INF2080

Church Turing Thesis and Decidability

Daniel Lupp

Universitetet i Oslo

28th February 2017

Department of Informatics

University of Oslo

Short Recap

- We have looked at Turing machines as a computational model

Short Recap

- We have looked at Turing machines as a computational model
- a finite state machine with an infinite tape, upon which a head can move, read, and write

Short Recap

- We have looked at Turing machines as a computational model
- a finite state machine with an infinite tape, upon which a head can move, read, and write
- have looked at Turing machine variants, seen that they are equivalent:

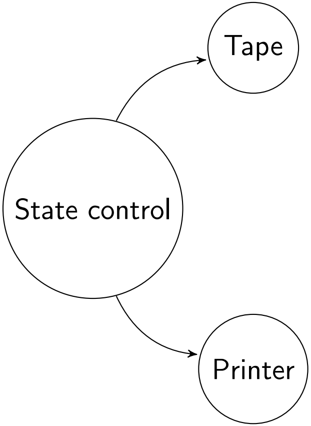
Short Recap

- We have looked at Turing machines as a computational model
- a finite state machine with an infinite tape, upon which a head can move, read, and write
- have looked at Turing machine variants, seen that they are equivalent:
- the LRS Turing machine (the head can move left, right, or stay put)
- the multitape Turing machine (multiple tapes, multiple heads)
- the nondeterministic Turing machine

This week

- one more TM variant: the enumerator
- Church Turing thesis
- Decidability

Enumerators



- An enumerator is a slightly altered turing machine:
- It has a working tape and an attached printer
- initializes with an empty working tape, taking no input
- throughout its computation, it can output strings using the printer
- If the enumerator does not halt, it can potentially output infinitely many strings

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Assume we have an enumerator E. We want to construct a Turing machine A that accepts all the words that E enumerates.

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Assume we have an enumerator E. We want to construct a Turing machine A that accepts all the words that E enumerates.

$A =$ On input w

1. Run E. Every time E prints a string, compare to w.
2. If w appears in the output of E, *accept*.

Enumerators

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Now, let A be a Turing machine. We want to construct an enumerator that enumerates $L(A)$.

Enumerators

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Now, let A be a Turing machine. We want to construct an enumerator that enumerates $L(A)$.
- Le Σ be the alphabet of $L(A)$. Then we can order all strings in Σ^* (first list all strings of length 1, then of length 2, etc).

Enumerators

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Now, let A be a Turing machine. We want to construct an enumerator that enumerates $L(A)$.
- Le Σ be the alphabet of $L(A)$. Then we can order all strings in Σ^* (first list all strings of length 1, then of length 2, etc). Label them $s_1, s_2, s_3, \ldots$. Then we can construct an enumerator:

$E =$ Ingore input

1. Repeat for $i = 1, 2, 3, \ldots$.
2. Run A on $s_1, \ldots, s_i$ for i steps.
3. If any computation accepts, print corresponding s_j.

Church Turing Thesis

- So, all variants of Turing machines we've seen are equivalent in expressivity.
- This is no coincidence: all can perform finite work in a single step, all have unlimited access to infinite memory.
- In fact, Turing machines capture *all* such computational models

Church-Turing Thesis

- the notion of *algorithm* is not new
- yet a formal description of what an algorithm is, or what is solvable using algorithms, did not appear until the 20th century.
- Many mathematicians assumed that one needed only to *find* the right "method", did not even consider something might be unsolvable.

Church-Turing Thesis

- Church and Turing independently formalized the notion of algorithm
- Previous, intuitive notion: a method according to which after a finite number of operations an answer is given (paraphrased, many formulations)
- Formal: an algorithm is a decidable Turing machine (deciders)
- Church Turing thesis: each intuitive definition of algorithms can be described by decidable Turing machines

Decidability

Definition

A language L is *decidable* if a Turing machine M_L exists that *decides* it, that is, if M_L either accepts or rejects any input w.

