6.045: Automata, Computability, and Complexity (GITCS)

Class 17

Nancy Lynch

Today

- Probabilistic Turing Machines and Probabilistic Time Complexity Classes
- Now add a new capability to standard TMs: random choice of moves.
- Gives rise to new complexity classes: BPP and RP
- Topics:
  - Probabilistic polynomial-time TMs, BPP and RP
  - Amplification lemmas
  - Example 1: Primality testing
  - Example 2: Branching-program equivalence
  - Relationships between classes
- Reading:
  - Sipser Section 10.2

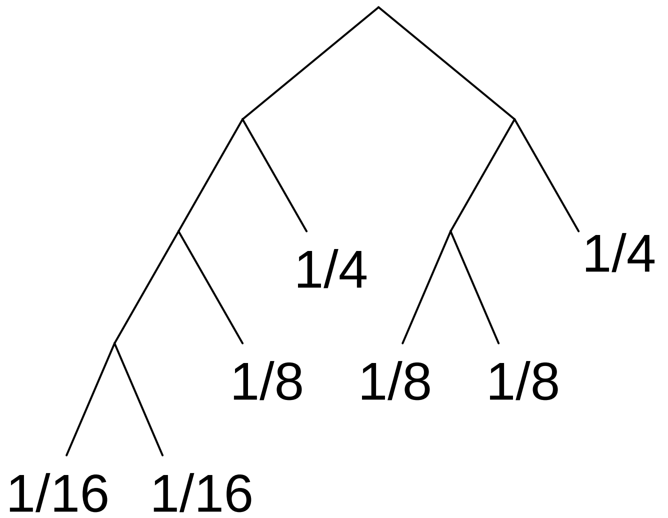
Probabilistic Polynomial-Time Turing Machines, BPP and RP

Probabilistic Polynomial-Time TM

- New kind of NTM, in which each nondeterministic step is a coin flip: has exactly 2 next moves, to each of which we assign probability ½.
- Example:
  - To each maximal branch, we assign a probability:

    $\underbrace{\tfrac{1}{2} \times \tfrac{1}{2} \times \ldots \times \tfrac{1}{2}}_{\text{number of coin flips on the branch}}$

Computation on input w



- Has accept and reject states, as for NTMs.
- Now we can talk about probability of acceptance or rejection, on input w.

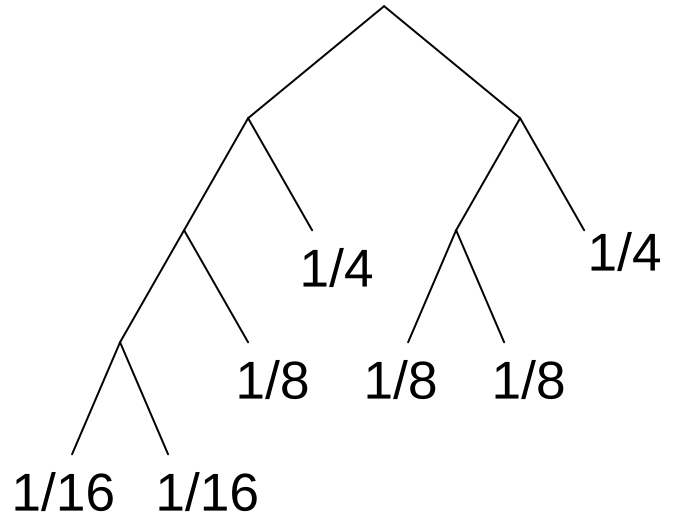
Probabilistic Poly-Time TMs

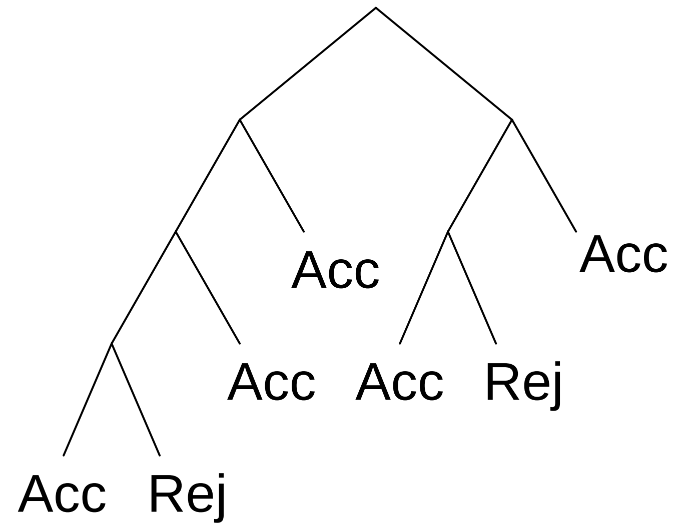
- Probability of acceptance =

  $\Sigma_{\text{b an accepting branch}}$ Pr(b)
- Probability of rejection =

  $\Sigma_{\text{b a rejecting branch}}$ Pr(b)
- Example:
  - Add accept/reject information
  - Probability of acceptance = 1/16 + 1/8 + 1/4 + 1/8 + 1/4 = 13/16
  - Probability of rejection = 1/16 + 1/8 = 3/16
- We consider TMs that halt (either accept or reject) on every branch---deciders.
- So the two probabilities total 1.

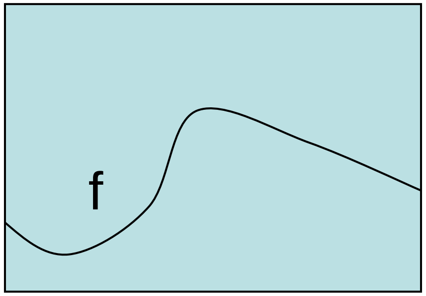
Computation on input w





Probabilistic Poly-Time TMs

- Time complexity:
  - Worst case over all branches, as usual.
- Q: What good are probabilistic TMs?
- Random choices can help solve some problems efficiently.
- Good for getting estimates---arbitrarily accurate, based on the number of choices.
- Example: Monte Carlo estimation of areas
  - E.g, integral of a function f.
  - Repeatedly choose a random point (x,y) in the rectangle.
  - Compare y with f(x).
  - Fraction of trials in which $y \leq f(x)$ can be used to estimate the integral of f.



Probabilistic Poly-Time TMs

- Random choices can help solve some problems efficiently.
- We'll see 2 languages that have efficient probabilistic estimation algorithms.
- Q: What does it mean to estimate a language?
- Each w is either in the language or not; what does it mean to "approximate" a binary decision?

- Possible answer: For "most" inputs w, we always get the right answer, on all branches of the probabilistic computation tree.
- Or: For "most" w, we get the right answer with high probability.
- Better answer: For every input w, we get the right answer with high probability.

Probabilistic Poly-Time TMs

- Better answer: For every input w, we get the right answer with high probability.
- Definition: A probabilistic TM decider M decides language L with error probability ε if
  - $w \in L$ implies that Pr[M accepts w] $\geq 1 - \varepsilon$, and
  - $w \notin L$ implies that Pr[M rejects w] $\geq 1 - \varepsilon$.
- Definition: Language L is in BPP (Bounded-error Probabilistic Polynomial time) if there is a probabilistic polynomial-time TM that decides L with error probability 1/3.
- Q: What's so special about 1/3?
- Nothing. We would get an equivalent definition (same language class) if we chose ε to be any value with $0 < \varepsilon < ½$.
- We'll see this soon---Amplification Theorem

Probabilistic Poly-Time TMs

- Another class, RP, where the error is 1-sided:
- Definition: Language L is in RP (Random Polynomial time) if there is a a probabilistic polynomial-time TM that decides L, where:
  - $w \in L$ implies that Pr[M accepts w] $\geq 1/2$, and
  - $w \notin L$ implies that Pr[M rejects w] = 1.
- Thus, absolutely guaranteed to be correct for words not in L---always rejects them.
- But, might be incorrect for words in L---might mistakenly reject these, in fact, with probability up to ½.
- We can improve the ½ to any larger constant < 1, using another Amplification Theorem.

