Polynomial Space

The classes **PS** and **NPS**

Relationship to Other Classes

Equivalence **PS** = **NPS**

A PS-Complete Problem

Polynomial-Space-Bounded TM's

- A TM M is said to be *polyspace-bounded* if there is a polynomial p(n) such that, given input of length n, M never uses more than p(n) cells of its tape.
- L(M) is in the class *polynomial space*, or **PS**.

Nondeterministic Polyspace

- If we allow a TM M to be nondeterministic but to use only p(n) tape cells in any sequence of ID's when given input of length n, we say M is a *nondeterministic polyspace-bounded* TM.
- And L(M) is in the class *nondeterministic polyspace*, or **NPS**.

Relationship to Other Classes

- Obviously, **P** $\subseteq$ **PS** and **NP** $\subseteq$ **NPS**.
  - If you use polynomial time, you cannot reach more than a polynomial number of tape cells.
- Alas, it is not even known whether **P** = **PS** or **NP** = **PS**.
- On the other hand, we shall show **PS** = **NPS**.

Exponential Polytime Classes

- A DTM M runs in *exponential polytime* if it makes at most $c^{p(n)}$ steps on input of length n, for some constant c and polynomial p.
- Say L(M) is in the class **EP**.
- If M is an NTM instead, say L(M) is in the class **NEP** (*nondeterministic exponential polytime*).

More Class Relationships

- **P** $\subseteq$ **NP** $\subseteq$ **PS** $\subseteq$ **EP**, and at least one of these is proper.
  - A diagonalization proof shows that **P** $\neq$ **EP**.
- **PS** $\subseteq$ **EP** requires proof.
- Key Point: A polyspace-bounded TM has only $c^{p(n)}$ different ID's.
  - We can count to $c^{p(n)}$ in polyspace and stop it after it surely repeated an ID.

Proof **PS $\subseteq$ EP**

- Let M be a p(n)-space bounded DTM with s states and t tape symbols.
- Assume M has only one semi-infinite tape.
- The number of possible ID's of M is $sp(n)t^{p(n)}$.

States

Positions of tape head

Tape contents

Proof **PS** $\subseteq$ **EP** – (2)

- Note that $(t+1)^{p(n)+1} \geq p(n)t^{p(n)}$.
  - Use binomial expansion $(t+1)^{p(n)+1} = t^{p(n)+1} + (p(n)+1)t^{p(n)} + \ldots$
- Also, $s = (t+1)^c$, where $c = \log_{t+1}s$.
- Thus, $sp(n)t^{p(n)} \leq (t+1)^{p(n)+1+c}$.
- We can count to the maximum number of ID's on a separate tape using base t+1 and p(n)+1+c cells – a polynomial.

Proof **PS** $\subseteq$ **EP** – (2)

- Redesign M to have a second tape and to count on that tape to $sp(n)t^{p(n)}$.
- The new TM M' is polyspace bounded.
- M' halts if its counter exceeds $sp(n)t^{p(n)}$.
  - If M accepts, it does so without repeating an ID.
- Thus, M' is exponential-polytime bounded, proving L(M) is in **EP**.

Savitch's Theorem: **PS = NPS**

- Key Idea: a polyspace NTM has "only" $c^{p(n)}$ different ID's it can enter.
- Implement a deterministic, recursive function that decides, about the NTM, whether $I \vdash^* J$ in at most m moves.
- Assume $m \leq c^{p(n)}$, since if the NTM accepts, it does so without repeating an ID.

