Recap: What does it mean for a language to be decidable?

EECS 376 Discussion 7

Sec 27: Th 5:30-6:30 DOW 1017

IA: Eric Khiu

Slide deck available at course drive/Discussion/Slides/Eric Khiu

Announcement

- HW6 due tonight at 8pm
- Past midterms are up: course drive/Exam
- Piazza poll @782- vote problems to be discussed during midterm review on 3/4

Agenda

- Turing Reductions
- Proving Undecidability
- More Turing Reductions
- Midterm Review
  - Potential Method
  - Divide and Conquer
  - Dynamic Programming
  - Greedy Algorithms
  - DFA
  - Turing Machines

Midterm Review

Recap: Decidable Language

- A language L is decidable iff there exists a TM that
  - accepts all $x \in L$
  - rejects all $x \notin L$
  - halts on all inputs
- One way to prove that a language is decidable is by describing a TM that decides it (i.e., a decider)
- **Note 1:** You are allowed to hardcode constant values in your machine (seen in HW6)
- **Note 2:** If some deciders are known to exists, you can use them in your machine (e.g., D_S and D_T from last week for $S \setminus T$)

Today: Proving *Un*decidability

- We know that the following languages are undecidable:
  - $L_{BARBER} = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$
  - $L_{HALT} = \{(\langle M \rangle, x) : M \text{ does not halt on } x\}$
  - $L_{ACC} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$
- Now, say we want to prove that L is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L is decidable
  - …[something happen]…
  - Therefore, [insert undecidable language] is now decidable. Contradiction.

Starter

Discuss: I have a C++ function **odd** that tells me if an input integer is odd. I don't know how it works. Now I want to write a function **even** to determine if x is even. What can I do?
Note: I'm smart enough to know that a number is either odd or even. I also know that even numbers are divisible by 2.

- Option 1: Call odd on x and return opposite

```
bool even(x){
     return (!odd(x));
}
```

- Option 2: Perform modular arithmetic myself

```
bool even(x){
     return (x % 2 == 0);
}
```

Turing Reductions

[Course Notes]

Turing Reducible

- A language A is *Turing reducible* to language B, written as

$$A \leq_T B$$

which means

- If, I have a TM that decides B, say M_B (black box/ oracle)
- Then, I can decide A

Warning: This tells me the *relationship* between languages A and B, I don't know anything (most importantly, decidability) of A and B *individually*

Turing Reduction Example

- Let A and B be defined as follows:

$$L_A = \{a^n : n \geq 0\} \quad L_B = \{b^n : n \geq 0\}$$

- **By using blackbox decider of $\boldsymbol{B}$**, prove that $A \leq_T B$
- Want to show: If I have a decider D_B for B, then I can decide A.
- To do so, we build a decider for A, say D_A that uses D_B.

- **Step 1: Identify the inputs of $\boldsymbol{D_A}$ and $\boldsymbol{D_B}$**
  - We will pass in a string, say x, into D_A and check if $x \in L_A$
  - We will pass in a string, say x', into D_B and check if $x' \in L_B$
  - **Note:** x' *may* be the same as x, but we don't know yet so assume they're different for now

Desired Behavior of D_A

- **Step 2: Draft Desired Behavior of $\boldsymbol{D_A}$** (can be used as correctness proof later)
  - $x \in A \Rightarrow x = a^n \Rightarrow \cdots \Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow \cdots \Rightarrow D_A(x)$ rejects
- Now, let's work backwards from the other direction.
- Using the blackbox decider essentially means using the output of that decider
- Which means we have two choices:
  - Do the same as the blackbox decider ("return same")
  - Do the opposite as the blackbox decider ("return opposite")
- Often, one choice is more intuitive than another, so just pick one and try!

