Decidability and
The Halting Problem

Foundations of Computer
Science Theory
Decidability

• A problem is **decidable** if there is an algorithm to solve it
  – An algorithm is a Turing machine that halts on all inputs (accepts or rejects)
  – Therefore, an algorithm must always halt

• Problems that are not decidable are called **undecidable** (also called **semi-decidable**, **Turing-recognizable**, or **recursively enumerable**)
  – In addition, there are problems that are not even Turing-recognizable
Decidable Languages

• A language $L$ is **decidable** if and only if there is a Turing machine $M$ that decides it

• $M$ decides a language $L \subseteq \Sigma^*$ if and only if:
  – For any string $w \in \Sigma^*$
    • if $w \in L$ then $M$ accepts $w$
    • if $w \notin L$ then $M$ rejects $w$
  – In this case, we will say that $L$ is in language class $D$, the set of **decidable** (recursive) languages
Decidable Languages

• Every regular language is decidable
  – We could create a Turing machine to decide any regular language
  – Turing machines are like “compilers” for regular languages, in that any given string (“program”) in the language is either accepted or rejected

• Similarly, every context-free language is decidable

• Similarly, any programming language you can think of is decidable
Some Decidable Problems

• Does a particular DFA accept a given input string?
  – Check if it lands in accepting or rejecting state

• Does a DFA accept any string at all?
  – Check if the DFA can reach an accept state from the start state by traveling along the transition arrows

• Do two DFAs (DFA A and DFA B) accept the same language?
  – Construct a new DFA C that accepts only those strings that are accepted by either A or B but not both. $L(C)$ is the symmetric difference between A and B:
    
    $L(C) = (L(A) \cap \neg L(B)) \cup (\neg L(A) \cap L(B))$. $L(C) = \emptyset$ iff $L(A) = L(B)$
Some Decidable Problems

• Does a regular expression generate a particular string?
  – Convert the RE to an NFA and check for accept or reject

• Does a CFG generate a particular string?
  – Convert to Chomsky Normal Form, check derivations that have at most $2n - 1$ steps where $|w| = n$

• Does a CFG generate any string at all?
  – Check a variable to determine if it is capable of generating a string of terminals; if not, then mark that variable and try the next one until all have been tried
Non-Deterministic Deciding

• Let $M$ be a non-deterministic Turing machine and let $w$ be a string in $\Sigma^*$
  
  – $M$ **accepts** $w$ if and only iff at least one of its computations accepts $w$
  
  – $M$ **rejects** $w$ if and only if all of its computations reject $w$
  
  Therefore, a non-deterministic $M$ **decides** a language if and only if, for any $w$ in $\Sigma^*$
    
    • There is a finite number of paths that $M$ can follow on input $w$
    • All of those paths halt by either accepting or rejecting $w$, and
    • If $w \in L$, then there is at least one path such that $M$ accepts $w$
Non-Deterministic Deciding

• What happens if some paths halt and others don’t?
  – This is called “semi-deciding”

• Semi-deciding requires only that there exists at least one path that halts and accepts $w$

• With semi-deciding, we don’t care how many non-accepting (i.e., looping or rejecting) paths there are
Semi-Decidable Languages

• A language $L$ is **semi-decidable** if and only if there is a Turing machine that semi-decides it

• $M$ **semi-decides** $L \subseteq \Sigma^*$ if and only if
  – For any string $w \in \Sigma^*$
    • $w \in L \rightarrow M$ accepts $w$
    • $w \notin L \rightarrow M$ does not accept $w$ (in this case, $M$ may either reject or it may fail to halt)
  – In this case, we will say that $L$ is in $SD$, the set of **semi-decidable** (undecidable, recursively enumerable, or Turing-recognizable) languages
Some Semi-Decidable Problems

• Given two context-free languages, do they generate the same language?
• Is a given context-free grammar ambiguous?
• Does a given Turing machine accept a given string?
• Can a particular line of code in a program ever be executed?
Example of a Semi-Decidable Language

Let \( L = b^*a(a \cup b)^* \) (every string has at least one \( a \))

We can build a Turing machine to semi-decide \( L \) (note that we could also build a machine that decides \( L \), since \( L \) is regular):

1. Loop:
   1.1 Move one square to the right.
   1.2 If the character under the head is an \( a \), halt and accept.

A deciding Turing machine for \( L \): If the character is a blank, halt and reject.

