In the following languages, three are decidable and three are undecidable. Which are which?

- $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a } CFG \text{ that generates string } w \}$.
- $E_{CFG} = \{ \langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset \}$.
- $ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \Sigma^* \}$.
- $A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is a } LBA \text{ that accepts string } w \}$.
- $E_{LBA} = \{ \langle M \rangle \mid M \text{ is a } LBA \text{ where } L(M) = \emptyset \}$.
- $ALL_{LBA} = \{ \langle M \rangle \mid M \text{ is a } LBA \text{ where } L(M) = \Sigma^* \}$. 
ECE-374-B: Lecture 25 - Midterm 3 Review

Instructor: Abhishek Kumar Umrawal
November 05, 2023

University of Illinois at Urbana-Champaign
In the following languages, three are decidable and three are undecidable. Which are which?

- $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a } CFG \text{ that generates string } w \}$.  
- $E_{CFG} = \{ \langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset \}$.  
- $ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \Sigma^* \}$.  
- $A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is a } LBA \text{ that accepts string } w \}$.  
- $E_{LBA} = \{ \langle M \rangle \mid M \text{ is a } LBA \text{ where } L(M) = \emptyset \}$.  
- $ALL_{LBA} = \{ \langle M \rangle \mid M \text{ is a } LBA \text{ where } L(M) = \Sigma^* \}$.  

$A_{CFG}$ decidable?
$A_{CFG}$ decidable?

YES!
YES!

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \rightarrow \epsilon | 0S0 | 1S1\}$
  (abbrev. for $S \rightarrow \epsilon, S \rightarrow 0S0, S \rightarrow 1S1$)
YES!

**Lemma**

A CFG in Chomsky normal form can derive a string $w$ in at most $2^n$ steps!

Knowing this, we can just simulate all the possible rule combinations for $2^n$ steps and see if any of the resulting strings matches $w$. 
$E_{CFG}$ decidable? (YES)

In this case, we just need to know if we can get from the start variable to a string with only terminal symbols.

1. Mark all terminal symbols in $G$
2. Repeat until no new variables get marked:
   2.1 Mark any variable $A$ where $G$ has the rule $A \rightarrow U_1 U_2 ... U_k$ where $U_i$ is a marked terminal/variable
3. If start variable is not marked, accept. Otherwise reject.

$V = \{S\}$

$T = \{0, 1\}$

$P = \{S \rightarrow \varepsilon | S \rightarrow S_0 | S \rightarrow S_1\}$ (abbrev. for $S \rightarrow \varepsilon$, $S \rightarrow S_0$, $S \rightarrow S_1$)
YES!
YES!

In this case, we just need to know if we can get from the start variable to a string with only terminal symbols.

1. Mark all terminal symbols in $G$
2. Repeat until no new variables get marked:
   2.1 Mark any variable $A$ where $G$ has the rule $A \rightarrow U_1 U_2 \ldots U_k$ where $U_i$ is a marked terminal/variable
3. If start variable is not marked, accept. Otherwise reject.

\begin{itemize}
  \item $V = \{S\}$
  \item $T = \{0, 1\}$
  \item $P = \{S \rightarrow \epsilon \mid OS0 \mid 1S1\}$
    (abbrev. for $S \rightarrow \epsilon, S \rightarrow OS0, S \rightarrow 1S1$)
\end{itemize}
ALL$_{CFG}$ decidable? (Nope!)

Proof requires computation histories which are outside the scope of this course.
ALL_{CFG} decidable? (Nope!)
Nope!

Proof requires computation histories which are outside the scope of this course.
$A_{LBA}$ decidable?

Remember an LBA has a finite tape. Therefore we know:

1. A tape of length $n$ where each cell can contain $g$ symbols, you have $g^n$ possible configurations.
2. The tape head can be in one of $n$ positions and has $q$ states yielding a tape that can be in $q^n$ configurations.
3. Therefore the machine can be in $q^n g^n$ configurations.

