Finishing touches!

- Part I: models of computation (reg exps, DFA/NFA, CFGs, TMs)
- Part II: (efficient) algorithm design
- Part III: intractability via reductions
    - Undecidablity: problems that have no algorithms
    - NP-Completeness: problems unlikely to have efficient algorithms unless $P = NP$

CS/ECE-374: Lecture 22 - Reductions

Lecturer: Nickvash Kani
Chat moderator: Samir Khan

April 15, 2021

University of Illinois at Urbana-Champaign

Finishing touches!

- Part I: models of computation (reg exps, DFA/NFA, CFGs, TMs)
- Part II: (efficient) algorithm design
- Part III: intractability via reductions
    - Undecidablity: problems that have no algorithms
    - NP-Completeness: problems unlikely to have efficient algorithms unless $P = NP$

Turing Machines and Church-Turing Thesis

Turing defined TMs as a machine model of computation

Church-Turing thesis: any function that is computable can be computed by TMs

Efficient Church-Turing thesis: any function that is computable can be computed by TMs with only a polynomial slow-down

Computability and Complexity Theory

- What functions can and *cannot* be computed by TMs?
- What functions/problems can and cannot be solved *efficiently*?

Why?

- Foundational questions about computation
- Pragmatic: Can we solve our problem or not?
- Are we not being clever enough to find an efficient algorithm or should we stop because there isn't one or likely to be one?

Reductions to Prove Intractability

A general methodology to prove impossibility results.

- Start with some *known* hard problem X
- *Reduce* X to your favorite problem Y

If Y can be solved then so can $X \Rightarrow Y$ is also *hard*

Reductions to Prove Intractability

A general methodology to prove impossibility results.

- Start with some *known* hard problem X
- *Reduce* X to your favorite problem Y

If Y can be solved then so can $X \Rightarrow Y$ is also *hard*

Caveat: In algorithms we reduce new problem to known solved one!

Reductions to Prove Intractability

A general methodology to prove impossibility results.

- Start with some *known* hard problem X
- *Reduce* X to your favorite problem Y

If Y can be solved then so can $X \Rightarrow Y$ is also *hard*

Caveat: In algorithms we reduce new problem to known solved one!

Who gives us the initial hard problem?

- Some clever person (Cantor/Gödel/Turing/Cook/Levin ...) who establish hardness of a fundamental problem
- Assume some core problem is hard because we haven't been able to solve it for a long time. This leads to *conditional* results

Reduction Question

A general methodology to prove impossibility results.

- Start with some *known* hard problem X
- *Reduce* X to your favorite problem Y

If Y can be solved then so can $X \Rightarrow Y$ is also *hard*

What if we want to prove a problem is easy?

Decision Problems, Languages, Terminology

When proving hardness we limit attention to *decision* problems

- A decision problem Π is a collection of instances (strings)
- For each instance I of Π, answer is YES or NO
- Equivalently: boolean function $f_\Pi : \Sigma^* \to \{0, 1\}$ where $f(I) = 1$ if I is a YES instance, $f(I) = 0$ if NO instance
- Equivalently: language $L_\Pi = \{I \mid I \text{ is a YES instance}\}$

Decision Problems, Languages, Terminology

When proving hardness we limit attention to *decision* problems

- A decision problem Π is a collection of instances (strings)
- For each instance I of Π, answer is YES or NO
- Equivalently: boolean function $f_\Pi : \Sigma^* \to \{0,1\}$ where $f(I) = 1$ if I is a YES instance, $f(I) = 0$ if NO instance
- Equivalently: language $L_\Pi = \{I \mid I \text{ is a YES instance}\}$

Notation about encoding: distinguish I from encoding $\langle I \rangle$

- n is an integer. $\langle n \rangle$ is the encoding of n in some format (could be unary, binary, decimal etc)
- G is a graph. $\langle G \rangle$ is the encoding of G in some format
- M is a TM. $\langle M \rangle$ is the encoding of TM as a string according to some fixed convention

Decision Problems, Languages, Terminology

Aside: Different problems can be formulated differently.
Example: Traveling Salesman

Common Formulation: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Decision Formulation: Given a list of cities and the distances between each pair of cities, is there a route route that visits each city exactly once and returns to the origin city **while having a shorter length than integer k**.

