix an arbitrary encoding of M_π and denote this encoding by $\langle M_\pi \rangle$. We begin by demonstrating that M_{nexp} halts on all inputs of the correct form. The result is analogous to Lemma 3.3 and the proof is similar.

Lemma 5.2. *M_{nexp} halts on all inputs of the form $\langle q_1, q_2, r, l, s, t_1, t_2 \rangle$.*

Proof. Given any input of the form $\langle q_1, q_2, r, l, s, t_1, t_2 \rangle$, M_{nexp} checks whether $\langle q_1 \rangle$ or $\langle q_2 \rangle$ contains U-encodings of any nondeterministic state transitions. If so, M_{nexp} halts with $\langle REJECT \rangle$ on its output tape. If not, M_{nexp} proceeds to check whether U has entered a halt state; either U has halted or it has not. If U has not halted, M_{nexp} rejects and halts. If U has halted, U will have some string $\langle x \rangle$, possibly the empty string, on its output tape. If $\langle x \rangle = \langle REJECT \rangle$, M_{nexp} accepts and halts. If $\langle x \rangle = \langle ACCEPT \rangle$, M_{nexp} evaluates the condition Q. If Q evaluates to $\top$, M_{nexp} accepts and halts; if Q evaluates to $\perp$, M_{nexp} rejects and halts. In all other cases, including any case where U halts with some string other than $\langle ACCEPT \rangle$ or $\langle REJECT \rangle$ on its output tape, M_{nexp} rejects and halts. $\square$

Our next result is adapted from Lemma 3.4, and the overall structure of the proof proceeds through an analysis of cases as in the proof of that Lemma.

LEMMA 5.3. *For any strings l, s and t_1, there exist strings $t_{2,0}$ and $t_{2,1}$ such that the following conditions hold simultaneously*:

a) M_{nexp} accepts on input $\langle\langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,1}\rangle$,

b) U halts within time bounded by $2^{|\langle t_{2,1}\rangle|}$ on input $\langle\langle M_\sigma\rangle, \langle M_{np}\rangle, l, s, t_1\rangle$ and on input $\langle\langle M_\pi\rangle, \langle M_{np}\rangle, l, s, t_1\rangle$,

c) On input

$$\langle\langle M_{nexp}\rangle, \langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle$$

U halts in time bounded by $2^{|\langle t_{2,1}\rangle|}$ with $\langle ACCEPT\rangle$ on its output tape,

d) $|\langle t_{2,0}\rangle| \leq |\langle t_{2,1}\rangle|$.

Proof. We know that M_{nexp} halts on all inputs of the form $\langle q_1, q_2, r, l, s, t_1, t_2\rangle$ by Lemma 5.2. In particular, M_{nexp} halts on input

$$\langle\langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle.$$

Then U must halt on input

$$\langle\langle M_{nexp}\rangle, \langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle.$$

since U simulates M_{nexp} in this case. Thus we can choose some string $t_{2,0}$ and let $t_{2,h}$ be any string with $|\langle t_{2,h}\rangle| \geq |\langle t_{2,0}\rangle|$ and $2^{|\langle t_{2,h}\rangle|}$ greater than or equal to the time required for U to halt on input

$$\langle\langle M_{nexp}\rangle, \langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle,$$

and also greater than or equal to the time required for U to halt on the inputs $\langle\langle M_\sigma\rangle, \langle M_{np}\rangle, l, s, t_1\rangle$ and $\langle\langle M_\pi\rangle, \langle M_{np}\rangle, l, s, t_1\rangle$. We consider three cases.

Case 1

If M_{nexp} accepts on input $\langle\langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,h}\rangle$ and U halts on input $\langle\langle M_{nexp}\rangle, \langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle$ within time bounded by $2^{|\langle t_{2,h}\rangle|}$ and with $\langle ACCEPT\rangle$ on its output tape, set $t_{2,1} = t_{2,h}$ and we are done.

