Formal Languages, Automata and Computation

Space Complexity

Space Complexity

- (Disk) Space – the final frontier!
- How much memory do computational problems require?
- We characterize problems based on their memory requirements.
- Space is reusable, time is not!
- We again use the Turing machine as our model of computation.

Space Complexity

Definition – Space Complexity

Let M be a deterministic Turing machine that halts on on inputs. The space complexity of M is the function $f : \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the **maximum number of tape cells** that M scans on any input of length n.

For nondeterministic TMs where all branches halt on all inputs, we take the maximum over all the branches of computation.

Space Complexity

Definition – Space Complexity Classes

Let $f : \mathcal{N} \longrightarrow \mathcal{R}^{+}$. The space complexity classes are defined as follows:

$$\begin{aligned} \text{SPACE}(f(n)) &= \{L \mid & L \text{ is a language decided by an } O(f(n)) \\ & & \text{space deterministic TM}\} \\ \text{NSPACE}(f(n)) &= \{L \mid & L \text{ is a language decided by an } O(f(n)) \\ & & \text{space nondeterministic TM}\} \end{aligned}$$

- SPACE($f(n)$) formalizes the class of problems that can be solved by computers with bounded memory. (Real world!)
- SPACE($f(n)$) problems could potentially take a long time to solve.
- Intutively space and time seem to be interchangeable.
- Just because a problem needs only *linear space* does not mean it can be solved in *linear time*.

Deterministic Space Complexity of *SAT*

- *SAT* is NP-complete.
- But *SAT* can be solved in linear space.
- M_1 = "On input $\langle\phi\rangle$, where ϕ is a Boolean formula:
  1. For each truth assignment to the variables $x_1, x_2, \ldots, x_m$ of ϕ:
  2. Evaluate ϕ on that truth assignment.
  3. If ϕ ever evaluates to 1, *accept*; if not, *reject*."

3SAT $\in$ **SPACE(n)**

(	x	∨	¬	y	∨	x	)		(	y	∨	x	∨	y	)						
(	x	∨	¬	y	∨	x	)		(	y	∨	x	∨	y	)		#	x		y	
(	x	∨	¬	y	∨	x	)		(	y	∨	x	∨	y	)		#	x	0	y	0
(	x	∨	¬	y	∨	x	)		(	y	∨	x	∨	y	)		#	x	0	y	1
(	x	∨	¬	y	∨	x	)		(	y	∨	x	∨	y	)		#	x	1	y	0

- Note that M_1 takes exponential time.

NONDETERMINISTIC SPACE COMPLEXITY OF ALL_{NFA}

- Consider $ALL_{NFA} = \{\langle A \rangle \mid A \text{ is a NFA and } L(A) = \Sigma^*\}$
- The following nondeterministic linear space algorithm decides $\overline{ALL_{NFA}}$.
- Nondeterministically guess an input string rejected by the NFA and use linear space to guess which states the NFA could be at a given time.
- $N =$ "On input $\langle M \rangle$ where M is an NFA.
  1. Place a marker on the start state of NFA.
  2. Repeat 2^q times, where q is the number of states of M.
     2.1 Nondeterministically select an input symbol and change the position of the markers on M's states, to simulate reading that symbol.
  3. *Accept* if stages 2 reveals some string that M rejects, i.e., if at some point none of the markers lie on accept states of M. Otherwise, *reject*."

NONDETERMINISTIC SPACE COMPLEXITY OF ALL_{NFA}

- Since there are at most 2^q subsets of the states of M, it must reject one of length at most 2^q, if M rejects any strings.
  - Remember that determinization could end up with at most 2^q states.
- N needs space for
  - storing the locations of the markers ($O(q) = O(n)$)
  - the repeat loop counter ($O(q) = O(n)$)
- Hence N runs in nondeterministic $O(n)$ space.
- Note that N runs in nondeterministic $2^{O(n)}$ time.
  - ALL_{NFA} is not known to be in NP or coNP.

Savitch's Theorem

- Remember that simulation of a nondeterministic TM with a deterministic TM requires an exponentional increase in time.
- Savitch's Theorem shows that any nondeterministic TM that uses $f(n)$ space can be converted to a deterministic TM that uses only $f^2(n)$ space, that is,

$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$$

- Obviously, there will be a slowdown.

Savitch's Theorem

Theorem

For any function $f : \mathcal{N} \longrightarrow \mathcal{R}^+$, where $f(n) \geq n$

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Proof Idea

- Let N be a nondeterministic TM with space complexity $f(n)$.
- Construct a deterministic machine M that tries every possible branch of N.
- Since each branch of N uses at most $f(n)$ space, then M uses space at most $f(n)$ space $+$ space for book-keeping.
- We need to simulate the nondeterministic computation and save as much space as possible.