Examples

- Given directed graph G, is it strongly connected? $\langle G \rangle$ is a YES instance if it is, otherwise NO instance
- Given number n, is it a prime number? $L_{PRIMES} = \{\langle n \rangle \mid n \text{ is prime}\}$
- Given number n is it a composite number? $L_{COMPOSITE} = \{\langle n \rangle \mid n \text{ is a composite}\}$
- Given $G = (V, E), s, t, B$ is the shortest path distance from s to t at most B? Instance is $\langle G, s, t, B \rangle$

Reductions: Overview

Reductions for decision problems|languages

For languages L_X, L_Y, a *reduction from L_X to L_Y* is:

- An algorithm ...
- Input: $w \in \Sigma^*$
- Output: $w' \in \Sigma^*$
- Such that:

$$\boxed{w \in L_X} \iff \boxed{w' \in L_Y}$$

Reductions for decision problems/languages

For decision problems X, Y, a *reduction from X to Y* is:

- An algorithm ...
- Input: I_X, an instance of X.
- Output: I_Y an instance of Y.
- Such that:

$$\boxed{I_Y \text{ is YES instance of } Y} \iff \boxed{I_X \text{ is YES instance of } X}$$

Using reductions to solve problems

- $\mathcal{R}$: Reduction $X \to Y$
- $\mathcal{A}_Y$: algorithm for Y:

Using reductions to solve problems

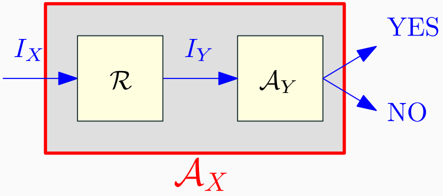
- $\mathcal{R}$: Reduction $X \to Y$
- $\mathcal{A}_Y$: algorithm for Y:
- $\implies$ New algorithm for X:

```
$\mathcal{A}_X(I_X)$:
          // $I_X$: instance of $X$.
          $I_Y \Leftarrow \mathcal{R}(I_X)$
          return $\mathcal{A}_Y(I_Y)$
```

Using reductions to solve problems

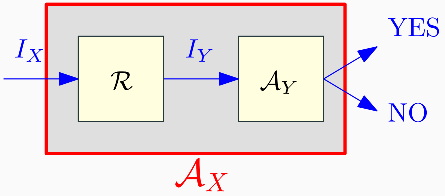
- $\mathcal{R}$: Reduction $X \to Y$
- $\mathcal{A}_Y$: algorithm for Y:
- $\implies$ New algorithm for X:

```
$\mathcal{A}_X(I_X)$:
        // $I_X$: instance of $X$.
        $I_Y \Leftarrow \mathcal{R}(I_X)$
        return $\mathcal{A}_Y(I_Y)$
```



In particular, if $\mathcal{R}$ and $\mathcal{A}_Y$ are polynomial-time algorithms, $\mathcal{A}_X$ is also polynomial-time.

Reductions and running time

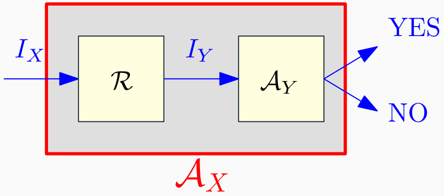


$R(n)$: running time of $\mathcal{R}$

$Q(n)$: running time of $\mathcal{A}_Y$

Question: What is running time of $\mathcal{A}_X$?

Reductions and running time



$R(n)$: running time of $\mathcal{R}$

$Q(n)$: running time of $\mathcal{A}_Y$

Question: What is running time of $\mathcal{A}_X$? $O(Q(R(n))$. Why?

- If I_X has size n, $\mathcal{R}$ creates an instance I_Y of size at most $R(n)$
- $\mathcal{A}_\mathcal{Y}$'s time on I_Y is by definition at most $Q(|I_Y|) \leq Q(R(n))$.

Example: If $R(n) = n^2$ and $Q(n) = n^{1.5}$ then $\mathcal{A}_X$ is $O(n^2 + n^3)$

Comparing Problems

- Reductions allow us to formalize the notion of "Problem X is no harder to solve than Problem Y".
- If Problem X reduces to Problem Y (we write $X \leq Y$), then X cannot be harder to solve than Y.
- More generally, if $X \leq Y$, we can say that X is no harder than Y, or Y is at least as hard as X. $X \leq Y$:
    - X is no harder than Y, or
    - Y is at least as hard as X.