This is an example of a language that is in both D and SD, i.e., D is a subset of SD. There are some languages that are in SD but are not in D, so sometime a semi-deciding Turing machine in the best we will be able to do.
The Universal Turing Machine

• An example of a semi-decidable, but not decidable, language is the language $L_u$ of a Universal Turing Machine (UTM)
  – A Universal Turing Machine is a Turing machine that accepts another Turing machine as input

• The UTM takes as input the code for some TM $M$, along with a binary string $w$, and accepts input string $<M, w>$ if and only if $M$ accepts $w$
The Language H

• Consider the language $H = \{<M, w>: M \text{ halts on input string } w\}$
  
  – $H$ is a Universal Turing Machine and $M$ is a Turing machine; thus $M$ can be thought of as an algorithm or a \textit{computer program} with input $w$

• $H$ is known as “The Halting Problem”

• \textit{Theorem}: The language $H$ is not decidable; it is semi-decidable
Proof that H is undecidable (proof by contradiction): Assume that H is decidable. If so, then you could write a function that implements H called $\text{halts}(<M, w>)$ that returns $\text{true}$ if $M$ halts on input $w$, or $\text{false}$ if $M$ is stuck in an infinite loop. Note that we are assuming that $\text{halts}$ is decidable (it always halts, returning $\text{true}$ or $\text{false}$).

Now suppose there is a program called $\text{Trouble}(x: \text{string})$ {
  
  if ($\text{halts}(x, x)$) then loop forever; else halt and return
}

Consider $\text{Trouble}(\text{Trouble})$. What is $\text{halts}(<\text{Trouble}, \text{Trouble}>)$?
- If $\text{halts}$ returns $\text{true}$ then $\text{Trouble}$ loops forever (but it halts!)
- If $\text{halts}$ returns $\text{false}$ then $\text{Trouble}$ halts (but it loops forever!)

Thus, since there is at least one $M$ for which $\text{halts}$ can never do the right thing, $\text{halts}(<M, w>)$ is undecidable. Since $\text{halts}$ implements H, the assumption that H is decidable is incorrect.
Halting is Undecidable

If we claim that H is decidable, then H should be able to get the correct value for any cell in this table. Let ‘1’ indicate machine$_i$ halts, and ‘blank’ indicate machine$_i$ loops forever.

<table>
<thead>
<tr>
<th></th>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$i_3$</th>
<th>...</th>
<th>&lt;Trouble&gt;</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$machine_1$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$machine_2$</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$machine_3$</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trouble</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

According to the table above, *Trouble* should loop forever on input <Trouble>. But since *halts* does the opposite of what *Trouble* wants it to do, *Trouble* will halt. If instead, Trouble should halt on input <Trouble>, then *halts* will cause *Trouble* to loop forever.
The Big Picture

• Some languages that are in SD are also in D:
  – \{w \in \{a, b, c\}^* : a^n b^n c^n, n \geq 0\}
  – \{w \in \{a, b\}^* : w \subseteq w\}
  – \{w \in \{a, b\}^* : ww\}
  – \{x, y, z \in \{0, 1\}^* : |x| \cdot |y| = |z|\}

• But there are languages that are in SD but not in D:
  – H = \{<M, w> : M halts on input w\}
  – L = \{w : w is the email address of someone who will respond to a message you just posted on your Facebook page\}
    • If someone responds, you know that their email address is in L. But if your best friend hasn’t responded yet, you don’t know that she isn’t going to. All you can do is wait.
Language Summary So Far

- SD
- H
- D
- $A^nB^nC^n$
- Context-Free
- $A^nB^n$
- Regular
- $a^*b^*$
Closure Properties of D and SD Languages

- Both D and SD are closed under union, intersection, concatenation, star, and reversal
- D is closed under difference and complement
- SD is not closed under difference and complement
Closure Under Union

• Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$
  – $L_1$ and $L_2$ are languages accepted by Turing machines $M_1$ and $M_2$
• Construct a 2-tape Turing machine $M$ that has a copy of the input on both tapes and simulates $M_1$ and $M_2$, each on one of the two tapes, “in parallel”
• $M$ accepts if either $M_1$ or $M_2$ accepts
D is Closed Under Union

