CS 301

Lecture 18 – Decidable languages

Decidable language

Recall, a language A is decidable if there is some TM M that

1. recognizes A (i.e., $L(M) = A$), and
2. halts on every input (i.e., $\forall w \in \Sigma^*$, M either accepts or rejects w)

Such a TM is called a decider

Acceptance

The acceptance problem for a type of machine[1] asks whether a given machine of the specified type accepts a string

Examples:

$A_{\mathsf{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts the string } w\}$

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

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

$A_{\mathsf{PDA}} = \{\langle B, w \rangle \mid B \text{ is a PDA that accepts the string } w\}$

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

$A_{\mathsf{TM}} = \{\langle B, w \rangle \mid B \text{ is a TM that accepts the string } w\}$

---

[1] For models of computation like regular expressions and grammars, the acceptance problems asks if it *generates* the string, rather than accepts it

Emptiness and equivalence

The emptiness problem asks whether the language of the machine is empty

Examples:

$$E_{\mathsf{DFA}} = \{\langle B \rangle \mid B \text{ is a DFA and } L(B) = \varnothing\}$$
$$E_{\mathsf{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \varnothing\}$$
$$E_{\mathsf{TM}} = \{\langle B \rangle \mid B \text{ is a TM and } L(B) = \varnothing\}$$

Emptiness and equivalence

The emptiness problem asks whether the language of the machine is empty

Examples:

$$E_{\mathsf{DFA}} = \{\langle B \rangle \mid B \text{ is a DFA and } L(B) = \varnothing\}$$
$$E_{\mathsf{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \varnothing\}$$
$$E_{\mathsf{TM}} = \{\langle B \rangle \mid B \text{ is a TM and } L(B) = \varnothing\}$$

The equivalence problem asks whether two machines have the same language

Examples:

$$EQ_{\mathsf{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$
$$EQ_{\mathsf{CFG}} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$
$$EQ_{\mathsf{TM}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are TMs and } L(A) = L(B)\}$$

A_{DFA} is decidable

Theorem
The language

$$A_{DFA} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts the string } w\}$$

is decidable.

A_{DFA} is decidable

Theorem
The language

$$A_{DFA} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts the string } w\}$$

is decidable.

Proof.
We want to build a TM M that decides A_{DFA}:

M = "On input $\langle B, w \rangle$,

1. Run (or simulate) B on w
2. If B ends in an accept state, *accept*; otherwise *reject*"

Since the simulation always ends after $|w|$ steps, M is a decider.
If $w \in L(B)$, then M will accept. If $w \notin L(B)$, then M will reject. $\qquad\square$

A_{DFA} is decidable

Theorem
The language

$$A_{DFA} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts the string } w\}$$

is decidable.

Proof.
We want to build a TM M that decides A_{DFA}:

M = "On input $\langle B, w \rangle$,

① Run (or simulate) B on w

② If B ends in an accept state, *accept*; otherwise *reject*"

Since the simulation always ends after $|w|$ steps, M is a decider.

If $w \in L(B)$, then M will accept. If $w \notin L(B)$, then M will reject. □

Note that the form of the input to M matches *exactly* the form of the strings in the language we want to decide

Using a TM as a subroutine in another TM

Given a TM R, we can construct another TM M that uses R

The language $A = \{\langle B, w \rangle \mid B$ is a DFA, $w \in L(B)$, and $w^{\mathcal{R}} \in L(B)\}$ is decidable

Let R be the decider for A_{DFA}.
$M = $ "On input $\langle B, w \rangle$,

1. Run R on w and if R rejects, *reject*
2. Run R on $w^{\mathcal{R}}$ and if R accepts, *accept*; otherwise *reject*"

How does this work?