Decidability

Definition

A language L is *decidable* if a Turing machine M_L exists that *decides* it, that is, if M_L either accepts or rejects any input w.

- This week we will discuss the decidability of various problems related to the classes of languages we have seen so far: regular, context-free, and Turing-recognizable.

Decidability

Definition

A language L is *decidable* if a Turing machine M_L exists that *decides* it, that is, if M_L either accepts or rejects any input w.

- This week we will discuss the decidability of various problems related to the classes of languages we have seen so far: regular, context-free, and Turing-recognizable.
- **Acceptance problem:** Given a DFA/NFA/CFG/PDA/TM/... and an input w, does the machine/grammar accept w?

Decidability

Definition

A language L is *decidable* if a Turing machine M_L exists that *decides* it, that is, if M_L either accepts or rejects any input w.

- This week we will discuss the decidability of various problems related to the classes of languages we have seen so far: regular, context-free, and Turing-recognizable.
- **Acceptance problem:** Given a DFA/NFA/CFG/PDA/TM/... and an input w, does the machine/grammar accept w?
- **Emptiness problem:** Given a DFA/NFA/CFG/PDA/TM/..., is its generated language empty?

Decidability

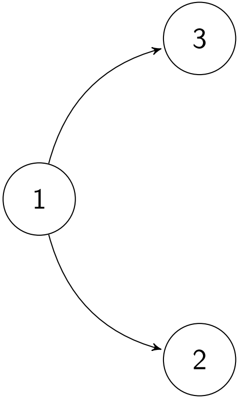
Definition

A language L is *decidable* if a Turing machine M_L exists that *decides* it, that is, if M_L either accepts or rejects any input w.

- This week we will discuss the decidability of various problems related to the classes of languages we have seen so far: regular, context-free, and Turing-recognizable.
- **Acceptance problem:** Given a DFA/NFA/CFG/PDA/TM/... and an input w, does the machine/grammar accept w?
- **Emptiness problem:** Given a DFA/NFA/CFG/PDA/TM/..., is its generated language empty?
- **Equality problem:** Given two DFA/NFA/CFG/PDA/TM/..., are the two generated languages equal?

Notation

For an object O (graph, automaton, Turing machine, etc.), let $\langle O \rangle$ represent its string representation. For example:



can be represented as the string $\{1, 2, 3, (1, 2), (1, 3)\}$

Acceptance problem - DFA

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

Acceptance problem - DFA

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

- Acceptance problem "Given B and w, does B accept w?" $\Leftrightarrow$ "$\langle B, w\rangle \in A_{DFA}$"?

Acceptance problem - DFA

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

- Acceptance problem "Given B and w, does B accept w?" $\Leftrightarrow$ "$\langle B, w\rangle \in A_{DFA}$"?

Theorem

A_{DFA} is a decidable language.

Proof idea: We create a Turing machine that simulates B on w:

Acceptance problem - DFA

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

- Acceptance problem "Given B and w, does B accept w?" $\Leftrightarrow$ "$\langle B, w\rangle \in A_{DFA}$"?

Theorem

A_{DFA} is a decidable language.

Proof idea: We create a Turing machine that simulates B on w:

M_{DFA} = On input $\langle B, w\rangle$

1. Simulate B on w.
2. If the simulation ends in an accept state, *accept*,
   if it ends in a nonaccepting state, *reject*.

Acceptance problem - DFA

Corollary

The class of regular languages is decidable.

Proof:

- Given a regular language L, we can encode its DFA B into a decider for L:

$$M_L = \text{On input } w$$

1. Simulate M_{DFA} on $\langle B, w \rangle$.
2. If M_{DFA} accepts, *accept*,
   if it rejects, *reject*.

Acceptance problem - NFA/RE

- What about NFAs and REs?

Acceptance problem - NFA/RE

- What about NFAs and REs?
- We have seen that they have equivalent expressive power to DFAs

Acceptance problem - NFA/RE

- What about NFAs and REs?
- We have seen that they have equivalent expressive power to DFAs
- So are the languages A_{NFA} and A_{RE} decidable?