Savitch's Theorem

- Given two configurations c_1 and c_2 of a $f(n)$ space TM N, and a number t, determine if we can get from c_1 to c_2 within t steps.
- $CANYIELD =$ " On input c_1, c_2 and t:
  1. If $t = 0$ accept iff $c_1 = c_2$
  2. If $t = 1$ accept iff $c_1 = c_2$ or c_1 yields c_2 in one step.
  3. If $t > 1$ then for every possible configuration c_m of N for w, using space $f(n)$
  4. Run $CANYIELD(c_1, c_m, \frac{t}{2})$.
  5. Run $CANYIELD(c_m, c_2, \frac{t}{2})$.
  6. If steps 4 and 5 both accept, then *accept*.
  7. If haven't yet accepted, *reject*."
- Space is reused during the recursive calls.
- The depth of the recursion is at most $\log t$.
- Each recursive steps uses $O(f(n))$ space and $t = 2^{O(f(n))}$ so $\log t = O(f(n))$. Hence total space used is $O(f^2(n))$.

Savitch's Theorem

- M simulates N using *CANYIELD*.
- If n is the length of w, we choose d so that N has no more than $2^{df(n)}$ configurations each using $f(n)$ tape.
- $2^{df(n)}$ provides an upper bound on the running time on any branch of N.
- $M =$ "On input w:
  1. Output the result of $CANYIELD(c_{start}, c_{accept}, 2^{df(n)})$."
- At each stage, *CANYIELD* stores c_1, c_2, and t for a total of $O(f(n))$ space.
- Minor technical points with the accepting configuration and the initial value of t (e.g., how does the TM know $f(n)$?) – See the book.

The Class PSPACE

Definition – PSPACE

PSPACE is the class of languages that are decidable in polynomial space on a deterministic TM.

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k).$$

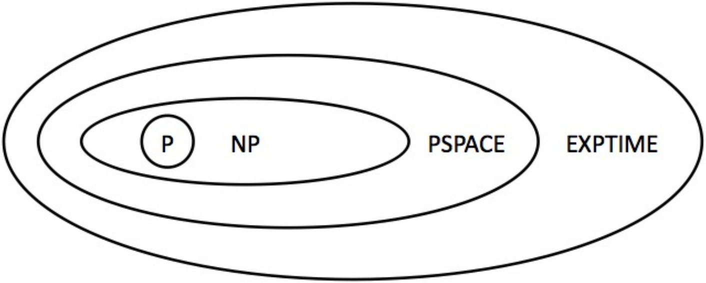
- NSPACE is defined analogously.
- But PSPACE = NSPACE, due to Savitch's theorem, because the square of a polynomial is also a polynomial.

The Class PSPACE – Some observations

- We know $SAT \in \text{SPACE}(n)$.
  - $\Rightarrow SAT \in$ PSPACE.
- We know $\overline{ALL_{NFA}} \in \text{NSPACE}(n)$ and hence $\overline{ALL_{NFA}} \in \text{SPACE}(n^2)$, by Savitch's theorem.
  - $\Rightarrow \overline{ALL_{NFA}} \in$ PSPACE.
- Deterministic space complexity classes are closed under complementation, so $ALL_{NFA} \in \text{SPACE}(n^2)$.
  - $\Rightarrow ALL_{NFA} \in$ PSPACE.
- A TM that operates in $f(n) \geq n$ time, can use at most $f(n)$ space.
  - $\Rightarrow$ P $\subseteq$ PSPACE
- NP $\subseteq$ NSPACE $\Rightarrow$ NP $\subseteq$ PSPACE.

The Class PSPACE – Some observations

- We can also bound the time complexity in terms of the space complexity.
- For $f(n) \geq n$, a TM that uses $f(n)$ space, can have at most $f(n)2^{O(f(n))}$ configurations.
  - $f(n)$ symbols on tape, so $|\Gamma|^{f(n)}$ possible strings and $f(n)$ possible state positions and $|Q|$ possible states $= 2^{O(f(n))}$
- PSPACE $\subseteq$ EXPTIME $= \bigcup_k \text{TIME}(2^{n^k})$.



PSPACE-Completeness

Definition – PSPACE-complete

A language B is PSPACE-complete if it satisfies two conditions:

1. B is in PSPACE, and
2. every A in PSPACE is polynomial time reducible to B.

- Note that we use polynomial-time reducibility!
- The reduction should be easy relative to the complexity of typical problems in the class.
- In general, whenever we define complete-problems for a complexity class, the reduction model must be more limited that the model use for defining the class itself.