Examples of Reductions

Independent Sets and Cliques

Given a graph G, a set of vertices V' is:

Independent Sets and Cliques

Given a graph G, a set of vertices V' is:

- An *independent set*: ifno two vertices of V' areconnected by an edgeof G.

Independent Sets and Cliques

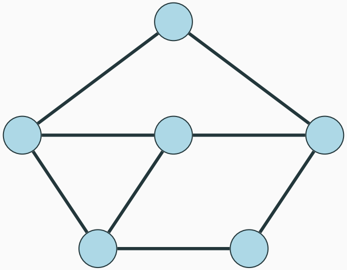
Given a graph G, a set of vertices V' is:

- An *independent set*: ifno two vertices of V' areconnected by an edgeof G.
- *clique*: *every* pair of vertices in V' is connected by an edge of G.

Independent Sets and Cliques

Given a graph G, a set of vertices V' is:

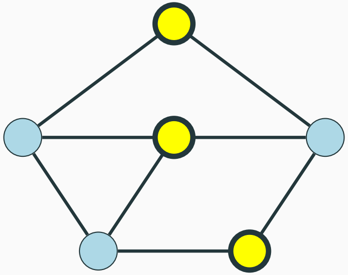
- An *independent set*: ifno two vertices of V' areconnected by an edgeof G.
- *clique*: *every* pair of vertices in V' is connected by an edge of G.



Independent Sets and Cliques

Given a graph G, a set of vertices V' is:

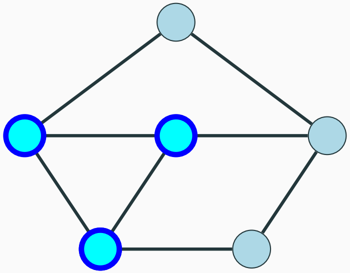
- An *independent set*: ifno two vertices of V' areconnected by an edgeof G.
- *clique*: *every* pair of vertices in V' is connected by an edge of G.



Independent Sets and Cliques

Given a graph G, a set of vertices V' is:

- An *independent set*: ifno two vertices of V' areconnected by an edgeof G.
- *clique*: *every* pair of vertices in V' is connected by an edge of G.



The Independent Set and Clique Problems

Problem: Independent Set

Instance: A graph G and an integer k.

Question: Does G has an independent set of size $\geq k$?

The Independent Set and Clique Problems

Problem: Independent Set

Instance: A graph G and an integer k.
Question: Does G has an independent set of size $\geq k$?

Problem: Clique

Instance: A graph G and an integer k.
Question: Does G has a clique of size $\geq k$?

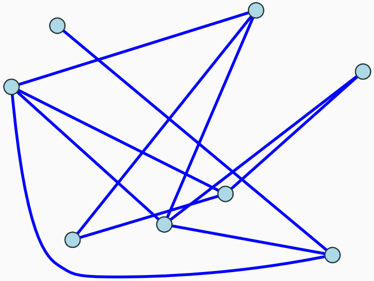
Recall

For decision problems X, Y, a reduction from X to Y is:

- An algorithm …
- that takes I_X, an instance of X as input …
- and returns I_Y, an instance of Y as output …
- such that the solution (YES/NO) to I_Y is the same as the solution to I_X.

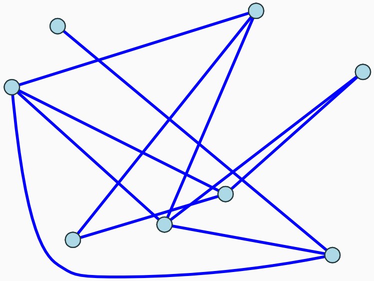
Reducing Independent Set to Clique

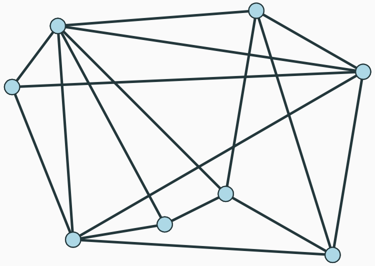
An instance of **Independent Set** is a graph G and an integer k.



