Lecture #13: What If...? More about Nondeterministic Computation

If $\mathcal{P} \neq \mathcal{NP}$ Then $\mathcal{NP}$-Intermediate Languages Exist

Recall that a language $L \subseteq \Sigma^\star$ is ***$\mathcal{NP}$-Intermediate*** if it satisfies the following properties.

(a) $L \in \mathcal{NP}$.

(b) $L \notin \mathcal{P}$.

(c) L is not $\mathcal{NP}$-complete.

If $\mathcal{P} = \mathcal{NP}$ then no $\mathcal{NP}$-intermediate languages can exist, because the above properties (a) and (b) would be contradictory. On the other hand, it has been shown by Ladner [2] that the assumption that $\mathcal{P} \neq \mathcal{NP}$ is sufficient to prove that these languages exist.

Unfortunately, the only languages that have been *proved* to be $\mathcal{NP}$-intermediate, assuming $\mathcal{P} \neq \mathcal{NP}$, are highly artificial (and complicated). A proof of the following claim can also be found in Section 3.3 of the text of Arora and Barak [1].

Claim 1. *If $\mathcal{P} \neq \mathcal{NP}$ then an $\mathcal{NP}$-Intermediate language exists.*

The proof of this claim is for interest only — CPSC 511 do not need to read this document in order to be successful in this course.

Recall the alphabet Σ_F and language $L_{\mathsf{F}} \subseteq \Sigma_F^\star$ introduced in Lecture #11 (on the Cook-Levin Theorem): L_{F} is the language of encodings of Boolean formulas and, as explained in that lecture, $L_{\mathsf{F}} \in \mathcal{P}$. Recall, as well, that $L_{\mathsf{FSAT}} \subseteq \Sigma_F^\star$ is the language of encodings of ***satisfiable*** Boolean formulas. As shown in Lecture #11, L_{FSAT} is $\mathcal{NP}$-complete.

Let

$$\Sigma_{F,\#} = \Sigma_F \cup \{\#\}.$$

Since $|\Sigma_F| = 16$ (as can be confirmed by consulting the notes for Lecture #11), $|\Sigma_{F,\#}| = 17$. For each total function $f : \mathbb{N} \to \mathbb{N}$, let

$$L_{\mathsf{FSAT},f} = \{\omega\#^{\left(|\omega|^{f(|\omega|)}\right)} \mid \omega \in L_{\mathsf{FSAT}}\} \subseteq \Sigma_{F,\#}^\star.^{1}$$

[1] To simplify the presentation, let us define 0^0 to be 0 here, and in the rest of this document.

Thus a string $\mu \in \Sigma_{F,\#}$ is in $L_{\textsf{FSAT},f}$ if and only if μ begins with a string $\omega \in L_{\textsf{F}}$ that encodes a satisfiable Boolean formula, and that continues and ends with $n^{f(n)}$ copies of #, where $n = |\omega|$.

Recall, as well, that the encodings of Turing machines described in Lecture #4 can be used to define an ***enumeration***

$$M_0, M_1, M_2, \dots$$

of all deterministic multi-tape Turing machines, such that every such Turing machine appears infinitely often in this listing.

The claim will be established by considering a ***recursively defined*** total function $H : \mathbb{N} \to \mathbb{N}$ that is defined as follows.

(a) $H(0) = H(1) = H(2) = 0.$

(b) For every integer n such that $n \geq 3$ consider the following set of ***properties*** of a non-negative integer i:

   i. $i < \log_2 \log_2 n.$
   ii. $M_i = (Q, \Sigma, \Gamma, \delta, q_0, q_{\textsf{accept}}, q_{\textsf{reject}})$ such that $|\Sigma| = 17$ — so that the language of M_i can be considered to be a subset of $\Sigma_{F,\#}^{\star}$.
   iii. For every string $\omega \in \Sigma_{F,\#}^{\star}$ with length at most $\log_2 n$ such that $\omega \in L_{\textsf{FSAT},H}$, M_i ***accepts*** ω after taking at most $i \cdot |\omega|^i$ steps.
   iv. For every string $\omega \in \Sigma_{F,\#}^{\star}$ with length at most $\log_2 n$ such that $\omega \notin L_{\textsf{FSAT},H}$, M_i ***rejects*** ω after taking at most $i \cdot |\omega|^i$ steps.

   If there exists a non-negative integer i satisfying properties (i)–(iv), above, then $H(n)$ is the ***smallest*** non-negative integer satisfying these properties. $H(n) = \lfloor \log_2 \log_2 n \rfloor$ if no non-negative integer that satisfies these properties exists.

This definition of H is certainly confusing — notice that the definition of the language $L_{\textsf{FSAT},H}$ depends on the function H, but the above definition of the function H also depends on the language $L_{\textsf{FSAT},H}$! Notice, though, that for $n \geq 3$ the definition of $H(n)$ only depends on the values of $H(k)$ for integers k such that $k \leq \log_2 n$. It turns out not only that H is a well-defined function but that the following is true as well.

