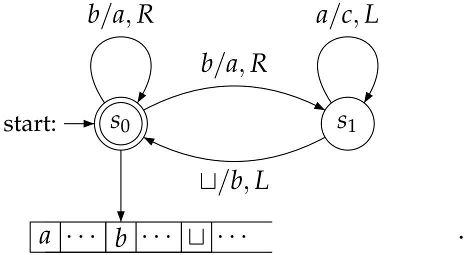
CS 208: Automata Theory and Logic

Part II, Lecture 4: PCP and Complexity

S Akshay



Department of Computer Science and Engineering,
Indian Institute of Technology Bombay.

Post's correspondence problem (PCP)

Post’s correspondence problem (PCP)

A dominoes matching puzzle

Can we arrange a set of domino tiles in such a way that the numbers read on top and bottom add up to the same?

- Not all dominoes need to be used
- Each domino can be used more than once

Post's correspondence problem (PCP)

PCP: A language-theoretic dominoes matching problem

Consider a set of dominoes as couples of strings, $a_i, b_i \in \Sigma^*$:

$$P = \left\{ \left[\frac{a_1}{b_1}\right], \left[\frac{a_2}{b_2}\right], \ldots, \left[\frac{a_k}{b_k}\right] \right\}$$

Does there exist a sequence $i_1, \ldots i_\ell$ such that the string read by the dominoes match? That is, $a_{i_1} \ldots a_{i_\ell} = b_{i_1} \ldots b_{i_\ell}$.

For e.g., a collection of dominoes may look like:

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

Then, a match/solution to the puzzle is:

$$\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right]$$

Post's correspondence problem (PCP)

PCP: A language-theoretic dominoes matching problem

Consider a set of dominoes as couples of strings, $a_i, b_i \in \Sigma^*$:

$$P = \left\{ \left[\frac{a_1}{b_1}\right], \left[\frac{a_2}{b_2}\right], \ldots, \left[\frac{a_k}{b_k}\right] \right\}$$

Does there exist a sequence $i_1, \ldots i_\ell$ such that the string read by the dominoes match? That is, $a_{i_1} \ldots a_{i_\ell} = b_{i_1} \ldots b_{i_\ell}$.

For e.g., a collection of dominoes may look like:

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

Then, a match/solution to the puzzle is:

$$\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right]$$

This problem is unsolvable by algorithms!

PCP is undecidable

Theorem

The Post's correspondence problem is undecidable for $|\Sigma| \geq 2$.

PCP is undecidable

Theorem

The Post's correspondence problem is undecidable for $|\Sigma| \geq 2$.

Proof sketch

- Step 1: Reduce to Modified PCP (MPCP) MPCP $=\{\langle P\rangle \mid P$ is an inst of PCP with a match starting from first domino.
- Step 2: Reduction from L_{TM}^{A} to MPCP. We construct MPCP P' whose matching/soln will solve the TM-acceptance problem.

PCP is undecidable

Theorem

The Post's correspondence problem is undecidable for $|\Sigma| \geq 2$.

Proof sketch

- Step 1: Reduce to Modified PCP (MPCP) MPCP $=\{\langle P\rangle \mid P$ is an inst of PCP with a match starting from first domino.
- Step 2: Reduction from L_{TM}^{A} to MPCP. We construct MPCP P' whose matching/soln will solve the TM-acceptance problem.
- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$.

1. Put $[\frac{\#}{\# q_0 \ldots w_n \#}]$ as first domino in P'.

Proof Contd.

2. for every tape alphabet a, b and states q, r s.t. $q \neq q_{rej}$

$$\text{if } \delta(q, a) = (r, b, R) \text{ put } [\frac{qa}{br}] \text{ in } P'$$

3. for every tape alphabet a, b, c and states q, r s.t. $q \neq q_{rej}$

$$\text{if } \delta(q, a) = (r, b, L) \text{ put } [\frac{cqa}{rcb}] \text{ in } P'$$

4. for tape alphabet a put $[\frac{a}{a}]$ in P'. (see board for sim)

Proof Contd.

2. for every tape alphabet a, b and states q, r s.t. $q \neq q_{rej}$

$$\text{if } \delta(q,a) = (r,b,R) \text{ put } [\frac{qa}{br}] \text{ in } P'$$

3. for every tape alphabet a, b, c and states q, r s.t. $q \neq q_{rej}$

$$\text{if } \delta(q,a) = (r,b,L) \text{ put } [\frac{cqa}{rcb}] \text{ in } P'$$

4. for tape alphabet a put $[\frac{a}{a}]$ in P'. (see board for sim)
5. for $\#$, put $[\frac{\#}{\#}]$ and $[\frac{\#}{\sqcup\#}]$ in P'.
6. for every tape alphabet a, put $[\frac{aq_{acc}}{q_{acc}}]$ and $[\frac{q_{acc}a}{q_{acc}}]$ in P'.
7. Complete by adding $[\frac{q_{acc}\#\#}{\#}]$ in P'.