Reducing Independent Set to Clique

An instance of **Independent Set** is a graph G and an integer k.

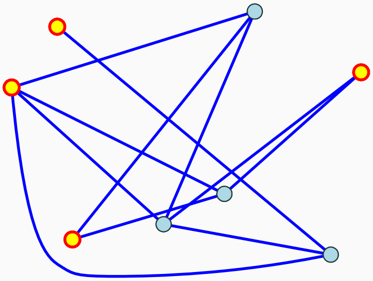


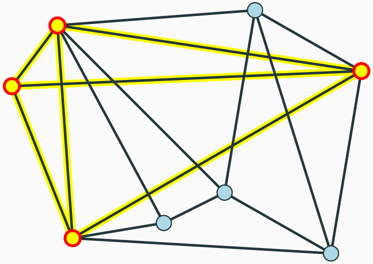


Reducing Independent Set to Clique

An instance of **Independent Set** is a graph G and an integer k.

Reduction given $\langle G, k \rangle$ outputs $\langle \overline{G}, k \rangle$ where $\overline{G}$ is the *complement* of G. $\overline{G}$ has an edge $uv \iff uv$ is not an edge of G.

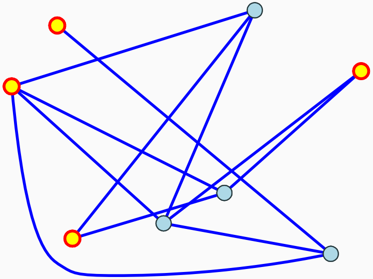


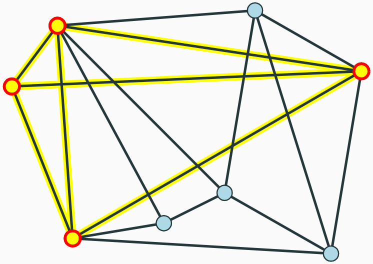


Reducing Independent Set to Clique

An instance of **Independent Set** is a graph G and an integer k.

Reduction given $\langle G, k \rangle$ outputs $\langle \overline{G}, k \rangle$ where $\overline{G}$ is the *complement* of G. $\overline{G}$ has an edge $uv \iff uv$ is not an edge of G.





A independent set of size k in $G \iff$ A clique of size k in $\overline{G}$

Correctness of reduction

Lemma
G has an independent set of size k $\iff$ $\overline{G}$ has a clique of size k.

Proof.
Need to prove two facts:

G has independent set of size at least k implies that $\overline{G}$ has a clique of size at least k.

$\overline{G}$ has a clique of size at least k implies that G has an independent set of size at least k.

Since $S \subseteq V$ is an independent set in G $\iff$ S is a clique in $\overline{G}$. $\square$

Independent Set and Clique

- Independent Set $\leq_P$ Clique.

Independent Set and Clique

- Independent Set $\leq_P$ Clique.
  What does this mean?
- If have an algorithm for Clique, then we have an algorithm for Independent Set.

Independent Set and Clique

- Independent Set $\leq_P$ Clique.
  What does this mean?
- If have an algorithm for Clique, then we have an algorithm for Independent Set.
- Clique is *at least as hard as* Independent Set.

Independent Set and Clique

- **Independent Set** $\leq_P$ **Clique**.
  What does this mean?
- If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.
- **Clique** is *at least as hard as* **Independent Set**.
- Also... **Clique** $\leq_P$ **Independent Set**. Why? Thus **Clique** and **Independent Set** are polnomial-time equivalent.

Visualize Clique and independent Set Reduction

I want to show Independent Set is atleast as has as Clique.

Visualize Clique and independent Set Reduction

I want to show **Independent Set** is atleast as has as **Clique**.

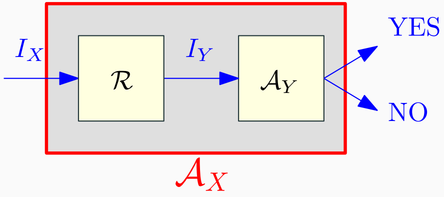
Write out the equality: **Clique** $\leq$ **Independent Set**

Visualize Clique and independent Set Reduction

I want to show **Independent Set** is atleast as has as **Clique**.