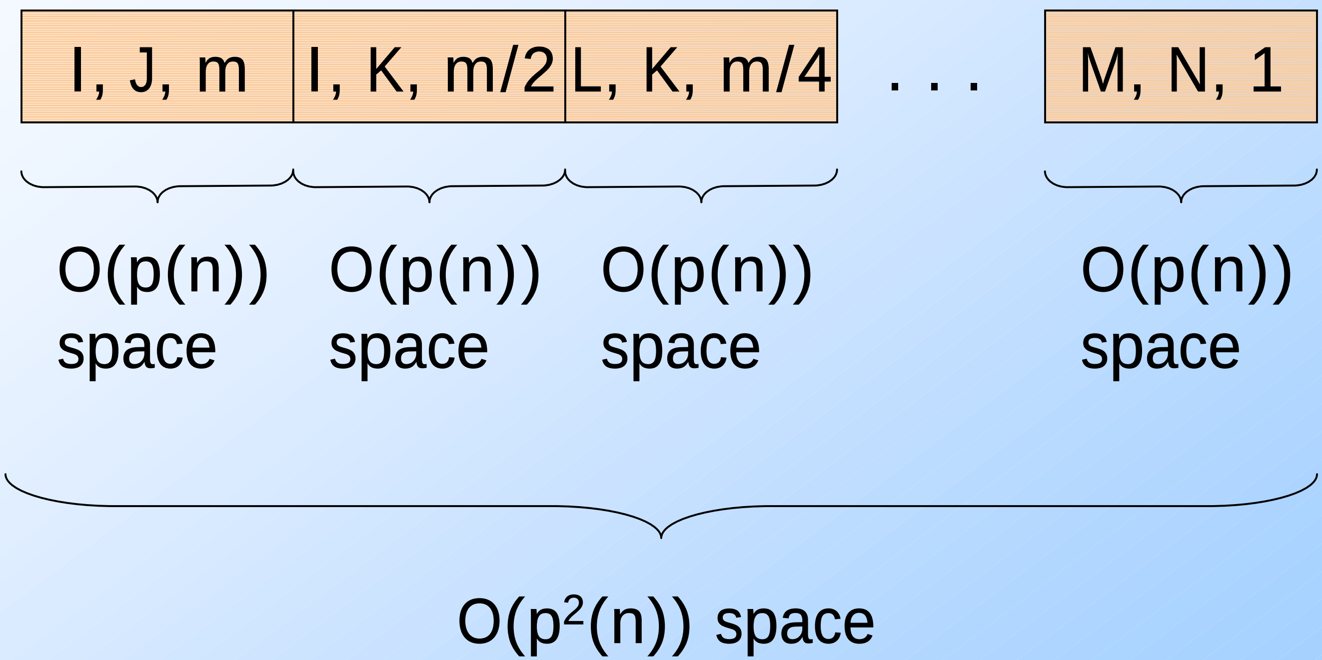
Savitch's Theorem – (2)

- *Recursive doubling* trick: to tell if $I \vdash^* J$ in $\leq$ m moves, search for an ID K such that $I \vdash^* K$ and $K \vdash^* J$, both in $\leq$ m/2 moves.
- Complete algorithm: ask if $I_0 \vdash^* J$ in at most $c^{p(n)}$ moves, where I_0 is the initial ID with given input w of length n, and J is any of the ID's with an accepting state and length $\leq$ p(n).

Recursive Doubling

```
boolean function f(I, J, m) {
  for (all ID's K using p(n) tape)
     if (f(I, K, m/2) && f(K, J, m/2))
          return true;
  return false;
}
```

Stack Implementation of *f*



Space for Recursive Doubling

- f(I, J, m) requires space O(p(n)) to store I, J, m, and the current K.
  - m need not be more than $c^{p(n)}$, so it can be stored in O(p(n)) space.
- How many calls to f can be active at once?
- Largest m is $c^{p(n)}$.

Space for Recursive Doubling – (2)

- Each call with third argument m results in only one call with argument m/2 at any one time.
- Thus, at most $\log_2 c^{p(n)} = O(p(n))$ calls can be active at any one time.
- Total space needed by the DTM is therefore $O(p^2(n))$ – a polynomial.

PS-Complete Problems

- A problem P in **PS** is said to be *PS-complete* if there is a polytime reduction from every problem in **PS** to P.
  - Note: it has to be polytime, not polyspace, because:
    1. Polyspace can exponentiate the output size.
    2. Without polytime, we could not deal with the question **P** = **PS**?

What PS-Completeness Buys

- If some PS-complete problem is:
  1. In **P**, then **P** = **PS**.
  2. In **NP**, then **NP** = **PS**.

Quantified Boolean Formulas

- We shall meet a PS-complete problem, called *QBF* : is a given quantified boolean formula true?
- But first we meet the QBF's themselves.
- We shall give a recursive (inductive) definition of QBF's along with the definition of free/bound variable occurrences.

QBF's – (2)

- First-order predicate logic, with variables restricted to true/false.
- Basis:
  1. Constants 0 (false) and 1 (true) are QBF's.
  2. A variable is a QBF, and that variable occurrence is *free* in this QBF.

