Turing Machines

Is the Halting problem recursively enumerable?

Is the halting problem rec.enum?

• Recall we used the solution to problem 8.3 of the textbook, given at the end of the exercises
• \( L_h = \{<M> : M \text{ halts on all inputs}\} \) is not recursively enumerable
• We prove that if \( L_h \) is rec. enum., then the property \( L = (0+1)^* \) would be rec. enum.

We need an \( M' \)

• Given \(<M>\), construct \( M' \) so that \( L(M) = L(M') \) and \( M' \) ONLY halts on \( L(M) \)
• \( M' \) shifts its input one space to the right, putting “$” at the first position and then simulates \( M \)
• If you reach “$” or reach a non-final state that has no transition move to new state \( p_1 \), from there oscillate between \( p_1 \) and new state \( p_2 \)
A non-trivial property would be decidable

- Suppose $L_h$ is rec. enum. and $M_h$ is the TM for $L_h$
- Construct an TM for $L_{S4} = \{<M>: L(M) = (0+1)^* \}$ as follows:

\[
\text{The monster for } \{<M>: L(M) = (0+1)^* \}
\]

\[<M> \quad \rightarrow \quad \text{make } M' \quad \rightarrow \quad \text{make } M'' \quad \rightarrow \quad \text{Yes}\]

- If $L(M) = (0+1)^*$, the monster machine says yes since $M'$ halts on all inputs
- If $L(M) \neq (0+1)^*$, the monster says no or never halts

Apply Rice's 2nd theorem

- All together, we have a TM for $L_{S4} = \{<M>: L(M) = (0+1)^* \}$
- Such a TM does not exist by Rice's second theorem

An easy result

- Suppose we take the problem “Does $M$ halt on a fixed $w$?”
- Just run $M$ on $w$, if it halts, output a “yes”
- This problem is recursively enumerable but not recursive
Marginally harder result

- Similarly, there is a TM to answer: “Does $M$ halt on some input?”
- On one tape generate $k = 2,3,4,...$
- Then on two more tapes, generate $i$ and $j$ so that $i + j = k$ ($i > 0, j > 0$)
- For each $k$, $i$, $j$ run $M$ for up to $i$ steps on the word $w_j$ in the canonical sequence
- Accept if and when $M$ halts for some $w_j$

State entry

- Do the same thing as on the previous slide: run $M$ on each $w_j$ for up to $i$ steps for all $i + j = k$ and $k = 2,3,4,...$
- This time accept if $M$ enters a pre-specified state
- We will accept every $M$ that enters a particular state on some input

Summary

- We saw that with regard to decidability the four problems: $M$ halts on all inputs, $M$ halts on a fixed $w$, $M$ halts on some $w$, $M$ reaches a specific state on some $w$
- all have the same property: they are undecidable
- But only the first of the 4 is not recursively enumerable
Context-free Grammars

Greibach Normal Form
Membership is decidable

Greibach Normal Form (GNF)

- Sheila Greibach's normal form (GNF) for a CFG is one where EVERY production has the form: $A \rightarrow a\alpha$, where $a \in T$ and $\alpha \in V^*$

Conversion to Greibach Normal Form is tedious
Start with CNF and number the symbols, e.g. $S = A_1$, $A = A_2$, $B = A_3$
$G = \{A_1, A_2, A_3\}, \{a, b\}, P, A_1$
$A_1 \rightarrow A_2 A_3$, $A_2 \rightarrow A_1 A_3 | A_3 A_1 | b$, $A_3 \rightarrow A_1 A_2 | a$
We need to convert all the productions with variables to the form $A_j \rightarrow A_k...$, where $j < k$

GNF--textbook's example modified

$A_1 \rightarrow A_2 A_3$ is OK but $A_2 \rightarrow A_1 A_3$ and $A_3 \rightarrow A_1 A_2$ need work:
Substitution:
$A_2 \rightarrow A_2 A_3 A_3 | A_3 A_1 | b$
Left-recursion removal
$A_2 \rightarrow A_3 A_1 | b | A_3 A_1 B_2 | bB_2$, $B_2 \rightarrow A_3 A_3 | A_3 A_3 B_2$
We are using: Lemma 4.3, Lemma 4.4

Lemma 4.3: Let \( A \rightarrow \alpha_1 B \alpha_2 \) be a production and let \( B \rightarrow \beta_1 | \beta_2 | \ldots | \beta_r \) be \textit{all} the productions for \( B \). Then the grammar can be modified by \textit{removing} \( A \rightarrow \alpha_1 B \alpha_2 \), provided the following productions are \textit{added}:
\[
A \rightarrow \alpha_1 \beta_1 \alpha_2 \mid \alpha_1 \beta_2 \alpha_2 \mid \ldots \mid \alpha_1 \beta_r \alpha_2
\]

Can be done differently

Note: the compiler books usually have a slightly different version to keep the grammar smaller.

