CSE 105
THEORY OF COMPUTATION

Friday, February 24, 2017

Winter 2017

http://cseweb.ucsd.edu/classes/wi17/cse105-ab/
# Recognition vs. Decision

<table>
<thead>
<tr>
<th></th>
<th>Recognizer for L</th>
<th>Decider for L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input in L</td>
<td>Accept</td>
<td>Accept</td>
</tr>
<tr>
<td>Input not in L</td>
<td>May Reject or Loop (Not Halt)</td>
<td>Reject</td>
</tr>
</tbody>
</table>
Variants of TMs

- Scratch work, copy input, …
- Multiple tapes
- Parallel computation
- Nondeterminism
- Printing vs. accepting
- Enumerators
- More flexible transition function
  - Can "stay put"
  - Can "get stuck"
  - *lots of examples in exercises to Chapter 3*

Also: wildly different models
- λ-calculus, Post canonical systems, URMś, etc.
Variants of TMs

- Scratch work, copy input, …
- Parallel computation
- Printing
- More flexible transition function
  - Can "stay put"
  - Can "get stuck"
  - Lots of examples in exercises to Chapter 3

Also: wildly different models

- $\lambda$-calculus, Post canonical systems, URMs, etc.

Multiple tapes

All these models are equally expressive...

capture the notion of "algorithm"
Algorithm

- Wikipedia "self-contained step-by-step set of operations to be performed"
- CSE 20 textbook "An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem."

Church-Turing thesis

Each algorithm can be implemented by some Turing machine.
Some algorithms

Examples of algorithms / algorithmic problems:
1. Recognize whether a string is a palindrome.
2. Reverse a string.
3. Recognize Pythagorean triples.
4. Compute the gcd of two positive integers.
5. Check whether a string is accepted by a DFA.
6. Convert a regular expression to an equivalent NFA.
7. Check whether the language of a PDA is infinite.
Some algorithms

Examples of algorithms / algorithmic problems:

1. Recognize whether a string is a palindrome.
2. Reverse a string.
3. Recognize Pythagorean triples.
4. Compute the gcd of two positive integers.
5. Check whether a string is accepted by a DFA.
6. Convert a regular expression to an equivalent NFA.
7. Check whether the language of a PDA is infinite.

Which of the following is true?

A. All these algorithms have inputs of the same type.
B. The inputs of each of these algorithms can be encoded as finite strings.
C. Some of these problems don't have algorithmic solutions.
D. I don't know.
Encoding input for TMs

By definition, TM inputs are **strings**

To define TM M:

"On input w ...

1. ...
2. ...
3. ...

For inputs that aren't strings, we have to **encode the object** (represent it as a string) first

**Notation:**

- \(<O>\) is the string that represents (encodes) the object \(O\)
- \(<O_1, \ldots, O_n>\) is the single string that represents the tuple of objects \(O_1, \ldots, O_n\)
Encoding inputs

Problems we care about can be reframed as languages of strings.

E.g. "Recognize whether a string is a palindrome."
\[ \{ w \mid w \text{ in } \{0,1\}^* \text{ and } w = w^R \} \]

E.g. "Recognize Pythagorean triples."
\[ \{ <a,b,c> \mid a,b,c \text{ integers and } a^2 + b^2 = c^2 \} \]

E.g. "Check whether a string is accepted by a DFA."
\[ \{ <B,w> \mid B \text{ is a DFA over } \Sigma, w \text{ in } \Sigma^*, \text{ and } w \text{ is in } L(B) \} \]

E.g. "Check whether the language of a PDA is infinite."
\[ \{ <A> \mid A \text{ is a PDA and } L(A) \text{ is infinite} \} \]
Regular

Context-Free

{Turing-Decidable

{Turing-Recognizable

{a^n b^n | n ≥ 0}

{{a^m b^n | m, n ≥ 0}

{a^n b^n c^n | n ≥ 0}

??

??