RP

- Definition: Language L is in RP (Random Polynomial time) if there is a a probabilistic polynomial-time TM that decides L, where:
  - w $\in$ L implies that Pr[M accepts w] $\geq$ 1/2, and
  - w $\notin$ L implies that Pr[M rejects w] = 1.
- Always correct for words not in L.
- Might be incorrect for words in L---can reject these with probability up to ½.
- Compare with nondeterministic TM acceptance:
  - w $\in$ L implies that there is some accepting path, and
  - w $\notin$ L implies that there is no accepting path.

Amplification Lemmas

Amplification Lemmas

- Lemma: Suppose that M is a PPT-TM that decides L with error probability ε, where $0 \leq \varepsilon < ½$.

  Then for any ε', $0 \leq \varepsilon' < ½$, there exists M′, another PPT-TM, that decides L with error probability ε'.
- Proof idea:
  - M′ simulates M many times and takes the majority value for the decision.
  - Why does this improve the probability of getting the right answer?
  - E.g., suppose $\varepsilon = 1/3$; then each trial gives the right answer at least 2/3 of the time (with 2/3 probability).
  - If we repeat the experiment many times, then with very high probability, we'll get the right answer a majority of the times.
  - How many times? Depends on ε and ε'.

Amplification Lemmas

- Lemma: Suppose that M is a PPT-TM that decides L with error probability ε, where $0 \leq \varepsilon < ½$.

  Then for any ε', $0 \leq \varepsilon' < ½$, there exists M′, another PPT-TM, that decides L with error probability ε'.
- Proof idea:
  - M′ simulates M many times, takes the majority value.
  - E.g., suppose $\varepsilon = 1/3$; then each trial gives the right answer at least 2/3 of the time (with 2/3 probability).
  - If we repeat the experiment many times, then with very high probability, we'll get the right answer a majority of the times.
  - How many times? Depends on ε and ε'.
  - 2k, where $(4\varepsilon (1- \varepsilon))^k \leq \varepsilon'$, suffices.
  - In other words $k \geq (\log_2 \varepsilon') / (\log_2 (4\varepsilon (1- \varepsilon)))$.
  - See book for calculations.

Characterization of BPP

- Theorem: $L \in BPP$ if and only for, for some ε, $0 \leq \varepsilon < ½$, there is a PPT-TM that decides L with error probability ε.

- Proof:
  - $\Rightarrow$ If $L \in BPP$, then there is some PPT-TM that decides L with error probability $\varepsilon = 1/3$, which suffices.
  - $\Leftarrow$ If for some ε, a PPT-TM decides L with error probability ε, then by the Lemma, there is a PPT-TM that decides L with error probability 1/3; this means that $L \in BPP$.

Amplification Lemmas

- For RP, the situation is a little different:
  - If $w \in L$, then Pr[M accepts w] could be equal to ½.
  - So after many trials, the majority would be just as likely to be correct or incorrect.
- But this isn't useless, because when $w \notin L$, the machine always answers correctly.
- Lemma: Suppose M is a PPT-TM that decides L, $0 \leq \varepsilon < 1$, and

  $w \in L$ implies Pr[M accepts w] $\geq 1 - \varepsilon$.

  $w \notin L$ implies Pr[M rejects w] = 1.

  Then for any ε', $0 \leq \varepsilon' < 1$, there exists M′, another PPT-TM, that decides L with:

  $w \in L$ implies Pr[M accepts w] $\geq 1 - \varepsilon'$.

  $w \notin L$ implies Pr[M rejects w] = 1.

Amplification Lemmas

- Lemma: Suppose M is a PPT-TM that decides L, $0 \leq \varepsilon < 1$,

  w $\in$ L implies Pr[M accepts w] $\geq 1 - \varepsilon$.

  w $\notin$ L implies Pr[M rejects w] = 1.

  Then for any ε', $0 \leq \varepsilon' < 1$, there exists M′, another PPT-TM, that decides L with:

  w $\in$ L implies Pr[M′ accepts w] $\geq 1 - \varepsilon'$.

  w $\notin$ L implies Pr[M′ rejects w] = 1.

- Proof idea:
  - M′: On input w:
    - Run k independent trials of M on w.
    - If any accept, then accept; else reject.
  - Here, choose k such that $\varepsilon^k \leq \varepsilon'$.
  - If w $\notin$ L then all trials reject, so M′ rejects, as needed.
  - If w $\in$ L then each trial accepts with probability $\geq 1 - \varepsilon$, so

    Prob(at least one of the k trials accepts)

    = 1 – Prob(all k reject) $\geq 1 - \varepsilon^k \geq 1 - \varepsilon'$.

Characterization of RP

- Lemma: Suppose M is a PPT-TM that decides L, $0 \leq \varepsilon < 1$,

  $w \in L$ implies Pr[M accepts w] $\geq 1 - \varepsilon$.

  $w \notin L$ implies Pr[M rejects w] = 1.

  Then for any ε', $0 \leq \varepsilon' < 1$, there exists M′, another PPT-TM, that decides L with:

  $w \in L$ implies Pr[M′ accepts w] $\geq 1 - \varepsilon'$.

  $w \notin L$ implies Pr[M′ rejects w] = 1.

- Theorem: $L \in RP$ iff for some ε, $0 \leq \varepsilon < 1$, there is a PPT-TM that decides L with:

  $w \in L$ implies Pr[M accepts w] $\geq 1 - \varepsilon$.

  $w \notin L$ implies Pr[M rejects w] = 1.

RP vs. BPP

- Lemma: Suppose M is a PPT-TM that decides L, $0 \leq \varepsilon < 1$,

  w $\in$ L implies Pr[M accepts w] $\geq 1 - \varepsilon$.

  w $\notin$ L implies Pr[M rejects w] = 1.

  Then for any ε', $0 \leq \varepsilon' < 1$, there exists M′, another PPT-TM, that decides L with:

  w $\in$ L implies Pr[M′ accepts w] $\geq 1 - \varepsilon'$.

  w $\notin$ L implies Pr[M′ rejects w] = 1.
- Theorem: RP $\subseteq$ BPP.
- Proof:
  - Given A $\in$ RP, get (by def. of RP) a PPT-TM M with:

    w $\in$ L implies Pr[M accepts w] $\geq$ ½.