Formal definition of mapping reducibility

- To reduce problem A to B, we use a computable function to convert instances of A to instances of B. Then we can solve A with a solver for B.

Formal definition of mapping reducibility

- To reduce problem A to B, we use a computable function to convert instances of A to instances of B. Then we can solve A with a solver for B.

A function $f : \Sigma^* \to \Sigma^*$ is called computable if there exists a TM M, which on every input w halts with just $f(w)$ on its tape.

Formal definition of mapping reducibility

- To reduce problem A to B, we use a computable function to convert instances of A to instances of B. Then we can solve A with a solver for B.

A function $f : \Sigma^* \to \Sigma^*$ is called computable if there exists a TM M, which on every input w halts with just $f(w)$ on its tape.

Formal definition of reduction

A language A is mapping reducible to B (denoted $A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$ s.t., for every w

$$w \in A \text{ iff } f(w) \in B$$

The function f is called the reduction of A to B.

So, to check if $w \in A$, use the reduction to map w to $f(w)$ and check if $f(w) \in B$.

Mapping reducibility

Theorem

1. If $A \leq_m B$ and B is decidable (resp. R.E), then A is decidable (resp. R.E).
2. If $A \leq_m B$ and A is decidable (resp. R.E), then B is decidable (resp. R.E).

Mapping reducibility

Theorem

1. If $A \leq_m B$ and B is decidable (resp. R.E), then A is decidable (resp. R.E).
2. If $A \leq_m B$ and A is decidable (resp. R.E), then B is decidable (resp. R.E).

Proof of 1. for decidable:

- Let M be the decider of B and f the reduction from A to B.
- Then define N a decider for A as follows: On input w
  1. Compute $f(w)$
  2. Run M on input $f(w)$ and output whatever M outputs.

Time Complexity

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.

Time Complexity

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.
- The time complexity of M is the function $T(n)$ that is the maximum, over all inputs w of length n, of the running time of M on w.

A time complexity class $TIME(t(n))$ is the set of all languages that can be decided by a TM in $O(t(n))$ time.

Time Complexity

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.
- The time complexity of M is the function $T(n)$ that is the maximum, over all inputs w of length n, of the running time of M on w.

A time complexity class $TIME(t(n))$ is the set of all languages that can be decided by a TM in $O(t(n))$ time.

- Every multi-tape TM with time complexity $t(n)$ can be simulated by a single-tape TM with time complexity $O(t^2(n))$.
- Every non-deterministic single-tape TM with time complexity $t(n)$ can be simulated by a deterministic single-tape TM with time complexity $2^{O(t(n))}$.

The complexity classes $\mathcal{P}$ and $\mathcal{NP}$

The class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in poly-time on a det 1-tape TM.
- $\mathcal{P} = \bigcup_k TIME(n^k)$.

The complexity classes $\mathcal{P}$ and $\mathcal{NP}$

The class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in poly-time on a det 1-tape TM.
- $\mathcal{P} = \bigcup_k TIME(n^k)$.

The class $\mathcal{NP}$

- $\mathcal{NP}$ is the class of languages decidable in poly-time on a non-det 1-tape TM.
- $\mathcal{NP}$ is the class of languages that guess a poly-length string and then verify membership in $\mathcal{P}$ (poly-time).

The complexity classes $\mathcal{P}$ and $\mathcal{NP}$

The class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in poly-time on a det 1-tape TM.
- $\mathcal{P} = \bigcup_k TIME(n^k)$.

The class $\mathcal{NP}$

- $\mathcal{NP}$ is the class of languages decidable in poly-time on a non-det 1-tape TM.
- $\mathcal{NP}$ is the class of languages that guess a poly-length string and then verify membership in $\mathcal{P}$ (poly-time).

Obviously $\mathcal{P} \subseteq \mathcal{NP}$, but the question is:

** **Is $\mathcal{P} = \mathcal{NP}$?** **

Examples of problems in $\mathcal{P}$ and $\mathcal{NP}$

Problems in $\mathcal{P}$

- *PATH*: In a directed graph G, is there a path from vertices s to t.
- *PRIMES*: Is a given number prime? (Solved by Agrawal-Kayal-Saxena in 2002).

