CSE 431 Theory of Computation **Spring 2014**

Lecture 16: May 22

Lecturer: James R. Lee *Scribe: Riley Klingler*

16.1 Savitch's Theorem

Theorem 16.1 *For any $f(n) \geq n$,*

$$NSPACE(f(n)) \subseteq SPACE(f(n)^2)$$

Proof: Suppose T is a non-deterministic Turing machine which requires at most $f(n)$ space on any input for some $f \geq n$. We will describe a deterministic Turing machine which requires $O(f(n)^2)$ space and has the same behavior as T. We first define the procedure CANYIELD, which given two configurations c_1 and c_2 of T and an integer t, accepts if there is a sequence of at most t which T can take starting at c_1 and ending at c_2.

```
CANYIELD($c_1, c_2, t$):
    **If** $t = 0$:
        Accept if $c_1 = c_2$.
    **Else if** $t = 1$:
        Accept if $T$ can take a single step from $c_1$ to $c_2$.
    **Else**:
        **For** all possible configurations $c_m$ of $T$:
            Accept if CANYIELD($c_1, c_m, \lfloor t/2 \rfloor$) and CANYIELD($c_m, c_2, \lceil t/2 \rceil$) both accept.
    Reject
```

Without loss of generality, we can assume that before T accepts, it writes 0's on each cell of the tape it used and moves the tapehead to the far left of the tape. Call this configuration c_{acc}. Let c be the the number of states of T plus the number of symbols in the tape alphabet of T. Then there are at most $c^{f(n)}$ valid configurations of T (because each configuration is at most $f(n)$ long and each thing in the configuration is either a member of the tape alphabet or a state of T). This means T takes at most $c^{f(n)}$ steps on any input on which it terminates, because otherwise it must repeat a configuration and run forever.

Let c_{start} be the start configuration of T on a given input w. Then T accepts w if and only if

$$\text{CANYIELD}(c_{start}, c_{acc}, c^{f(n)})$$

accepts. In addition, we can see that this will take at most $O(f(n)^2)$ space. This is because each call to CANYIELD requires $O(f(n))$ space to store the encoded c_m it chooses plus space for its recursive calls (it will make multiple recursive calls, but only one at any time, so the space for those can be reused). Because we halve t each time, there will be $\log(c^{f(n)}) = O(f(n))$ recursive calls, so the total space need is $O(f(n))O(f(n)) = O(f(n)^2)$.

If $f(n)$ is computable with $O(f(n)^2)$ space, then we can compute $c^{f(n)}$ before our first call to CANYIELD. Otherwise, we can add a size parameter s to CANYIELD, and instead of iterating over all configurations of T, only iterate over configurations which use at most s space. Define $c_{acc,s}$ to be the configuration of T that is the accept state followed by s zeros. Then we can model T using the following algorithm:

```
**For** $i = 1, 2, 3 \ldots$
    **For** each configuration $c_G$ with length less than $i + 1$:
        **If** CANYIELD($c_1, c_G, c^{i+1}, i + 1$) accepts:
            Accept if $c_G = c_{acc,i+1}$
            Increment $i$ and loop.
        Reject
```

This algorithm accepts if at any time CANYIELD tells us we can reach an accept state. In addition, it rejects if we reach an i such that T cannot yield any configuration of length $i + 1$. As T never uses more than $f(n)$ space, we will thus reject once $i = f(n) + 1$, which can only happen when T never reaches an accept state of length $f(n)$, so we know T rejects this input.

We have proven that whenever $f(n) \geq n$, every non-deterministic Turing machine T which requires at most $f(n)$ space on input of length n can be simulated by a deterministic Turing machine which requires $O(f(n))$ space, so we have shown $PSPACE(f(n)) \subseteq SPACE(f(n)^2)$.

■

An important corollary to this is that $PSPACE$ (the set of all languages which can be decided using a Turing machine with polynomial space) is equal to $NPSPACE$ (the set of all languages which can be decided using a non-deterministic Turing machine with polynomial space).

16.2 Complexity Hierarchy

Now that we have seen $PSPACE = NPSPACE$, we might wonder how they relate to other complexity classes.

Theorem 16.2

$$NP \subseteq PSPACE$$

Proof: We have previously seen SAT is NP-complete. We also know SAT $\in PSPACE$ because to implement the brute force solution we only need to store the current possible assignment to the variables, which takes only linear space. Then, given $L \in NP$, we can reduce it to SAT with only a polynomial blowup in size, then solve the SAT instance with polynomial space, allowing us to solve L with polynomial space, so $L \in PSPACE$. As this is true for all $L \in NP$, $NP \subseteq PSPACE$. ■

Definition 16.3 *Let*

$$EXPTIME = \bigcup_k TIME(2^{n^k}).$$

Theorem 16.4

$$PSPACE \subseteq EXPTIME$$

Proof: As discussed in the proof of Savitch's theorem, A decider which uses $f(n)$ space on input n can take at most $c^{f(n)}$ steps before halting. Therefore, a Turing machine which uses polynomial space must run in exponential time. ■

These theorems give us the following hierarchy of complexity classes:

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME.$$

Any of the three subset relations above could in fact be equals. However, the time hierarchy theorem (discussed on Tuesday) tells us that at least one must be proper, and it is widely expected that all three inclusions are proper.