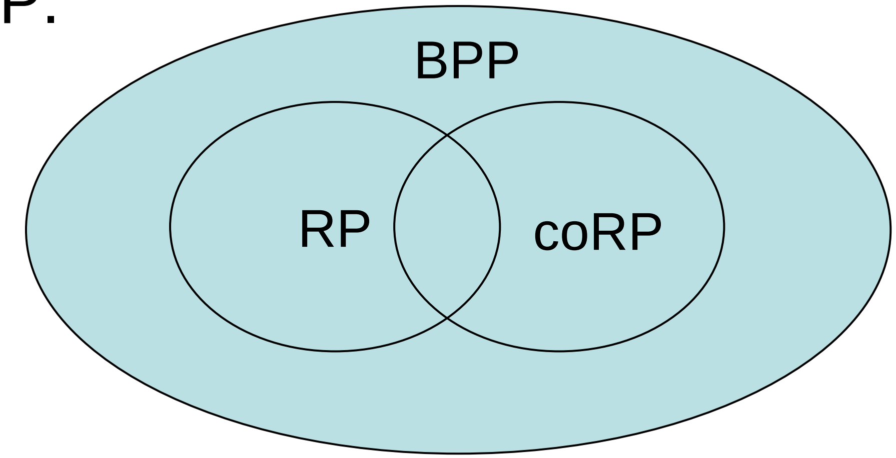
    w $\notin$ L implies Pr[M rejects w] = 1.
  - By Lemma, get another PPT-TM for A, with:

    w $\in$ L implies Pr[M accepts w] $\geq$ 2/3.

    w $\notin$ L implies Pr[M rejects w] = 1.
  - Implies A $\in$ BPP, by definition of BPP.

RP, co-RP, and BPP

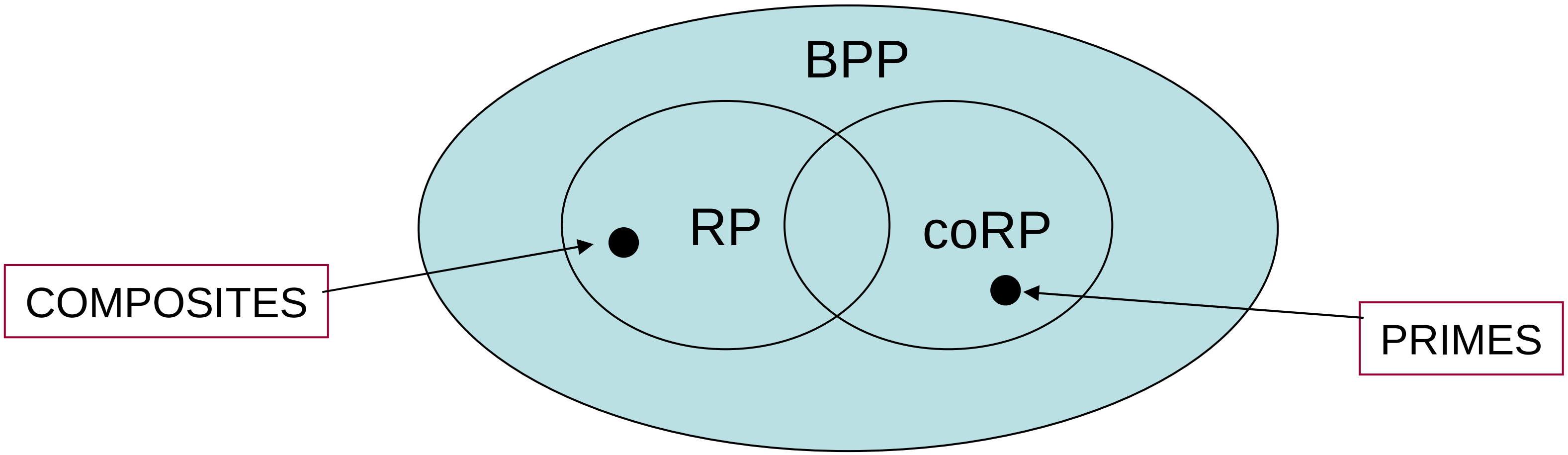
- Definition: coRP = { L | $L^c \in$ RP }
- coRP contains the languages L that can be decided by a PPT-TM that is always correct for w $\in$ L and has error probability at most ½ for w $\notin$ L.
- That is, L is in coRP if there is a PPT-TM that decides L, where:
  - w $\in$ L implies that Pr[M accepts w] = 1, and
  - w $\notin$ L implies that Pr[M rejects w] $\geq$ 1/2.
- Theorem: coRP $\subseteq$ BPP.
- So we have:



Example 1: Primality Testing

Primality Testing

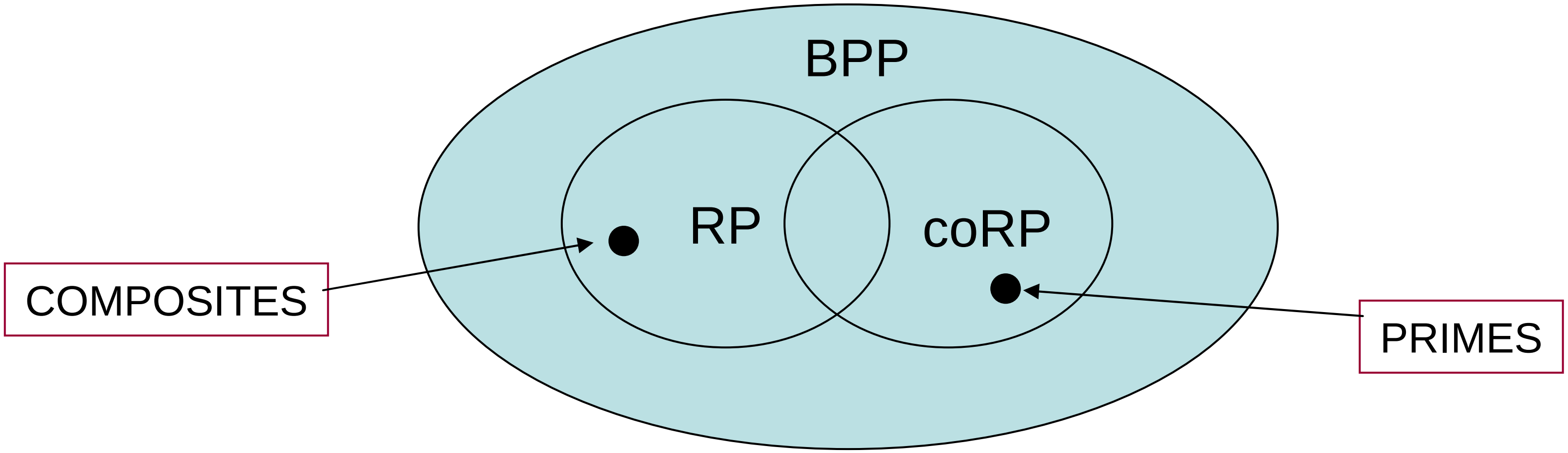
- PRIMES = { <n> | n is a natural number > 1 and n cannot be factored as q r, where 1 < q, r < n }
- COMPOSITES = { <n> | n > 1 and n can be factored…}
- We will show an algorithm demonstrating that PRIMES ∈ coRP.
- So COMPOSITES ∈ RP, and both ∈ BPP.



- This is not exciting, because it is now known that both are in P. [Agrawal, Kayal, Saxema 2002]
- But their poly-time algorithm is hard, whereas the probabilistic algorithm is easy.
- And anyway, this illustrates some nice probabilistic methods.

Primality Testing

- PRIMES = { <n> | n is a natural number > 1 and n cannot be factored as q r, where 1 < q, r < n }
- COMPOSITES = { <n> | n > 1 and n can be factored…}



- Note:
  - Deciding whether n is prime/composite isn't the same as factoring.
  - Factoring seems to be a much harder problem; it's at the heart of modern cryptography.

Primality Testing

- PRIMES = { <n> | n is a natural number > 1 and n cannot be factored as q r, where 1 < q, r < n }
- Show PRIMES $\in$ coRP.
- Design PPT-TM (algorithm) M for PRIMES that satisfies:
  - n $\in$ PRIMES $\Rightarrow$ Pr[M accepts n] = 1.
  - n $\notin$ PRIMES $\Rightarrow$ Pr[M accepts n] $\leq 2^{-k}$.
- Here, k depends on the number of “trials” M makes.
- M always accepts primes, and almost always correctly identifies composites.