??
Computational problems

A computational problem is **decidable** iff the language encoding the problem instances is decidable.
Sample computational problems and their encodings:

- \(A_{DFA}\) "Check whether a string is accepted by a DFA."
  \[\{ <B,w> \mid B \text{ is a DFA over } \Sigma, w \text{ in } \Sigma^*, \text{ and } w \text{ is in } L(B) \}\]

- \(E_{DFA}\) "Check whether the language of a DFA is empty."
  \[\{ <A> \mid A \text{ is a DFA over } \Sigma, L(A) \text{ is empty} \}\]

- \(E_{DFA}\) "Check whether the languages of two DFA are equal."
  \[\{ <A, B> \mid A \text{ and } B \text{ are DFA over } \Sigma, L(A) = L(B)\}\]

We will show that all of these problems are decidable!
Proving decidability: DFA Acceptance

Claim: $A_{DFA}$ is decidable

Proof: WTS that \{ <B,w> | B is a DFA over $\Sigma$, w in $\Sigma^*$, and w is in $L(B)$ \} is decidable.

Step 1: construction

*How would you check if w is in $L(B)$?*
Proving decidability

Claim: $A_{\text{DFA}}$ is decidable

Proof: WTS that \{ <B,w> | B is a DFA over $\Sigma$, w in $\Sigma^*$, and w is in $L(B)$ \} is decidable.

Step 1: construction

Define TM M by: M = "On input <B,w>:

1. Check whether B is a valid encoding of a DFA and w is a valid input for B. If not, reject.
2. Simulate running B on w (by keeping track of states in B, transition function of B, etc.)
3. When the simulation ends, by finishing to process all of w, check current state of B: if it is final, accept; if it is not, reject."

How are we describing the machine M here?

A. Formal definition of M
B. Implementation-level description of M
C. High-level description of M
D. I don't know
Step 1: construction
Define TM M by M = "On input <B,w>
1. Check whether B is a valid encoding of a DFA and w is a valid input for B. If not, reject.
2. Simulate running B on w (by keeping track of states in B, transition function of B, etc.)
3. When the simulation ends, by finishing to process all of w, check current state of B: if it is final, accept; if it is not, reject."

Step 2: correctness proof
WTS (1) L(M) = A_{DFA} and (2) M is a decider.
Claim: $E_{DFA}$ is decidable

Proof: WTS that $\{ <A> | A \text{ is a DFA over } \Sigma, L(A) \text{ is empty} \}$ is decidable.

Idea: give high-level description

Step 1: construction

What condition characterizes DFAs whose language is empty?

A. $<A>$ is in $E_{DFA}$ iff A's initial state is accepting.
B. $<A>$ is in $E_{DFA}$ iff A' set of accepting states is empty.
C. $<A>$ is in $E_{DFA}$ iff A is the empty set.
D. None of the above.
E. I don't know.
Proving decidability: DFA Emptiness

Claim: $E_{DFA}$ is decidable
Proof: WTS that $\{ <A> \mid A \text{ is a DFA over } \Sigma, L(A) \text{ is empty} \}$ is decidable.
Idea: give high-level description
Step 1: construction

What condition characterizes DFAs whose language is empty?
$<A>$ is in $E_{DFA}$ iff an accept state is reachable from the start state.
Claim: $E_{DFA}$ is decidable

Proof: WTS that \{ $<A>$ | $A$ is a DFA over $\Sigma$, $L(A)$ is empty \} is decidable.

Idea: give high-level description

Step 1: construction

Reachability Algorithm from CSE21: Graph Search
Idea: Repeatedly mark neighbors until nothing more can be marked

What condition characterizes DFAs whose language is empty?

\(<A> \text{ is in } E_{DFA} \text{ iff an accept state is } \text{reachable} \text{ from the start state.}\)
Proving decidability: DFA Emptiness

Claim: $E_{DFA}$ is decidable
Proof: WTS that $\{ <A> | A$ is a DFA over $\Sigma$, $L(A)$ is empty $\}$ is decidable. Idea: give high-level description

Step 1: construction
Define TM $K$ by: $K = "$On input $<A>$:

1. Check whether $A$ is a valid encoding of a DFA; if not, reject.
2. Mark the start state of $A$.
3. Repeat until no new states get marked:
   - Loop over states of $A$ and mark any unmarked state that has an incoming edge from a marked state.
4. If no final state of $A$ is marked, accept; otherwise, reject."
Proving decidability

Step 1: construction
Define TM K by: K = "On input A:
1. Check whether A is a valid encoding of a DFA; if not, reject.
2. Mark the start state of A.
3. Repeat until no new states get marked:
   - Loop over states of A and mark any unmarked state that has an
     incoming edge from a marked state.
4. If no final state of A is marked, accept; otherwise, reject.

