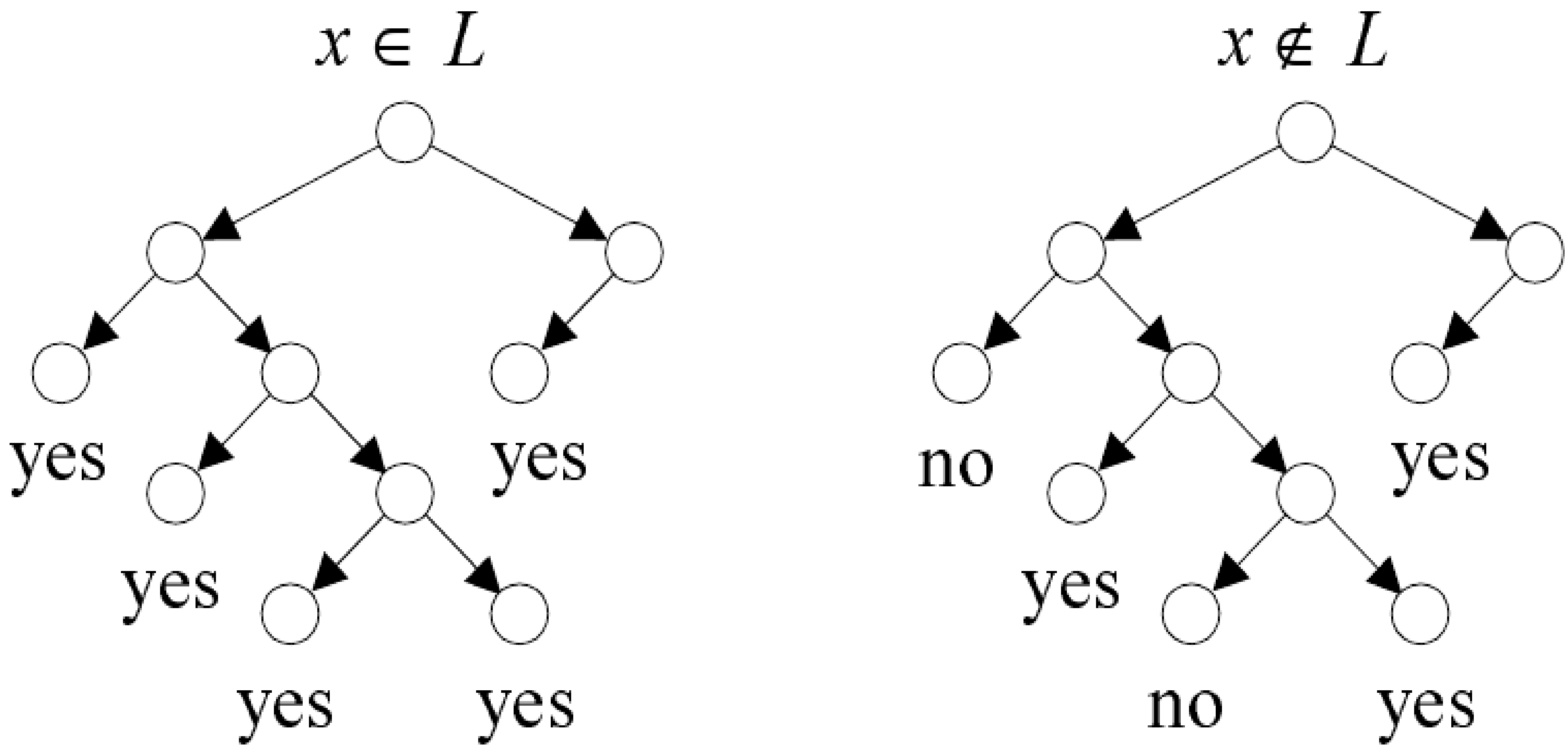
CoNP and Function Problems

coNP

- By definition, coNP is the class of problems whose complement is in NP.
- NP is the class of problems that have succinct certificates.
- coNP is therefore the class of problems that have succinct disqualifications:
  - A “no” instance of a problem in coNP possesses a short proof of its being a “no” instance.
  - Only “no” instances have such proofs.

coNP (continued)

- Suppose L is a coNP problem.
- There exists a polynomial-time nondeterministic algorithm M such that:
  - If $x \in L$, then $M(x)$ = "yes" for all computation paths.
  - If $x \in L$, then $M(x)$ = "no" for some computation path.