QBF's – (3)

- Induction: If E and F are QBF's, so are:
  1. E AND F, E OR F, and NOT F.
     - Variables are bound or free as in E or F.
  2. $(\forall x)E$ and $(\exists x)E$ for any variable x.
     - All free occurrences x are bound to this *quantifier*, and other occurrences of variables are free/bound as in E.
- Use parentheses to group as needed.
  - Precedence: quantifiers, NOT, AND, OR.

Example: QBF

bound

$$(\forall x)(\exists y)(((\exists x)(x \text{ OR } y)) \text{ AND NOT } (x \text{ AND } y))$$

bound

bound

Evaluating QBF's

- In general, a QBF is a function from truth assignments for its free variables to {0, 1} (false/true).
- Important special case: no free variables; a QBF is either true or false.
- We shall give the evaluation only for these formulas.

Evaluating QBF's – (2)

- Induction on the number of operators, including quantifiers.
- Stage 1: eliminate quantifiers.
- Stage 2: evaluate variable-free formulas.
- Basis: 0 operators.
  - Expression can only be 0 or 1, because there are no free variables.
  - Truth value is 0 or 1, respectively.

Induction

1. Expression is NOT E, E OR F, or E AND F.
   - Evaluate E and F; apply boolean operator to the results.
2. Expression is $(\forall x)E$.
   - Construct E_0 = E with each x bound to this quantifier replaced by 0, and analogously E_1.
   - E is true iff both E_0 and E_1 are true.
3. Expression is $(\exists x)E$.
   - Same, but E is true iff either E_0 or E_1 is true.

Example: Evaluation

$(\forall x)(\exists y)(((\exists x)(x \text{ OR } y)) \text{ AND NOT } (x \text{ AND } y))$

- Substitute x = 0 for outer quantifier:

$(\exists y)(((\exists x)(x \text{ OR } y)) \text{ AND NOT } (0 \text{ AND } y))$

- Substitute x = 1 for outer quantifier:

$(\exists y)(((\exists x)(x \text{ OR } y)) \text{ AND NOT } (1 \text{ AND } y))$

Example: Evaluation – (2)

- Let's follow the x = 0 subproblem:

(∃y)(((∃x)(x OR y)) AND NOT (0 AND y))

- Two cases: y = 0 and y = 1.

((∃x)(x OR 0)) AND NOT (0 AND 0)

((∃x)(x OR 1)) AND NOT (0 AND 1)

Example: Evaluation – (3)

- Let's follow the y = 0 subproblem:

((∃x)(x OR 0)) AND NOT (0 AND 0)

- Need to evaluate (∃x)(x OR 0).
  - x = 0: 0 OR 0 = 0.
  - x = 1: 1 OR 0 = 1.
  - Hence, value is 1.
- Answer is 1 AND NOT (0 AND 0) = 1.

Example: Evaluation – (4)

- Let's follow the y = 1 subproblem:

(($\exists$x)(x OR 1)) AND NOT (0 AND 1)

- Need to evaluate ($\exists$x)(x OR 1).
  - x = 0: 0 OR 1 = 1.
  - x = 1: 1 OR 1 = 1.
- Hence, value is 1.
- Answer is 1 AND NOT (0 AND 1) = 1.

Example: Evaluation – (5)

- Now we can resolve the (outermost) x = 0 subproblem:

$(\exists y)(((\exists x)(x \text{ OR } y)) \text{ AND NOT } (0 \text{ AND } y))$

- We found both of its subproblems are true.
- We only needed one, since the outer quantifier is ∃y.
- Hence, 1.

Example: Evaluation – (6)

- Next, we must deal with the x = 1 case:

$(\exists y)(((\exists x)(x \text{ OR } y)) \text{ AND NOT } (1 \text{ AND } y))$

- It also has the value 1, because the subproblem y = 0 evaluates to 1.
- Hence, the entire QBF has value 1.

The QBF Problem

- The problem *QBF* is:
  - Given a QBF with no free variables, is its value 1 (true)?
- Theorem: QBF is PS-complete.
- Comment: What makes QBF extra hard? Alternation of quantifiers.
  - Example: if only ∃ used, then the problem is really SAT.

Part I: QBF is in **PS**

- Suppose we are given QBF F of length n.
- F has at most n operators.
- We can evaluate F using a stack of subexpressions that never has more than n subexpressions, each of length $\leq$ n.
- Thus, space used is $O(n^2)$.

QBF is in **PS** – (2)

- Suppose we have subexpression E on top of the stack, and E = G OR H.