Case 2

On the other hand, if M_{nexp} accepts on input $\langle\langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,h}\rangle$ but U halts on input

$$\langle\langle M_{nexp}\rangle, \langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle$$

within time bounded by $2^{|\langle t_{2,h}\rangle|}$ and with $\langle REJECT\rangle$ on its output tape, let t_2' be any string with $|\langle t_2'\rangle| \geq |\langle t_{2,h}\rangle|$. Then, on input

$$\langle\langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_2'\rangle,$$

M_{nexp} has an accepting computational path since M_{nexp} may select $t_\epsilon = t_{2,0}$ and run U on input $\langle\langle M_{nexp}\rangle, \langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_{2,0}\rangle$ for time bounded by $2^{|\langle t_2' \rangle|}$. By assumption, on this input U halts with $\langle REJECT \rangle$ on its output tape, and so M_{nexp} must accept. Then set $t_{2,0} = t_2'$ and let $t_{2,k}$ be a string with $2^{|\langle t_{2,k} \rangle|}$ greater than or equal to the time required for U to halt on input

$$\langle \langle M_{nexp} \rangle, \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_{2,0} \rangle$$

as well as for U to halt on input $\langle \langle M_\sigma \rangle, \langle M_{np} \rangle, l, s, t_1 \rangle$ and on input $\langle \langle M_\pi \rangle, \langle M_{np} \rangle, l, s, t_1 \rangle$. Choose any $t_{2,1}$ satisfying $|\langle t_{2,1} \rangle| \geq \max \{|\langle t_{2,k} \rangle|, |\langle t_{2,0} \rangle|\}$. Then, on input

$$\langle \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_{2,1} \rangle,$$

M_{nexp} has an accepting path as follows: M_{nexp} may choose $t_\epsilon = t_{2,0}$ and run U on input

$$\langle \langle M_{nexp} \rangle, \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_{2,0} \rangle.$$

for time bounded by $2^{|\langle t_{2,1} \rangle|}$. U will then halt with $\langle ACCEPT \rangle$ on its output tape. We already know that M_{nexp} accepts on input

$$\langle \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_{2,1} \rangle,$$

since $|\langle t_{2,1} \rangle| \geq |\langle t_{2,0} \rangle|$ and $t_{2,0} = t_2'$.

Case 3

There remains the possibility that M_{nexp} rejects on input

$$\langle \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_{2,h} \rangle,$$

In this event, set $t_{2,0} = t_{2,h}$, and determine a new $t_{2,h}$ with $|\langle t_{2,h} \rangle| \geq |\langle t_{2,0} \rangle|$ and $2^{|\langle t_{2,h} \rangle|}$ greater than or equal to the time required for U to halt on inputs

$$\langle \langle M_{nexp} \rangle, \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_{2,0} \rangle,$$

$\langle \langle M_\sigma \rangle, \langle M_{np} \rangle, l, s, t_1 \rangle$ and $\langle \langle M_\pi \rangle, \langle M_{np} \rangle, l, s, t_1 \rangle$. Then we can show that we have reduced this case to Case 2 above, as in the proof of Lemma 3.4. This completes the proof. □

Thus we can, for any strings l, s and t_1, find t_2 such that M_{nexp} accepts on input

$$\langle \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_2 \rangle,$$

and U halts on input $\langle \langle M_{nexp} \rangle, \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_\epsilon \rangle$: we obtain $t_{2,0}$ and $t_{2,1}$ from Lemma 5.3 and set $t_2 = t_{2,1}$, $t_\epsilon = t_{2,0}$.

We can now prove a result corresponding to Lemma 3.5.

Lemma 5.4. *Let t_2 be such that U halts on input $\langle \langle M_\sigma \rangle, \langle M_{np} \rangle, l, s, t_1 \rangle$ and on input $\langle \langle M_\pi \rangle, \langle M_{np} \rangle, l, s, t_1 \rangle$ and such that M_{nexp} accepts on input*

$$\langle \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_2 \rangle.$$

Let there exist t_ϵ, with $|\langle t_\epsilon \rangle| \leq |\langle t_2 \rangle|$, such that U halts on the input

$$\langle \langle M_{nexp} \rangle, \langle M_\sigma \rangle, \langle M_\pi \rangle, \langle M_{nexp} \rangle, l, s, t_1, t_\epsilon \rangle,$$

with $\langle ACCEPT\rangle$ on U's output tape in time bounded by $2^{|\langle t_2\rangle|}$. Then condition Q evaluates to $\top$.