$x \in L$
yes
yes
yes
yes
yes
$x \notin L$
no
yes
yes
no
yes

coNP (concluded)

- Clearly P ⊆ coNP.
- It is not known if P = NP ∩ coNP.
  - Contrast this with R = RE ∩ coRE

Some coNP Problems

- VALIDITY $\in$ coNP.
  - If φ is not valid, it can be disqualified very succinctly: a truth assignment that does not satisfy it.
- SAT complement $\in$ coNP.
  - The disqualification is a truth assignment that satisfies it.
- HAMILTONIAN PATH complement $\in$ coNP.
  - The disqualification is a Hamiltonian path.

An Alternative Characterization of coNP

- Let $L \subseteq \Sigma*$ be a language. Then $L \in$ coNP if and only if there is a polynomially decidable and polynomially balanced relation R such that L = {x : \forall y (x, y) ∈ R}.
- $L' = \{x : (x, y) \in \neg R$ for some $y\}$.
- Because $\neg R$ remains polynomially balanced, $L \in$ NP
- Hence $L \in$ coNP by definition.

coNP Completeness

- L is NP-complete if and only if its complement L' = $\Sigma* - L$ is coNP-complete.
- Proof (⇒; the ⇐ part is symmetric)
  - Let L1' be any coNP language.
  - Hence L1 $\in$ NP.
  - Let R be the reduction from L1 to L.
  - So $x \in L1$ if and only if $R(x) \in L$.
  - So $x \in L1'$ if and only if $R(x) \in L'$.
  - R is a reduction from L1' to L'.

Some coNP-Complete Problems

- SAT complement is coNP-complete.
  - SAT complement is the complement of sat.
- VALIDITY is coNP-complete.
  - φ is valid if and only if $\neg \varphi$ is not satisfiable.
  - The reduction from sat complement to VALIDITY is hence easy.
- HAMILTONIAN PATH complement is coNP-complete.

Possible Relations between P, NP, coNP

1. $P = NP = coNP$.
2. $NP = coNP$ but $P \neq NP$.
3. $NP \neq coNP$ and $P \neq NP$.
   - This is current “consensus.”

coNP Hardness and NP Hardness

- If a coNP-hard problem is in NP, then NP = coNP.
- Let L $\in$ NP be coNP-hard.
- Let PNTM M decide L.
- For any L1 $\in$ coNP, there is a reduction R from L1 to L.
- L1 $\in$ NP as it is decided by PNTM M(R(x)).
  - Alternatively, NP is closed under complement.
- Hence coNP $\subseteq$ NP.
- The other direction NP $\subseteq$ coNP is symmetric.

coNP Hardness and NP Hardness (concluded)

- Similarly, If an NP-hard problem is in coNP, then NP = coNP.
- Hence NP-complete problems are unlikely to be in coNP and coNP-complete problems are unlikely to be in NP.

The Primality Problem

- An integer p is prime if $p > 1$ and all positive numbers other than 1 and p itself cannot divide it.
- PRIMES asks if an integer N is a prime number
- Dividing N by 2, 3, . . . , $\sqrt{N}$ is not efficient.
  - The length of N is only $\log N$, but $\sqrt{N} = 2^{0.5 \log N}$.
- A polynomial-time algorithm for primes was not found until 2002 by Agrawal, Kayal, and Saxena!
- We will focus on efficient "probabilistic" algorithms for primes (used in practice).

ΔNP

- $\Delta NP \equiv NP \cap coNP$ is the class of problems that have succinct certificates and succinct disqualifications.
  - Each "yes" instance has a succinct certificate.
  - Each "no" instance has a succinct disqualification.
  - No instances have both.
- $P \subseteq \Delta NP$.
- We will see that primes $\in$ DP.
  - In fact, primes $\in$ P as mentioned earlier.