Lemma 2. *Let $\Sigma_0 = \{1\}$. Then the function $f_H : \Sigma_F^{\star} \to \Sigma_0^{\star}$ such that*

$$f(\omega) = 1^{H(|\omega|)},$$

mapping every string $\omega \in \Sigma_F^{\star}$ to the unary representation of $H(|\omega|)$, is computable deterministically in polynomial time.

Note: The proof of Lemma 2 is long and complicated — and the proof of the main claim can be completed without understanding how this lemma is proved. The proof of this lemma — which is found in Appendix A — can be skipped or left until a later reading of this document.

Lemma 3. *$L_{\mathsf{FSAT},H} \in \mathcal{NP}$.*

Proof. Consider a verification algorithm that uses a new symbol, "♠", as the separator between encodings of instances of the language it should verify and encodings of certificates. The algorithm should begin by checking whether its input begins, either with a string "$\omega\#$" or with a string "ω♠", where $\omega \in \Sigma_F^\star$ — rejecting if is this is not the case. Note that, if the input string *does* have one of these forms then at most $|\omega| + 1$ are needed to confirm this.

The algorithm should next check whether $\omega \in L_{\mathsf{F}}$, rejecting if *this* is not the case. Since $L_{\mathsf{F}} \in \mathcal{P}$, as discussed in Lecture #11, this step can also be carried out, deterministically, using a number of steps that is at most polynomial in $|\omega|$.

The algorithm should then use ω to compute the string $\omega\#^{H(|\omega|)}$; it follows by Lemma 2 that this step can also be carried out, deterministically, using a number of steps that is at most polynomial in $|\omega|$. The input should be rejected if it does not begin with $\omega\#^{H(|\omega|)}$♠ — rejecting if this is not also true. Since the length of $\omega\#^{H(|\omega|)}$ is at most polynomial in $|\omega|$, this step can also be carried out using time at most polynomial in the length of ω.

The algorithm should then, effectively, ignore the copies of "#" in the input string and proceed as a verification algorithm for L_{FSAT} would in order to verify ω. That is, it should continue by confirming that the input ends with the encoding of a satisfying truth assignment for the Boolean formula encoded by ω — accepting if this is the case, and rejecting otherwise. Since the verification algorithm for L_{FSAT} would use a number of steps at most polynomial in the length of ω, it follows that the rest of the verification algorithm for $L_{\mathsf{FSAT},H}$ use at most this number of steps too — as needed to conclude that $L_{\mathsf{FSAT},H} \in \mathcal{NP}$. □

Lemma 4. *If $L_{\mathsf{FSAT},H} \in \mathcal{P}$ then there exists a constant c such that $H(n) \leq c$ for all $n \in \mathbb{N}$.*

Proof. Suppose that $L_{\mathsf{FSAT},H} \in \mathcal{P}$. Then there exists a deterministic Turing machine M that decides this language, using at most $d \cdot n^d$ steps when executed on any string with positive length n, for some positive integer constant d.

Since M has infinitely many encodings there is a number h such that $h > d$ and $M = M_h$.

The constant c in the claim can now be chosen as

$$\max\left(h, \max_{0 \leq i \leq 2^{2^h}} (H(i))\right).$$

Then it follows that if $n > 2^{2^h}$, so that $h < \log_2 \log_2 n$, h satisfies all the properties in part (b) of the recursive definition of H, so that $H(n) \leq h \leq c$. On the other hand, it is clear from the above definition of c that $H(n) \leq c$ if $0 \leq n \leq 2^{2^h}$ as well. Thus the constant c satisfies the condition given in the claim. □

Lemma 5. *If $\mathcal{P} \neq \mathcal{NP}$ then $L_{\mathsf{FSAT},H} \notin \mathcal{P}$.*

Proof. By contradiction. Suppose that $\mathcal{P} \neq \mathcal{NP}$ but $L_{\mathsf{FSAT},H} \in \mathcal{P}$.

Consider a function $f_{\Sigma_F^\star} \to \Sigma_{F,\#}^\star$ such that, for all $\omega \in \Sigma_F^\star$,

$$f(\omega) = \begin{cases} \omega\#^{|\omega|^{H(|\omega|)}} & \text{if } \omega \in L_{\mathsf{F}}, \\ \lambda & \text{if } \omega \notin L_{\mathsf{F}}. \end{cases}$$

Note that $\omega \in L_{\mathsf{FSAT}}$ if and only if $f(\omega) \in L_{\mathsf{FSAT},H}$.