Acceptance problem - NFA/RE

- What about NFAs and REs?
- We have seen that they have equivalent expressive power to DFAs
- So are the languages A_{NFA} and A_{RE} decidable?
- We can use the known procedures to convert NFA→DFA and RE→NFA!

Acceptance problem - NFA/RE

- What about NFAs and REs?
- We have seen that they have equivalent expressive power to DFAs
- So are the languages A_{NFA} and A_{RE} decidable?
- We can use the known procedures to convert NFA→DFA and RE→NFA!

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

Theorem

The language A_{NFA} is decidable.

Acceptance problem - NFA/RE

- What about NFAs and REs?
- We have seen that they have equivalent expressive power to DFAs
- So are the languages A_{NFA} and A_{RE} decidable?
- We can use the known procedures to convert NFA→DFA and RE→NFA!

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

Theorem

The language A_{NFA} is decidable.

Proof:

M_{NFA} = On input $\langle B, w\rangle$

1. Convert B to an equivalent DFA C.
2. Simulate M_{DFA} on input $\langle B, w\rangle$

   if it accepts, *accept*; if it rejects, *reject*.

Acceptance problem - NFA/RE

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

Theorem

The language A_{RE} is decidable.

Acceptance problem - NFA/RE

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

Theorem

The language A_{RE} is decidable.

Proof: Similar to before, however now we reduce to NFA case:

M_{RE} = On input $\langle R, w\rangle$

1. Convert R to an equivalent NFA B.
2. Simulate M_{NFA} on input $\langle B, w\rangle$

   if it accepts, *accept*; if it rejects, *reject*.

Acceptance problem - Regular languages

- So we see that it is does not matter which computational model we use to represent the regular language; this has no effect on decidabillity

Acceptance problem - Regular languages

- So we see that it is does not matter which computational model we use to represent the regular language; this has no effect on decidabillity
- Recall the Church-Turing thesis: intuitive notion of algorithm/procedure $\Leftrightarrow$ Turing machine algorithm

Acceptance problem - Regular languages

- So we see that it is does not matter which computational model we use to represent the regular language; this has no effect on decidabillity
- Recall the Church-Turing thesis: intuitive notion of algorithm/procedure $\Leftrightarrow$ Turing machine algorithm
- Our "procedures" of converting NFA$\rightarrow$DFA, RE$\rightarrow$NFA, CFG$\leftrightarrow$PDA can be formally described using a decidable TM!

Emptiness problem - Regular languages

Next "decision problem:" Given a DFA A, is the language generated by A empty?

Emptiness problem - Regular languages

Next “decision problem:” Given a DFA A, is the language generated by A empty?
$\Leftrightarrow \langle A \rangle \in E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$?

Emptiness problem - Regular languages

Next "decision problem:" Given a DFA A, is the language generated by A empty?
$\Leftrightarrow \langle A \rangle \in E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$?

- When does a DFA accept a string w?

Emptiness problem - Regular languages

Next "decision problem:" Given a DFA A, is the language generated by A empty?
$\Leftrightarrow \langle A \rangle \in E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$?

- When does a DFA accept a string w? When it reaches an accept state!

Emptiness problem - Regular languages

Next "decision problem:" Given a DFA A, is the language generated by A empty?
$\Leftrightarrow \langle A \rangle \in E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$?

- When does a DFA accept a string w? When it reaches an accept state!
- So all the TM has to do is check whether an accept state is reachable from the start state.

Emptiness problem - Regular languages

Next "decision problem:" Given a DFA A, is the language generated by A empty?
$\Leftrightarrow \langle A\rangle \in E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$?

- When does a DFA accept a string w? When it reaches an accept state!
- So all the TM has to do is check whether an accept state is reachable from the start state.
- We use the "marking" technique we have previously seen to keep track of the DFA's states that have been reached.

Emptiness problem - Regular languages

Theorem

The language E_{DFA} is decidable.