Using a TM as a subroutine in another TM

Given a TM R, we can construct another TM M that uses R

The language $A = \{\langle B, w\rangle \mid B$ is a DFA, $w \in L(B)$, and $w^{\mathcal{R}} \in L(B)\}$ is decidable

Let R be the decider for A_{DFA}.
M = "On input $\langle B, w\rangle$,

1. Run R on w and if R rejects, *reject*
2. Run R on $w^{\mathcal{R}}$ and if R accepts, *accept*; otherwise *reject*"

How does this work?
The details aren't important, but here's one way we could incorporate R's implementation into M's:

1. Construct M' as a 2-TM that uses its second tape for the implementation of R

Using a TM as a subroutine in another TM

Given a TM R, we can construct another TM M that uses R

The language $A = \{\langle B, w \rangle \mid B$ is a DFA, $w \in L(B)$, and $w^{\mathcal{R}} \in L(B)\}$ is decidable

Let R be the decider for A_{DFA}.
M = "On input $\langle B, w \rangle$,

1. Run R on w and if R rejects, *reject*
2. Run R on $w^{\mathcal{R}}$ and if R accepts, *accept*; otherwise *reject*"

How does this work?
The details aren't important, but here's one way we could incorporate R's implementation into M's:

1. Construct M' as a 2-TM that uses its second tape for the implementation of R
2. So M' first copies w to the second tape and then simulates R on it. If R doesn't reject, it copies $w^{\mathcal{R}}$ to the second tape and simulates R a second time

Using a TM as a subroutine in another TM

Given a TM R, we can construct another TM M that uses R

The language $A = \{\langle B, w\rangle \mid B$ is a DFA, $w \in L(B)$, and $w^{\mathcal{R}} \in L(B)\}$ is decidable

Let R be the decider for A_{DFA}.
M = "On input $\langle B, w\rangle$,

1. Run R on w and if R rejects, *reject*
2. Run R on $w^{\mathcal{R}}$ and if R accepts, *accept*; otherwise *reject*"

How does this work?
The details aren't important, but here's one way we could incorporate R's implementation into M's:

1. Construct M' as a 2-TM that uses its second tape for the implementation of R
2. So M' first copies w to the second tape and then simulates R on it. If R doesn't reject, it copies $w^{\mathcal{R}}$ to the second tape and simulates R a second time
3. Construct the basic TM M by the procedure that lets us simulate a k-TM on a basic TM

UIC

TM subroutines

We're going to be doing this a lot so there are some important things to watch out for

- If R isn't a decider, then the simulation of R on some input may not halt; if it doesn't the machine we're building won't either!
- We can't say, "Run R on some input and if it loops, do something" because we don't know how to tell if a TM is looping

A_{NFA} is decidable

Theorem
The language

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

is decidable.

A_{NFA} is decidable

Theorem
The language

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

is decidable.

Proof.
We want to build a TM M that decides A_{NFA}. Let R be the TM that decides A_{DFA} and build
$M =$ "On input $\langle B, w \rangle$,

A_{NFA} is decidable

Theorem
The language

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

is decidable.

Proof.
We want to build a TM M that decides A_{NFA}. Let R be the TM that decides A_{DFA} and build

M = "On input $\langle B, w \rangle$,

1. Convert the NFA B to an equivalent DFA C using the power set construction
2. Run R on $\langle C, w \rangle$. If R accepts, *accept*; otherwise *reject*"

A_{NFA} is decidable

Theorem
The language

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

is decidable.

Proof.
We want to build a TM M that decides A_{NFA}. Let R be the TM that decides A_{DFA} and build

M = "On input $\langle B, w \rangle$,

1. Convert the NFA B to an equivalent DFA C using the power set construction

2. Run R on $\langle C, w \rangle$. If R accepts, *accept*; otherwise *reject*"

Since R is a decider, the simulation in step 2 will always halt so M is a decider.

If $w \in L(B)$, then $w \in L(C)$ so R will accept $\langle C, w \rangle$ and thus M accepts. If $w \notin L(B)$, then $w \notin L(C)$ so both R and M will reject. $\qquad\square$

A_{REX} is decidable

Theorem
The language

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

is decidable.

A_{REX} is decidable

Theorem
The language

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

is decidable.

Proof.
We want to build a TM M that decides A_{REX}. Let R be the TM that decides A_{NFA} and build

M = "On input $\langle R, w \rangle$,

A_{REX} is decidable

Theorem
The language

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

is decidable.