Proof. Backtracking (as in the proof of Lemma 3.5) from the unique step in Algorithm 5.1 where M_{nexp} accepts , we see that the only computational path in M_{nexp} for which the conditions in the hypothesis of the lemma are possible is the path on which Q evaluates to $\top$. □

In any case, for any strings l, s, and t_1, there always exists some string t_2 such that M_{nexp} accepts on input $\langle\langle M_\sigma\rangle, \langle M_\pi\rangle, \langle M_{nexp}\rangle, l, s, t_1, t_2\rangle$. This gives us the following Lemma:

Lemma 5.5. *Any encoding $\langle M_\pi\rangle$ of M_π has infinite length.*

Proof. By Lemmas 5.3 and 5.4, for an infinite number of strings l, s, and t_1, condition Q evaluates to $\top$, that is $|\langle M_\sigma\rangle| K_1(\langle l, s, t_1\rangle) \leq |\langle M_\pi\rangle| K_2(\langle l, s, t_1\rangle)$ holds. M_σ explores all computational paths of M_{np}, thus $K_1(\langle l, s, t_1\rangle)$ is bounded below by $2^{|\langle l,s,t_1\rangle|}$ by Lemma 4.6. Also, by assumption M_π is able to execute an algorithm of Savitch type with input $\langle\langle M_{np}\rangle, l, s, t_1\rangle$ in polynomial time. We may take $|\langle M_{np}\rangle|$ as a fixed positive integer and conclude that is there is some polynomial f satisfying $K_2(\langle l, s, t_1\rangle) \leq f(|\langle l, s, t_1\rangle|)$. Combining all of these inequalities we have

$$2^{|\langle l,s,t_1\rangle|} \leq |\langle M_\sigma\rangle| K_1(\langle l, s, t_1\rangle) \leq |\langle M_\pi\rangle| K_2(\langle l, s, t_1\rangle) \leq f(|\langle l, s, t_1\rangle|)$$

and $|\langle M_\sigma\rangle| 2^{|\langle l,s,t_1\rangle|}/f(|\langle l, s, t_1\rangle|) \leq |\langle M_\pi\rangle|$ follows. But this inequality holds for infinitely many strings s, l, and t_1 and so $|\langle M_\pi\rangle|$, bounded below by the quotient of an exponential function and a polynomial function, must be infinite. Since $\langle M_\pi\rangle$ is an arbitrary fixed encoding of M_π, the result follows. □

We conclude that

Corollary 5.6. *No Turing machine of Savitch type exists that recognizes $L(M_{np})$ in polynomial time.*

Combining the results of this Section and the previous one, we have the following theorem:

Theorem 5.7. *No algorithm exists which recognizes the language $L(M_{np})$ in polynomial time.*

Then, since $L(M_{np})$ is in NP but not in P, we have the corollary:

Corollary 5.8. $\mathsf{NP} \not\subseteq \mathsf{P}$.

Acknowledgments. This paper is a synthesis of ideas found in the references cited below. The author here acknowledges this debt, with gratitude to the authors of those works. In particular, the author's interest in the problem began with a talk given by Hartmanis. The initial idea underlying the results given here is derived from the theorems of Karp and Lipton (with Kannan and Sipser); this idea was brought into its present form through the influence of Savitch's work.

Any errors are, of course, the responsibility of the author alone.

References

[Aar16] S. Aaronson, $P \overset{?}{=} NP$, in *Open Problems in Mathematics* (John Forbes Nash, Jr. and M. T. Rassias, eds.), Springer, 2016, see also http://www.scottaaronson.com/papers/pvsnp.pdf, pp. 1–122.

[AW09] S. Aaronson and A. Wigderson, Algebrization: A new barrier in complexity theory, *ACM Transactions on Computation Theory* **1** (2009), 2.

[All09] E. Allender, A status report on the P versus NP question, *Advances in Computers* **77** (2009), 117–147.

[AB09] S. Arora and B. Barak, *Computational complexity: a modern approach*, Cambridge University Press, 2009.