- Algorithm rests on some number-theoretic facts about primes (just state them here):

Fermat's Little Theorem

- PRIMES = { <n> | n is a natural number > 1 and n cannot be factored as q r, where 1 < q, r < n }
- Design PPT-TM (algorithm) M for PRIMES that satisfies:
  - $n \in$ PRIMES $\Rightarrow$ Pr[M accepts n] = 1.
  - $n \notin$ PRIMES $\Rightarrow$ Pr[M accepts n] $\leq 2^{-k}$.
- Fact 1: Fermat's Little Theorem: If n is prime and $a \in Z_n^+$ then $a^{n-1} \equiv 1 \mod n$.

Integers mod n except for 0, that is, {1,2,…,n-1}

- Example: $n = 5$, $Z_n^+ = \{1,2,3,4\}$.
  - a = 1: $1^{5-1} = 1^4 = 1 \equiv 1 \mod 5$.
  - a = 2: $2^{5-1} = 2^4 = 16 \equiv 1 \mod 5$.
  - a = 3: $3^{5-1} = 3^4 = 81 \equiv 1 \mod 5$.
  - a = 4: $4^{5-1} = 4^4 = 256 \equiv 1 \mod 5$.

Fermat's test

- Design PPT-TM (algorithm) M for PRIMES that satisfies:
  - $n \in$ PRIMES $\Rightarrow$ Pr[M accepts n] = 1.
  - $n \notin$ PRIMES $\Rightarrow$ Pr[M accepts n] $\leq 2^{-k}$.
- Fermat: If n is prime and $a \in Z_n^+$ then $a^{n-1} \equiv 1 \bmod n$.
- We can use this fact to identify some composites without factoring them:
- Example: n = 8, a = 3.
  - $3^{8-1} = 3^7 \equiv 3 \bmod 8$, not $1 \bmod 8$.
  - So 8 is composite.
- Algorithm attempt 1:
  - On input n:
    - Choose a number a randomly from $Z_n^+ = \{ 1,\ldots,n-1 \}$.
    - If $a^{n-1} \equiv 1 \bmod n$ then accept (passes Fermat test).
    - Else reject (known not to be prime).

Algorithm attempt 1

- Design PPT-TM (algorithm) M for PRIMES that satisfies:
  - $n \in \text{PRIMES} \Rightarrow \Pr[\text{ M accepts n}] = 1$.
  - $n \notin \text{PRIMES} \Rightarrow \Pr[\text{ M accepts n}] \leq 2^{-k}$.
- Fermat: If n is prime and $a \in Z_n^+$ then $a^{n-1} \equiv 1 \bmod n$.
- First try: On input n:
  - Choose number a randomly from $Z_n^+ = \{ 1,\ldots,n-1 \}$.
  - If $a^{n-1} \equiv 1 \bmod n$ then accept (passes Fermat test).
  - Else reject (known not to be prime).
- This guarantees:
  - $n \in \text{PRIMES} \Rightarrow \Pr[\text{ M accepts n}] = 1$.
  - $n \notin \text{PRIMES} \Rightarrow$??
  - Don't know. It could pass the test, and be accepted erroneously.
- The problem isn't helped by repeating the test many times, for many values of a---because there are some non-prime n's that pass the test for all values of a.

Carmichael numbers

- Fermat: If n is prime and $a \in Z_n^+$ then $a^{n-1} \equiv 1 \bmod n$.
- On input n:
  - Choose a randomly from $Z_n^+ = \{ 1,\ldots,n-1 \}$.
  - If $a^{n-1} \equiv 1 \bmod n$ then accept (passes Fermat test).
  - Else reject (known not to be prime).
- Carmichael numbers: Non-primes that pass all Fermat tests, for all values of a.
- Fact 2: Any non-Carmichael composite number fails at least half of all Fermat tests (for at least half of all values of a).
- So for any non-Carmichael composite, the algorithm correctly identifies it as composite, with probability $\geq ½$.
- So, we can repeat k times to get more assurance.
- Guarantees:
  - $n \in$ PRIMES $\Rightarrow$ Pr[M accepts n] = 1.
  - n a non-Carmichael composite number $\Rightarrow$ Pr[M accepts n] $\leq 2^{-k}$.
  - n a Carmichael composite number $\Rightarrow$ Pr[M accepts n] = 1 (wrong)

Carmichael numbers

- Fermat: If n is prime and $a \in Z_n^+$ then $a^{n-1} \equiv 1 \bmod n$.
- On input n:
  - Choose a randomly from $Z_n^+ = \{ 1,\ldots,n-1 \}$.
  - If $a^{n-1} \equiv 1 \bmod n$ then accept (passes Fermat test).
  - Else reject (known not to be prime).
- Carmichael numbers: Non-primes that pass all Fermat tests.
- Algorithm guarantees:
  - $n \in$ PRIMES $\Rightarrow$ Pr[M accepts n] = 1.
  - n a non-Carmichael composite number $\Rightarrow$ Pr[M accepts n] $\leq 2^{-k}$.
  - n a Carmichael composite number $\Rightarrow$ Pr[M accepts n] = 1.
- We must do something about the Carmichael numbers.
- Use another test, based on:
- Fact 3: For every Carmichael composite n, there is some b $\neq 1, -1$ such that $b^2 \equiv 1 \bmod n$ (that is, 1 has a nontrivial square root, mod n). No prime has such a square root.

Primality-testing algorithm

- Fact 3: For every Carmichael composite n, there is some b $\neq$ 1, -1 such that $b^2 \equiv 1$ mod n. No prime has such a square root.
- Primality-testing algorithm: On input n:
  - If n = 1 or n is even: Give the obvious answer (easy).
  - If n is odd and > 1: Choose a randomly from Z_n^+.
    - (Fermat test) If a^{n-1} is not congruent to 1 mod n then reject.
    - (Carmichael test) Write $n - 1 = 2^h s$, where s is odd (factor out twos).
      - Consider successive squares, a^s, a^{2s}, a^{4s}, a^{8s} ..., $a^{2^h s} = a^{n-1}$.
      - If all terms are $\equiv$ 1 mod n, then accept.
      - If not, then find the last one that isn't congruent to 1.
      - If it's $\equiv$ -1 mod n then accept else reject.

Primality-testing algorithm

- If n is odd and > 1:
  - Choose a randomly from Z_n^+.
  - (Fermat test) If a^{n-1} is not congruent to 1 mod n then reject.
  - (Carmichael test) Write $n - 1 = 2^h s$, where s is odd.
    - Consider successive squares, a^s, a^{2s}, a^{4s}, a^{8s} ..., $a^{2^h s} = a^{n-1}$.
    - If all terms are $\equiv$ 1 mod n, then accept.
    - If not, then find the last one that isn't congruent to 1.
    - If it's $\equiv$ -1 mod n then accept else reject.

- Theorem: This algorithm satisfies:
  - $n \in \text{PRIMES} \Rightarrow \Pr[\text{accepts } n] = 1$.
  - $n \notin \text{PRIMES} \Rightarrow \Pr[\text{accepts } n] \le ½$.
- By repeating it k times, we get:
  - $n \notin \text{PRIMES} \Rightarrow \Pr[\text{accepts } n] \le (½)^k$.