Proof.
We want to build a TM M that decides A_{REX}. Let R be the TM that decides A_{NFA} and build

M = "On input $\langle R, w \rangle$,

1. Convert the regular expression R to an equivalent NFA N using the standard construction
2. Run R on $\langle N, w \rangle$. If R accepts, *accept*; otherwise *reject*"

A_{REX} is decidable

Theorem
The language

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

is decidable.

Proof.
We want to build a TM M that decides A_{REX}. Let R be the TM that decides A_{NFA} and build

M = "On input $\langle R, w \rangle$,

1. Convert the regular expression R to an equivalent NFA N using the standard construction
2. Run R on $\langle N, w \rangle$. If R accepts, *accept*; otherwise *reject*"

Since R is a decider, the simulation in step 2 will always halt so M is a decider.

If $w \in L(R)$, then $w \in L(N)$ so R will accept $\langle N, w \rangle$ and thus M accepts. If $w \notin L(R)$, then $w \notin L(N)$ so both R and M will reject. □

DFAs, NFAs, and regular expressions are equivalent

For the purposes of constructing TMs, it doesn't matter if the input is a DFA, NFA, or regular expression

The TM can always convert from one to another using the procedures we've been learning about all semester

Emptiness problem

Consider the DFAs

M_1:
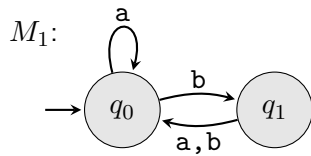
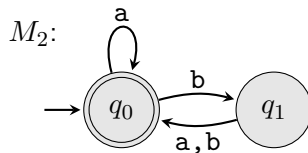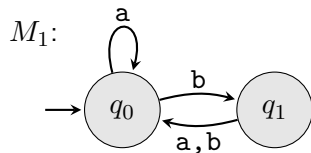Is $L(M_1) = \varnothing$?

Emptiness problem

Consider the DFAs

M_1:

Is $L(M_1) = \varnothing$? Yes

Emptiness problem

Consider the DFAs

M_1: with states q_0 (start) and q_1, transition a loop at q_0, b from q_0 to q_1, and a,b from q_1 to q_0.

M_2: with states q_0 (start, accepting) and q_1, transition a loop at q_0, b from q_0 to q_1, and a,b from q_1 to q_0.
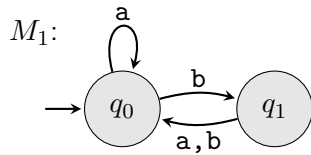
Is $L(M_1) = \varnothing$? Yes

Is $L(M_2) = \varnothing$?

Emptiness problem
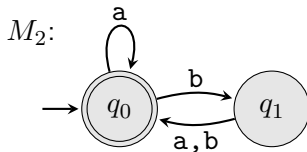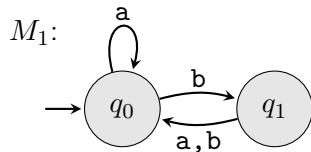
Consider the DFAs

Is $L(M_1) = \varnothing$? Yes

Is $L(M_2) = \varnothing$? No

Emptiness problem

Consider the DFAs

M_1:

M_2:

M_3:
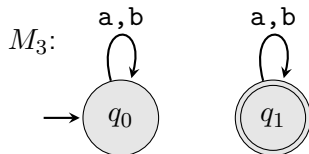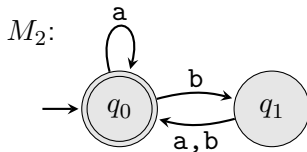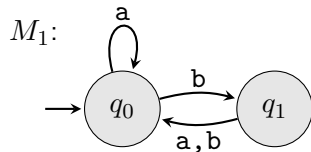
Is $L(M_1) = \varnothing$? Yes

Is $L(M_2) = \varnothing$? No

Is $L(M_3) = \varnothing$?