Step 2: correctness proof

WTS (1) \( L(K) = E_{\text{DFA}} \) and
(2) K is a decider.

K will mark
A. all the states of the DFA A
B. all the states of the DFA A that are reachable from the start state
C. some states in the DFA more than once
D. more than one of the above
Proving decidability: DFA Equality

Claim: $\text{EQ}_{\text{DFA}}$ is decidable

Proof: WTS that $\{ <A, B> \mid A, B \text{ are DFA over } \Sigma, L(A) = L(B) \}$ is decidable. Idea: give high-level description

Step 1: construction

Will we be able to simulate $A$ and $B$?  
What does set equality mean?  
Can we use our previous work?
Proving decidability: DFA Equality

Claim: \( \text{EQ}_{\text{DFA}} \) is decidable

Proof: WTS that \( \{ <A, B> \mid A, B \text{ are DFA over } \Sigma, L(A) = L(B) \} \) is decidable. Idea: give high-level description

Step 1: construction

Very high-level:
Build new DFA recognizing symmetric difference of \( L(A) \) and \( L(B) \). Check if this set is empty.
Proving decidability: DFA Equality

**Claim:** $\text{EQ}_{\text{DFA}}$ is decidable

**Proof:** WTS that $\{ <A, B> | A, B \text{ are DFA over } \Sigma, L(A) = L(B) \}$ is decidable. *Idea: give high-level description*

**Step 1: construction**

Define TM $J$ by: $J = "\text{On input } <A, B>:\"

1. Check whether $A, B$ are valid encodings of DFA; if not, reject.
2. Construct a new DFA, $D$, from $A,B$ using algorithms for complement, union, intersection of regular languages such that $L(D) = \text{symmetric difference of } L(A) \text{ and } L(B)$.
3. Run machine $K$ on $<D>$.
4. If $K$ accepts, then accept; if $K$ rejects, then reject."
Proving decidability: DFA Equality

Step 1: construction
Define TM J by: J = "On input <A,B>:
1. Check whether A,B are valid encodings of DFA; if not, reject.
2. Construct a new DFA, D, from A,B using algorithms for complement, union, intersection of regular languages such that \( L(D) = \text{symmetric difference of } L(A) \text{ and } L(B) \).
3. Run machine K on <D>.
4. If K accepts, then accept; if K rejects, then reject."

Step 2: correctness proof
WTS (1) \( L(J) = \text{EQ}_{\text{DFA}} \) and
(2) J is a decider.
Proving decidability: More Examples

All proven in Sipser 4.1.

• **A\textsubscript{NFA}**  "Check whether a string is accepted by an NFA."
  \[
  \{ \langle B, w \rangle \mid B \text{ is an NFA over } \Sigma, w \text{ in } \Sigma^*, \text{ and } w \text{ is in } L(B) \}\]

• **A\textsubscript{REX}**  "Check whether a regular expression generates a string."
  \[
  \{ \langle R, w \rangle \mid R \text{ is a regular expression over } \Sigma \text{ that generates } w \text{ in } \Sigma^* \}\]

• **A\textsubscript{CFG}**  "Check whether a string is generated by a CFG."
  \[
  \{ \langle G, w \rangle \mid G \text{ is a CFG over } \Sigma \text{ that generates } w \text{ in } \Sigma^* \}\]

• **E\textsubscript{CFG}**  "Check whether the language of a CFG is empty."
  \[
  \{ \langle G \rangle \mid G \text{ is a CFG over } \Sigma \text{ and } L(G) \text{ is empty } \}\]

• **EQ\textsubscript{CFG}**  "Check whether the languages of two CFGs are equal."
  \[
  \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFGs over } \Sigma, L(G) = L(H) \}\]

not responsible for proof of this
Techniques for proving decidability

- **Subroutines**: can use decision procedures of decidable problems as subroutines in other algorithms
  - Examples: $A_{DFA}$, $E_{DFA}$, $EQ_{DFA}$
- **Constructions**: can use algorithms for constructions as subroutines in other algorithms
  - Converting DFA to DFA recognizing complement (or Kleene star).
  - Converting two DFA/NFA to one recognizing union (or intersection, concatenation).
  - Converting NFA to equivalent DFA.
  - Converting regular expression to equivalent NFA.
  - Converting DFA to equivalent regular expression.