Proof:

N_{DFA} = On input $\langle A \rangle$

1. Mark the start state of A.
2. Repeat 3. until no new states are marked:
3. Mark any state with an incoming transition from a marked state.
4. If no accept state is reached, *accept*; else, *reject*.

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?
$\Leftrightarrow \langle A, B\rangle \in EQ_{DFA} = \{\langle A, B\rangle \mid \ A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$?

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?
$\Leftrightarrow \langle A, B \rangle \in EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$?

- Now we use the set theoretic notion of *symmetric difference* to help us!

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?
$\Leftrightarrow \langle A, B\rangle \in EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$?

- Now we use the set theoretic notion of *symmetric difference* to help us!
- The symmetric difference of two languages $L(A)$ and $L(B)$ is defined as

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?
$\Leftrightarrow \langle A, B\rangle \in EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$?

- Now we use the set theoretic notion of *symmetric difference* to help us!
- The symmetric difference of two languages $L(A)$ and $L(B)$ is defined as

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

- intuitively: the symmetric difference contains everything that is in precisely one of the two languages, but not both.

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?
$\Leftrightarrow \langle A, B\rangle \in EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$?

- Now we use the set theoretic notion of *symmetric difference* to help us!
- The symmetric difference of two languages $L(A)$ and $L(B)$ is defined as

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

- intuitively: the symmetric difference contains everything that is in precisely one of the two languages, but not both.
- Two sets are equal if and only if their symmetric difference is empty!

Equality problem - Regular languages

What if we have two regular languages, accepted by DFAs A and B, and want to check whether they are equal?
$\Leftrightarrow \langle A, B\rangle \in EQ_{DFA} = \{\langle A, B\rangle \mid$ A and B are DFAs and $L(A) = L(B)\}$?

- Now we use the set theoretic notion of *symmetric difference* to help us!
- The symmetric difference of two languages $L(A)$ and $L(B)$ is defined as

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

- intuitively: the symmetric difference contains everything that is in precisely one of the two languages, but not both.
- Two sets are equal if and only if their symmetric difference is empty! $\rightarrow$ emptiness problem!

Equality problem - Regular languages

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Recall closure properties of regular languages:

- closed under union, intersection, and complement (among other things)

Equality problem - Regular languages

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Recall closure properties of regular languages:

- closed under union, intersection, and complement (among other things)
- have seen procedures for constructing the DFA for unions/intersections/complements of regular languages.

Equality problem - Regular languages

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Recall closure properties of regular languages:

- closed under union, intersection, and complement (among other things)
- have seen procedures for constructing the DFA for unions/intersections/complements of regular languages.
- Using these, we can construct a DFA that accepts the symmetric difference of two regular languages.

Equality problem - Regular languages

Theorem

The language EQ_{DFA} is decidable.

Equality problem - Regular languages

Theorem

The language EQ_{DFA} is decidable.

Proof:

S_{DFA} = On input $\langle A, B \rangle$

1. Construct C, the DFA of the symmetric difference of $L(A)$ and $L(B)$.
2. Run N_{DFA} on C. (checks whether $L(C)$ is empty)
3. If N_{DFA} accepts, *accept*; if N_{DFA} rejects, *reject*.

Summary - Regular languages

- Regular languages are decidable:

Summary - Regular languages

- Regular languages are decidable:
- the acceptance problem (does A accept w?) is decidable, independent of the computational model in which we chose to describe regular languages;

Summary - Regular languages

- Regular languages are decidable:
- the acceptance problem (does A accept w?) is decidable, independent of the computational model in which we chose to describe regular languages;
- the emptiness problem (is $L(A)$ empty?) is decidable;

Summary - Regular languages

- Regular languages are decidable:
- the acceptance problem (does A accept w?) is decidable, independent of the computational model in which we chose to describe regular languages;
- the emptiness problem (is $L(A)$ empty?) is decidable;
- the equality problem (are $L(A)$ and $L(B)$ equal?) is decidable.