Example: compare the derivations
in the old grammar
\[
A \Rightarrow A \Rightarrow A \alpha_5 \Rightarrow A \alpha_1 \alpha_5 \Rightarrow \ldots \Rightarrow A \alpha_3 \alpha_8 \ldots \alpha_5
\Rightarrow \beta_2 \alpha_3 \alpha_8 \ldots \alpha_5
\]

in the new grammar
\[
A \Rightarrow \beta_2 B \Rightarrow \beta_2 \alpha_3 B \Rightarrow \beta_2 \alpha_3 \alpha_8 \Rightarrow \ldots \Rightarrow \beta_2 \alpha_3 \alpha_8 \ldots \alpha_5
\]

Back to the GNF example

Now for \( A_3 \rightarrow A_1 A_2 \)

first substitution: \( A_3 \rightarrow A_2 A_3 A_2 | a \)

then replace \( A_2 \)

\[
A_3 \rightarrow A_3 A_1 A_2 A_3 A_2 \mid b A_3 A_2 \mid A_3 A_1 B_2 A_3 A_2
\mid b B_2 A_3 A_2 \mid a
\]
End of Phase 1

- Now remove left recursion
- $A_3 \rightarrow bA_3 A_2 | bB_2 A_3 A_2 | a | bA_3 A_2 B_3$
  | $bB_2 A_3 A_2 B_3 | aB_3$
- $B_3 \rightarrow A_1 A_3 A_2 | A_1 B_2 A_3 A_2 | A_1 A_3 A_2 B_3$
  | $A_1 B_2 A_3 A_2 B_3$
- Since there is no $A_4$, all of $A_3$'s productions must be in GNF

Substitute, substitute, substitute

- $A_3 \rightarrow bA_3 A_2 | bB_2 A_3 A_2 | a | bA_3 A_2 B_3$
  | $bB_2 A_3 A_2 B_3 | aB_3$
- hence $A_2 \rightarrow A_3 A_1 | b | A_3 A_1 B_2 | bB_2$
become
- $A_2 \rightarrow bA_3 A_2 A_1 | bB_2 A_3 A_2 A_1 | aA_1$
  | $bA_3 A_2 B_3 A_1 | bB_2 A_3 A_2 B_3 A_1$
  | $aB_3 A_1 | b | bA_3 A_2 A_1 B_2$
  | $bB_2 A_3 A_2 A_1 B_2 | aA_1 B_2$
  | $bA_3 A_2 B_3 A_1 B_2$
  | $bB_2 A_3 A_2 B_3 A_1 B_2 | aB_3 A_1 B_2$
  | $bB_2$

Finish with $A_1$, $A_2$, $A_3$

- Now we have to fix $A_1 \rightarrow A_2 A_3$:
  - $A_1 \rightarrow bA_3 A_2 A_1 A_3 | bB_2 A_3 A_2 A_1 A_3$
    | $aA_1 A_3 | bA_3 A_2 B_3 A_1 A_3$
    | $bB_2 A_3 A_2 B_3 A_1 A_3 | aB_3 A_1 A_3 | bA_3$
    | $bA_3 A_2 A_1 B_2 A_3 | bB_2 A_3 A_2 A_1 B_2 A_3$
    | $aA_1 B_2 A_3 | bA_3 A_2 B_3 A_1 B_2 A_3$
    | $bB_2 A_3 A_2 B_3 A_1 B_2 A_3 | aB_3 A_1 B_2 A_3$
    | $bB_2 A_3$
- That leaves $B_2$ and $B_3$

Then do $B_2$

- $B_2 \rightarrow A_3 A_3 | A_3 A_3 B_2$
become
- $B_2 \rightarrow bA_3 A_2 A_3 | bB_2 A_3 A_2 A_3 | aA_3$
  | $bA_3 A_2 B_3 A_3 | bB_2 A_3 A_2 B_3 A_3 | aB_3 A_3$
  | $bA_3 A_2 A_3 B_2 | bB_2 A_3 A_2 A_3 B_2 | aA_3 B_2$
  | $bA_3 A_2 B_3 A_3 B_2 | bB_2 A_3 A_2 B_3 A_3 B_2$
  | $aB_3 A_3 B_2$
- and finally $B_3$:
and I don't care if there is a mistake!