• For decidable languages: M accepts if either $M_1$ or $M_2$ accept; rejects if both $M_1$ and $M_2$ reject
  • If $M_1$ and $M_2$ are both algorithms, then M is also an algorithm

Remember: Reject means “halt without accepting”
SD is Closed Under Union

• For SD languages: M accepts if either $M_1$ or $M_2$ accept
  • Either $M_1$ or $M_2$ might run forever if they don’t accept
  • If either $M_1$ or $M_2$ are not necessarily algorithms, then M is also not necessarily an algorithm

M, with input w

- $M_1$ $ightarrow$ Accept
- $M_2$ $ightarrow$ Accept

OR $\rightarrow$ M Accepts
D is Closed Under Intersection

M, with input w

M₁
Accept
Reject

M₂
Accept
Reject

AND
M Accepts

OR
M Rejects
SD is Closed Under Intersection

M, with input w

M₁

Accept

AND

M₂

Accept

M Accepts
D is Closed Under Concatenation

- Partition \( w = xy \in L_D \) where \( x \in L_{Dx} \) and \( y \in L_{Dy} \)
- Let \( M = M_1M_2 \) and run \( x \) on \( M_1 \) and \( y \) on \( M_2 \)
- Since \( M_1 \) and \( M_2 \) are in \( D \), \( M \) will eventually halt for each partition
- Accept if both accept for any one partition
- Reject if all partitions are tried and none lead to acceptance
SD is Closed Under Concatenation

• Use non-determinism to partition the string, since we don’t know when to reject (M may never halt)

• Construct a 2-tape non-deterministic TM M:
  1. Guess a way to break input into 2 pieces, \( w = xy \)
  2. Move \( x \) to first tape and \( y \) to second tape
  3. Simulate \( M_1 \) on \( x \) and \( M_2 \) on \( y \)
  4. Accept if both \( M_1 \) and \( M_2 \) accept
D and SD are Closed Under Star

- The same ideas work for both decidable and semi-decidable languages
- For decidable languages: systematically try all ways to partition the input string into pieces
- For semi-decidable languages: non-deterministically guess many partitions, accept if each piece is accepted
D and SD are Closed Under Reversal

• Reverse the input string
• Then simulate a Turing machine that will accept \( w \) if and only if \( w^R \) is also accepted
• Works for both decidable and semi-decidable languages
  – Because only acceptance is needed
Difference and Complement

• For decidable languages, both $M_1$ and $M_2$ will eventually halt
  – Accept $M_1 - M_2$ if $M_1$ accepts and $M_2$ rejects
  – Corollary: Decidable languages must also be closed under complement

• Semi-decidable languages are not closed under difference or complement
  – $M_2$ may never halt
Theorem: D is closed under complement.

Proof: (by construction) If $L$ is in D, then there is a deterministic Turing machine $M$ that decides it. $M$:

From $M$, we construct $M'$ to decide $\neg L$: Swap the accept and reject states. $M'$ halts and accepts whenever $M$ would halt and reject; $M'$ halts and rejects whenever $M$ would halt and accept. Since $M$ always halts, so does $M'$. 

D is Closed Under Complement
**SD is Not Closed Under Complement**

*Theorem:* SD is not closed under complement.

*Proof:* (by contradiction) Suppose that SD were closed under complement. Then, given any language \( L \) in SD, \( \neg L \) would also be in SD. So there would be a Turing machine \( M \) that semi-decides \( L \) and another Turing machine \( \neg M \) that semi-decides \( \neg L \). From \( M \) and \( \neg M \) we could construct another Turing machine \( M\# \) that decides \( L \): On input \( w \), \( M\# \) will simulate \( M \) and \( \neg M \) in “parallel”. Since \( w \) must be an element of either \( L \) or \( \neg L \), one of \( M \) or \( \neg M \) must eventually accept. If \( M \) accepts then \( M\# \) halts and accepts. If \( \neg M \) accepts, then \( M\# \) halts and rejects. So if the SD languages were closed under complement, then both \( L \) and \( \neg L \) would be in D (all SD languages would be in D).