Summary - Regular languages

- Regular languages are decidable:
- the acceptance problem (does A accept w?) is decidable, independent of the computational model in which we chose to describe regular languages;
- the emptiness problem (is $L(A)$ empty?) is decidable;
- the equality problem (are $L(A)$ and $L(B)$ equal?) is decidable.
- in each case: we reduced the question to checking membership in a language.

Decision problems - CFLs

What about the decision problems for context-free languages?

Decision problems - CFLs

What about the decision problems for context-free languages?
Are the languages

$$\begin{aligned} A_{CFG} =& \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\} \\ E_{CFG} =& \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\} \\ EQ_{CFG} =& \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\} \end{aligned}$$

decidable?

Acceptance problem - CFLs

Theorem

The language A_{CFG} is decidable.

Proof:

Acceptance problem - CFLs

Theorem

The language A_{CFG} is decidable.

Proof:

- We cannot do the proof analogously to the DFA case: PDAs do not necessarily always terminate (they can endlessly loop, writing on to the stack).

Acceptance problem - CFLs

Theorem

The language A_{CFG} is decidable.

Proof:

- We cannot do the proof analogously to the DFA case: PDAs do not necessarily always terminate (they can endlessly loop, writing on to the stack).
- Instead, we use the fact that every CFG can be converted to a grammar in Chomsky Normal Form.

Acceptance problem - CFLs

Theorem

The language A_{CFG} is decidable.

Proof:

- We cannot do the proof analogously to the DFA case: PDAs do not necessarily always terminate (they can endlessly loop, writing on to the stack).
- Instead, we use the fact that every CFG can be converted to a grammar in Chomsky Normal Form.
- One can show (Problem 2.38 in Sipser) that if a grammar is CNF, then every derivation of w has length $2n - 1$, where n is the length of w.

Acceptance problem - CFLs

Theorem

The language A_{CFG} is decidable.

Proof:

- We cannot do the proof analogously to the DFA case: PDAs do not necessarily always terminate (they can endlessly loop, writing on to the stack).
- Instead, we use the fact that every CFG can be converted to a grammar in Chomsky Normal Form.
- One can show (Problem 2.38 in Sipser) that if a grammar is CNF, then every derivation of w has length $2n - 1$, where n is the length of w.
- That way we only need to check all derivations of length $2n - 1$ to see if any generates w!

Acceptance problem - CFLs

Theorem

The language A_{CFG} is decidable.

Proof:

M_{CFG} = On input $\langle G, w \rangle$

1. Convert G to a CFG in Chomsky Normal Form.
2. If $n = 0$, where n is the length of w, list all derivations with 1 step.
   Else, list all derivations with $2n - 1$ steps.
3. If any of the derivations generate w *accept*; otherwise, *reject*.

Decidability of CFLs

As in the regular language case, we can use this last result to show:

Corollary

Every context-free language is decidable.

Decidability of CFLs

As in the regular language case, we can use this last result to show:

Corollary

Every context-free language is decidable.

Proof: completely analogous to the DFA/regular case:

M_L = On input w

1. Simulate M_{CFG} on $\langle B, w \rangle$.
2. If M_{CFG} accepts, *accept*,
   if it rejects, *reject*.

Emptiness problem - CFLs

Theorem

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

Proof idea:

- In the DFA case, we checked reachability of accept states from the start state through a marking procedure.

Emptiness problem - CFLs

Theorem

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

Proof idea:

- In the DFA case, we checked reachability of accept states from the start state through a marking procedure.
- Can we do the same here?

Emptiness problem - CFLs

Theorem

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

Proof idea:

- In the DFA case, we checked reachability of accept states from the start state through a marking procedure.
- Can we do the same here?
- Yes! but slightly differently.
- Consider the grammar consisting of only $S \to S$. If we were to start with S and iteratively generate all derivations, we would never terminate.