However $L_{\mathsf{F}} \in \mathcal{P}$, and it follows by Lemma 2 that a unary representation of $H(|\omega|)$ is computable deterministically, from ω, in polynomial time. Furthermore, it follows by Lemma 4 that $H(|\omega|) \leq c$ for some constant c — so that the number of copies of "$\#$" to be appended to ω, in order to compute $f(\omega)$ when $\omega \in L_{\mathsf{F}}$, is also polynomial in the length of ω Thus the function f is, therefore, computable deterministically in polynomial time — and this a polynomial-time many-one reduction from L_{FSAT} to $L_{\mathsf{FSAT},H}$:

$$L_{\mathsf{FSAT}} \preceq_{\mathsf{P,\,M}} L_{\mathsf{FSAT},H}.$$

Since $L_{\mathsf{FSAT},H} \in \mathcal{P}$ (by assumption) it now follows that $L_{\mathsf{FSAT}} \in \mathcal{P}$ too. Since L_{FSAT} is $\mathcal{NP}$-hard it now follows that $\mathcal{P} = \mathcal{NP}$ ***contradicting*** the assumption that $\mathcal{P} \neq \mathcal{NP}$. The claim now follows. □

Lemma 6. *If $\mathcal{P} \notin \mathcal{NP}$ then*

$$\lim_{n \to +\infty} H(n) = +\infty.$$

Proof. Suppose that $\mathcal{P} \neq \mathcal{NP}$. Then Lemma 5 implies that $L_{\mathsf{FSAT},H} \notin \mathcal{P}$. Furthermore, it follows by the argument used to establish this lemma that either the above limit must not exist, at all, or it must by $+\infty$.

Now, if this claim is false (so that the limit is not $+\infty$) then there must exist a constant c such that $H(n) = c$ for infinitely many non-negative integers n.

However, the ***definition*** of H now implies that the deterministic Turing machine M_c correctly decides whether $\omega \in L_{\mathsf{FSAT},H}$ for *every* string ω with *any* positive length n: If the machine failed to do this for a string with positive length d then it would not be possible that $H(n) = c$ for $n > 2^{2^d}$.

Modifying M_c (if needed) to ensure that the machine correctly decides membership of λ in the language, a deterministic Turing machine that decides $L_{\mathsf{FSAT},H}$ in polynomial time is now

```
On input $\omega \in \Sigma_F^\star$
1. If $|\omega| \leq N$ then use the finite control to remember whether $\omega \in L_{\text{FSAT}}$. ***Accept*** if
   $\omega \in L_{\text{FSAT}}$ and ***reject***, otherwise.
2. Compute $f(\omega) \in \Sigma_{F,\#}^\star$.
3. If $f(\omega)$ does not have the form $\mu\#^m$ for a string $\mu \in L_{\text{F}}$, and a natural number $m$,
   then ***reject***.
4. If $m \neq k^{H(k)}$ where $k = |\mu|$, for $m$ and $\mu$ as above, then ***reject***.
5. Apply this algorithm to recursively decide whether $\mu \in L_{\text{FSAT}}$. ***Accept*** if $\mu \in L_{\text{FSAT}}$
   and ***reject*** otherwise.
```

Figure 1: Algorithm Used to Obtain a Contradiction

obtained. It follows that $L_{\text{FSAT},H} \in \mathcal{P}$, giving us a ***contradiction***, as needed to establish the claim. □

Lemma 7. *If $\mathcal{P} \neq \mathcal{NP}$ then $L_{\textit{FSAT},H}$ is not $\mathcal{NP}$-complete.*

Proof. Suppose, to obtain a contradiction, that $\mathcal{P} \neq \mathcal{NP}$ and that $L_{\text{FSAT},H}$ is $\mathcal{NP}$-complete.

Then $L_{\text{FSAT}} \preceq_{\text{P, M}} L_{\text{FSAT},H}$, since $L_{\text{FSAT}} \in \mathcal{NP}$, so there exists a function $f : \Sigma_F^\star \to \Sigma_{F,\#}^\star$ such that

- $\omega \in L_{\text{FSAT}}$ if and only if $f(\omega) \in L_{\text{FSAT},H}$ for every string $\omega \in \Sigma_F^\star$, and
- there is a deterministic Turing machine computing f in polynomial time. In particular, there exists a constant c such that the number of steps used by this machine, given an input with positive length n, is at most $c \cdot n^c$.

Since $\mathcal{P} \neq \mathcal{NP}$, it follows by Lemma 6 that there are only finitely many numbers $n \in \mathbb{N}$ such that $H(n) \leq 2c$ — so that there is a natural number N such that $H(n) \geq 2c$ whenever $n \in \mathbb{N}$ and $n \geq N$. We may assume $N \geq 9$, so that $2\sqrt{N} < N$.

A deterministic Turing machine can use a (greatly expanded) finite control to remember whether $\omega \in L_{\text{FSAT}}$ for every string $\omega \in \Sigma_F^\star$ such that $|\omega| < N$ — so that it will correctly report whether $\omega \in L_{\text{FSAT}}$ on input ω, for every such input string.