But we know that there is at least one language (\( H \)) that is in SD but not in D. Contradiction; therefore, SD can not be closed under complement.
Theorem: A language is in D if and only if both it and its complement are in SD

Proof:

• $L$ in D implies $L$ and $\neg L$ are in SD:
  – $L$ is in SD because $D \subseteq SD$
  – $D$ is closed under complement
  – So $\neg L$ is also in D and thus in SD
• $L$ and $\neg L$ are in SD implies $L$ is in D:
  – $M$ semi-decides $L$
  – $\neg M$ semi-decides $\neg L$
  – From these two, construct $M\#$ to decide $L$:
    • Run $M$ and $\neg M$ in parallel on $w$
    • Exactly one of them will eventually accept
1. D is a subset of SD. In other words, every decidable language is also semi-decidable.

2. There exists at least one language that is in SD but not in D (the donut in the figure below).

3. There exist languages that are not in SD. In other words, the gray area of the figure below is not empty.
Languages That are Not in SD

• **Theorem:** There are languages that are not in SD (i.e., there are languages that are not Turing-recognizable)

• **Proof:** We will use a counting argument:
  – **Lemma:** There is a countably infinite number of SD languages over $\Sigma$
  – **Lemma:** There is an uncountably infinite number of languages over $\Sigma$. So there are more languages than there are languages in SD. Thus there must exist at least one language that is in $\neg$SD.
\[ \neg H \text{ is Not in SD} \]

- The language \( \neg H = \{<M, w>: M \text{ does not halt on input string } w\} \) is not in SD

- **Proof:**
  - \( H \) is in SD
  - If \( \neg H \) were also in SD then \( H \) would be in D
  - But \( H \) is not in D
  - So \( \neg H \) can not be in SD
## Summary of Decidability

<table>
<thead>
<tr>
<th>The Problem View</th>
<th>The Language View</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does TM $M$ have an even number of states?</td>
<td>${&lt;M&gt; : M \text{ has an even number of states}}$</td>
<td>D</td>
</tr>
<tr>
<td>Does TM $M$ halt on $w$?</td>
<td>${&lt;M, w&gt; : M \text{ halts on } w}$</td>
<td>SD, ¬D</td>
</tr>
<tr>
<td>Does TM $M$ halt on the empty tape?</td>
<td>${&lt;M&gt; : M \text{ halts on } \varepsilon}$</td>
<td>SD, ¬D</td>
</tr>
<tr>
<td>Is there any string on which TM $M$ halts?</td>
<td>${&lt;M&gt; : \text{there exists at least one string on which TM } M \text{ halts}}$</td>
<td>SD, ¬D</td>
</tr>
<tr>
<td>Does TM $M$ accept $w$?</td>
<td>${&lt;M, w&gt; : M \text{ accepts } w}$</td>
<td>SD, ¬D</td>
</tr>
<tr>
<td>Does TM $M$ accept $\varepsilon$?</td>
<td>${&lt;M&gt; : M \text{ accepts } \varepsilon}$</td>
<td>SD, ¬D</td>
</tr>
<tr>
<td>Is there any string that TM $M$ accepts?</td>
<td>${&lt;M&gt; : \text{there exists at least one string that TM } M \text{ accepts}}$</td>
<td>SD, ¬D</td>
</tr>
</tbody>
</table>
## Summary of Decidability

<table>
<thead>
<tr>
<th>Question</th>
<th>Description</th>
<th>Decidability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does TM $M$ not halt on any string?</td>
<td>$&lt;M&gt;$: there does not exist any string on which $M$ halts</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ halt on all strings?</td>
<td>$&lt;M&gt;$: $M$ halts on $\Sigma^*$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ accept all strings?</td>
<td>$&lt;M&gt;$: $L(M) = \Sigma^*$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Do TMs $M_a$ and $M_b$ accept the same languages?</td>
<td>$&lt;M_a, M_b&gt;$: $L(M_a) = L(M_b)$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ not halt on its own description?</td>
<td>$&lt;M&gt;$: TM $M$ does not halt on input $&lt;M&gt;$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Is TM $M$ minimal?</td>
<td>$&lt;M&gt;$: $M$ is minimal</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Is the language that TM $M$ accepts regular?</td>
<td>$&lt;M&gt;$: $L(M)$ is regular</td>
<td>$\neg$SD</td>
</tr>
</tbody>
</table>