Desired Behavior of D_A

- **Step 2: Draft Desired Behavior of D_A** (can be used as correctness proof later)
  - $x \in A \Rightarrow x = a^n \Rightarrow \cdots \Rightarrow D_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow \cdots \Rightarrow D_B(x')$ rejects $\Rightarrow D_A(x)$ rejects
- Suppose we picked "return same"
- Since we assumed the correctness of D_B, we must have $x' \in B$ if $D_B(x')$ accepts, otherwise $x' \notin B$
- So we have
  - $x \in A \Rightarrow \cdots \Rightarrow x' \in B \Rightarrow D_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow \cdots \Rightarrow x' \notin B \Rightarrow D_B(x')$ rejects $\Rightarrow D_A(x)$ rejects

Input(s) For D_B

- Desired Behavior of D_A: On input x:
  - $x \in A \Rightarrow x = a^n \Rightarrow \cdots \Rightarrow x = b^n \Rightarrow x' \in B \Rightarrow D_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow \cdots \Rightarrow x \neq b^n \Rightarrow x' \notin B \Rightarrow D_B(x')$ rejects $\Rightarrow D_A(x)$ rejects
- **Step 3: Generate input(s) for D_B**
  - We want $x' = b^n$ if $x = a^n$
  - We want $x' \neq b^n$ if $x \neq a^n$
  - Note: Technically we can just check x (since we can hardcode 'a' into the machine), but since we want to use D_B, we need to come out with the mapping
  - Brainstorm: What can we do to generate x'? Possibly by making use of x?
    - Flip all characters of x (aaa→bbb, aba→bab)

Putting Everything together

- Build D_A as follows:

```
D_A = "On input x:
    x' = Flip all bits in x
    Run D_B on x'
    If D_B(x') rejects then Accept
    Reject"
```

- Or equivalently,

```
Run D_L on x and return opposite"
```

- Correctness Proof:
  - $x \in A \Rightarrow x = a^n \Rightarrow x' = b^n \Rightarrow x' \in B \Rightarrow D_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow x' \neq b^n \Rightarrow x' \notin B \Rightarrow D_B(x')$ rejects $\Rightarrow D_A(x)$ rejects

Turing Reductions Overview

- Suppose we want to show that $A \leq_T B$
- **Step 1: Identify the inputs of D_A and D_B**
  - Is the input a number? A string? Multiple strings? A machine?
- **Step 2: Draft Desired Behavior of D_A**
  - Choose between "return same" and "return opposite"
  - Return same: $x \in A \Leftrightarrow \cdots \Leftrightarrow x' \in B \Leftrightarrow D_B(x')$ accepts $\Leftrightarrow D_A(x)$ accepts
  - Return opposite: $x \in A \Leftrightarrow \cdots \Leftrightarrow x' \notin B \Leftrightarrow D_B(x')$ rejects $\Leftrightarrow D_A(x)$ accepts
- **Step 3: Generate input(s) for D_B**
  - Return same: How to generate x', possibly using x, such that $x \in A \Rightarrow x' \in B$ and $x \notin A \Rightarrow x' \notin B$?
  - Return opposite: How to generate x', possibly using x, such that $x \in A \Rightarrow x' \notin B$ and $x \notin A \Rightarrow x' \in B$?

Note: Here we condense the two cases using iff

Turing Reductions Exercise

- Recall that

$$L_{ACC} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$$

- Now consider the languages

$$L_{DOUBLE-ACCEPT} = \{(\langle M \rangle, x, y) : M \text{ accepts } x \text{ and } y\}$$

- Prove that

$$L_{ACC} \leq L_{DOUBLE-ACCEPT}$$

Turing Reductions Exercise

Solution: Let W be a blackbox decider for $L_{\text{DOUBLE-ACCEPT}}$. Define a decider A for L_{ACC} as follows:

$$\boxed{\begin{array}{l} A = \text{"on input } (\langle M\rangle, x)\text{:} \\ \quad 1\text{: Run } W \text{ on } (\langle M\rangle, x, x) \text{ and return same"} \end{array}}$$

Since W is a decider, A necessarily halts. Therefore it remains to show that A is a decider for L_{ACC} :

- $(\langle M\rangle, x) \in L_{\text{ACC}} \implies M$ accepts $x \implies W$ accepts $(\langle M\rangle, x, x) \implies A$ accepts $(\langle M\rangle, x)$
- $(\langle M\rangle, x) \notin L_{\text{ACC}} \implies M$ does not accept $x \implies W$ rejects $(\langle M\rangle, x, x) \implies A$ rejects $(\langle M\rangle, x)$

Therefore, we have shown that $L_{\text{ACC}} \leq_T L_{\text{DOUBLE-ACCEPT}}$.

