Today's learning goals

- Design TMs using different levels of descriptions.
- Give high-level description for TMs (recognizers and enumerators) used in constructions.
- Prove properties of the classes of recognizable and decidable sets.
- Describe several variants of Turing machines and informally explain why they are equally expressive.
- State and use the Church-Turing thesis.
An example

\[ L = \{ w#w \mid w \text{ is in } \{0,1\}^* \} \]

Idea for Turing machine

- Zig-zag across tape to corresponding positions on either side of '#' to check whether these positions agree. If they do not, or if there is no '#', reject. If they do, cross them off.
- Once all symbols to the left of the '#' are crossed off, check for any symbols to the right of '#': if there are any, reject; if there aren't, accept.

How would you use this machine to prove that L is decidable?
TM $M$ **deciding** language $L$

If $x \in L \Rightarrow M(x)$ accepts
If $x \notin L \Rightarrow M(x)$ rejects

---

TM $M$ **recognizing** language $L$

If $x \in L \Rightarrow M(x)$ accepts
If $x \notin L \Rightarrow M(x)$ reject or loop

---

Input $x$

$M(x) = \{\begin{array}{ll}
\text{accept} & \\
\text{reject} & \\
\text{loop} & \\
\end{array}$

$f(x) = \begin{cases} 
\text{return bool value} 
\end{cases}$
$Q = \{ q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_{\text{accept}}, q_{\text{reject}} \}$

$\Sigma = \{ 0, 1, \# \}$

$\Gamma = \{ 0, 1, \#, x, _ \}$

All missing transitions have output $(q_{\text{reject}}, _, R)$
Configuration $uqv$ for current tape $uv$ (and then all blanks), current head location is first symbol of $v$, current state $q$.

Computation on input $0\#0$?

$q, 0\#0$

$x, q_2, \#0$

$x, \#, q_6, \#x$

$q_7, x\#x$

$x, q, \#x$

$x, \#\#x$
Computation on input 0# ?

q₁₀#
x₉₂#
x₉₄#
x₉₇
Describing TMs

- **Formal definition**: set of states, input alphabet, tape alphabet, transition function, start state, accept state, reject state.
- **Implementation-level definition**: English prose to describe Turing machine head movements relative to contents of tape.
- **High-level description**: Description of algorithm, without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.
An example

Which of the following is an implementation-level description of a TM which decides the empty set?

M = "On input w:
A. reject."
B. sweep right across the tape until find a non-blank symbol. Then, reject."
C. If the first tape symbol is blank, accept. Otherwise, reject."
D. More than one of the above."
E. I don't know."
Context-free languages

Turing recognizable languages

Turing decidable languages

Regular languages

State control (NFA)

Stack

Push/pop to stack

Read tape head

Input
Closure

**Theorem:** The class of decidable languages over fixed alphabet $\Sigma$ is closed under union.

**Proof:** Let …

**WTS** …
Closure

**Theorem**: The class of decidable languages over fixed alphabet \( \Sigma \) is closed under union.

Proof: Let \( L_1 \) and \( L_2 \) be languages over \( \Sigma \) and suppose \( M_1 \) and \( M_2 \) are TMs deciding these languages. We will define a new TM, \( M \), via a high-level description. We will then show that \( L(M) = L_1 \cup L_2 \) and that \( M \) always halts.

\[ g(x) \begin{array}{c}
g(x) \text{ True} \\
x \in L_2 \\
\text{return value} \\
g(x) \text{ False} \\
x \notin L_1
\end{array} \]
Closure

Theorem: The class of decidable languages over fixed alphabet $\Sigma$ is closed under union.

Proof: Let $L_1$ and $L_2$ be languages and suppose $M_1$ and $M_2$ are TMs deciding these languages. Construct the TM $M$ as "On input $w$,

1. Run $M_1$ on input $w$. If $M_1$ accepts $w$, accept. Otherwise, go to 2.
2. Run $M_2$ on input $w$. If $M_2$ accepts $w$, accept. Otherwise, reject."