Primality-testing algorithm

- If n is odd and > 1:
  - Choose a randomly from Z_n^+.
  - (Fermat test) If a^{n-1} is not congruent to 1 mod n then reject.
  - (Carmichael test) Write $n - 1 = 2^h s$, where s is odd.
    - Consider successive squares, a^{s}, a^{2s}, a^{4s}, a^{8s} ..., $a^{2^h s} = a^{n-1}$.
    - If all terms are $\equiv$ 1 mod n, then accept.
    - If not, then find the last one that isn't congruent to 1.
    - If it's $\equiv$ -1 mod n then accept else reject.

- Theorem: This algorithm satisfies:
  - $n \in \text{PRIMES} \Rightarrow \Pr[\text{accepts } n] = 1$.
  - $n \notin \text{PRIMES} \Rightarrow \Pr[\text{accepts } n] \leq ½$.
- Proof: Suppose n is odd and > 1.

Proof

- If n is odd and > 1:
  - Choose a randomly from Z_n^+.
  - (Fermat test) If a^{n-1} is not congruent to 1 mod n then reject.
  - (Carmichael test) Write $n – 1 = 2^h s$, where s is odd.
    - Consider successive squares, $a^s, a^{2s}, a^{4s}, a^{8s} \ldots, a^{2^h s} = a^{n-1}$.
    - If all terms are $\equiv$ 1 mod n, then accept.
    - If not, then find the last one that isn't congruent to 1.
    - If it's $\equiv$ -1 mod n then accept else reject.
- Proof that $n \in$ PRIMES $\Rightarrow$ Pr[accepts n] = 1.
  - Show that, if the algorithm rejects, then n must be composite.
  - Reject because of Fermat: Then not prime, by Fact 1 (primes pass).
  - Reject because of Carmichael: Then 1 has a nontrivial square root b, mod n, so n isn't prime, by Fact 3.
  - Let b be the last term in the sequence that isn't congruent to 1 mod n.
  - b^2 is the next one, and is $\equiv$ 1 mod n, so b is a square root of 1, mod n.

Proof

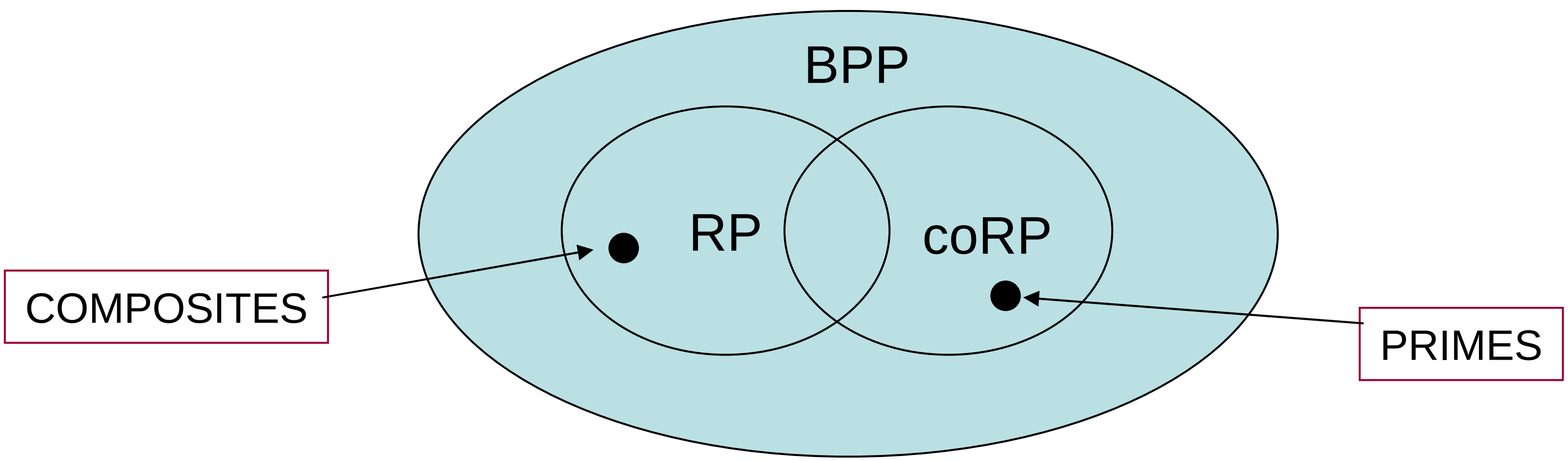
- If n is odd and > 1:
  - Choose a randomly from Z_n^+.
  - (Fermat test) If a^{n-1} is not congruent to 1 mod n then reject.
  - (Carmichael test) Write $n - 1 = 2^h s$, where s is odd.
    - Consider successive squares, a^s, a^{2s}, a^{4s}, a^{8s} ..., $a^{2^h s} = a^{n-1}$.
    - If all terms are $\equiv$ 1 mod n, then accept.
    - If not, then find the last one that isn't congruent to 1.
    - If it's $\equiv$ -1 mod n then accept else reject.
- Proof that $n \notin$ PRIMES $\Rightarrow$ Pr[accepts n] $\leq$ ½.
  - Suppose n is a composite.
  - If n is not a Carmichael number, then at least half of the possible choices of a fail the Fermat test (by Fact 2).
  - If n is a Carmichael number, then Fact 3 says that some b fails the Carmichael test (is a nontrivial square root).
  - Actually, when we generate b using a as above, at least half of the possible choices of a generate bs that fail the Carmichael test.
  - Why: Technical argument, in Sipser, p. 374-375.

Primality-testing algorithm

- So we have proved:
- Theorem: This algorithm satisfies:
  - $n \in$ PRIMES $\Rightarrow$ Pr[accepts n] = 1.
  - $n \notin$ PRIMES $\Rightarrow$ Pr[accepts n] $\leq$ ½.
- This implies:
- Theorem: PRIMES $\in$ coRP.
- Repeating k times, or using an amplification lemma, we get:
  - $n \in$ PRIMES $\Rightarrow$ Pr[accepts n] = 1.
  - $n \notin$ PRIMES $\Rightarrow$ Pr[accepts n] $\leq (½)^k$.
- Thus, the algorithm might sometimes make mistakes and classify a composite as a prime, but the probability of doing this can be made arbitrarily low.
- Corollary: COMPOSITES $\in$ RP.

Primality-testing algorithm

- Theorem: PRIMES $\in$ coRP.
- Corollary: COMPOSITES $\in$ RP.
- Corollary: Both PRIMES and COMPOSITES $\in$ BPP.

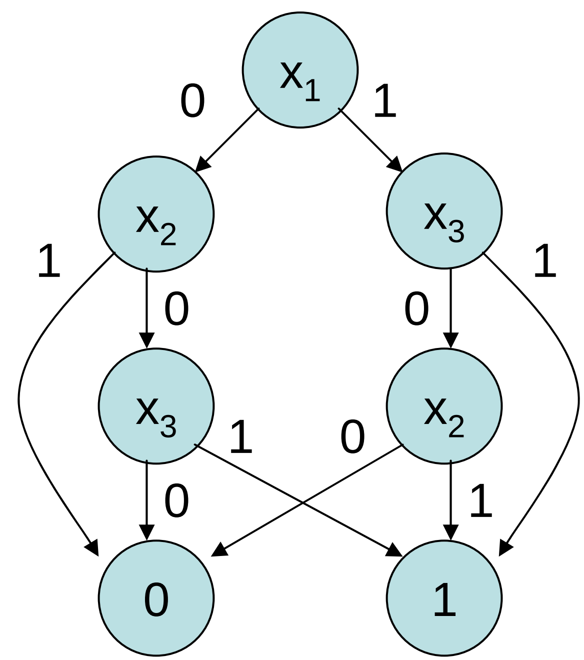


Example 2: Branching-Program Equivalence

Branching Programs

- Branching program: A variant of a decision tree. Can be a DAG, not just a tree:
- Describes a Boolean function of a set $\{ x_1, x_2, x_3, \ldots \}$ of Boolean variables.
- Restriction: Each variable appears at most once on each path.
- Example:

x_1	x_2	x_3	result
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Branching Programs

- Branching program representation for Boolean functions is used by system modeling and analysis tools, for systems in which the state can be represented using just Boolean variables.
- Programs called Binary Decision Diagrams (BDDs).
- Analyzing a model involves exploring all the states, which in turn involves exploring all the paths in the diagram.
- Choosing the “right” order of evaluating the variables can make a big difference in cost (running time).