Emptiness problem

Consider the DFAs

M_1:
M_2:
M_3:
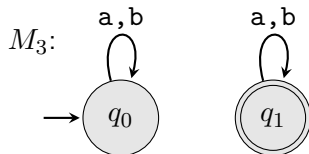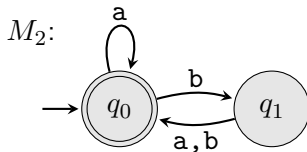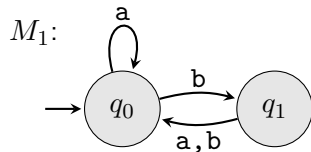
Is $L(M_1) = \varnothing$? Yes

Is $L(M_2) = \varnothing$? No

Is $L(M_3) = \varnothing$? Yes

Emptiness problem
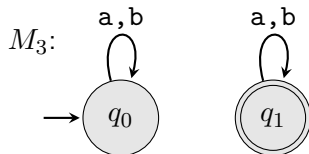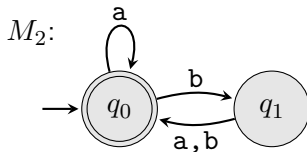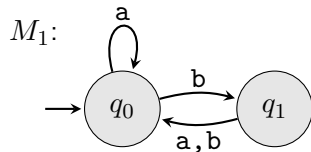
Consider the DFAs

Is $L(M_1) = \varnothing$? Yes

Is $L(M_2) = \varnothing$? No

Is $L(M_3) = \varnothing$? Yes

What are the necessary and sufficient conditions for a DFA to recognize the empty language?

Emptiness problem
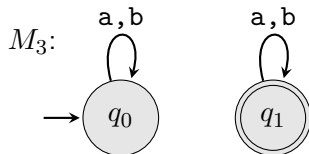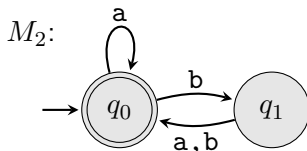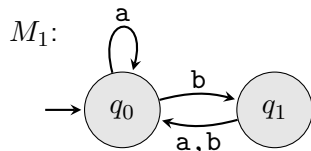
Consider the DFAs

Is $L(M_1) = \varnothing$? Yes

Is $L(M_2) = \varnothing$? No

Is $L(M_3) = \varnothing$? Yes

What are the necessary and sufficient conditions for a DFA to recognize the empty language? There must be no path from the initial state to an accepting state

Algorithm to check if the language of a DFA is empty

We can use a graph algorithm to determine which states are reachable from the initial state

Start by marking the initial state in some way, then iteratively mark all of the neighbors of all marked states until no new states are marked

If an accepting state is marked, the language is not empty, otherwise it is empty

Algorithm to check if the language of a DFA is empty

We can use a graph algorithm to determine which states are reachable from the initial state

Start by marking the initial state in some way, then iteratively mark all of the neighbors of all marked states until no new states are marked

If an accepting state is marked, the language is not empty, otherwise it is empty

Algorithm to check if the language of a DFA is empty

We can use a graph algorithm to determine which states are reachable from the initial state

Start by marking the initial state in some way, then iteratively mark all of the neighbors of all marked states until no new states are marked

If an accepting state is marked, the language is not empty, otherwise it is empty
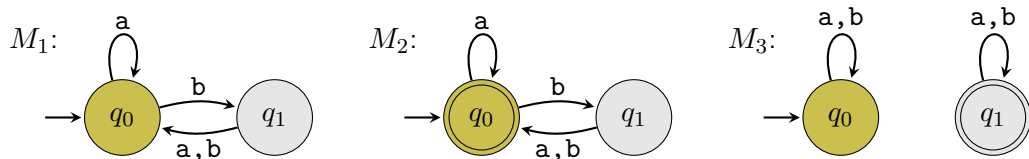
E_{DFA} is decidable

Theorem
The language $E_{DFA} = \{\langle B \rangle \mid B$ is a DFA and $L(B) = \varnothing\}$ is decidable

E_{DFA} is decidable

Theorem
The language $E_{DFA} = \{\langle B \rangle \mid B \text{ is a DFA and } L(B) = \varnothing\}$ is decidable

Proof.
Let's build a TM M to decide E_{DFA}:

M = "On input $\langle B \rangle$,

1. Mark the initial state of B
2. Repeat until no new states are marked
3.   Mark any state that has an incoming transition from a marked state
4. If any accept state is marked, *reject*; otherwise *accept*"

E_{DFA} is decidable

Theorem
The language $E_{DFA} = \{\langle B \rangle \mid B$ is a DFA and $L(B) = \varnothing\}$ is decidable

Proof.
Let's build a TM M to decide E_{DFA}:

M = "On input $\langle B \rangle$,

1 Mark the initial state of B

2 Repeat until no new states are marked

3   Mark any state that has an incoming transition from a marked state

4 If any accept state is marked, *reject*; otherwise *accept*"

If $L(B) \neq \varnothing$, then there must be a path from the initial state to an accept state so M will reject. If $L(B) = \varnothing$, then no such path exists and M will accept

M is a decider because step 2 can happen at most $|Q| - 1$ times before no new states are marked $\qquad\square$

Emptiness problems for NFAs and regular expression

How would we show that the analogous emptiness problems for NFAs and regular expressions are decidable?

Emptiness problems for NFAs and regular expression

How would we show that the analogous emptiness problems for NFAs and regular expressions are decidable?

Let R be a decider for E_{DFA} and then build a TM that first converts the NFA or regular expression to a DFA and then runs R on it

Equivalence of DFAs

The equivalence problem for DFAs asks if two DFAs recognize the same language

Using a decider for E_{DFA}, we could check if both languages are empty, but that doesn't really help

We also have a decider to check if a string is in a language

An idea that doesn't work

Let's try to build a decider for
$EQ_{\mathsf{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

Let R be a decider for A_{DFA} and build
$M =$ "On input $\langle A, B \rangle$,

1. Repeat for each string $w \in \Sigma^*$
2.   Run R on $\langle A, w \rangle$
3.   Run R on $\langle B, w \rangle$
4.   If one of A or B accepts w and the other rejects, *reject*
5. Otherwise, A and B agree on all strings so *accept*"

If there is a string w that is accepted by A but rejected by B or vice versa, M will eventually reject in step 4. Otherwise $L(A) = L(B)$ so M will accept in step 5

An idea that doesn't work

Let's try to build a decider for
$EQ_{\mathsf{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

Let R be a decider for A_{DFA} and build
$M =$ "On input $\langle A, B \rangle$,

  1 Repeat for each string $w \in \Sigma^*$

  2   Run R on $\langle A, w \rangle$

  3   Run R on $\langle B, w \rangle$

  4   If one of A or B accepts w and the other rejects, *reject*

  5 Otherwise, A and B agree on all strings so *accept*"

If there is a string w that is accepted by A but rejected by B or vice versa, M will
eventually reject in step 4. Otherwise $L(A) = L(B)$ so M will accept in step 5

Why doesn't this idea work?

An idea that doesn't work

Let's try to build a decider for
$EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

Let R be a decider for A_{DFA} and build
$M =$ "On input $\langle A, B\rangle$,

1. Repeat for each string $w \in \Sigma^*$
2.   Run R on $\langle A, w\rangle$
3.   Run R on $\langle B, w\rangle$
4.   If one of A or B accepts w and the other rejects, *reject*
5. Otherwise, A and B agree on all strings so *accept*"

If there is a string w that is accepted by A but rejected by B or vice versa, M will eventually reject in step 4. Otherwise $L(A) = L(B)$ so M will accept in step 5

Why doesn't this idea work?
There are infinitely many strings in Σ^* so the loop on line 1 will never end if
$L(A) = L(B)$

A better idea

Given two languages X and Y, consider the language $D = (X \setminus Y) \cup (Y \setminus X)$

Three cases:

1 If there is a string x which is in X but not in Y, then $X \setminus Y \neq \varnothing$ so $D \neq \varnothing$

A better idea

Given two languages X and Y, consider the language $D = (X \setminus Y) \cup (Y \setminus X)$

Three cases:

1. If there is a string x which is in X but not in Y, then $X \setminus Y \neq \varnothing$ so $D \neq \varnothing$
2. If there is a string w which is in Y but not in X, then $Y \setminus X \neq \varnothing$ so $D \neq \varnothing$

A better idea

Given two languages X and Y, consider the language $D = (X \smallsetminus Y) \cup (Y \smallsetminus X)$

Three cases:

1. If there is a string x which is in X but not in Y, then $X \smallsetminus Y \neq \varnothing$ so $D \neq \varnothing$
2. If there is a string w which is in Y but not in X, then $Y \smallsetminus X \neq \varnothing$ so $D \neq \varnothing$
3. If $X = Y$, then $X \smallsetminus Y$ and $Y \smallsetminus X$ are both $\varnothing$ so $D = \varnothing$

A better idea

Given two languages X and Y, consider the language $D = (X \smallsetminus Y) \cup (Y \smallsetminus X)$

Three cases:

1. If there is a string x which is in X but not in Y, then $X \smallsetminus Y \neq \varnothing$ so $D \neq \varnothing$
2. If there is a string w which is in Y but not in X, then $Y \smallsetminus X \neq \varnothing$ so $D \neq \varnothing$
3. If $X = Y$, then $X \smallsetminus Y$ and $Y \smallsetminus X$ are both $\varnothing$ so $D = \varnothing$

If X and Y are regular languages, then D is regular because $X \smallsetminus Y = X \cap \overline{Y}$ and regular languages are closed under union, intersection, and complement

EQ_{DFA} is decidable

Theorem
The language $EQ_{DFA} = \{\langle A, B \rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$ is decidable

EQ_{DFA} is decidable

Theorem
The language $EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is decidable

Proof.
Let R be a decider for E_{DFA} and build
M = "On input $\langle A, B \rangle$,

1. Construct DFA C where $L(C) = (L(A) \smallsetminus L(B)) \cup (L(B) \smallsetminus L(A))$
2. Run R on $\langle C \rangle$
3. If R accepts, *accept*; otherwise *reject*"

EQ_{DFA} is decidable

Theorem
The language $EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is decidable

Proof.
Let R be a decider for E_{DFA} and build
$M = $ "On input $\langle A, B\rangle$,

1. Construct DFA C where $L(C) = (L(A) \smallsetminus L(B)) \cup (L(B) \smallsetminus L(A))$

2. Run R on $\langle C\rangle$

3. If R accepts, *accept*; otherwise *reject*"

The construction in step 1 can be carried out using the constructions we've seen earlier in the semester
M is a decider because R is a decider
$L(C) = \varnothing$ iff $L(A) = L(B)$ so M accepts iff $L(A) = L(B)$ □

Recapitulation

The acceptance, emptiness, and equivalence problems for DFAs/NFAs/regular expressions are all decidable

For some of these, we used deciders for other languages as subroutines

Analogous problems for context-free languages

We can consider the acceptance, emptiness, and equivalence problems for context-free languages

Like the situation with DFAs, NFAs, and regular expressions, we can easily convert between CFGs and PDAs, so it suffices to consider CFGs

A_{CFG} is decidable

Theorem
The language $A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG and } G \text{ generates } w\}$ is decidable.

A_{CFG} is decidable

Theorem
The language $A_{CFG} = \{\langle G, w \rangle \mid G$ is a CFG and G generates $w\}$ is decidable.

This is trickier than DFAs, but we can use Chomsky normal form (CNF)

If $w \in L(G)$ and G is in CNF, how many steps does it take for G to derive w?

A_{CFG} is decidable

Theorem
The language A_{CFG} = $\{\langle G, w \rangle \mid G$ is a CFG and G generates $w\}$ is decidable.

This is trickier than DFAs, but we can use Chomsky normal form (CNF)

If $w \in L(G)$ and G is in CNF, how many steps does it take for G to derive w?
If $w = \varepsilon$, it takes 1 step
If $w \neq \varepsilon$, it takes $2|w| - 1$ steps

Let's use this fact along with our procedure to convert a CFG to CNF

Proof

Proof.