Correctness of construction:

WTS $L(M) = L_1 \cup L_2$ and $M$ is a decider.
**Closure**

Good exercises – can't use without proof! (Sipser 3.15, 3.16)

<table>
<thead>
<tr>
<th>The class of decidable languages is closed under</th>
<th>The class of recognizable languages is closed under</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union ✓</td>
<td>Union</td>
</tr>
<tr>
<td>Concatenation</td>
<td>Concatenation</td>
</tr>
<tr>
<td>Intersection</td>
<td>Intersection</td>
</tr>
<tr>
<td>Kleene star</td>
<td>Kleene star</td>
</tr>
<tr>
<td>Complementation</td>
<td></td>
</tr>
</tbody>
</table>
Variants of TMs

Section 3.2

- 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*

Payoff: In high-level description of TM, can simulate conversion to other variant.
"Equally expressive"

Model 1 is **equally expressive** as Model 2 iff

- every language recognized by some machine in Model 1 is recognizable by some machine in Model 2, **and**

- every language recognized by some machine in Model 2 is recognizable by some machine in Model 1.
Nondeterministic TMs

• Transition function

\[ \delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\}) \]

Sketch of proof of equivalence: A. Given TM, build nondeterministic TM recognizing same language. B. Given nondeterministic TM, build (deterministic) TM recognizing same language.

Idea: Try all possible branches of nondeterministic computation. 3 tapes: "read-only" input tape, simulation tape, tape tracking nondeterministic branching.
Multitape TMs

- As part of construction of machine, declare some finite number of tapes that are available.
- Input given on tape 1, rest of the tapes start blank.
- Each tape has its own read/write head.
- Transition function

\[ \delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k \]

Sketch of proof of equivalence: Given TM, build multitape TM recognizing same language. Given k-tape TM, build (1-tape) TM recognizing same language.

Idea: Use delimiter to keep tape contents separate, use special symbol to indicate location of each read/write head.
Very different model: Enumerators

Produce language as output rather than recognize input

Finite State Control

Computation proceeds according to transition function.

At any point, machine may "send" a string to printer.

L(E) = \{ w \mid E \text{ eventually, in finite time, prints } w \}
Enumerators

• What about machines that produce output rather than accept input?

Finite State Control

Unlimited work tape

Computation proceeds according to transition function.

At any point, machine may "send" a string to printer.

$L(E) = \{ w | E \text{ eventually, in finite time, prints } w\}$

Can $L(E)$ be infinite?

A. No, strings must be printed in finite time.
B. No, strings must be all be finite length.
C. Yes, it may happen if $E$ does not halt.
D. Yes, all $L(E)$ are infinite.
E. I don't know.
Set of all strings $\Sigma^*$

"For each $\Sigma$, there is an enumerator whose language is the set of all strings over $\Sigma$.

A. True
B. False
C. Depends on $\Sigma$.
D. I don't know.
Set of all strings

"For each $\Sigma$, there is an enumerator whose language is the set of all strings over $\Sigma$."

A. True  
B. False  
C. Depends on $\Sigma$.  
D. I don't know.
Recognition and enumeration

Theorem: A language L is Turing-recognizable iff some enumerator enumerates L.

Proof:

⇒ Assume L is Turing-recognizable. WTS some enumerator enumerates it.

⇐ Assume L is enumerated by some enumerator. WTS L is Turing-recognizable.
Recognition and enumeration  

Assume the enumerator $E$ enumerates $L$. WTS $L$ is Turing-recognizable.

We'll use $E$ in a subroutine for high-level description of Turing machine $M$ that will recognize $L$.

Define $M$ as follows: $M = \text{"On input } w, \text{\quad 1. Run } E. \text{ Every time } E \text{ prints a string, compare it to } w. \text{\quad 2. If } w \text{ ever appears as the output of } E, \text{ accept.}'}$

Correctness?
Recognition and enumeration

Assume $L$ is Turing-recognizable. WTS some enumerator enumerates it.

Let $M$ be a TM that recognizes $L$. We'll use $M$ in a subroutine for high-level description of enumerator $E$.

Let $s_1, s_2, \ldots$ be a list of all possible strings of $\Sigma^*$. Define $E$ as follows:

$E = \text{"Repeat the following for each value of } i = 1, 2, 3, \ldots$

1. Run $M$ for $i$ steps on each input $s_1, \ldots, s_i$
2. If any of the $i$ computations of $M$ accepts, print out the accepted string.

Correctness?
Variants of TMs

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

Multiple tapes

Nondeterminism

Enumerators

Also: wildly different models

• λ-calculus, Post canonical systems, URMs, etc.
Variants of TMs

- Scratch work, copy input
- Parallel computation
- Printing vs. accepting
- More flexible transition function
  - Can "stay put"
  - Can "get stuck"
  - Lots of examples in exercises

Also: wildly different models
- $\lambda$-calculus, Post canonical systems, URM$s$, etc.

All these models are equally expressive!
Church-Turing thesis

- 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."

Each algorithm can be implemented by some Turing machine.