Primitive Roots in Finite Fields

- Theorem (Lucas and Lehmer (1927)) A number $p > 1$ is prime if and only if there is a number $1<r<p$ (called the primitive root or generator) s.t.
- 1. $r^{p-1} = 1 \bmod p$, and
- 2. $r^{(p-1)/q} = 1 \bmod p$ for all prime divisors q of $p-1$.
- Proof excluded.

Pratt's Theorem

- (Pratt (1975)) PRIMES $\in$ NP $\cap$ coNP.
- primes is in coNP because a succinct disqualification is a divisor.
- Suppose p is a prime.
- p's certificate includes the r in L.L. Theorem
- Use recursive doubling to check if $r^{p-1} = 1 \bmod p$ in time polynomial in the length of the input, log p.
- We also need all prime divisors of $p - 1$: $q1$, $q2$, . . . , qk .
- Checking $r^{(p-1)/qi} \neq 1 \bmod p$ is also easy.

The Proof (concluded)

- Checking $q_1, q_2, \ldots, q_k$ are all the divisors of $p - 1$ is easy.
- We still need certificates for the primality of the q_i's.
- The complete certificate is recursive and tree-like: $C(p) = (r; q_1, C(q_1), q_2, C(q_2), \ldots, q_k, C(q_k))$.
- $C(p)$ can also be checked in polynomial time.
- We next prove that $C(p)$ is succinct.

The Succinctness of the Certificate

- The length of C(p) is at most quadratic at $5 \log^2 p$.
- This claim holds when $p = 2$ or $p = 3$.
- In general, $p - 1$ has $k < \log p$ prime divisors $q1 = 2, q2, \ldots, qk$.
- C(p) requires: 2 parentheses and $2k < 2 \log p$ separators (length at most $2 \log p$ long), r (length at most $\log p$), $q1 = 2$ and its certificate 1 (length at most 5 bits), the qi 's (length at most $2 \log p$), and the C(qi)s.

The Proof (concluded)

- C(p) is succinct because

$$
\begin{aligned}
|C(p)| &\leq 5 \log p + 5 + 5\ \Sigma_{i=2}^{k} \log^2 qi \\
&\leq 5 \log p + 5 + 5\ \ (\Sigma_{i=2}^{k} \log 2\ qi)^2 \\
&\leq 5 \log p + 5 + 5 \log (p-1)/2 \\
&< 5 \log p + 5 + 5(\log 2\ p - 1)^2 \\
&= 5 \log^2 p + 10 - 5 \log 2\ p \leq 5 \log^2 p
\end{aligned}
$$

for $p \geq 4$.

Function Problems

- Decisions problem are yes/no problems (sat, tsp (d),etc.).
- Function problems require a solution (a satisfying truth assignment, a best tsp tour, etc.).
- Optimization problems are clearly function problems.
- What is the relation between function and decision problems?
- Which one is harder?

Function Problems Cannot Be Easier than Decision Problems

- If we know how to generate a solution, we can solve the corresponding decision problem.
  - If you can find a satisfying truth assignment efficiently, then sat is in P.
  - If you can find the best tsp tour efficiently, then tsp(d) is in P.
- But decision problems can be as hard as thecorresponding function problems.

FSAT

- FSAT is this function problem:
  - Let φ(x1 , x2 , . . . , xn) be a boolean expression.
  - If φ is satisfiable, then return a satisfying truth assignment.
  - Otherwise, return “no.”
- We next show that if SAT ∈ P, then FSAT has a polynomial-time algorithm.

An Algorithm for FSAT Using SAT

```
t := ε;
if φ ∈ SAT then
  for i = 1, 2, . . . , n do
    if φ[ xi = true ] ∈ SAT then
      t := t ∪ { xi = true };
      φ := φ[ xi = true ];
    else
      t := t ∪ { xi = false };
      φ := φ[ xi = false ];
    end if
  end for
  return t;
else
  return “no”;
end if
```

Analysis

- There are $\leq n + 1$ calls to the algorithm for SAT
- Shorter boolean expressions than φ are used in each call to the algorithm for sat.
- So if SAT can be solved in polynomial time, so can FSAT.
- Hence SAT and FSAT are equally hard (or easy).