Lemma
If an LBA does not accept or reject in $q^n g^n$ then it is stuck in a loop forever.
$A_{LBA}$ decidable? \underline{YES!}

Remember an LBA has a finite tape. Therefore we know:

1. A tape of length $n$ where each cell can contain $g$ symbols, you have $g^n$ possible configurations.

2. The tape head can be in one of $n$ positions and has $q$ states yielding a tape that can be in $qn$ configurations.

3. Therefore the machine can be in $q^ng^n$ configurations.

Lemma: If an LBA does not accept or reject in $q^ng^n$ then it is stuck in a loop forever.
A_{LBA} decidable? (RIV)

YES!

Remember a LBA has a finite tape. Therefore we know:

1. A tape of length $n$ where each cell can contain $g$ symbols, you have $g^n$ possible configurations.
2. The tape head can be in one of $n$ positions and has $q$ states yielding a tape that can be in $qn$ configurations.
3. Therefore the machine can be in $qng^n$ configurations.
YES!

Remember a LBA has a finite tape. Therefore we know:

1. A tape of length $n$ where each cell can contain $g$ symbols, you have $g^n$ possible configurations.
2. The tape head can be in one of $n$ positions and has $q$ states yielding a tape that can be in $qn$ configurations.
3. Therefore the machine can be in $qng^n$ configurations.

Lemma

If an LBA does not accept or reject in $qng^n$ then it is stuck in a loop forever.
Decider for $A_{LBA}$ will do the following.

1. Simulate $\langle M \rangle$ on $w$ for $qng^n$ steps.
   1.1 if accepts, then accept
   1.2 if rejects, then reject

2. If neither accepts or rejects, means it’s in a loop in which case, reject.
$E_{LBA}$ decidable? (RNW)
Nope!
Nope!

Proof requires computational history trick, a story for another time ...
ALL_{LBA} decidable? (RUY)
ALL_{LBA} decidable? (RIY)

Nope!
Nope!

No standard proof for this, but let’s look at a pattern as follows.
### Decidability across grammar complexities

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>CFG</th>
<th>PDA</th>
<th>LBA</th>
<th>TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>ALL</td>
<td>D</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

Eventually problems get too tough...
Nope!

No standard proof for this, but let’s look at a pattern:

So we sort of know that \( \text{ALL}_{\text{LBA}} \) isn’t decidable because we knew \( \text{ALL}_{\text{CFG}} \) wasn’t (though intuition is never sufficient evidence).
Rice’s theorem: Any ‘non-trivial’ property about the language recognized by a Turing machine is undecidable.
Un-/decidability practice problems
Available Undecidable languages

Undecidable due to the theorem we proved. (Anchor point)

- \[ L_{\text{Accept}} = \left\{ \langle M, w \rangle \mid M \text{ is a TM and accepts } w \right\} \].
- \[ L_{\text{HALT}} = \left\{ \langle M \rangle \mid M \text{ is a TM and halts on } \varepsilon \right\} \].

Undecidable as we did: \( L_{\text{Accept}} \Rightarrow L_{\text{HALT}} \)
Is the following language **undecidable**?

\[ L_{\text{HaltOnInput}} = \{ \langle M, w \rangle \mid M \text{ is a TM and halts on } w \} \]
Is the following language undecidable?

\[ L_{HasFooling} = \left\{ \langle M \rangle \mid M \text{ is a } TM \text{ and } L(M) \text{ has a fooling set} \right\} . \]
Reduction:
\[ \langle M \rangle, \langle w \rangle \rightarrow \langle M' \rangle \]

\[ M'(x) : \]
\[ \text{if } \overline{2} \text{ is of the form } 0^n1^n \]
\[ \text{accept} \quad \text{has a fooling set} \]
\[ \text{Else:} \]
\[ \text{Run } M \text{ on } w \]
\[ \text{accept} \]
\[ \text{if } M \text{ halts on } w \text{ then accept} \]
\[ \text{if } M \text{ doesn't halt then } M' \text{ accepts } 0^n1^n : 1FS \]
\[ \text{if } M \text{ halts then what's the language of } M' : \]
\[ L(C(M')) = \{0^n1^n : n \geq 0\} \]
\[ L(C(M)) = \Sigma^* \]
\[ \text{Regex: } (0+1)^* \]
\[ \text{doesn't have an 1FS!} \]
NP-Complete practice problems
A centipede is an undirected graph formed by a path of length $k$ with two edges (legs) attached to each node on the path as shown in the below figure. Hence, the centipede graph has $3k$ vertices. The **CENTIPEDE** problem is the following: given an undirected graph $G = (V, E)$ and an integer $k$, does $G$ contain a centipede of $3k$ distinct vertices as a subgraph? Prove that **CENTIPEDE** is NP-Complete.
What do we need to do to prove \textbf{Centipede is NP-Complete}? 