Now consider the algorithm shown in Figure 1, above. If this recursive algorithm — which decides membership of its input in L_{FSAT} — rejects ω at any of lines 1, 3 or 4, then its output is correct. This is also true if it accepts ω at line 1. Otherwise step 5 is reached and either

(a) $|\mu| < N$, so that the finite control will be used to decide whether $\mu \in L_{\text{FSAT}}$, and this execution of the algorithm will terminate with the correct output provided, or

(b) $|\mu| \geq N$. In this case $|\mu| \leq 2\sqrt{|\omega|} < |\omega|$ — for if $|\mu| \geq N$ and $|\mu| > 2\sqrt{|\omega|}$ then $|f(\omega)| > |\mu|^{H(|\mu|)} \geq c \cdot n^c$ for $n = |\omega|$, contradicting the fact that f can be computed using an efficient deterministic Turing machine, as described above.

It is now easily proved, by induction on the length of the input string, that this algorithm correctly decides membership in L_{FSAT}.

It is also easily proved (using the above bounds on the length of μ if the last step is reached) that the number of steps used by this recursive algorithm is at most polynomial in the length of the input string in the worst case. This implies that $L_{\mathsf{FSAT}} \in \mathcal{P}$.

Since L_{FSAT} is $\mathcal{NP}$-hard, it follows that $\mathcal{P} = \mathcal{NP}$ — giving us the ***contradiction*** needed to establish the claim. $\square$

Proof of Claim 1. Claim 1 now follows by Lemmas 3, 5, and 7. $\square$

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.

A Proof of Lemma 2

As its title indicates, this appendix includes a proof of Lemma 2 (as stated on page 2).

Consider algorithms that make use of the following global data.

- $\mathtt{H}$ is an integer array with length $1 + \lfloor \log_2 n \rfloor$, for $n = |\omega|$. The entries of this array will initially be undefined but, for every integer m such that $0 \leq m \leq \lfloor \log_2 n \rfloor$, $\mathtt{H}[m]$ will eventually be set to be $H(m)$.
- A partial function φ from the set of strings in $\Sigma_{F,\#}^{\star}$ to $\{\mathtt{T}, \mathtt{F}\}$. This partial function will initially be undefined everywhere — and it will never be defined on more than a finite set of input strings. Eventually, $\varphi(\mu)$ will be set to be $\mathtt{T}$ if $\mu \in L_{\mathsf{FSAT},H}$ and to be $\mathtt{F}$ if $\mu \notin L_{\mathsf{FSAT},H}$, for every string $\mu \in \Sigma_{F,\#}^{\star}$ such that $|\mu| \leq \log_2 n$.

These data structures can be encoded as strings (for computations by Turing machines) in a variety of ways. For example, the array $\mathtt{H}$ could be encoded as a set of ordered pairs, each consisting of the decimal representation of an integer m such that $0 \leq m \leq \lfloor \log_2 n \rfloor$ and the decimal representation of $\mathtt{H}[m]$, once this value has been defined. Similarly, φ could be encoded as a set of ordered pairs, each consisting of a string $\mu \in \Sigma_{F,\#}^{\star}$ with length at most $\lfloor \log_2 n \rfloor$, and the value $\varphi(\mu) \in \{\mathtt{T}, \mathtt{F}\}$, when this value is defined.

A string encoding H, in this way, would certainly have length at most polynomial in $n = |\omega|$. Indeed, the encoding of an ordered pair storing the decimal representations of an integer m and of $H(m)$ would have length in $O(\log_2 m)$, so that the length of the encoding of H would have length in $O(\log_2 n \log_2 \log_2 n)$. Furthermore, if a Turing machine with two-way infinite tapes is used (or simulated) then, for each integer m such that $0 \leq m \leq \lfloor \log_2 n \rfloor$, the cost to either use this encoding to *obtain* the value of $H(m)$, or to modify the string to *set* the value of $H(m)$, can be shown to be in $O(\log_2 m \log_2 \log_2 m)$.

Since the number of strings in $\Sigma_{F,\#}^{\star}$ with length at most $\lfloor \log_2 n \rfloor$ is

$$\begin{aligned}
\sum_{i=0}^{\lfloor \log_2 n \rfloor} |\Sigma_{F,\#}|^i &= \sum_{i=0}^{\lfloor \log_2 n \rfloor} 17^i \\
&= \frac{17^{\lfloor \log_2 n \rfloor + 1} - 1}{17 - 1} \\
&< \tfrac{17}{16} \cdot 17^{\log_2 n} \\
&= \tfrac{17}{16} \cdot n^{\log_2 17},
\end{aligned}$$

it can be argued that the length of an encoding of the partial function φ will be at most polynomial in n. Furthermore, if the above ordered pairs are sorted (in the encoding) by nondecreasing order of the length of the string that is the first element of the ordered pair, then it can be argued that the distance one must move to, from the beginning of the encoding, to determine whether $\varphi(\mu)$ is defined is at most singly exponential in the length of μ — that is, in $O(c^{|\mu|})$ for some positive constant c. Assuming the use of a multi-tape Turing machine (or a simulation of one) it can be argued that the cost to change the encoding, to *set* the value of $\varphi(\mu)$, is in $O(c^{|\mu|})$ for a positive constant c as well.

