Outline

- The class L and NL
- Log space computable functions
- NL-complete languages
- PATH is NL-complete
- The coNL class

From Sipser Chapter 8.4-8.6
Chaining polytime reductions

• So far, when describing the space requirements of TM we have considered at the same time:
  – The locations of the tape used to initially store the input
  – The locations of the tape used to store the output at the end of the computation
  – The locations of the tape used by the TM during its execution to maintain “work-in-progress” data

• Under this characterization, all languages are in SPACE(n), where n is the input size!
  – Space requirement is always at least linear w.r.t. the input size

• In some circumstances we want to focus on the space used in the execution of the input of the algorithm excluding the space for input and output
  – Under this assumption, we can characterize algorithms which req
Log space transducer

• A log-time transducer is a TM with
  – A read-only input tape for the input
  – A write-only, move-to-tight-only output tape
  – A work tape, which uses $O(\log n)$ space, where $n$ is the size of the input
  – The TM always halts

• Example: The language $\{0^n1^n\}$ can be decided by a log time transducer
  – We only need to count the number of 0s and 1s
  – At most we need to keep track of a number $n$
  – Required $\log n$ space
The classes L and NL

- **L** = \{L|L=L(M) \ where \ M \ is \ a \ deterministic \ TM \ which \ uses \ at \ most \ log(n) \ work \ space\}

- **NL** = \{L|L=L(N) \ where \ N \ is \ a \ non-deterministic \ TM \ which \ uses \ at \ most \ log(n) \ work \ space\}
  
  - The work-space of a Non-deterministic TM is the space used at most for any input for any non-deterministic execution branch of the TM

- Is L = NL? Open question!
  
  - Savitch’s Thm. Only implies that there is at most a quadratic blowup
The PATH language

• PATH = \{<G,s,t>| G = (V,E) is a directed graph with a path from the vertex s to the vertex t}\}
  – Also referred as the path-connection problem!

• PATH ∈ NL
  – N = “On input <G,s,t>”:
  – r = s //r is a reference to the current step of the path
  – For i =2 to n=|V|
    • Not deterministically pick a vertex v among the successors of r
    • If (r,v) ∈ E:
      – If v= t then ACCEPT
    • Else: REJECT
  – REJECT
Analysis

• Correctness: If there is a path from $s$ to $t$ it will be selected by one sequence of non-deterministic choices
  – Such path will have length at most $n = |V|$

• Analysis
  – The algorithm only needs enough work space to keep track of the position of the current vertex in the construction of the path $(r)$ on the input tape
  – Overall $O(\log n)$ space
PAHT ∈ L?

- We need to avoid cycles
  - Otherwise the TM is not guaranteed to halt!
- Keeping track of the current position (r) is not enough
- We would need to keep track of the entire path so far to avoid cycles
- But a path can have length O(n)
- We would require O(n) work-space
- A deterministic variation of the discussed algorithm does not work
Log space reductions

- A log space transducer $M$ computes a function $f : \Sigma^* \rightarrow \Sigma^*$ where $f(w)$ is the string remaining on the output tape at the end of the computation after $M$ halts with $w$ on its input tape.
- Such a function is a log space computable function.
- A language $A$ is log space reducible to a language $B$, if $A$ is mapping reducible to $B$, written $A \leq_L B$, by means of a log space computable function.
  - That is, it there is a log space computable function such that $\forall w \in \Sigma^*$
    $w \in A \iff f(w) \in B$
NL-complete languages

• A language L is **NL-hard** if for any language A in NL, we have A \(\leq_L L\)
  
  – NL-hard languages are as hard (with respect to space requirement as any language in NL)

• We say that a language L is **NL-complete** if
  
  – \(L \in \text{NL}\)
  
  – L is NL-hard
Consequences

• Theorem 1: if $A \leq_L B$ and $B \in L$, then $A \in L$
  – There exist a log space computable function such that $w \in A \iff f(w) \in B$
  – By definition, there exists a log space transducer that decides $B$

• Theorem 2: if any NL-complete language is in $L$, then $L = \text{NL}$
  – Let $B$ be a NL complete language in $L$
  – there exists a log space transducer that decides $B$
  – By definition of NL-completeness, for any $A$ in $L$ we have $A \leq_L B$
PATH is NL-complete

• We already proved that PATH $\in$ NL
• We need to argue that PATH is NL-hard
• Proof idea: For any language $L \in$ NL we want to have a reduction such that given $w$ we construct an instance $<G,s,t>$ of the PATH decision problem such that $w \in L \iff <G,s,t> \in PATH$
• Recall analogous previous results:
  – We consider the computational history of a TM for $L$
  – We consider the sequence of configurations form a starting one on input $w$ to an accepting one
  – We use such sequence of configurations to build $<G,s,t>$
PATH is NL-complete

• We already proved that PATH $\in$ NL
• We need to argue that PATH is NL-hard
• Proof idea: For any language $L \subseteq$ NL we want to have a reduction such that given $w$ we construct an instance $<G,s,t>$ of the PATH decision problem such that $w \in L$ $\iff <G,s,t> \in PATH$

• Recall analogous previous results:
  – We consider the computational history of a TM for $L$
  – We consider the sequence of configurations form a starting one on input $w$ to an accepting one
  – We use such sequence of configurations to build $<G,s,t>$
Proving PATH is NL-complete

• Given $w$, $L$, let $N$ be a log space non-deterministic transducer such that $L(N)=L$

• Construct a directed graph $G=(V,E)$
  
  – $V$ is such that every vertex correspond to a possible configuration of $N$ on $w$
  
  – $E$ is the set of edges such that $(u, v) \in E$ if is possible to reach the configuration corresponding to vertex $v$ from the configuration corresponding to vertex $u$ using a single applications of the transitions functions
  
  – Let $v_{\text{start}}$ (resp., $v_{\text{accept}}$) be the vertex corresponding to the starting (resp., accepting ) configuration

• Run the procedure for PATH on $<G, v_{\text{start}}, v_{\text{accept}}>$

• We set the length of the path to $d \cdot n$, where $d$ is a constant
Proving PATH is NL-complete

• Correctness: if in G there is a directed path from \( v_{\text{start}} \) to \( v_{\text{accept}} \), then there is a computational history on N for w such that w is accepted.
  – The inverse direction is true by construction

• Analysis: The reduction needs to give \( <G,s,t> \)
  – By definition, each configuration of N uses at most \( O(\log n) \) space
  – We can generate one at a time the vertices, and output them on the output tape
  – From Savitch’s theorem we have \( \text{SPACE} (\log n) \subseteq \text{TIME} (2^{\log n}) \)
  – An accepting computation history will require at most n steps
  – Log n space is sufficient
Relation between P and NL

– We have \( L \subseteq P \), since:

\[
SPACE(\log n) \subseteq TIME(2^{\log n}) = TIME(n)
\]

– What about NL?

• A TM which uses space \( f(n) \) can have at most \( f(n)2^{O(f(n))} \) distinct configurations

• We can bound the time by the number of such configurations

• For \( f(n) = \log n \) we have \( n \log n \) configurations \( \rightarrow \) polynomial in \( n \)

• Hence, \( NL \subseteq P \)
The class coNL

- $\text{coNL} = \{ L | \overline{L} \in \text{NL} \}$
- $\text{PATH} \in \text{coNL}$
  - See section 8.6 for detailed proof
- Important consequence!
  - Since PATH is also NL-complete we can conclude NL=coNL