- Centipede is in \textit{NP}:
  - certificate: 3 arrays of vertices: \( s = \{ v_1, v_2, \ldots, v_k \} \): backbone
  - \( s' = \{ v_1', v_2', \ldots, v_k' \} \): upper leg
  - \( s'' = \{ v_1'', v_2'', \ldots, v_k'' \} \): lower leg
  - certifier: \( |s| = k, \ |s'| = k, \ |s''| = k \)
    \( v_i \) in \( s \) have edges with \( v_i' \) in \( s' \) and \( v_i'' \) in \( s'' \)
- Hardness:

Hamiltonian Path $\Rightarrow$ Centipede.

**E.g.**

$G$: $G$ has a Hamiltonian Path! (HP)

$G'$: $G'$ has a centipede of length $k = n$

Centipede of size $n$ is in $G'$

Centipede of size $n$ is not in $G$

Reduction:

- $<G>$
  - $<G'>$ see above
  - ALGO Centipede
  - accept
  - reject

Accept: $G$ has an HP

Reject: $G$ doesn't have an HP
Prove Centipede is in NP:

The problem is in NP. We let the certificate be three ordered lists of length \( k \). The first list is the main path and the other two lists form the legs. We can easily verify in polynomial time that the lists form a \textit{centipede} by checking that in the first list any two consecutive vertices have an edge between them in \( G \) and checking that a vertex from the second or third list has an edge to a vertex in the first list of the same order (position in the list).
Prove Centipede is in \textbf{NP-hard}:
Prove Centipede is in NP-hard:

Hamiltonian Path (HP): Given a graph G (either directed or undirected), is there a path that visits every vertex exactly once.
The problem is NP-Hard by reduction from HAMILTONIAN-PATH to CENTIPEDE. Given an instance of HAMILTONIAN-PATH, a graph $G$ with $n$ vertices $v_1, v_2, \cdots, v_n$, create a new graph $G'$ by adding $2n$ vertices: $u_1, u_2, \cdots, u_n$ and $x_1, x_2, \cdots, x_n$. Then, add an edge $(u_i, v_i)$ and $(x_i, v_i)$ for $1 \leq i \leq n$. The reduction is polynomial time since we only added $2n$ of vertices and edges to the graph.
A quasi-satisfying assignment (quasiSAT) for a 3CNF boolean formula $\Phi$ is an assignment of truth values to the variables such that at most one clause in $\Phi$ does not contain a true literal. Prove that it is NP-complete to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

quasiSAT is NP-complete.
A quasi-satisfying assignment (quasiSAT) for a 3CNF boolean formula $\Phi$ is an assignment of truth values to the variables such that at most one clause in $\Phi$ does not contain a true literal. Prove that it is NP-complete to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

Prove quasiSAT is in NP.
A quasi-satisfying assignment (quasiSAT) for a 3CNF boolean formula $\Phi$ is an assignment of truth values to the variables such that at most one clause in $\Phi$ does not contain a true literal. Prove that it is NP-complete to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

Prove quasiSAT is NP-hard.
Prove quasiSAT is NP-hard.

3SAT: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment.

Join a clause to $\phi$ with $\land$ such that $c$ is never satisfied.

You can join a bunch of clauses to $\phi$ such that $\phi'$ (the resulting clause) is never satisfied! How?

Define: $\phi' = \phi \land (x \lor y \lor z) \land (\neg x \lor y \lor z) \land \ldots$
Prove quasiSAT is NP-hard.

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment.
Good luck on the exam