- Q: Given two branching programs, B_1 and B_2, do they compute the same Boolean function?
- That is, do the same values for all the variables always lead to the same result in both programs?

Branching-Program Equivalence

- Q: Given two branching programs, B_1 and B_2, do they compute the same Boolean function?
- Express as a language problem:

  EQ_{BP} = { < B_1, B_2 > | B_1 and B_2 are BPs that compute the same Boolean function }.

- Theorem: EQ_{BP} is in coRP $\subseteq$ BPP.
- Note: Need the restriction that a variable appears at most once on each path. Otherwise, the problem is coNP-complete.

- Proof idea:
  - Pick random values for x_1, x_2, … and see if they lead to the same answer in B_1 and B_2.
  - If so, accept; if not, reject.
  - Repeat several times for extra assurance.

Branching-Program Equivalence

EQ_{BP} = { < B_1, B_2 > | B_1 and B_2 are BPs that compute the same Boolean function }

- Theorem: EQ_{BP} is in coRP $\subseteq$ BPP.
- Proof idea:
  - Pick random values for x_1, x_2, … and see if they lead to the same answer in B_1 and B_2.
  - If so, accept; if not, reject.
  - Repeat several times for extra assurance.
- This is not quite good enough:
  - Some inequivalent BPs differ on only one assignment to the vars.
  - Unlikely that the algorithm would guess this assignment.
- Better proof idea:
  - Consider the same BPs but now pretend the domain of values for the variables is Z_p, the integers mod p, for a large prime p, rather than just {0,1}.
  - This will let us make more distinctions, making it less likely that we would think B_1 and B_2 are equivalent if they aren't.

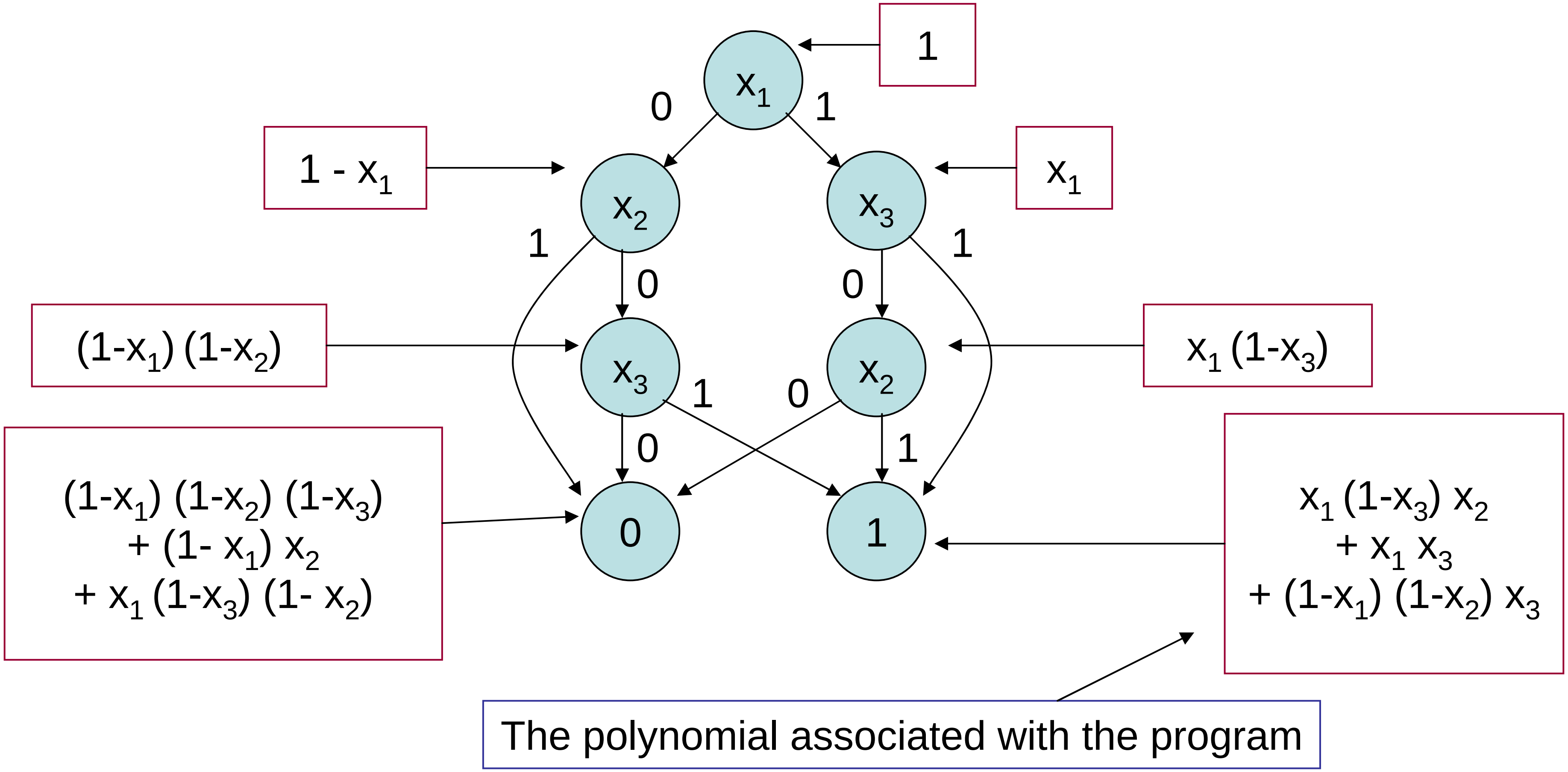
Branching-Program Equivalence

EQ_{BP} = { < B_1, B_2 > | B_1 and B_2 are BPs that compute the same Boolean function }

- Theorem: EQ_{BP} is in coRP $\subseteq$ BPP.
- Proof idea:
  - Pick random values for x_1, x_2, … and see if they lead to the same answer in B_1 and B_2.
  - If so, accept; if not, reject.
  - Repeat several times for extra assurance.
- Better proof idea:
  - Pretend that the domain of values for the variables is Z_p, the integers mod p, for a large prime p, rather than just {0,1}.
  - This lets us make more distinctions, making it less likely that we would think B_1 and B_2 are equivalent if they aren't.
  - But how do we apply the programs to integers mod p?
  - By associating a multi-variable polynomial with each program:

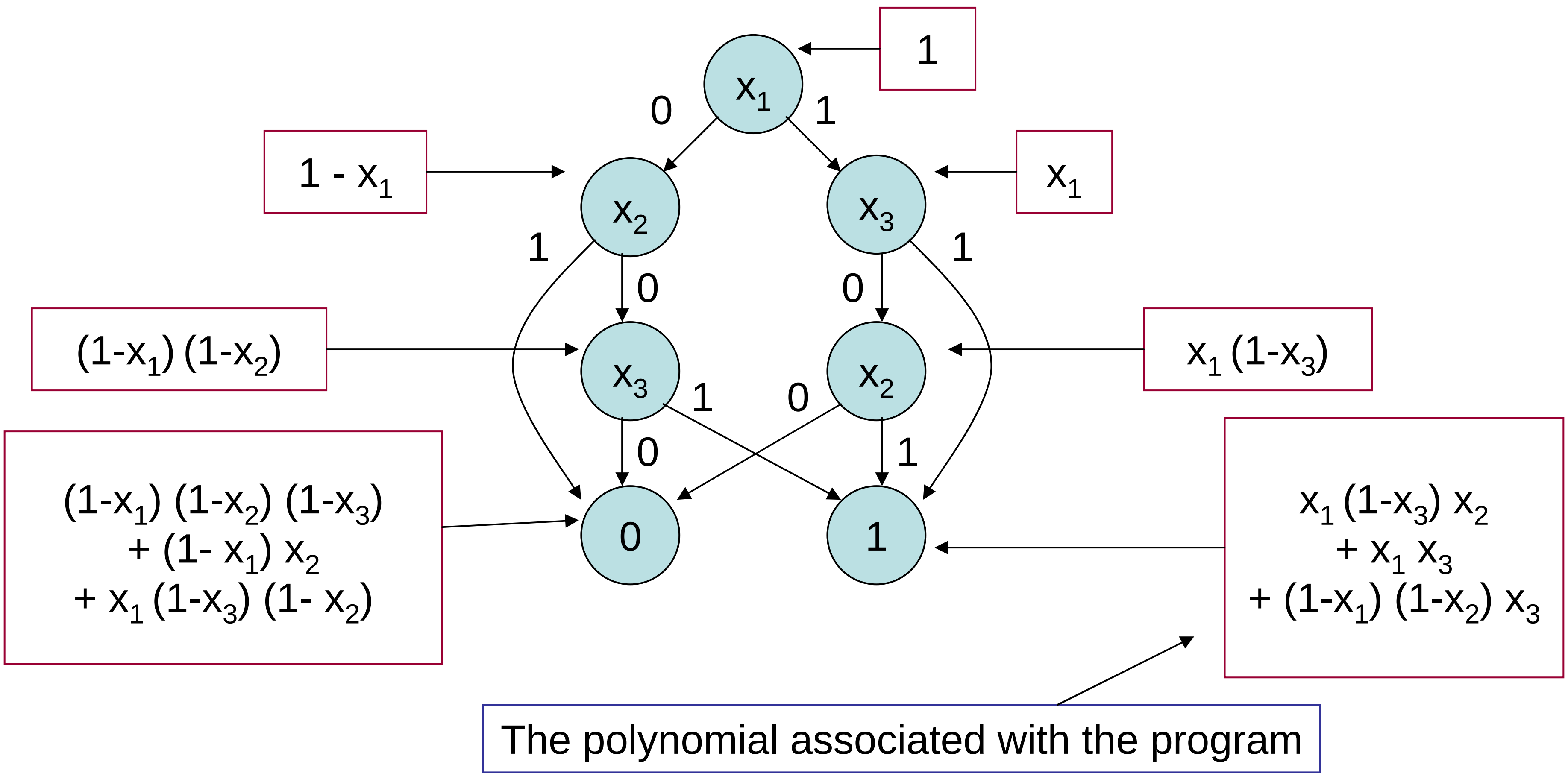
Associating a polynomial with a BP

- Associate a polynomial with each node in the BP, and use the poly associated with the 1-result node as the poly for the entire BP.



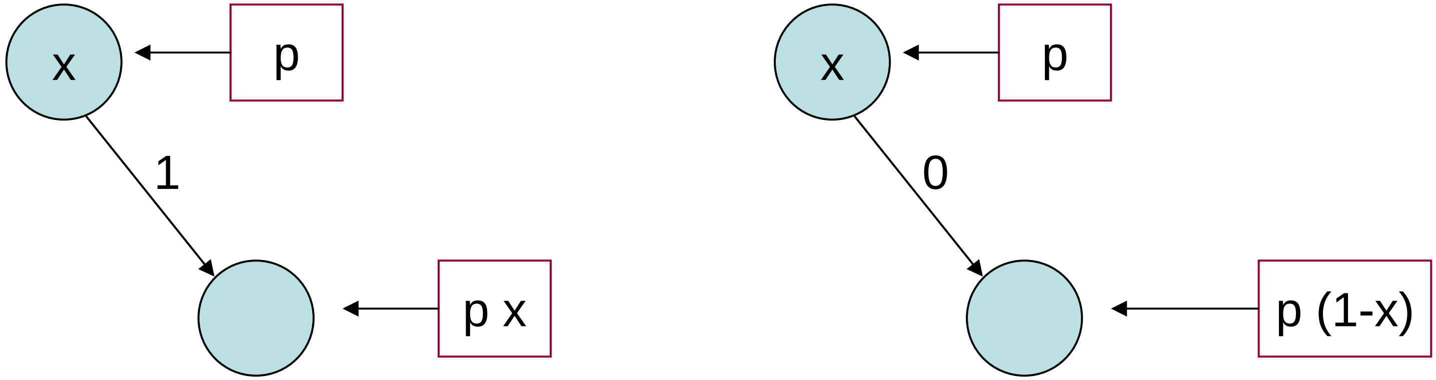
Labeling rules

- Top node: Label with polynomial 1.
- Non-top node: Label with sum of polys, one for each incoming edge:
  - Edge labeled with 1, from x, labeled with p, contributes p x.
  - Edge labeled with 0, from x, labeled with p, contributes p (1-x).



Labeling rules

- Top node: Label with polynomial 1.
- Non-top node: Label with sum of polys, one for each incoming edge:
  - Edge labeled with 1, from x labeled with p, contributes p x.
  - Edge labeled with 0, from x labeled with p, contributes p (1-x).



Associating a polynomial with a BP

- What do these polynomials mean for Boolean values?
- For any particular assignment of { 0, 1 } to the variables, each polynomial at each node evaluates to either 0 or 1 (because of their special form).
- The polynomials on the path followed by that assignment all evaluate to 1, and all others evaluate to 0.
- The polynomial associated with the entire program evaluates to 1 exactly for the assignments that lead there = those that are assigned value 1 by the program.
- Example: Above.
  - The assignments leading to result 1 are:
  - Which are exactly the assignments for which the program's polynomial evaluates to 1.

x_1	x_2	x_3
0	0	1
1	0	1
1	1	0
1	1	1

$x_1 (1-x_3) x_2$
$+ x_1 x_3$
$+ (1-x_1) (1-x_2) x_3$

Branching-Program Equivalence

- Now consider Z_p, integers mod p, for a large prime p (much bigger than the number of variables).

- Equivalence algorithm: On input < B_1, B_2 >, where both programs use m variables:
  - Choose elements $a_1, a_2, \ldots, a_m$ from Z_p at random.
  - Evaluate the polynomials p_1 associated with B_1 and p_2 associated with B_2 for $x_1 = a_1, x_2 = a_2, \ldots, x_m = a_m$.
    - Evaluate them node-by-node, without actually constructing all the polynomials for both programs.
    - Do this in polynomial time in the size of < B_1, B_2 >, LTTR.
  - If the results are equal (mod p) then accept; else reject.

- Theorem: The equivalence algorithm guarantees:
  - If B_1 and B_2 are equivalent BPs (for Boolean values) then Pr[algorithm accepts n] = 1.
  - If B_1 and B_2 are not equivalent, then Pr[algorithm rejects n] $\geq$ 2/3.