Proving Undecidability

Course Notes

Very Important Theorem

- $\boxed{\text{Suppose } A \leq_T B\text{. If } B \text{ is decidable, then } A \text{ is decidable.}}$
- Analogy: B is a *harder* problem than A. If I can solve the harder problem, then I can solve the easier problem.

- Contrapositive: $p = B$ is decidable; q = A is decidable
  - If p then q is equivalent to If $\neg q$ then $\neg p$
  - If A is <u>un</u>decidable, then B is <u>un</u>decidable

Known Fact	Conclusion
A decidable	?
A undecidable	B undecidable
B decidable	A decidable
B undecidable	?

Undecidability Proof Outline

- Know: [insert undecidable language] is undecidable
- Task: Prove L is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L is decidable
  - …[something happen]…
  - Therefore, [insert undecidable language] is now decidable. Contradiction.

Undecidability Proof Outline

- Know: L_{BARBER} is undecidable
- Task: Prove L_{ACC} is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L_{ACC} is decidable
  - …[something happen]…
  - Therefore, L_{BARBER} is now decidable. Contradiction.

Undecidability Proof Outline

- Know: L_{BARBER} is undecidable
- Task: Prove L_{ACC} is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L_{ACC} is decidable
  - We have shown that $L_{BARBER} \leq_T L_{ACC}$
  - By this theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.
  - Since we have $L_{BARBER} \leq_T L_{ACC}$ and L_{ACC} is decidable by assumption
  - Therefore, L_{BARBER} is now decidable. Contradiction.

Sanity Check: Who goes on which side?

- Which of the following proves that L is <u>un</u>decidable? [Select all applies]
  - $\Sigma^* \leq_T L$
  - $L \leq_T \emptyset$
  - $L \leq_T L_{BARBER}$
  - $L_{HALT} \leq_T L$

More Turing Reductions

Course Notes

More Turing Reductions

- Another thing you are allowed to do is to create a machine (without running it) and pass it to the backbox decider
- For example, I want to use the black box decider D_E that decides

  $$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$

  to build a decider for L_{ACC}. I can do the following:

```
D_A = "On input (⟨M⟩,x):
    Construct M' as follows: TODO
    Run D_E on ⟨M'⟩ and return same"
```

- **Discuss:** Why do I have to create M'? Why can't I just use M?
  - The input of D_E must be a machine
  - We don't know $L(M)$, so we can't tell if D_E accepts/ rejects M

More Turing Reductions

- For example, I want to use the black box decider D_E that decides
$$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$
to build a decider for L_{ACC}. I can do the following:

```
DA = “On input (⟨M⟩,x):
    Construct M’ as follows: TODO
    Run DE on ⟨M’⟩ and return same”
```

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ … $\Rightarrow D_E(\langle M' \rangle)$ accepts $\Rightarrow D_A(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ … $\Rightarrow D_E(\langle M' \rangle)$ rejects $\Rightarrow D_A(\langle M \rangle, x)$ rejects

More Turing Reductions

- For example, I want to use the black box decider D_E that decides

$$L_E = \{\langle M\rangle : L(M) = \emptyset\}$$

  to build a decider for L_{ACC}. I can do the following:

  D$_A$ = “On input (⟨M⟩,x):

  Construct M’ as follows: TODO

  Run D$_E$ on ⟨M’⟩ and return same”

- Correctness proof draft
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ ... $\Rightarrow L(M') = \emptyset \Rightarrow D_E(\langle M'\rangle)$ accepts $\Rightarrow D_A(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ ... $\Rightarrow L(M') \neq \emptyset \Rightarrow D_E(\langle M'\rangle)$ rejects $\Rightarrow D_A(\langle M\rangle, x)$ rejects

