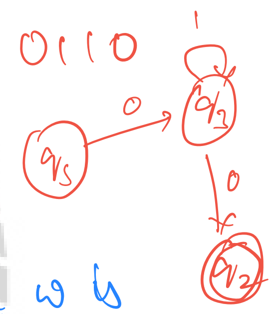
Decidable Problems from Regular Languages

Theorem 4.1: $A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts } w\}$

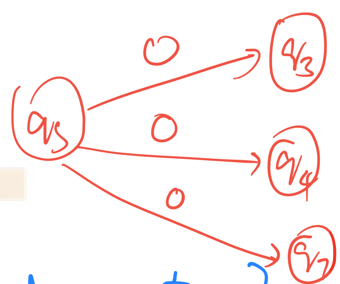
A_{DFA} is decidable.

Proof: The decider is the following TM.



M: On input $\langle B, w\rangle$, B is a DFA and w is a string.

1. Simulate B on w
2. If simulation ends in an accept state, accept. Else reject.



Theorem 4.2: A_{NFA} is decidable.

$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts } w\}$

Proof idea: First convert NFA to DFA say C using conversion process explained in Chapter 1. After this the resulting DFA can be checked if it accepts w using Theorem 4.1.

Theorem 4.3: A_{REX} is decidable.

$$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$$

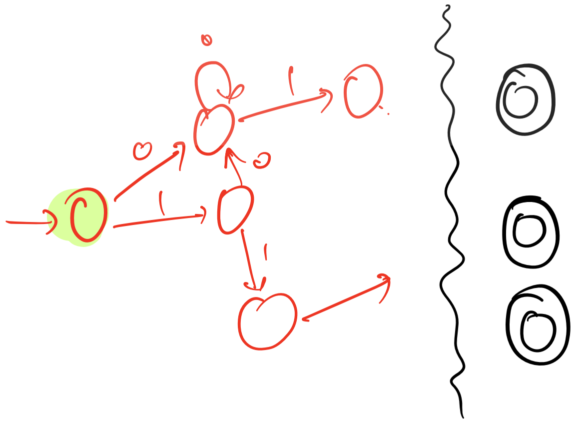
Proof: Convert REX to NFA, and then run decider for NFA.

Theorem 4.4: E_{DFA} is decidable.

$$E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA}, L(A) = \phi\}$$

How can we check each and every string? → Not clear.

Instead, we view the DFA as a graph. Is there a path from the starting state to an accepting state?



If there is a path, language is not empty.

We can use BFS/DFS to check if an accepting state is reachable from the start state. If it is reachable, reject. If no accept state is reachable, then accept.

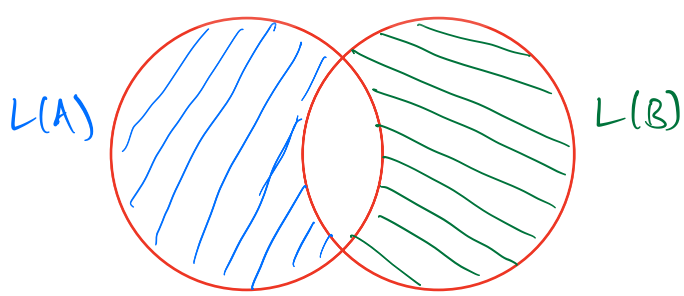
Exercise: Read theorems 4.1, 4.2, 4.3 & 4.4.

Theorem 4.5: EQ_{DFA} is decidable.

$$EQ_{DFA} = \{\langle A,B\rangle \mid A, B \text{ are DFA's}, L(A) = L(B)\}$$

Proof: Given A, B, we can use closure properties to construct a DFA C such that

$$L(C) = \underbrace{\left[\overline{L(A)} \cap L(B)\right] \cup \left[L(A) \cap \overline{L(B)}\right]}_{\text{Symmetric difference of } L(A) \text{ \& } L(B)}$$



$L(A) = L(B) \iff L(C) = \phi.$

On input A, B, which are DFA's.

1. Construct DFA C using closure properties.
2. Run the decider for E_{DFA} on C
3. If the E_{DFA} decider accepts, accept.
   Else reject.

Decidable Problems using CFL's

$A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates } w \}$

Can we try all possible derivations? It is not clear how to do that. The TM may never halt. This idea can be used to obtain a recognizer, but not a decider.

Instead, we can try using Chomsky Normal Form. All derivations used to derive a string of length n must have exactly 2n-1 steps. We can try all derivations of exactly 2n-1 steps.

Theorem 4.7: A_{CFG} is decidable.

Proof: S = On input $\langle G, w \rangle$, where G is a CFG and w is a string.

1. Convert G into Chomsky Normal Form.
2. List all possible derivations of $2n-1$ steps, where $|n| = w$. If $w = \varepsilon$, then list all derivations of 1 step.
3. Accept if w is in the list. Else reject.

Note: A more efficient way is to use CYK algorithm, which we saw in Chapter 2.

$E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG}, L(G) = \Phi \}$.

Theorem 4.8: E_{CFG} is decidable.

We cannot keep checking for each w if $w \in L(G)$. This will not be a decider. Instead, we use a strategy similar to the one used for E_{DFA}.

Tape 1 | Rules G

Tape 1: $\langle G \rangle$

Tape 2 | a b 0 1 A B E |

Tape 2: Initially list all the terminals symbols.

$A \to aba$
$B \to 00A$
$C \to DA$
$E \to AB$

- Repeat until no new variable is added.

A is added to tape 2 if A can generate a string of terminals

For all A, variable in G:
If there is a rule $A \to U_1 U_2 \dots U_k$ where all of $U_1, U_2, \dots U_k$ are in tape 2, then add A to tape 2.

- Check if start variable is in tape 2.
- If it is not in tape 2, accept.
- Else reject.

$L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}$

Theorem 4.9: Every CFL is decidable.

Read the proof. (Uses A_{CFG})

$EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ are CFG's}, L(G) = L(H) \}$

Can we use the same trick as in EQ_{DFA}? NO.

CFL's are not closed under intersection and complement. In fact, EQ_{CFG} is undecidable.