In any case it will be assumed that this or some other reasonable encoding of both H and φ is used, so that the operations

- checking whether $\mathtt{H}[m]$ is defined for an integer i such that $0 \leq m \leq \lfloor \log_2 n \rfloor$, returning this value if it is;
- setting the value of $\mathtt{H}[m]$ to be a given value, if this is not defined already,
- checking whether φ is defined at a given string $\mu \in \Sigma_{F,\#}^{\star}$ with length at most $\log_2 n$, and returning the value of φ at this string if it is defined,
- setting the value of φ at a given string $\mu \in \Sigma_{F,\#}^{\star}$ with length at most $\log_2 n$ to be a given value if this is not already defined

```
integer setH (integer m) {
 1. if (H[m] is defined) {
 2.   return H[m]
 3. } else if $(\mathtt{m} \le 2)$ {
 4.   H[m] := $0$
 5.   return $0$
    } else {
 6.   int i := $0$
 7.   while $(\mathtt{i} < \lfloor \log_2 \log_2 \mathtt{m} \rfloor)$ {
 8.     if (i encodes a Turing machine $M_{\mathtt{i}}$ whose input alphabet $\Sigma$
                                  has size $17$ — so $\Sigma = \Sigma_{F,\#}$) {
 9.       boolean passed := true
10.       for each (string $\mu \in \Sigma_{F,\#}^{\star}$ such that $|\mu| \le \lfloor \log_2 \mathtt{m} \rfloor$) {
11.         int t := i $\cdot$ $|\mu|^{\mathtt{i}}$
12.         if $(\mathtt{set}\varphi(\mu))$ {
13.           Simulate the first t steps of the execution of $M_{\mathtt{i}}$ on input $\mu$. Set
              passed to be false if this machine either fails to halt after this num-
              ber of steps, or rejects $\mu$ after at most this number of steps have
              been used.
            } else {
14.           Simulate the first t steps of the execution of $M_{\mathtt{i}}$ on input $\mu$. Set
              passed to be false if this machine either fails to halt after this num-
              ber of steps, or accepts $\mu$ after at most this number of steps have
              been used.
            }
          }
```

Figure 2: Algorithm to Evaluate H on Small Integers

can each be carried out using times that are as described for these functions, above.

Now consider the functions shown in Figures 2–4 on pages 8–10.

It is sufficient to prove the following in order to establish that the function H is well-defined and computable.

Lemma 8. *Let m be an integer such that $0 \le m \le \lfloor \log_2 n \rfloor$.*

(a) If the algorithm setH *is executed with the integer m as input then this execution of the*

```
15.         if ( passed ) {
16.           H[m] := i
17.           return i
            } else {
18.           i := i + 1
            }
          } else {
19.         i := i + 1
          }
        }
20.     H[m] := $\lfloor \log_2 \log_2 \mathtt{m} \rfloor$
21.     return $\lfloor \log_2 \log_2 \mathtt{m} \rfloor$
      }
}
```

Figure 3: Algorithm to Evaluate H on Small Integers, Concluded

algorithm eventually ends. On termination of this execution of the algorithm `H[`m`]` *has the value* $H(m)$*, and this value is returned as output.*

(b) *If the algorithm* `set`φ *is executed with a string* $\mu \in \Sigma_{F,\#}^{\star}$ *with length* m *as input, then this execution of the algorithm eventually ends. On termination of this execution of the algorithm* $\varphi(\mu)$ *is equal to* `true` *if* $\mu \in L_{\mathit{FSAT},H}$ *and is equal to* `false` *otherwise. Furthermore, this value* $\varphi(\mu)$ *has been returned as output.*

Proof of Lemma 8. The result can be proved by induction on m, using the strong form of mathematical induction and considering the cases $0 \leq m \leq 2$ in the basis.

Basis: One can see by inspection of the code that, for an integer m such that $0 \leq m \leq \lfloor \log_2 n \rfloor$, the array entry `H[`$m$`]` is only set when the algorithm `setH` is executed with the same integer m as input, and that, for a string $\mu \in \Sigma_{F,\#}^{\star}$ with length m, the value of $\varphi(\mu)$ is only defined when the algorithm `set`φ is executed with the same string μ as input. It follows that `H[`m`]` is undefined at the beginning of the first execution of the `setH` algorithm with input m and that $\varphi(\mu)$ is undefined at the beginning of the first execution of the `set`φ algorithm with input μ as well.

