COMPSCI 501: Formal Language Theory

Lecture 29: Hierarchy Theorems

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

5 April 2019

Recap and Preview

L $\subseteq$ NL = coNL $\subseteq$ **P** $\subseteq$ NP $\subseteq$ **PSPACE** $\subseteq$ EXPTIME

- Refine this hierarchy
- Separation between classes
- Separate by space and time complexity *within* PSPACE and P
- Clearly, with more space or time, a TM can decide more languages
- Can it decide **strictly** more?

Space constructible functions

Informally:
f is **space constructible** if $f(n)$ is computable in space $O(f(n))$.

Def. A function $f : \mathbb{N} \to N$ that is at least $O(\log n)$ is *space constructible* if there is a $O(f(n))$ space TM that computes $f(n)$ from 1^n (string of n 1s).

"Usual" functions (polynomial, log) are space constructible

n^2: convert n to binary ($\log n$ space), multiply with itself – space proportional to length, $O(\log n)$

$\log n$: convert n to binary ($\log n$ space), count bits (likewise)

Space Hierarchy Theorem

We're given a space bound $O(f(n))$

Some languages can be decided in $O(f(n))$ space

If we now have asymptotically less space, $o(f(n))$, can we decide strictly less ?

Recall: $g(n)$ is $o(f(n))$ if $\lim_{n\to\infty} g(n)/f(n) = 0$

Space Hierarchy Theorem: For *any* space constructible function $f : \mathbb{N} \to \mathbb{N}$, there exists a language that is decidable in $O(f(n))$ space but not $o(f(n))$ space.

Proof: Contradiction using Diagonalization

Recall idea of diagonalization proof for A_{TM}:

If you have a decider, you can simulate it, then answer the opposite!

But simulation adds some space/time overhead

Proof idea: difference between $o(f(n))$ and $O(f(n))$ enough for simulation.

In space $O(f(n))$, can simulate anything that's in $o(f(n))$ and then do the opposite $\implies$ we have a separator!

We'll describe the language A in terms of a decider D for it (a TM).

D must be different from any TM M that runs in $o(f(n))$ space.
Diagonalization: if input is $\langle M\rangle$, do opposite of M
(if input does not represent TM, don't care $\implies$ *reject*)

Proof of Space Hierarchy Theorem

Two technical details:

- Input M may not be a decider $\implies$ must avoid nontermination
  count steps while simulating M, *reject* if more than $2^{f(n)}$
  (we know $o(f(n))$ space runs in $2^{o(f(n))}$ time)
- $o(n)$ condition for M is true only for $n \geq n_0$.
  run M on w for any input $\langle M\rangle 10^k$
  (input will eventually be long enough for M to complete)

Space analysis

Must represent any M using D' symbols: constant factor, depending on M

Step counter only adds $\log f(n)$ space. $\implies$ total $O(f(n))$ space.

Corollary: Space complexity classes

For any functions $f_1, f_2 : \mathbb{N} \to \mathbb{N}$, with $f_1(n) = o(f_2(n))$, and f_2 space constructible, we have SPACE($f_1(n)$) $\subsetneq$ SPACE($f_2(n)$).

In particular, we have SPACE(n^{c_1}) $\subsetneq$ SPACE(n^{c_2}).

- for all naturals c_i (since n^c is space constructible)
- can prove n^c is space constructible for rationals
- between any two reals we have two rationals
  $\Longrightarrow$ different space complexity for any *real* c_1, c_2

NL $\subsetneq$ PSPACE

By Savitch's theorem, NL $\subseteq$ SPACE($\log^2 n$) $\subsetneq$ SPACE(n)

In particular, *TQBF* $\notin$ NL, because it's PSPACE-complete *with respect to log space reducibility*

Time constructible functions

Informally:
t is **time constructible** if $t(n)$ is computable in time $O(t(n))$.

Def. A function $t : \mathbb{N} \to N$ that is at least $O(n \log n)$ is *time constructible* if $t(n)$ is computable in time $O(t(n))$ from 1^n.

Why $\geq O(n \log n)$? We've seen binary conversion needs this time.

Time Hierarchy Theorem

Time Hierarchy Theorem: For *any* time constructible function $t : \mathbb{N} \to \mathbb{N}$, there exists a language that is decidable in $O(t(n))$ time but not in time $o(t(n)/\log t(n))$.

Why log factor? Simulation may be done with constant factor for space overhead, but not for time overhead:
tape head needs to move on input, more than constant per move

To keep moves efficient, divide tape into three *tracks*

1	2	3	1	2	3	1	2	3	1	2	3	...

1. information on M's tape
2. M's transition function, and current state
3. simulation time counter

Key: keep tracks 2 and 3 contents close to head position on track 1 (need to access them together)
track 2: constant space $\Longrightarrow$ constant time to move
track 3: $O(\log t(n))$ space $\Longrightarrow$ $O(\log t(n))$ time to move

Corollary: Time complexity classes

For any functions $t_1, t_2 : \mathbb{N} \to \mathbb{N}$, with $t_1(n) = o(t_2(n)/\log t_2(n))$, and t_2 time constructible, we have TIME($t_1(n)$) $\subsetneq$ TIME($t_2(n)$).

In particular, we have TIME(n^{c_1}) $\subsetneq$ TIME(n^{c_2}) for any real numbers $1 \leq c_1 < c_2$

P $\subsetneq$ EXPTIME

Exponential Space Completeness

Def A language B is **EXPSPACE-complete** if
1. $B \in$ EXPSPACE
2. every A in EXPSPACE is polynomially reducible to B

We identify a particular language that's EXPSPACE-complete.

Generalized regular expressions:
allow R^k as shorthand for $R \cdot R \cdot \ldots \cdot R$ (k concatenations)

We show $EQ_{\mathsf{REX}\uparrow}$ is EXPSPACE-complete
(equivalence of two regexes with exponentiation)

Proof: $EQ_{\mathsf{REX}\uparrow}$ is in EXPSPACE

On input $\langle R_1, R_2 \rangle$

1. Convert generalized to basic regexes B_1, B_2 (exponential space)
2. Convert B_1, B_2 to NFAs (linear space)
3. Check their equivalence

nondeterministic linear space
guess distinguishing string, like in $\overline{ALL_{\mathsf{NFA}}}$
convert to $O(n^2)$ space deterministic by Savitch's theorem

Proof: $EQ_{\mathsf{REX}\uparrow}$ is EXPSPACE-hard

Take any language A decided by TM M in SPACE(2^{n^k})

Reduce using computation histories

$R_1 = \Delta^*$, with $\Delta = \Gamma \cup Q \cup \{\#\}$ (configuration alphabet)

$R_2 =$ all strings which are **not** rejecting computation histories

Not equivalent precisely if M accepts w

Rejecting CH may have: bad start, bad middle ("window"), bad reject

Exponentiation key to efficient encoding
e.g. all strings that don't have $\#$ in position $2^{n^k} + 1$: $\Delta^{2^{n^k}} \Delta_{\#} \Delta^*$