TSP and TSP (d) Revisited

- We are given n cities 1, 2, . . . , n and integer distances dij = dji between any two cities i and j.
- The TSP asks for a tour with the shortest total distance (not just the shortest total distance, as earlier).
  - The shortest total distance must be at most $2^{|x|}$ where x is the input.
- TSP (d) asks if there is a tour with a total distance at most B.
- We next show that if TSP (d) $\in$ P, then TSP has a polynomial-time algorithm.

An Algorithm for tsp Using tsp (d)

- 1: Perform a binary search over interval [0, $2^{|x|}$] by calling tsp (d) to obtain the shortest distance C;
- 2: for i, j = 1, 2, . . . , n do
- 3: Call tsp (d) with B = C and dij = C + 1;
- 4: if “no” then
- 5: Restore dij to old value; {Edge [i, j] is critical.}
- 6: end if
- 7: end for
- 8: return the tour with edges whose dij ≤ C;

Analysis

- An edge that is not on any optimal tour will be eliminated, with its dij set to C + 1.
- An edge which is not on all remaining optimal tours will also be eliminated.
- So the algorithm ends with n edges which are not eliminated.
- There are $O(| x | + n^2)$ calls to the algorithm for tsp (d).
- So if tsp (d) can be solved in polynomial time, so can tsp.
- Hence tsp (d) and tsp are equally hard (or easy).

FNP and FP

- L € NP iff
  there exists poly-time computable $R_L(x,y)$ s.t.

  X € L ⇔ y { |y| ≤ p(|x|) & $R_L(x,y)$ }

  - Note how R_L defines the problem-language L
- The corresponding search problem $\Pi_{R(L)}$ € FNP is: given an x find any y s.t. $R_L(x,y)$ and reply “no” if none exist
  - Are all FNP problems self-reducible like FSAT? [open?]
- FP is the subclass of FNP where we only consider problems for which a poly-time algorithm is known

FP <?> FNP

- A proof a-la-Cook shows that FSAT is FNP-complete
- Hence, if FSAT$\in$FP then FNP = FP
- But we showed self-reducibility for SAT, so the theorem follows:
- **Theorem:** FP = FNP iff P=NP

TFNP

- What happens if the relation R is total?

i.e., for each x there is at least one y s.t. R(x,y)

- Define TFNP to be the subclass of FNP where the relation R is total
  - TFNP contains problems that always have a solution, e.g. factoring, fix-point theorems, graph-theoretic problems, …
  - How do we know a solution exists?

  By an “inefficient proof of existence”, i.e. non-(efficiently)-constructive proof
- The idea is to categorize the problems in TFNP based on the type of inefficient argument that guarantees their solution

Properties of TFNP

1. FP ⊆TFNP ⊆FNP
2. TFNP = F(NP ∩ coNP)
   - NP = problems with “yes” certificate y s.t. $R_1(x,y)$
   - coNP = problems with “no” certificate z s.t. $R_2(x,y)$
   - for TFNP   F(NP ∩ coNP) take R = R_1 U R_2
   - for F(NP ∩ coNP) TFNP take R_1 = R and R_2 = ø
3. There is an FNP-complete problem in TFNP iff NP = coNP
   - ➔: If NP = coNP then trivially FNP = TFNP
   - ➔: If the FNP-complete problem Π_R is in TFNP then: FSAT reduces to Π_R via f and g, hence any unsatisfiable formula φ has a “no” certificate y, s.t. R(f(φ),y) (y exists since Π_R is in TFNP) and g(y)=“no”
4. TFNP is a semantic complexity class ➔ no complete problems!
   - note how telling whether a relation is total is undecidable (and not even RE!!)

ANOTHER HC is in TFNP

- Thm: any graph with odd degrees has an even number of HC through edge xy
- Proof Idea:
  - take a HC
  - remove edge (1,2) & take a HP
  - fix endpoint 1 and start “rotating” from the other end
  - each HP has two “valid” neighbors (d=3) except for those paths with endpoints 1,2