Now if $0 \leq m \leq 2$ and the algorithm `setH` is executed with input m then the test at line 1 fails, the test at line 3 passes, and steps 4 and 5 are executed so that `H[`m`]` is correctly set to $H(m) = 0$ and this value is returned, as required. On all subsequent applications with

```
boolean set$\varphi$ ($\mu$: $\Sigma^\star_{F,\#}$) {
 1.  if ($\varphi(\mu)$ is defined) {
 2.    return $\varphi(\mu)$
     } else {
 3.    Set $\nu$ to be the longest prefix of $\mu$ that is in $\Sigma^\star_F$
 4.    if ($\nu \in L_F$) {
 5.      int h := setH[$|\nu|$]
 6.      if $\left(\mu == \nu\#^{\left(|\nu|^{\mathtt{h}}\right)}\right)$ {
           The following test should be carried out by using exhaustive search —
           checking partial truth assignments on the set of variables included in the
           formula encoded by $\nu$ until either a satisfying truth assignment has been
           found for $\nu$ or all such partial truth assignments have been checked.
 7.        if ($\nu \in L_{\mathsf{FSAT}}$) {
 8.          Set $\varphi(\mu)$ to be true
           } else {
 9.          Set $\varphi(\mu)$ to be false
           }
         } else {
10.        Set $\varphi(\mu)$ to be false
         }
       } else {
11.      Set $\varphi(\mu)$ to be false
       }
12.    return $\varphi(\mu)$
     }
}
```

Figure 4: Algorithm to Decide Membership in $L_{\mathsf{FSAT},H}$ on Short Strings

input m the test at line 1 is passed and $H(m) = 0$ is correctly returned, without H[m] being changed, as desired.

Similarly, if the setφ algorithm is executed with a string $\mu \in \Sigma^\star_{F,\#}$ with length m, where $0 \leq m \leq 2$, and this is the first execution of this algorithm, then $\varphi(\mu)$ is initially undefined and the test at line 1 fails, so that the execution continues with the step at line 3. If ν is the longest prefix of μ in $\Sigma^\star_F$ then $|\mu| \leq |\mu| \leq 2$, so that h is set to be $H(|\nu|) = 0$ at line 5. Now, if $\mu = \nu\#^{|\nu|^h}$

then either $|\nu| = |\mu| = 0$ or $|\mu > 0$ and $|\nu| = |\mu| - 1 \leq 1$. Consequently either the test at line 6 fails or it is passes and the test at lne 7 fails. In either case $\varphi(\mu)$ is set to be `false` and this value is returned, as required to satisfy the claim in this case. On subsequent applications of the algorithm with input μ the test at line 1 is passed and $\varphi(\mu)$ is returned at line 2, as required.

For the inductive step one should first consider an execution of the `setH` algorithm with input m, where $3 \leq m \leq \lfloor \log_2 n \rfloor$. Once again, if this is the first execution of the algorithm with this input then the test at line 1 fails. This time the test at line 3 fails as well and the steps at lines 6–21 are executed. Now, a comparison of these steps with the definition of the function H suffices to confirm that if the call to `set`φ returns the expected result (`true` if $\mu \in L_{\mathsf{FSAT},H}$ and `false` otherwise) then `H[`m`]` is correctly set to be $H(m)$, when either the step at line 16 or the step at line 20. Now, since each string μ considered at line 12 has length between 0 and $\lfloor \log_2 m \rfloor \leq m - 1$, it follows by the inductive hypothesis that these calls each return the required value, as needed to establish the claim in this case.

Once again, when considering a later execution of this algorithm on the same input value m, it suffices to note that the test at line 1 will pass and that the required value will be returned as output, establishing the claim for this case as well.

A similar argument can be used to establish that if $3 \leq m \leq \lfloor \log_2 n \rfloor$ and the `set`φ algorithm is executed, with a string $\mu \in \Sigma^{\star}_{F,\#}$ with length m as input, then the execution terminates with $\varphi(\mu)$ set to be `true` if $\mu \in L_{\mathsf{FSAT},H}$ and `false` otherwise, and with this value returned as output: If this is the first execution of the algorithm with μ as input then the test at line 1 fails and the steps at lines 3–12 are executed. Comparing these steps to the definition of the language $L_{\mathsf{FSAT},H}$ one can see that $\varphi(\mu)$ is correctly defined and the desired output is returned, as long as the execution of `setH[`$|\nu|$`]` at line 5 terminates, returning $H(|\nu|)$ as output. Now, $0 \leq |\nu| \leq |\mu| = m$; this is, therefore, a consequence of the inductive hypothesis if $|\nu| < |\mu|$, and it follows by the proof of part (a) (in the inductive step), above, if $|\nu| = |\mu|$. This establishes the desired result for this case.

It remains only to consider later executions of the `set`φ algorithm on the same input string μ. Once again, this is easily established by noting that the test at line 1 is passed and the required output is returned when the step at line 2 is executed.

The lemma now follows by induction on m. ☐

Bounding the time used to execute these algorithms is complicated by the fact that they are given as pseudocode that is not entirely complete or detailed. For example, steps 13 and 14 of the `setH` involve the use of a universal Turing machine to carry out a set number of initial steps of an algorithm, while step 7 of the `set`φ algorithm includes an exhaustive search of all partial truth assignments including truth values for the Boolean variables included in a given Boolean formula — so that the executions of these steps certainly require more than constant time.