Brainstorm Time

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ ... $\Rightarrow L(M') = \emptyset \Rightarrow D_A(\langle M' \rangle)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ ... $\Rightarrow L(M') \neq \emptyset \Rightarrow D_A(\langle M' \rangle)$ rejects
- **Brainstorm 1:** How to make $L(M') = \emptyset$ happen?
  - Just make M' rejects all inputs!
    - `M'= "On input w: Reject"`

Trigger this if M accepts x

- **Brainstorm 2:** How to make $L(M') \neq \emptyset$ happen?
  - Just make M' accepts *some* inputs, say "duck"
    - `M'= "On input w: If w = 'duck' then Accept Else Reject"`
  - Even easier: make M' accepts all inputs, i.e., $L(M') = \Sigma^*$
    - `M'= "On input w: Accept"`

Trigger this if M does not accepts x

Putting Everything together…

- D_A = "On input (⟨M⟩,x):

  Construct M' as follows:

  ```
  M' = "On input w:
      Run M on x
      If M accepts x then Reject
      Else Accept"
  ```

  Run D_E on ⟨M'⟩ and return same"
- Correctness Proof:
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow M'$ rejects all inputs $\Rightarrow L(M') = \emptyset \Rightarrow D_E(\langle M'\rangle)$ accepts $\Rightarrow D_A(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow M'$ accepts all inputs $\Rightarrow L(M') \neq \emptyset \Rightarrow D_E(\langle M'\rangle)$ rejects $\Rightarrow D_A(\langle M\rangle, x)$ rejects

Midterm Review

See source cited for solutions

Potential Method (WN22 MCQ 1)

Consider the following code. Note that the variables x and y are real numbers.

Require: $x > y > 0$ are real numbers ▷ Hint: This actually matters...

function $\text{FOO376}(x, y)$
  if $x \leq 0$ **then return** 1;
  $z \leftarrow \text{FOO376}(x - \log y, y)$
  return $(z + 1)$

Which of the following is a valid potential function for the algorithm FOO376 (above)?

- ◯ $s = x + y$
- ◯ $s = e^y - x$
- ◯ $s = x$
- ◯ None of the above

Divide and Conquer (WS7 Review 2)

Given an n-digit positive integer k, the algorithm below computes 376^k using a naïve recursive strategy.

```
function NaivePow(k: positive integer)
    if k = 1 then return 376
    return 376 · NaivePow(k − 1)
```

Describe an efficient divide and conquer algorithm for solving this problem. For simplicity, you may assume that k is a power of 2. Your solution should include a correctness and runtime analysis in terms of n (assuming multiplication takes constant time).

Dynamic Programming (WS7 Review 3)

Give a recurrence relation (including base cases) that is suitable for dynamic programming solutions to the following problem. You do not need to prove your correctness.

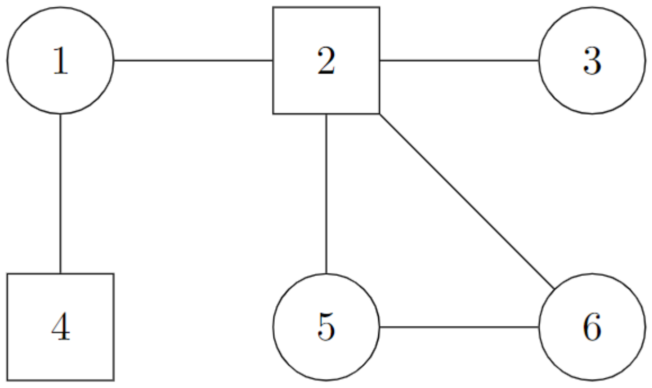
LONGEST-ARITHMETIC-SUBSEQUENCE(A, d): Given an array of integers A and a difference d, return the length of the longest arithmetic subsequence in A with difference d. That is, return the longest subsequence S such that $S[i+1] - S[i] = d$ for each i.

Greedy Algorithms (WN23 Short 3)

A dominating set S in a graph G is a set of vertices for which every vertex of G either is in S, or is adjacent to some vertex in S.