1. Push G onto the stack.
2. Evaluate it recursively.
3. If true, return true.
4. If false, replace G by H, and return what H returns.

QBF is in **PS** – (3)

- Cases E = G AND H and E = NOT G are handled similarly.
- If E = (∃x)G, then treat E as if it were E = E_0 OR E_1.
  - Observe: difference between ∃ and OR is succinctness; you don't write both E_0 and E_1.
    - But E_0 and E_1 must be almost the same.
- If E = (∀x)G, then treat E as if it were E = E_0 AND E_1.

Part II: All of **PS** Polytime Reduces to QBF

- Recall that if a polyspace-bounded TM M accepts its input w of length n, then it does so in $c^{p(n)}$ moves, where c is a constant and p is a polynomial.
- Use recursive doubling to construct a QBF saying "there is a sequence of $c^{p(n)}$ moves of M leading to acceptance of w."

Polytime Reduction: The Variables

- We need collections of boolean variables that together represent one ID of M.
- A *variable ID* I is a collection of O(p(n)) variables $y_{j,A}$.
  - True iff the j-th position of the ID I is A (a state or tape symbol).
  - $0 \leq j \leq p(n)+1$ = length of an ID.

The Variables – (2)

- We shall need O(p(n)) variable ID's.
  - So the total number of boolean variables is $O(p^2(n))$.
- Shorthand: $(\exists I)$, where I is a variable ID, is short for $(\exists y_1)(\exists y_2)(\ldots)$, where the y's are the boolean variables belonging to I.
- Similarly $(\forall I)$.

Structure of the QBF

- The QBF is $(\exists I_0)(\exists I_f)$(S AND N AND F AND U), where:
  1. I_0 and I_f are variable ID's representing the start and accepting ID's respectively.
  2. U = "unique" = one symbol per position.
  3. S = "starts right": $I_0 = q_0w$.
  4. F = "finishes right" = I_f accepts.
  5. N = "moves right."

Structure of U, S, and F

- U is as done for Cook's theorem.
- S asserts that the first n+1 symbols of I_0 are q_0w, and other symbols are blank.
- F asserts one of the symbols of I_f is a final state.
- All are easy to write in O(p(n)) time.

Structure of QBF N

- $N(I_0,I_f)$ needs to say that $I_0 \vdash^* I_f$ by at most $c^{p(n)}$ moves.
- We construct subexpressions N_0, N_1, N_2,... where $N_i(I,J)$ says "$I \vdash^* J$ by at most 2^i moves."
- N is N_k, where $k = \log_2 c^{p(n)} = O(p(n))$.

Note: differs from text, where the subscripts exponentiate.

Constructing the N_i's

- Basis: $N_0(I, J)$ says "I=J OR I⊦J."
- If I represents variables $y_{j,A}$ and J represents variables $z_{j,A}$, we say I=J by the boolean expression for $y_{j,A} = z_{j,A}$ for all j and A.
  - Remember: a=b is (a AND b) OR (NOT a AND NOT b).
- I⊦J uses the same idea as for SAT.

Induction

- Suppose we have constructed N_i and want to construct N_{i+1}.
- $N_{i+1}(I, J)$ = "there exists K such that $N_i(I, K)$ and $N_i(K, J)$."
- We must be careful:
  - We must write O(p(n)) formulas, each in polynomial time.

Induction – (2)

- If each formula used two copies of the previous formula, times and sizes would exponentiate.
- Trick: use $\forall$ to make one copy of N_i serve for two.
- $N_{i+1}(I, J)$ = "if (P,Q) = (I,K) or (P,Q) = (K,J), then $N_i(P, Q)$."

Express as boolean variables

Induction – (3)

- More formally, $N_{i+1}(I, J)$ =

$(\exists K)(\forall P)(\forall Q)(((P \neq I \text{ OR } Q \neq K) \text{ AND } (P \neq K \text{ OR } Q \neq J)) \text{ OR } N_i(P, Q))$

Pair (P,Q) is neither (I,K) nor (K,J)

Or $P \vdash^{*} Q$ in at most 2^i moves.

Induction – (4)

- We can thus write N_{i+1} in time O(p(n)) plus the time it takes to write N_i.
- Remember: N is N_k, where $k = \log_2 c^{p(n)} = O(p(n))$.
- Thus, we can write N in time $O(p^2(n))$.
- Finished!! The whole QBF for w can be written in polynomial time.