Write out the equality: **Clique** $\leq$ **Independent Set**

Draw reduction figure:

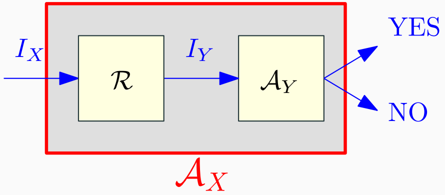


Visualize Clique and independent Set Reduction

I want to show **Independent Set** is atleast as has as **Clique**.

Write out the equality: **Clique** $\leq$ **Independent Set**

Draw reduction figure:



Fill in the blanks:

- $I_X = \langle G \rangle$
- $A_X = \text{Clique}$
- $I_Y = \langle \overline{G} \rangle$
- $A_Y = \text{Independent Set}$
- $R : \overline{G} = \{V, \overline{E}\}$

Review: Independent Set and Clique

Assume you can solve the **Clique** problem in $T(n)$ time. Then you can solve the **Independent Set** problem in

(A) $O(T(n))$ time.

(B) $O(n \log n + T(n))$ time.

(C) $O(n^2 T(n^2))$ time.

(D) $O(n^4 T(n^4))$ time.

(E) $O(n^2 + T(n^2))$ time.

(F) Does not matter - all these are polynomial if $T(n)$ is polynomial, which is good enough for our purposes.

Independent Set and Vertex Cover

Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

Vertex Cover

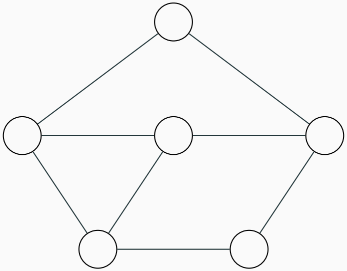
Given a graph $G = (V, E)$, a set of vertices S is:

- A *vertex cover* if every $e \in E$ has at least one endpoint in S.

Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

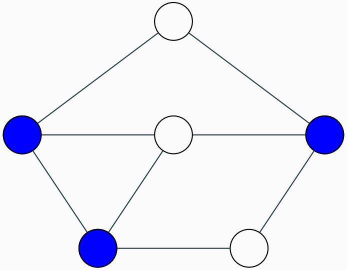
- A *vertex cover* if every $e \in E$ has at least one endpoint in S.



Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

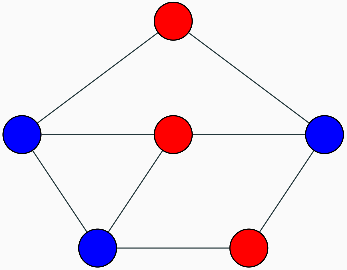
- A *vertex cover* if every $e \in E$ has at least one endpoint in S.



Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

- A *vertex cover* if every $e \in E$ has at least one endpoint in S.



The Vertex Cover Problem

Problem (Vertex Cover)

Input: *A graph G and integer k.*

Goal: *Is there a vertex cover of size $\leq k$ in G?*

The Vertex Cover Problem

Problem (Vertex Cover)

Input: *A graph G and integer k.*

Goal: *Is there a vertex cover of size $\leq k$ in G?*

Can we relate **Independent Set** and **Vertex Cover**?

Relationship between Vertex Cover and Independent Set

Lemma
Let $G = (V, E)$ be a graph. S is an Independent Set $\iff$ $V \setminus S$ is a vertex cover.

Relationship between Vertex Cover and Independent Set

Lemma
Let $G = (V, E)$ be a graph. S is an Independent Set $\iff$ $V \setminus S$ is a vertex cover.

Proof.

$(\Rightarrow)$ Let S be an independent set

- Consider any edge $uv \in E$.
- Since S is an independent set, either $u \notin S$ or $v \notin S$.
- Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
- $V \setminus S$ is a vertex cover.

Relationship between Vertex Cover and Independent Set

Lemma
Let $G = (V, E)$ be a graph. S is an Independent Set $\iff$ $V \setminus S$ is a vertex cover.

Proof.

($\Rightarrow$) Let S be an independent set

- Consider any edge $uv \in E$.
- Since S is an independent set, either $u \notin S$ or $v \notin S$.
- Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
- $V \setminus S$ is a vertex cover.

($\Leftarrow$) Let $V \setminus S$ be some vertex cover:

- Consider $u, v \in S$
- uv is not an edge of G, as otherwise $V \setminus S$ does not cover uv.
- $\implies$ S is thus an independent set. $\square$

Independent Set $\leq_P$ Vertex Cover

- G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.