Build M = "On input $\langle G, w \rangle$,

1. Convert G to CNF

2. If $w = \varepsilon$, then if $S \to \varepsilon$ is a rule in G, *accept*; otherwise *reject*

3. Otherwise, list all derivations of length $2|w| - 1$

4. If any of these derivations is w, *accept*; otherwise *reject*"

This is a decider because there are finitely many derivations of length $2|w| - 1$.
If $w \in L(G)$, then M will accept, otherwise M will reject. $\qquad\square$

Emptiness problem for CFGs

Given a CFG, we want to figure out if the CFG can derive *any* string of terminals

Let's figure out which variables in G can derive strings of terminals and then we can look the start variable to see if it's one that can

E_{CFG} is decidable

Theorem

The language $E_{CFG} = \{\langle G\rangle \mid G$ is a CFG and $L(G) = \varnothing\}$ is decidable.

E_{CFG} is decidable

Theorem
The language $E_{CFG} = \{\langle G \rangle \mid G$ is a CFG and $L(G) = \varnothing\}$ is decidable.

Proof.
$M =$ "On input $\langle G \rangle$,

1. Mark all terminal symbols in the rules of G
2. Repeat until no new variables get marked
3. 　　Mark any variable A in G where $A \to u_1 u_2 \cdots u_k$ is a rule and each u_i is marked
4. If the start variable is marked, *reject*; otherwise *accept*"

E_{CFG} is decidable

Theorem
The language $E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \varnothing\}$ is decidable.

Proof.
M = "On input $\langle G \rangle$,

1. Mark all terminal symbols in the rules of G
2. Repeat until no new variables get marked
3. Mark any variable A in G where $A \rightarrow u_1 u_2 \cdots u_k$ is a rule and each u_i is marked
4. If the start variable is marked, *reject*; otherwise *accept*"

M is a decider because the loop can happen only finitely many times

If the start variable is marked, then it can derive a string of terminals so $L(G) \neq \varnothing$ and M rejects

If the start variable is not marked, then it cannot derive any string of terminals so $L(G) = \varnothing$ and M accepts $\qquad\square$

What about equivalence of CFGs?

Theorem
The language $A = \{\langle G, H \rangle \mid G$ and H are CFGs and $L(G) \neq L(H)\}$ is Turing-recognizable (RE)

What about equivalence of CFGs?

Theorem
The language $A = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) \neq L(H)\}$ is Turing-recognizable (RE)

Proof.
Let R be a decider for A_{CFG} and build $M = $ "On input $\langle G, H \rangle$,

1. For each $w \in \Sigma^*$,
2.   Run R on $\langle G, w \rangle$
3.   Run R on $\langle H, w \rangle$
4.   If w was generated by only one of G or H, *accept*

If $L(G) \neq L(H)$, then M will accept at some iteration of the loop
If $L(G) = L(H)$, then M will run forever $\qquad\qquad\square$

Co-Turing-recognizable (coRE)

The language A is clearly related to the complement of EQ_{CFG}

$\overline{EQ_{\mathsf{CFG}}} = A \cup \{w \mid w$ is not a valid representation of $\langle G, H \rangle$ for CFGs G and $H\}$

Co-Turing-recognizable (coRE)

The language A is clearly related to the complement of EQ_{CFG}

$$\overline{EQ_{\mathsf{CFG}}} = A \cup \{w \mid w \text{ is not a valid representation of } \langle G, H \rangle \text{ for CFGs } G \text{ and } H\}$$

We say that a language is co-Turing-recognizable (coRE) if its complement is Turing-recognizable (RE)

The language EQ_{CFG} is coRE

Co-Turing-recognizable (coRE)

The language A is clearly related to the complement of EQ_{CFG}

$$\overline{EQ_{\mathsf{CFG}}} = A \cup \{w \mid w \text{ is not a valid representation of } \langle G, H \rangle \text{ for CFGs } G \text{ and } H\}$$

We say that a language is co-Turing-recognizable (coRE) if its complement is Turing-recognizable (RE)

The language EQ_{CFG} is coRE

We'll see later that EQ_{CFG} is not RE and thus not decidable (think about what that means for grading homework problems about CFGs)