[BGS75] T. Baker, J. Gill, and R. Solovay, Relativizations of the $P =?NP$ question, *SIAM Journal on Computing* **4** (1975), 431–442.

[BDG88] J. L. Balcázar, J. Diaz, and J. Gabarró, *Structural Complexity* I, *volume* 11 *of* EATCS *Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, 1988.

[BG14] J. Beall and M. Glanzberg, Liar paradox, in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), 2014, http://plato.stanford.edu/archives/fall2014/entries/liar-paradox/.

[BC04] Y. Bertot and P. Castéran, *Interactive theorem proving and program development.* T*exts in Theoretical Computer Science*, *an* EATCS *Series*, Springer-Verlag, 2004.

[Bol15] T. Bolander, Self-reference, in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), 2015, http://plato.stanford.edu/archives/spr2015/entries/self-reference/.

[Boo74] R. V. Book, Comparing complexity classes, *Journal of Computer and System Sciences* **9** (1974), 213–229.

[Boo76] R. V. Book, Translational lemmas, polynomial time, and $(log\ n)^j$-space, *Theoretical Computer Science* **1** (1976), 215–226.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf, Nonrelativizing separations, in *Proceedings*, *Thirteenth Annual* IEEE *Conference on Computational Complexity*, IEEE, 1998, pp. 8–12.

[Bus12] S. R. Buss, Towards $NP - P$ via proof complexity and search, *Annals of Pure and Applied Logic* **163** (2012), 906–917.

[Coo03] S. Cook, The importance of the P versus NP question, *Journal of the ACM* **50** (2003), 27–29.

[Coo71] S. A. Cook, The complexity of theorem-proving procedures, in *Proceedings of the third annual* ACM *symposium on Theory of computing*, ACM, 1971, pp. 151–158.

[CR72] S. A. Cook and R. A. Reckhow, Time-bounded random access machines, in *Proceedings of the fourth annual* ACM *symposium on Theory of computing*, ACM, 1972, pp. 73–80.

[CS93] J. N. Crossley and J. C. Shepherdson, Extracting programs from proofs by an extension of the curry-howard process, in *Logical Methods*, Springer, 1993, pp. 222–288.

[Dav04] M. Davis, *The undecidable*: *Basic papers on undecidable propositions, unsolvable problems and computable functions*, Dover, 2004, Corrected republication of Raven Press edition of 1965.

[Flo67] R. W. Floyd, Nondeterministic algorithms, *Journal of the ACM* **14** (1967), 636–644.

[For09] L. Fortnow, The status of the P versus NP problem, *Communications of the ACM* **52** (2009), 78–86.

[GMA80] J. Gallant, D. Maier, and J. Astorer, On finding minimal length superstrings, *Journal of Computer and System Sciences* **20** (1980), 50–58.

[Gon08] G. Gonthier, Formal proof–the four-color theorem, *Notices of the AMS* **55** (2008), 1382–1393.

[HS65] J. Hartmanis and R. E. Stearns, On the computational complexity of algorithms, *Transactions of the American Mathematical Society* **117** (1965), 285–306.

[HO02] L. A. Hemaspaandra and M. Ogihara, *The complexity theory companion*, Springer, 2002.

[HS66] F. C. Hennie and R. E. Stearns, Two-tape simulation of multitape turing machines, *Journal of the ACM* **13** (1966), 533–546.

[HS08] J. R. Hindley and J. P. Seldin, *Lambda-calculus and combinators*: *an introduction*, Cambridge University Press, 2008.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Massachusetts, 1979.

[Iba72] O. H. Ibarra, A note concerning nondeterministic tape complexities, *Journal of the ACM* **19** (1972), 608–612.

[IKK09] R. Impagliazzo, V. Kabanets, and A. Kolokolova, An axiomatic approach to algebrization, in *Proceedings of the forty-first annual* ACM *symposium on Theory of computing*, ACM, 2009, pp. 695–704.

[Joh90] D. S. Johnson, A catalog of complexity classes, in *Handbook of Theoretical Computer Science Volume A*) (J. van Leeuwen, ed.), MIT Press, Cambridge, MA, USA, 1990, pp. 67–161.