Independent Set $\leq_P$ Vertex Cover

- G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
- G has an independent set of size $\geq k \iff G$ has a vertex cover of size $\leq n - k$

Independent Set $\leq_P$ Vertex Cover

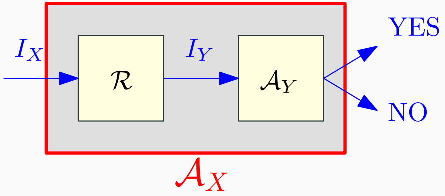
- G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
- G has an independent set of size $\geq k \iff G$ has a vertex cover of size $\leq n - k$
- (G, k) is an instance of **Independent Set**, and $(G, n - k)$ is an instance of **Vertex Cover** with the same answer.

Independent Set $\leq_P$ Vertex Cover

- G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
- G has an independent set of size $\geq k \iff G$ has a vertex cover of size $\leq n - k$
- (G, k) is an instance of **Independent Set**, and $(G, n - k)$ is an instance of **Vertex Cover** with the same answer.
- Therefore, **Independent Set** $\leq_P$ **Vertex Cover**. Also **Vertex Cover** $\leq_P$ **Independent Set**.

Independent Set $\leq_P$ Vertex Cover

- G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
- G has an independent set of size $\geq k \iff G$ has a vertex cover of size $\leq n - k$



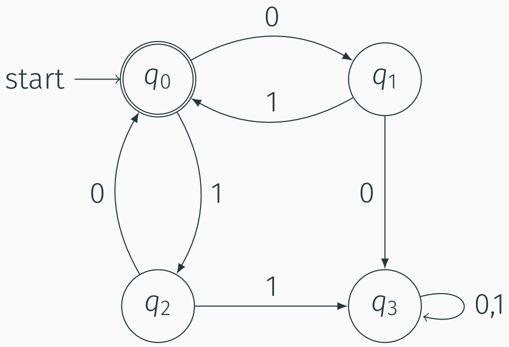
- $I_X = \langle G \rangle$
- $A_X = \text{Independent Set}(G, k)$
- $I_Y = \langle G \rangle$
- $A_Y = \text{Vertex Cover}(G, n - k)$
- $R : G' = G$

NFAs|DFAs and Universality

DFA Accepting a String

Given DFA M and string $w \in \Sigma^*$, does M accept w?

- Instance is $\langle M, w \rangle$
- Algorithm: given $\langle M, w \rangle$, output YES if M accepts w, else NO



Does above DFA accept 0010110?

DFA Accepting a String

Given DFA M and string $w \in \Sigma^*$, does M accept w?

- Instance is $\langle M, w \rangle$
- Algorithm: given $\langle M, w \rangle$, output YES if M accepts w, else NO

Question: Is there an (efficient) algorithm for this problem?

DFA Accepting a String

Given DFA M and string $w \in \Sigma^*$, does M accept w?

- Instance is $\langle M, w \rangle$
- Algorithm: given $\langle M, w \rangle$, output YES if M accepts w, else NO

Question: Is there an (efficient) algorithm for this problem?

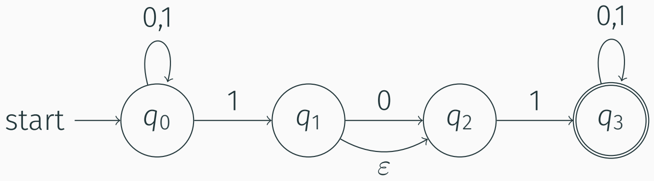
Yes. Simulate M on w and output YES if M reaches a final state.

Exercise: Show a linear time algorithm. Note that linear is in the input size which includes both encoding size of M and $|w|$.

NFA Accepting a String

Given NFA N and string $w \in \Sigma^*$, does N accept w?

- Instance is $\langle N, w \rangle$
- Algorithm: given $\langle N, w \rangle$, output YES if N accepts w, else NO



Does above NFA accept 0010110?

NFA Accepting a String

Given NFA N and string $w \in \Sigma^*$, does N accept w?

- Instance is $\langle N, w \rangle$
- Algorithm: given $\langle N, w \rangle$, output YES if N accepts w, else NO

Question: Is there an algorithm for this problem?