- $B_3 \rightarrow A_1 A_3 A_2 | A_1 B_2 A_3 A_2 | A_1 A_3 A_2 B_3$
  \quad | A_1 B_2 A_3 A_2 B_3$

become

- $B_3 \rightarrow bA_3 A_2 A_1 A_3 A_2$
  \quad | bB_2 A_3 A_2 A_1 A_3 A_2 | aA_1 A_3 A_2 A_2$
  \quad | bA_3 A_2 B_3 A_1 A_3 A_2$
  \quad | bB_2 A_3 A_2 B_1 A_3 A_2 | aA_1 B_2 A_3 A_2$
  \quad | bA_3 A_2 B_3 A_1 A_2 A_2$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 | aA_1 A_3 A_2 A_2$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 | aA_1 B_2 A_3 A_2$
  \quad | bB_2 A_3 A_2 A_1 A_3 A_2 B_2 | bA_3 A_2 B_3 A_2 A_2$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 | aA_1 A_3 A_2 A_2$
  \quad | bB_2 A_3 A_2 B_1 A_3 A_2 | aA_1 B_2 A_3 A_2$
  \quad | aB_3 A_1 B_2 A_3 A_2 | bB_2 A_3 A_2 A_2$

and going...

- $B_3 \rightarrow bA_3 A_2 A_1 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 A_3 A_2 B_3 | aA_1 A_3 A_2 B_3$
  \quad | bA_3 A_2 B_3 A_1 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 B_1 A_3 A_2 B_3 | aB_3 A_1 A_3 A_2 B_3$
  \quad | bA_3 A_2 B_3 A_1 A_2 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | aA_1 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | aA_1 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 A_3 A_2 B_3 | bB_2 A_3 A_2 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | aB_3 A_1 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | bB_2 A_3 A_2 A_2 B_3$

and done!

- $B_3 \rightarrow bA_3 A_2 A_1 A_3 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 A_3 B_2 A_3 A_2 B_3 | aA_1 A_3 B_2 A_3 A_2 B_3$
  \quad | bA_3 A_2 B_3 A_1 A_3 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 B_1 A_3 B_2 A_3 A_2 B_3 | aA_1 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | aA_1 A_3 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | aA_1 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 A_3 A_2 B_3 | bB_2 A_3 A_2 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | aB_3 A_1 B_2 A_3 A_2 B_3$
  \quad | bB_2 A_3 A_2 A_1 B_2 A_3 A_2 B_3 | bB_2 A_3 A_2 A_2 B_3$
Membership

- Here is an potentially exponential algorithm to test if \( w \) is in a CFL
- If \( w \) is the empty string, simply test if the start symbol \( S \) is nullable
- If \( w \neq \varepsilon \), convert the grammar of the CFL to GNF
- If \( w = a_1a_2\ldots a_n \) we consider all \( S \rightarrow a_1\alpha_1 \)
- Follow up on all such productions that could give \( w \):

Keep track of all possibilities

- Look a little deeper: Each possibility for \( \alpha_1 \) will begin with some \( A_i \) and then we list all productions \( A_i \rightarrow a_2\alpha_2 \) since the first symbol of \( \alpha_1 \) must be the one that derives \( a_2 \).
- For all possible \( \alpha_2 \) look at the first variable \( A_j \). To be used for \( w \), it must have one or more productions \( A_j \rightarrow a_3\alpha_3 \)

If there is a derivation, we found it

- Keep accumulating all these derivations and stop at length \( |w| \)
- Since a production the GNF can never derive a shorter string, one of the sequences found must be a derivation of \( w \) if there is one
- This process is potentially exponential but there is an \( n^3 \) algorithm

Cocke, Younger, Kasami algorithm

CYK algorithm to test for membership
The CYK algorithm

- We may also use the CNF to show that there is an $n^3$ algorithm to determine membership.
- Given a CFL $L$ and string $w$, it is decidable whether $w$ belongs to $L$.
- The algorithm has complexity $O(n^3)$, where $n = |w|$.

The CYK algorithm - II

- The Cocke-Younger-Kasami algorithm works as follows:
  - Find a CNF grammar for $L$.
  - Take a string $a_1a_2...a_n$.
  - We first find all the variables that generate the individual terminals, e.g. $A_1 \rightarrow a_1$, $A_3 \rightarrow a_1$, $A_4 \rightarrow a_2$, $A_6 \rightarrow a_2$, ...
    $A_1 \rightarrow a_n$.