With that noted, let $T_H : \mathbb{N} \to \mathbb{N}$ such that, for every integer m such that $0 \leq m \leq \log_2 n$,

$H(m)$ can be computed and $\mathtt{H}[m]$ can be set, by an execution of the `setH` algorithm using the execution of at most $T_H(m)$ steps. Increasing the value of this function slightly (by replacing it with $\widehat{T}_H(m) = \max(T_H(i) : 0 \leq i \leq m)$, if necessary) it can be assumed that this is a non-decreasing function of m.

Similarly, let $T_\varphi : \mathbb{N} \to \mathbb{N}$ such that, for every integer m such that $0 \leq m \leq \log_2 n$, and every string $\mu \in \Sigma^\star_{\mathsf{SAT},H}$ with length at most m, $\varphi(\nu)$ can be set, using an execution of the $\mathtt{set}\varphi$ algorithm with at most $T_\varphi(m)$ steps. This can be assumed to be a non-decreasing function of m as well.

Consider an execution of the `setH` algorithm with input m where $0 \leq m \leq 2$. If this is the first execution of the algorithm with this input then the test at line 1 fails. The test at line 3 is then checked and passed, and the execution of the algorithm ends after the steps at lines 4 and 5 are executed: Four steps are executed in total. On the other hand, if this is not the first execution of the algorithm with this input then the test at line 1 is passed and the execution of the algorithm ends after the execution of the step at line 2, so that two steps are executed. We may therefore set $T_H(m)$ to be 4 when $0 \leq m \leq 2$.

Now consider an execution of this algorithm on input m where $m \geq 3$. If this is the first execution of the algorithm with this input then, as above, the test at line 1 fails. The test at line 3 is checked and also fails, so that the execution of the algorithm continues with the steps at lines 6–21. Consider, first, the body of the inner loop (at lines 11–14). During this execution the string μ has length at most $\log_2 \mathtt{m}$, and $\mathtt{i}$ is an integer such that $0 \leq \mathtt{i} \leq \log_2 \log_2 \mathtt{m}$. Thus

$$\begin{aligned} \mathtt{t} &= \mathtt{i} \cdot |\mu|^{\mathtt{i}} \\ &= (\log_2 \log_2 \mathtt{m}) \cdot (\log_2 \mathtt{m})^{\log_2 \log_2 \mathtt{m}} \end{aligned}$$

While this grows asymptotically more slowly than $\mathtt{m}^\varepsilon$ for any positive real constant ε, it may exceed this bound when $\mathtt{m}$ is small. It can be checked numerically that $\mathtt{t} \leq \mathtt{m}^{3/2}$ for every integer $\mathtt{m} \geq 2$, and this bound will be sufficient to establish the claim here — for, with care, it can be shown that the cost to compute a unary representation of $\mathtt{t}$ at line 11 is at most linear in $\mathtt{t}$. The cost to execute the step at line 12 is bounded by the cost to compute $\mathtt{set}_\varphi(\mu)$, which is at most $T_\varphi(\lfloor \log_2 \mathtt{m} \rfloor)$.

Either the step at line 13 of the step at line 14 is executed. The bounds given at the end of Lecture #4 — and the supplement for these notes, which describes a better emulation — can be used to establish that each of these steps can be carried out using $O(T(\log T)(\log \mathtt{m}))$ moves — and this is certain in $O(\mathtt{m}^2)$.

Thus each execution of the body of the loop at lines 11–14 can be carried out using at most $T_\varphi(\lfloor \log_2 \mathtt{m} \rfloor) + cm^2$, for some positive constant c.

It follows by the above analysis (replacing n with $\mathtt{m}$) that the number of strings $\mu \in \Sigma^\star_{F,\#}$ with length at most $\lfloor \log_2 \mathtt{m} \rfloor$ is at most $\frac{17}{16} \cdot \mathtt{m}^{\log_2 17}$. One can iterate over the set of all such strings using a number of steps at most linear in the number of these, and this can be used that the number of steps used for an execution of the loop at lines 10–14 is at most

$$\tfrac{17}{16}\mathtt{m}^{\log_2 17} T_\varphi(\lfloor \log_2 \mathtt{m} \rfloor) + \widehat{c}\,\mathtt{m}^{2+\log_2 17}$$

for a positive constant $\widehat{c}$.

Consider, next, the body of the loop at lines 9–18. As suggested above, accessing and modifying the array can certainly be carried out using time at most linear in $\mathtt{m}$, so that the cost of an execution of the body of this loop is dominated by the cost of an execution of the inner `for` loop and, indeed is less than the expression given above if the value for the constant $\widehat{c}$ is slightly increased.