NFA Accepting a String

Given NFA N and string $w \in \Sigma^*$, does N accept w?

- Instance is $\langle N, w \rangle$
- Algorithm: given $\langle N, w \rangle$, output YES if N accepts w, else NO

Question: Is there an algorithm for this problem?

- Convert N to equivalent DFA M and use previous algorithm!
- Hence a reduction that takes $\langle N, w \rangle$ to $\langle M, w \rangle$
- Is this reduction efficient?

NFA Accepting a String

Given NFA N and string $w \in \Sigma^*$, does N accept w?

- Instance is $\langle N, w \rangle$
- Algorithm: given $\langle N, w \rangle$, output YES if N accepts w, else NO

Question: Is there an algorithm for this problem?

- Convert N to equivalent DFA M and use previous algorithm!
- Hence a reduction that takes $\langle N, w \rangle$ to $\langle M, w \rangle$
- Is this reduction efficient? No, because $|M|$ is exponential in $|N|$ in the worst case.

Exercise: Describe a polynomial-time algorithm.

Hence reduction may allow you to see an easy algorithm but not necessarily best algorithm!

DFA Universality

A DFA M is universal if it accepts every string.

That is, $L(M) = \Sigma^*$, the set of all strings.

Problem (DFA universality)

Input: *A DFA M.*

Goal: *Is M universal?*

How do we solve **DFA Universality**?

We check if M has *any* reachable non-final state.

NFA Universality

An NFA N is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

Problem (NFA universality)

Input: *A NFA M.*

Goal: *Is M universal?*

How do we solve **NFA Universality**?

NFA Universality

An NFA N is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

Problem (NFA universality)

Input: *A NFA M.*

Goal: *Is M universal?*

How do we solve **NFA Universality**?

Reduce it to **DFA Universality**?

NFA Universality

An NFA N is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

Problem (NFA universality)

Input: *A NFA M.*

Goal: *Is M universal?*

How do we solve **NFA Universality**?

Reduce it to **DFA Universality**?

Given an NFA N, convert it to an equivalent DFA M, and use the **DFA Universality** Algorithm.

What is the problem with this reduction?

NFA Universality

An NFA N is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

Problem (NFA universality)

Input: *A NFA M.*

Goal: *Is M universal?*

How do we solve **NFA Universality**?

Reduce it to **DFA Universality**?

Given an NFA N, convert it to an equivalent DFA M, and use the **DFA Universality** Algorithm.

What is the problem with this reduction? The reduction takes exponential time!

NFA Universality is known to be PSPACE-Complete.

Polynomial time reductions

Polynomial-time reductions

We say that an algorithm is *efficient* if it runs in polynomial-time.

Polynomial-time reductions

We say that an algorithm is *efficient* if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

Polynomial-time reductions

We say that an algorithm is *efficient* if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

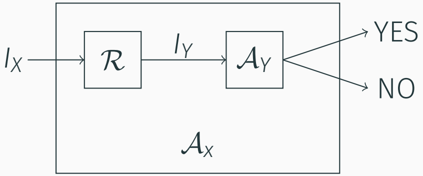
If we have a polynomial-time reduction from problem X to problem Y (we write $X \leq_P Y$), and a poly-time algorithm $\mathcal{A}_Y$ for Y, we have a polynomial-time/efficient algorithm for X.

Polynomial-time reductions

We say that an algorithm is *efficient* if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem X to problem Y (we write $X \leq_P Y$), and a poly-time algorithm $\mathcal{A}_Y$ for Y, we have a polynomial-time/efficient algorithm for X.



Polynomial-time Reduction

A polynomial time reduction from a *decision* problem X to a *decision* problem Y is an *algorithm* $\mathcal{A}$ that has the following properties:

- given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y
- $\mathcal{A}$ runs in time polynomial in $|I_X|$.
- Answer to I_X YES $\iff$ answer to I_Y is YES.

Polynomial-time Reduction

A polynomial time reduction from a *decision* problem X to a *decision* problem Y is an *algorithm* $\mathcal{A}$ that has the following properties:

- given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y
- $\mathcal{A}$ runs in time polynomial in $|I_X|$.
- Answer to I_X YES $\iff$ answer to I_Y is YES.