The CYK algorithm - III

- Then we try to find the variables that generate pairs of terminals:
  - We had $A_1 \rightarrow a_1$, $A_6 \rightarrow a_2$.
  - Perhaps $A_2 \rightarrow A_1A_6$.
  - So $A_2 \Rightarrow a_1a_2$.

The CYK algorithm - IV

- Next we try to find the variables that generate triples of terminals:
  - We had $A_2 \Rightarrow a_1a_2$, $A_4 \rightarrow a_3$.
    Perhaps $A_3 \rightarrow A_2A_4$.
    So $A_3 \Rightarrow a_1a_2a_3$.
  - We may also have $A_3 \rightarrow a_1$, $A_5 \Rightarrow a_2a_3$.
    Perhaps $A_5 \rightarrow A_3A_5$ so $A_5 \Rightarrow a_1a_2a_3$. 
The CYK algorithm - V

- This process proceeds looking for all possible ways of generating the string of length \( n \).
- Example: \( S \rightarrow AB, A \rightarrow BC / SA / a, B \rightarrow CC / b, C \rightarrow CA / a \).
- We consider the string \( abaab \).

CYK: Steps 1 and 2

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

since

\( S \rightarrow AB \quad A \rightarrow BC \quad B \rightarrow CC \quad S \rightarrow AB \quad C \rightarrow CA \)

CYK: Step 3, strings of length 3 (\( aba \))

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1)

For location (1), we have to look for variables that generate \( AA, CA, SA, \) or \( SC \), since we can then generate \( aba \).
CYK: Step 3, strings of length 3 (baa)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>A,C</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

since

\[ A \rightarrow BC \]

CYK: Step 3, strings of length 3 (aab)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>A,C</td>
<td>A</td>
<td>(\emptyset)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

nothing generates \(AS, CS, BB\) or \(CB\)

CYK: Step 4, strings of length 4 (abaa)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>A,C</td>
<td>A</td>
<td>(\emptyset)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1)

For location (1), we have to look for variables that generate \(AA, CA, SB, SC, AA, AC, CA\) or \(CC\) since we can then generate \(abaa\)

since

\[ B \rightarrow CC \]
\[ C \rightarrow CA \]
CYK: Step 4, strings of length 4 (baab)

\[
\begin{array}{|c|c|c|c|}
\hline
a & b & a & a \\
\hline
A, C & B & A, C & A, C \\
S & A & B, C & S \\
A, C & A & \emptyset & A \\
B, C & S & & \\
\hline
\end{array}
\]

since

\[S \rightarrow AB\]

CYK: Step 5, string of length 5 (abaab)

\[
\begin{array}{|c|c|c|c|c|}
\hline
a & b & a & a & b \\
\hline
S & A & B, C & A, C & \emptyset \\
A, C & A & \emptyset & A \\
B, C & S & & S \\
\hline
\end{array}
\]

- Since S does not derive abaab, the string is not in L

Second example abaaa

\[
\begin{array}{|c|c|c|c|}
\hline
a & b & a & a \\
\hline
A, C & B & A, C & A, C \\
S & A & B, C & B, C \\
\hline
\end{array}
\]

Grammar: \[S \rightarrow AB, A \rightarrow BC / SA / a, B \rightarrow CC / b, C \rightarrow CA / a\]

Partial grammar: \[S \rightarrow AB, A \rightarrow BC | SA, B \rightarrow CC, C \rightarrow CA\]

Step 3

\[
\begin{array}{|c|c|c|c|c|}
\hline
a & b & a & a & a \\
\hline
A, C & A & S, A, B, C & B, C \\
\hline
\end{array}
\]
**Step 4**

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>A,C</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>B,C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A,C</td>
<td>A</td>
<td>S,A,B,C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B,C</td>
<td>S,A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partial grammar: \( S \rightarrow AB, A \rightarrow BC \mid SA, B \rightarrow CC, C \rightarrow CA \)

**Step 5**

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,C</td>
<td>B</td>
<td>A,C</td>
<td>A,C</td>
<td>A,C</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>B,C</td>
<td>B,C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A,C</td>
<td>A</td>
<td>S,A,B,C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B,C</td>
<td>S,A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partial grammar: \( S \rightarrow AB, A \rightarrow BC \mid SA, B \rightarrow CC, C \rightarrow CA \)

S derives \( abaaa \), so the string is in \( L \)