We are interested in a smallest dominating set of a given graph, i.e., one that has the fewest possible vertices. (There may be more than one smallest dominating set.)

For example, the following graph has a smallest dominating set $S^* = \{2, 4\}$: every vertex other than 2 and 4 is adjacent to 2 or 4 (or both), and there is no dominating set consisting of a single vertex.



Greedy Algorithm (WN23 Short 3), cont.

A dominating set S in a graph G is a set of vertices for which every vertex of G either is in S, or is adjacent to some vertex in S.

We are interested in a smallest dominating set of a given graph, i.e., one that has the fewest possible vertices. (There may be more than one smallest dominating set.)

Consider the following greedy algorithm for finding a dominating set in a graph.

function GreedyDS(G)
    $S \leftarrow \emptyset$
    while G has at least one vertex **do**
        Select any vertex v in G that has largest degree (i.e., the most neighbors)
        Add v to S
        Remove v and all its neighbors, including all incident edges, from G
    return S

Give a small graph G on which the algorithm **might not return a smallest dominating set**. Specifically, **give a sequence of vertices that the algorithm might choose** to make up its final output set, and **give an optimal dominating set of G that is smaller** than this output set.

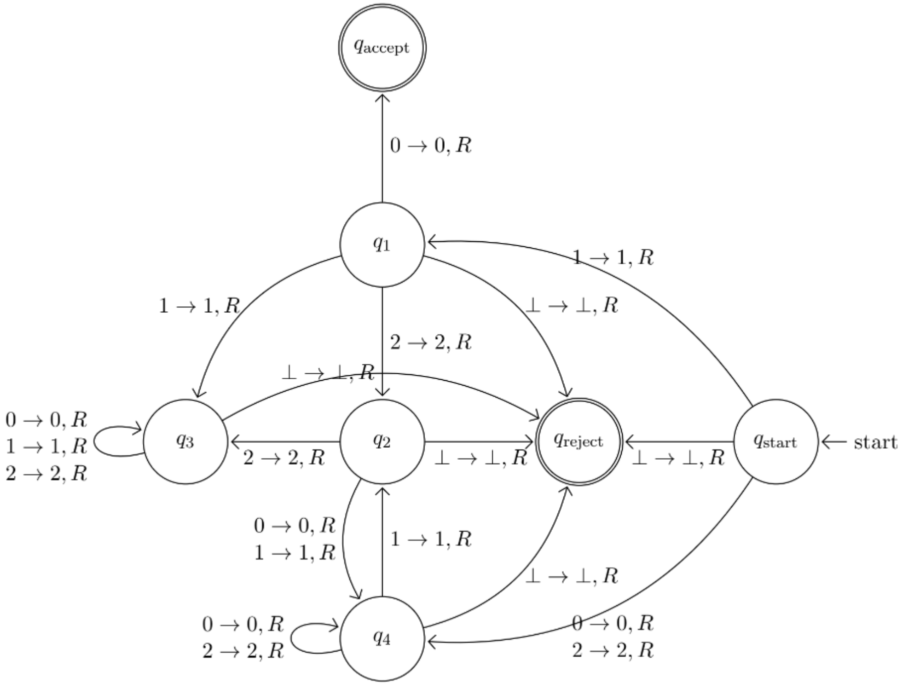
DFA (FA22 9b)

Let $L_2 \subseteq \{a, b, c\}^*$ be the set of all strings over the alphabet $\{a, b, c\}$ **except** those that contain both at least one b and at least one c. For example, aa, aba, cca are all in L_2, but abc is not as it contains both a b and a c.

Write a DFA over the alphabet $\{a, b, c\}$ that decides the language L_2.

Turing Machines (WS6 TM2)

Consider the Turing Machine whose state diagram is given below:



Which of the following statements is true about this Turing Machine?

- ◯ It accepts all strings that contain the substring "10."
- ◯ It loops on any input string that contains only 2s.
- ◯ It loops on any string that contains the substring "11" until it reaches $\perp$.
- ◯ None of the above