Lemma
If $X \leq_P Y$ then a polynomial time algorithm for Y implies a polynomial time algorithm for X.

Such a reduction is called a *Karp reduction*. Most reductions we will need are Karp reductions.Karp reductions are the same as mapping reductions when specialized to polynomial time for the reduction step.

Review question: Reductions again...

Let X and Y be two decision problems, such that X can be solved in polynomial time, and $X \leq_P Y$. Then

(A) Y can be solved in polynomial time.

(B) Y can NOT be solved in polynomial time.

(C) If Y is hard then X is also hard.

(D) None of the above.

(E) All of the above.

Be careful about reduction direction

Note: $X \leq_P Y$ does not imply that $Y \leq_P X$ and hence it is very important to know the FROM and TO in a reduction.

To prove $X \leq_P Y$ you need to show a reduction FROM X TO Y

That is, show that an algorithm for Y implies an algorithm for X.

Turing machines and reductions

Reasoning about TMs/Programs

- $\langle M \rangle$ is encoding of a TM M.
- Equivalently think of $\langle M \rangle$ as the code of a program in some high-level programming language

Reasoning about TMs/Programs

- $\langle M \rangle$ is encoding of a TM M.
- Equivalently think of $\langle M \rangle$ as the code of a program in some high-level programming language

Three related problems:

- Given $\langle M \rangle$ does M halt on blank input? (Halting Problem)
- Given $\langle M, w \rangle$ does M halt on input w?
- Given $\langle M, w \rangle$ does M accept w? (Universal Language)

Question: Do any of the above problems have an algorithm?

Reasoning about TMs/Programs

- $\langle M \rangle$ is encoding of a TM M.
- Equivalently think of $\langle M \rangle$ as the code of a program in some high-level programming language

Three related problems:

- Given $\langle M \rangle$ does M halt on blank input? (Halting Problem)
- Given $\langle M, w \rangle$ does M halt on input w?
- Given $\langle M, w \rangle$ does M accept w? (Universal Language)

Question: Do any of the above problems have an algorithm?

Theorem (Turing)
All the three problems are undecidable! No algorithm/program/TM.

Undecidability Reductions

CS 125 auto grading problem:

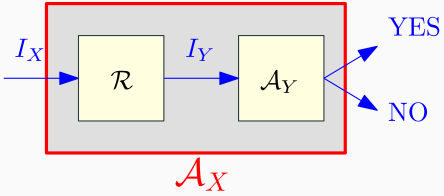
- student assignment: write program to print “Hello World”
- autograder: given student’s code $\langle S \rangle$ check if it prints “Hello World” correctly

Undecidability Reductions

How do we reduce the halting problem to the autograding problem?! Want to prove $\mathrm{HALT} \leq_P \mathrm{Grader}$

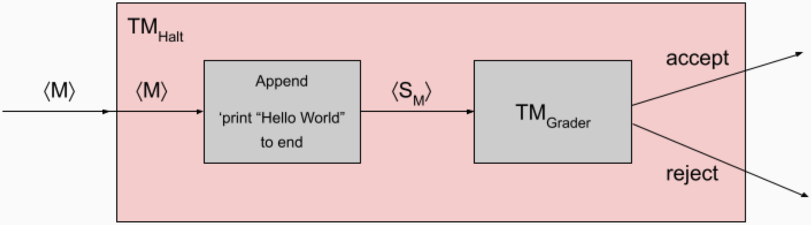
Undecidability Reductions

How do we reduce the halting problem to the autograding problem?! Want to prove $\mathrm{HALT} \leq_P \mathrm{Grader}$



Undecidability Reductions

How do we reduce the halting problem to the autograding problem?! Want to prove $\mathrm{HALT} \leq_P \mathrm{Grader}$



Undecidability Reductions

CS 125 auto grading problem:

- student assignment: write program to print “Hello World”
- autograder: given student’s code $\langle S \rangle$ check if it prints “Hello World” correctly

Impossible! Why? Reduce Halting problem to CS125 autograding

Given arbitrary program $\langle M \rangle$ reduction generates program $\langle S_M \rangle$ such that S prints “Hello World” iff M halts

- Reduction is linear time algorithm. Just copies code of M to create code for S_M with additional couple of lines
- Main point: algorithm should work correctly for *every* input not just some simple cases.