[Joh12] D. S. Johnson, A brief history of NP-completeness, 1954–2012, *Documenta Mathematica* **Extra Volume: Optimization Stories** (2012), 359–376, 21st International Symposium on Mathematical Programming, Berlin, August 19-24 2012.

[KL82] R. M. Karp and R. J. Lipton, Turing machines that take advice, *ENSEIGN: L'Enseignement Mathematique Revue Internationale* **28** (1982), 191–209.

[KMP77] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt, Fast pattern matching in strings, *SIAM Journal on Computing* **6** (1977), 323–350.

[Lad75] R. E. Ladner, On the structure of polynomial time reducibility, *Journal of the ACM* **22** (1975), 155–171.

[Lan10] J. M. Landsberg, P versus NP and geometry, *Journal of Symbolic Computation* **45** (2010), 1359–1377.

[Lev73] L. Levin, Universal sequential search problems (Russian), *Problems of Information Transmission* **9** (1973), 115–116, A translation into English is found in the appendix to [Tra84].

[Mul12] K. D. Mulmuley, The *GCT* program toward the P vs. NP problem, *Communications of the ACM* **55** (2012), 98–107.

[NG14] R. Nederpelt and H. Geuvers, *Type Theory and Formal Proof: An Introduction*, Cambridge University Press, 2014.

[Pap95] C. H. Papadimitriou, *Computational Complexity*, Reprinted with corrections, Addison-Wesley, 1995.

[PMW93] C. Paulin-Mohring and B. Werner, Synthesis of ml programs in the system coq, *Journal of Symbolic Computation* **15** (1993), 607–640.

[PCG$^+$10] B. C. Pierce, C. Casinghino, M. Greenberg, V. Sjoberg, and B. Yorgey, *Software foundations*, Webpage: http://www. cis. upenn. edu/bcpierce/sf/current/index. html, 2010.

[PCW05] I. Poernomo, J. N. Crossley, and M. Wirsing, *Adapting Proofs-as-Programs: The* Curry–H*oward Protocol*, Springer Science+Business Media Incorporated, 2005.

[Pos47] E. L. Post, Recursive unsolvability of a problem of Thue, *The Journal of Symbolic Logic* **12** (1947), 1–11, Also in [Dav04].

[RR97] A. A. Razborov and S. Rudich, Natural proofs, *Journal of Computer and System Sciences* **55** (1997), 24–35.

[Sav70] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, *Journal of Computer and System Sciences* **4** (1970), 177–192.

[Sip83] M. Sipser, A complexity theoretic approach to randomness, in *Proceedings of the fifteenth annual* ACM *symposium on Theory of computing*, ACM, 1983, pp. 330–335.

[Sip92] M. Sipser, The history and status of the P versus NP question, in *Proceedings of the twenty-fourth annual* ACM *symposium on theory of computing*, ACM, 1992, pp. 603–618.

[Sma98] S. Smale, Mathematical problems for the next century, *The Mathematical Intelligencer* **20** (1998), 7–15.

[SU06] M. H. Sørensen and P. Urzyczyn, *Lectures on the Curry-Howard isomorphism*, *Studies in Logic and the Foundations of Mathematics* **149**, Elsevier, 2006.

[Sto87] L. STOCKMEYER, Classifying the computational complexity of problems, *The journal of symbolic logic* **52** (1987), 1–43.

[Str03] T. STRAHM, Theories with self-application and computational complexity, *Information and Computation* **185** (2003), 263–297.

[Tra84] B. A. TRAKHTENBROT, A survey of Russian approaches to perebor (brute-force searches) algorithms, *Annals of the History of Computing* **6** (1984), 384–400.

[Tur36] A. M. TURING, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* **42** (1936), 230–265, corrections, Ibid, vol. 43 (1937), pp. 544-546. Also in [Dav04].

[Wig06] A. WIGDERSON, *P*, *NP* and mathematics–a computational complexity perspective, in *Proceedings of the* 2006 I*nternational* C*ongress of* M*athematicians*, 2006.

[Wil16] R. R. WILLIAMS, Natural proofs versus derandomization, *SIAM Journal on Computing* **45** (2016), 497–529.