Problems in $\mathcal{NP}$

- *HAMPATH*: In a directed graph G, is there a path from vertices s to t, which visits each vertex exactly once.
- *k-CLIQUE*: Does a given undir graph have a clique of size k?

NP-completeness

NP-complete problems

A class of languages of $\mathcal{NP}$ such that if one of them is in $\mathcal{P}$, then all of $\mathcal{NP}$ is in $\mathcal{P}$.

NP-completeness

NP-complete problems

A class of languages of $\mathcal{NP}$ such that if one of them is in $\mathcal{P}$, then all of $\mathcal{NP}$ is in $\mathcal{P}$.

Satisfiability (SAT)

- Boolean variables $x, y, z, \ldots$ taking values 0 (false) or 1 (true).
- Boolean operations: AND, OR and NOT.
- Boolean formulas: $\varphi = (\neg x \wedge y) \vee (x \wedge \neq z)$.
- A satisfying assignment is an assignment $x = 0, y = 1, z = 0$ s.t the formula evaluate to 1 (true)?
- A formula is satisfiable if it has a satisfying assignment.

Qn: Given a formula φ, is it satisfiable?

NP-completeness

NP-complete problems

A class of languages of $\mathcal{NP}$ such that if one of them is in $\mathcal{P}$, then all of $\mathcal{NP}$ is in $\mathcal{P}$.

Satisfiability (SAT)

- Boolean variables $x, y, z, \ldots$ taking values 0 (false) or 1 (true).
- Boolean operations: AND, OR and NOT.
- Boolean formulas: $\varphi = (\neg x \wedge y) \vee (x \wedge \neq z)$.
- A satisfying assignment is an assignment $x = 0, y = 1, z = 0$ s.t the formula evaluate to 1 (true)?
- A formula is satisfiable if it has a satisfying assignment.

Qn: Given a formula φ, is it satisfiable?

Theorem (Cook-Levin '70s)

$SAT \in \mathcal{P}$ iff $\mathcal{P} = \mathcal{NP}$

Poly-time reducibility

A function $f : \Sigma^* \to \Sigma^*$ is called poly-time computable if there exists a poly-time TM M, which on every input w halts with just $f(w)$ on its tape.

Poly-time reducibility

A function $f : \Sigma^* \rightarrow \Sigma^*$ is called poly-time computable if there exists a poly-time TM M, which on every input w halts with just $f(w)$ on its tape.

Formal definition of reduction

A language A is P-time reducible to B (denoted $A \leq_m B$) if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$ s.t., for every w

$$w \in A \text{ iff } f(w) \in B$$

The function f is called the P-time reduction of A to B.

Poly-time reducibility

A function $f : \Sigma^* \to \Sigma^*$ is called poly-time computable if there exists a poly-time TM M, which on every input w halts with just $f(w)$ on its tape.

Formal definition of reduction

A language A is P-time reducible to B (denoted $A \leq_m B$) if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$ s.t., for every w

$$w \in A \text{ iff } f(w) \in B$$

The function f is called the P-time reduction of A to B.

NP-complete problems

B is NP-complete if $B \in \mathcal{NP}$ and every $A \in \mathcal{NP}$ is P-time reducible to B.

Thus, to show that a problem B is NP-complete it suffices to show a P-time reduction to an already known NP-complete problem (e.g., SAT) to B.

Examples of NP-complete problems

- *SAT* was the first example of an NP-complete problem.
  - For proof, read Hopcroft-Motwani-Ullman or Sipser.
- But now by showing *P*-time reduction from *SAT* we can easily show other *NP*-complete problems!

Some *NP*-complete problems (prove by reduction!)

- *3-SAT*: satisfiability of 3-CNF formulae. E.g.,$(\neg x \vee y \vee \neg z) \wedge (\neg y \vee \neg z)$.
- *k-CLIQUE, HAMPATH*: As defined before.
- *3COLOR*: Can the vertices of a graph be colored with 3 colors so that no 2 adj vertices have the same color?
- Bounded PCP: Given PCP instance $\{\left[\frac{a_1}{b_1}\right], \left[\frac{a_2}{b_2}\right], \ldots, \left[\frac{a_k}{b_k}\right]\}$ and a bound L, does there exist a sequence $i_1, \ldots i_\ell$ of length at most L, i.e., $\ell \leq L$ s.t $a_{i_1} \ldots a_{i_\ell} = b_{i_1} \ldots b_{i_\ell}$.