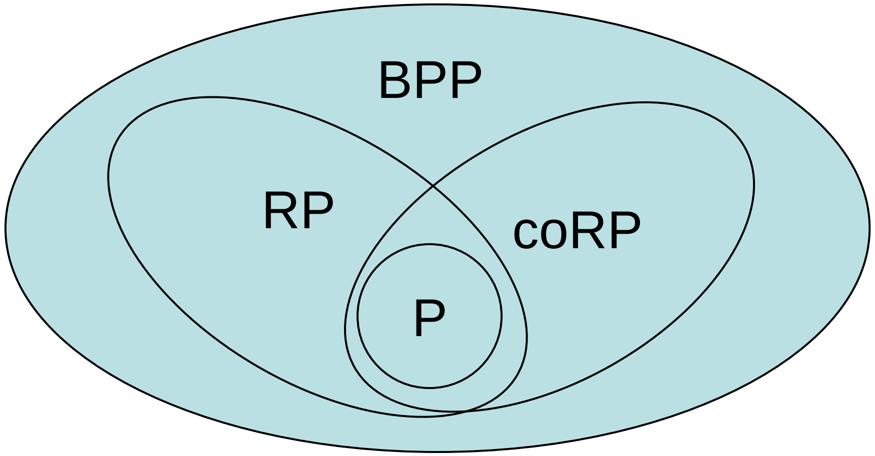
Branching-Program Equivalence

- Equivalence algorithm: On input < B_1, B_2 >:
  - Choose elements $a_1, a_2, \ldots, a_m$ from Z_p at random.
  - Evaluate the polynomials p_1 associated with B_1 and p_2 associated with B_2 for $x_1 = a_1, x_2 = a_2, \ldots, x_m = a_m$.
  - If the results are equal (mod p) then accept; else reject.
- Theorem: The equivalence algorithm guarantees:
  - If B_1 and B_2 are equivalent BPs then Pr[accepts n] = 1.
  - If B_1 and B_2 are not equivalent, then Pr[rejects n] $\geq$ 2/3.
- Proof idea: (See Sipser, p. 379)
  - If B_1 and B_2 are equivalent BPs (for Boolean values), then p_1 and p_2 are equivalent polynomials over Z_p, so always accepts.
  - If B_1 and B_2 are not equivalent (for Boolean values), then at least 2/3 of the possible sets of choices from Z_p yield different values, so Pr[rejects n] $\geq$ 2/3.
- Corollary: $EQ_{BP} \in coRP \subseteq BPP$.

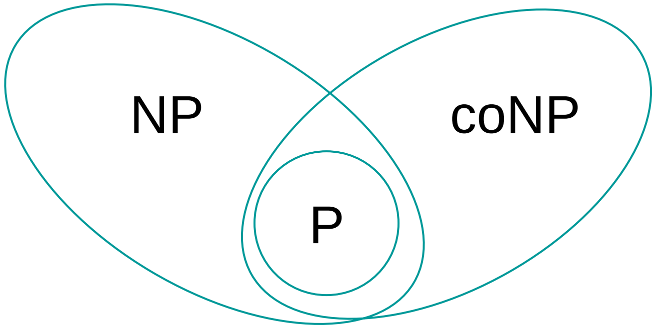
Relationships Between Complexity Classes

Relationships between complexity classes

- We know:



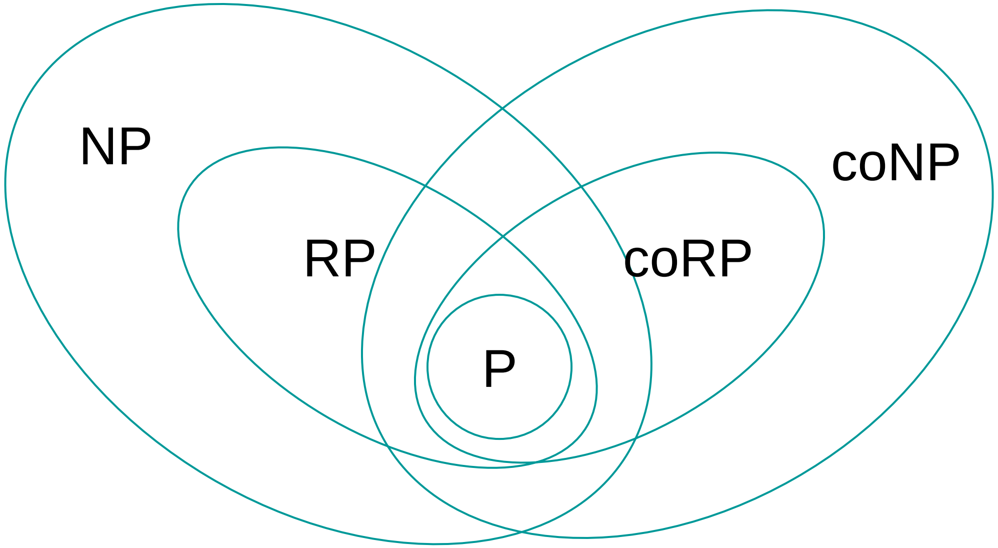
- Also recall:



- From the definitions, RP $\subseteq$ NP and coRP $\subseteq$ coNP.
- So we have:

Relationships between classes

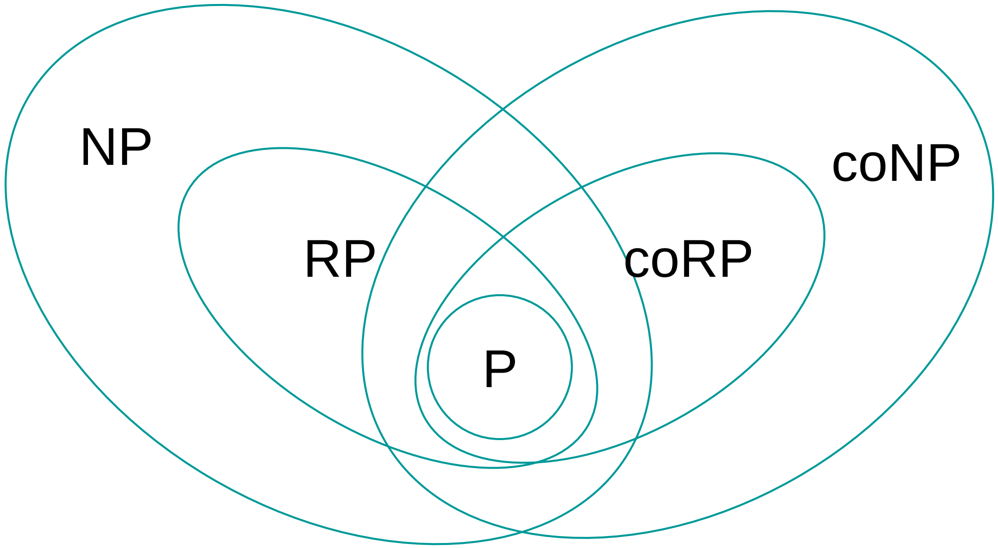
- So we have:



- Q: Where does BPP fit in?

Relationships between classes

- Where does BPP fit?
  - NP $\cup$ coNP $\subseteq$ BPP ?
  - BPP = P ?
  - Something in between ?
- Many people believe BPP = RP = coRP = P, that is, that randomness doesn't help.



- How could this be?
- Perhaps we can emulate randomness with pseudo-random generators---deterministic algorithms whose output "looks random".
- What does it mean to "look random"?
- A polynomial-time TM can't distinguish them from random.
- Current research!

Next time…

- Cryptography!