The TQBF Problem

- Quantified Boolean Formulas are exactly like the Boolean formulas we define for the *SAT* problem, but additionally have existential ($\exists$) and universal ($\forall$) quantifiers.
  - $\forall x[x \vee y]$
  - $\exists x \exists y[x \vee \overline{y}]$
  - $\forall x[x \vee \overline{x}]$
  - $\forall x[x]$
  - $\forall x \exists y[(x \vee y) \wedge (\overline{x} \vee \overline{y})]$
- A fully quantified Boolean formula is a quantified formula where every variable is quantified.
  - All except the first above are fully quantified.
  - A fully quantified Boolean formula is also called a sentence, and is either true or false.

Definition – TQBF

$TQBF = \{\langle \phi \rangle \mid \phi$ is a true fully quantified Boolean formula$\}$

THE TQBF PROBLEM

THEOREM

TQBF $= \{\langle\phi\rangle \mid \phi$ is a true fully quantified Boolean formula$\}$ is PSPACE-complete.

- Assume T decides *TQBF*.
- If ϕ has no quantifiers, it is an expression with only constants! Evaluate ϕ and accept if result is 1.
- If $\phi = \exists x \psi$, recursively call T on ψ, first with $x = 0$ and then with $x = 1$. Accept if either returns 1.
- If $\phi = \forall x \psi$, recursively call T on ψ, first with $x = 0$ and then with $x = 1$. Accept if both return 1.

The TQBF Problem

Claim

Every language A in PSPACE is polynomial-time reducible to *TQBF*.

- We build a polynomial time reduction from A to *TQBF*
- The reduction turns a string w into a TQBF ϕ that simulates a PSPACE TM M for A on w.
- Essentially the same as in the proof of the NP-completeness of *SAT* – build a formula from the accepting computation history.
- But uses the approach in Savitch's Theorem.
- Details in section 8.3 in the book.
- PSPACE is often called the class of games.
    - Formalizations of many popular games are PSPACE-complete.

The Classes L and NL

- We have so far considered time and space complexity bounds that are at least linear.
- We now examine smaller, sublinear space bounds.
  - For time complexity, sublinear bounds are insufficient to read the entire input!
- For sublinear space complexity, the TM is able to read the whole input but not store it.
- We must modify the computational model!

The Classes L and NL

- We introduce a TM with two-tapes:
  1. A read-only input tape.
  2. A read/write work tape.
- On the input tape, the head always stays in the region where the input is.
- The work tape can be read and written in the usual way.
- Only the cells scanned on the work tape contribute to the space complexity.

Definitions– Log space complexity classes

$$\mathrm{L} = \mathrm{SPACE}(\log n)$$

$$\mathrm{NL} = \mathrm{NSPACE}(\log n)$$

An Algorithm in L

- Consider the (good old) language $A = \{0^k 1^k \mid k \geq 0\}$
- Previous algorithm (zig-zag and cross out symbols) used linear space.
- We can not do this now since the input tape is read-only.
- Once the machine is certain the string is of the desired pattern, it can count the number of 0's and 1's.
- The only additional space needed are for the two counters (in binary).
- A binary counter uses only logarithmic space, $O(\log k)$.

AN ALGORITHM IN NL

- Consider the *PATH* problem
  $PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$
- *PATH* is in P, but that algorithm uses linear space.
- It is not known if *PATH* can be solved in deterministic log space.
- It can be solved in nondeterministic log space:
  1. Starting with s, the nondeterministic log space TM guesses the next node to go to on the way to t.
  2. The TM only records the id or the position of the node (so needs log space).
  3. The TM nondeterministically guesses the next node, until either it reaches t or until it has gone for m steps where m is the number of nodes.

The Classes L and NL

- Log-space reducibility
- NL-completeness
- *PATH* is NL-complete.
  - For a given log space nondeterministic TM and input w, map the accepting computation history to a graph, with nodes representing configurations.
- NL $\subseteq$ P (remember *PATH* $\in$ P)
- NL $=$ coNL.
- L $\subseteq$ NL = coNL $\subseteq$P $\subseteq$ PSPACE.

And We are done for the semester (– the final)

- Thanks for your patience and for taking the occasional mental pain.
- But then, no pain no gain!
- We do review Thursday and (if you want) on Sunday, please come prepared and let me know what major concepts your still have problems with.
- Final on Monday, April 25, 2010, at 14:30-17:30
- Good luck!