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
  – In that sense, then, the class of regular languages is like a “programming language”

• 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 each variable to determine if it is capable of generating a string of terminals; if so, 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
  • 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?
  – This is however, Turing-recognizable
• 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.

Question: What if we have a very long input string? How long should we look for an \( a \) until we figure out that the string should be rejected?
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).
- 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 $H = \{ <M, w> : M \text{ halts on input string } w \}$
- This is known as “The Halting Problem”
- **Theorem:** The language $H$ is semi-decidable but is *not decidable*
H is Undecidable

Proof that H is undecidable (proof by contradiction): Assume that we have a function called \( \text{halts}(<M>, w) \) that returns \( \text{true} \) if \( w \) is accepted by \( M \), or \( \text{false} \) if \( w \) is rejected by \( M \). Note that we are assuming that \( \text{halts} \) is decidable (i.e., it always halts by either accepting or rejecting \( w \)).

Now define a program called \( \text{Trouble}(x: \text{string}) \) {
  if \( \text{halts}(x, x) \), then \( \text{Trouble} \) loops forever, otherwise \( \text{Trouble} \) halts}

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 should halt)
  ● If \( \text{halts} \) returns \( \text{false} \) then \( \text{Trouble} \) halts (but it should not halt)

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

Lexicographically enumerate all Turing machines and strings \( i \). Let ‘1’ indicate machine halts, and ‘blank’ indicate loops forever.

According to the table above, \textit{Trouble} should loop forever on input \(<\text{Trouble}>\). But because \textit{halts} does the opposite of what \textit{Trouble} wants it to do, \textit{Trouble} will halt. If instead, \textit{Trouble} should halt on input \(<\text{Trouble}>\), then \textit{halts} will cause \textit{Trouble} to loop forever.
The Big Picture

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

• But there are languages that are in SD but not in D:
  – \( H = \{<M, w> : M \text{ halts on input } w \} \)
  – \( E = \{ w : w \text{ is the email address of someone who will respond to a message you just posted on your Facebook page} \} \)
    • 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

- Regular Languages: $a^*b^*$
- Context-Free Languages: $A^nB^n$
- $A^nB^nC^n$
- $H$
- SD
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: If $M_1$ and $M_2$ are both algorithms, then $M$ will always halt for any accepted or rejected input string.

**Diagram:**

- **Input $w$**
- **$M_1$**
  - Accept
  - Reject
- **$M_2$**
  - Accept
  - Reject
- **$M$**
  - OR
    - Accept
  - AND
    - Reject

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

• For SD languages: accept if either accepts, but both $M_1$ and $M_2$ might run forever without halting or accepting
  – That is OK, since we only want to prove that $L_1 \cup L_2$ is semi-decidable in this case
D is Closed Under Intersection

Input w

$M_1 \rightarrow$ Accept
$M_1 \rightarrow$ Reject

$M_2 \rightarrow$ Accept
$M_2 \rightarrow$ Reject

$M \rightarrow$ Accept
$M \rightarrow$ Reject

AND $\rightarrow$ Accept
OR $\rightarrow$ Reject
SD is Closed Under Intersection

Input $w$

$M_1$ $\xrightarrow{\text{Accept}}$ $M$

$M_2$ $\xrightarrow{\text{Accept}}$ $M$

AND $\xrightarrow{\text{Accept}}$
D is Closed Under Concatenation

• Systematically try every way to partition $w$ into two pieces, $x$ and $y$
• $M_1$ and $M_2$ 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

• 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 non-deterministic TM $M$:
  1. Guess a break in input $w = xy$
  2. Move $y$ to second tape
  3. Simulate $M_1$ on $x$, $M_2$ on $y$
  4. Accept if both 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

• Start by reversing the input
• Then simulate a Turing machine for $L$ to accept $w$ if and only if $w^R$ is in $L$
• Works for both decidable and semi-decidable languages
Difference and Complement

• For decidable languages, both $M_1$ and $M_2$ will eventually halt
  – Accept if $M_1$ accepts and $M_2$ rejects
  – Corollary: Decidable languages are closed under complementation

• Semi-decidable languages are **not closed** under difference or complement
  – $M_2$ may never halt, so you can’t be sure the input is in the difference
**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 $\overline{L}$: Swap the *accept* and *reject* states. $M'$ halts and accepts whenever $M$ would halt and reject; $M'$ halts and accepts whenever $M$ would halt and accept. Since $M$ always halts, so does $M'$. 

![Diagram](image-url)
**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.
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 ¬L are in SD:
  – L is in SD because D ⊂ SD
  – D is closed under complement
  – So ¬L is also in D and thus in SD
• L and ¬L are in SD implies L is in D:
  – M semi decides L
  – ¬M semi decides ¬L
  – From these two, construct M# to decide L:
    • Run M and ¬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 \(\neg H\) is not 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>$H = {&lt;M, w&gt;: M \text{ halts on } w}$</td>
<td>SD, $\neg$D</td>
</tr>
<tr>
<td>Does TM $M$ halt on the empty tape?</td>
<td>$H_\varepsilon = {&lt;M&gt;: M \text{ halts on } \varepsilon}$</td>
<td>SD, $\neg$D</td>
</tr>
<tr>
<td>Is there any string on which TM $M$ halts?</td>
<td>$H_{\text{ANY}} = {&lt;M&gt;: \text{there exists at least one string on which TM } M \text{ halts}}$</td>
<td>SD, $\neg$D</td>
</tr>
<tr>
<td>Does TM $M$ halt on all strings?</td>
<td>$H_{\text{ALL}} = {&lt;M&gt;: M \text{ halts on } \Sigma^*}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ accept $w$?</td>
<td>$A = {&lt;M, w&gt;: M \text{ accepts } w}$</td>
<td>SD, $\neg$D</td>
</tr>
<tr>
<td>Does TM $M$ accept $\varepsilon$?</td>
<td>$A_\varepsilon = {&lt;M&gt;: M \text{ accepts } \varepsilon}$</td>
<td>SD, $\neg$D</td>
</tr>
<tr>
<td>Is there any string that TM $M$ accepts?</td>
<td>$A_{\text{ANY}} = {&lt;M&gt;: \text{there exists at least one string that TM } M \text{ accepts}}$</td>
<td>SD, $\neg$D</td>
</tr>
</tbody>
</table>
# Summary of Decidability

<table>
<thead>
<tr>
<th>Does TM $M$ accept all strings?</th>
<th>$A_{ALL} = {&lt;M&gt;: L(M) = \Sigma^*}$</th>
<th>$\neg$SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do TMs $M_a$ and $M_b$ accept the same languages?</td>
<td>$EqTMs = {&lt;M_a, M_b&gt;: L(M_a) = L(M_b)}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ not halt on any string?</td>
<td>$H_{\neg\text{ANY}} = {&lt;M&gt;: \text{there does not exist any string on which } M \text{ halts}}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ not halt on its own description?</td>
<td>${&lt;M&gt;: \text{TM } M \text{ does not halt on input } &lt;M&gt;}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Is TM $M$ minimal?</td>
<td>$TM_{MIN} = {&lt;M&gt;: M \text{ is minimal}}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Is the language that TM $M$ accepts regular?</td>
<td>$TM_{reg} = {&lt;M&gt;: L(M) \text{ is regular}}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ accept the language $A^nB^n$?</td>
<td>$A_{anbn} = {&lt;M&gt;: L(M) = A^nB^n}$</td>
<td>$\neg$SD</td>
</tr>
</tbody>
</table>