A binary or decimal counter can have value from 0 to $\lfloor \log_2 \log_2 \mathtt{m} \rfloor$ using a number of steps that is most linear in $\log_2 \log_2 \mathtt{m}$ — and a similar argument can be used to establish that (once $\lfloor \log_2 \log_2 \mathtt{m} \rfloor$ has been computed and available) the total cost to compare each counter value with $\lfloor \log_2 \log_2 \mathtt{m} \rfloor$ is in $O(\log_2 \log_2 \mathtt{m})$ as well. Thus this bounds the total cost of all executions of the loop test at line 7 and, since there are at most $\lfloor \log_2 \log_2 \mathtt{m} \rfloor$ executions of the loop body, it now follows that the total cost for the execution of the `while` loop at lines 7–19 is at most

$$\tfrac{17}{16}\mathtt{m}^{\log_2 17} \log_2 \log_2 \mathtt{m} \cdot T_\varphi(\lfloor \log_2 \mathtt{m} \rfloor) + \widehat{c}\,\mathtt{m}^{2+\log_2 17} \log_2 \log_2 \mathtt{m}.$$

Considering the remaining steps — whose cost is certainly dominated by the cost of of the execution of the loop, and which do not include additional uses of the method $\mathtt{set}_\varphi$, — we may now conclude that

$$T_H(\mathtt{m}) \le \begin{cases} 4 & \text{if } 0 \le \mathtt{m} \le 2, \\ \tfrac{17}{16}\mathtt{m}^{\log_2 17} \log_2 \log_2 \mathtt{m} \cdot T_\varphi(\lfloor \log_2 \mathtt{m} \rfloor) & \\ \qquad\qquad + \widehat{c}\,\mathtt{m}^{2+\log_2 17} \log_2 \log_2 \mathtt{m} & \text{if } \mathtt{m} \ge 3. \end{cases} \tag{1}$$

Consider, now, the execution of the $\mathtt{set}_\varphi$ method on an input string $\mu \in \Sigma_{F,\#}^\star$ with length ℓ. As noted above, the cost to check whether $\varphi(\mu)$ is defined, and return its value it is, is at most linear in $c^{|\mu|} = c^\ell$ for some positive constant c. It remains only to consider the case that $\varphi(\mu)$ is not defined, so that the test at line 1 fails and the steps at lines 3–12 are executed.

The step at line 3 can certainly be carried out using $O(|\mu|) = O(\ell)$ steps of a deterministic Turing machine. As discussed in Lecture #11, membership in L_{F} can be decided deterministically in quadratic time. Thus, since $|\nu| \le |\mu| = \ell$, the test at line 4 can be checked at cost $O(\ell^2)$, and the operation can be completed (by reaching and executing the steps at lines 11 and 12 using a number of steps that is at most linear in c^ℓ, once again.

It remains only to consider the case that $\nu \in L_F$, so that $|\nu| \ge 2$ and the steps at lines 5–10 and 12 are reached and executed. Since T_H is a non-decreasing function, and $|\nu| \le |\mu| = \ell$, the step at line 5 can be carried out using at most $T_H(\ell)$ steps.

The test at line 6 can be carried out by checking first whether $\mu = \nu^j$ for some positive integer j — rejecting if this is not the case — and repeatedly dividing by $|\nu|$ to confirm that j is a power of $|\nu|$ (rejecting it is not) and comparing $\log_{|\nu|} j$ to h otherwise. This can be carried out (using a multiple tape Turing machine) using $O(|\nu|) = O(\ell)$ steps. If this test fails then the steps at lines 10 and 12 are executed, so the rest of the computation is carried out using $O(c^\ell)$ steps. It remains only to consider the case that the test at line 6 passes, so that the steps at lines 7–9 and 12 are reached and executed.

Now, as noted in the description of the algorithm, the test at line 7 can be carried out by exhaustive search — checking all partial truth assignments on the variables in the formula encoded by ν, until either a satisfying truth assignment has been found or all truth assignments have been checked. There are certainly at most $2^\nu \leq 2^\ell$ truth assignments to check and — again, using a method described in Lecture #8 — the value for the formula under any given truth assignment can be checked using a number of steps at most linear in $|\nu|^2 \leq \ell^2$. Thus the cost of the test at line 7 is at most linear in $\ell^2 \cdot 2^\ell$. Increasing the value of the constant c if necessary (to ensure that $c > 2$) we may conclude that this step can be carried out using $O(c^\ell)$ steps as well. The remaining steps to be carried out can be completed at this cost too.

It therefore follows that

$$T_\varphi(\ell) \leq T_H(\ell) + c_1 \cdot c^\ell \tag{2}$$

for positive constants c_1 and c (where $c > 2$), for every non-negative integer ℓ.

Now — choosing multiplicative constants $\widetilde{c}$ and $\overline{c}$ to be large enough to establish the claimed bounds when $\mathtt{m}$ and ℓ are small — the equations at lines (1) and (2) can be used to show (by induction on $\mathtt{m} + \ell$) that

$$T_H(\mathtt{m}) \leq \widetilde{c}\mathtt{m}^{5+\max(2,c)}$$

and

$$T_\varphi(\ell) \leq \overline{c} \cdot c^\ell$$

for all integers $\mathtt{m}$ and ℓ. It now follows that the function $f_H \Sigma_F^\star \rightarrow \Sigma_0^\star$ such that

$$f(\omega) = 1^{H(|\omega|)}$$

for every string $\omega \in \Sigma_F^\star$ is computable deterministically in polynomial time, using the method `setH` — establishing Lemma 2.