Emptiness problem - CFLs

Theorem

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

Proof idea:

- In the DFA case, we checked reachability of accept states from the start state through a marking procedure.
- Can we do the same here?
- Yes! but slightly differently.
- Consider the grammar consisting of only $S \rightarrow S$. If we were to start with S and iteratively generate all derivations, we would never terminate.
- We're interested in finding out whether a string of terminals can be generated from S. So why not first mark terminals, then mark a variable A if there is a rule $A \rightarrow s$ where s consists of marked symbols?

Emptiness problem - CFLs

Theorem

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

Proof idea:

- In the DFA case, we checked reachability of accept states from the start state through a marking procedure.
- Can we do the same here?
- Yes! but slightly differently.
- Consider the grammar consisting of only $S \rightarrow S$. If we were to start with S and iteratively generate all derivations, we would never terminate.
- We're interested in finding out whether a string of terminals can be generated from S. So why not first mark terminals, then mark a variable A if there is a rule $A \rightarrow s$ where s consists of marked symbols?$\rightarrow$ go through derivations "backwards". If S is marked, then a string of terminals can be generated.

Emptiness problem - CFLs

Theorem

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

Example: Grammar

$$S \to ARB$$
$$A \to a$$
$$B \to b$$
$$R \to aRb \mid \varepsilon$$

Emptiness problem - CFLs

Theorem

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

Example: Grammar

$$
\begin{aligned}
S &\to ARB \\
A &\to \dot{a} \\
B &\to \dot{b} \\
R &\to \dot{a}R\dot{b} \mid \dot{\varepsilon}
\end{aligned}
$$

Emptiness problem - CFLs

Theorem

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

Example: Grammar

$$\begin{aligned} S &\rightarrow \dot{A}\dot{R}\dot{B} \\ \dot{A} &\rightarrow \dot{a} \\ \dot{B} &\rightarrow \dot{b} \\ \dot{R} &\rightarrow \dot{a}\dot{R}\dot{b} \mid \dot{\varepsilon} \end{aligned}$$

Emptiness problem - CFLs

Theorem

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

Example: Grammar

$$\dot{S} \rightarrow \dot{A}\dot{R}\dot{B}$$
$$\dot{A} \rightarrow \dot{a}$$
$$\dot{B} \rightarrow \dot{b}$$
$$\dot{R} \rightarrow \dot{a}\dot{R}\dot{b} \mid \dot{\varepsilon}$$

Emptiness problem - CFLs

Theorem

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

Example: Grammar

$$\dot{S} \rightarrow \dot{A}\dot{R}\dot{B}$$

$$\dot{A} \rightarrow \dot{a}$$

$$\dot{B} \rightarrow \dot{b}$$

$$\dot{R} \rightarrow \dot{a}\dot{R}\dot{b} \mid \dot{\varepsilon}$$

$\rightarrow$ S is marked, so language is not empty!

Emptiness problem - CFLs

Theorem

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

Proof:

N_{CFG} = On input $\langle G \rangle$

1. Mark all terminal symbols in G.
2. Repeat 3. until no new variables are marked:
3. Mark any variable A where G has a rule $A \rightarrow U_1 \ldots U_k$ and each symbol U_i has been marked.
4. If the start variable is not marked, *accept*. otherwise, *reject*.

Equality problem - CFLs

- So what about $EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$? Is it decidable?
- Before we used the symmetric difference $(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ to use the emptiness decider.

Equality problem - CFLs

- So what about $EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$? Is it decidable?
- Before we used the symmetric difference $(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ to use the emptiness decider.
- But context-free languages *are not closed under complementation or intersection!*

Equality problem - CFLs

- So what about $EQ_{CFG} = \{\langle G, H\rangle \mid G$ and H are CFGs and $L(G) = L(H)\}$? Is it decidable?
- Before we used the symmetric difference $(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ to use the emptiness decider.
- But context-free languages *are not closed under complementation or intersection!*
- in fact, EQ_{CFG} is *not* decidable. Next week we'll see techniques to show this.

Summary- CFLs

- the acceptance and emptiness decision problems are decidable for context-free languages
- hence, each context-free language is decidable.
- checking equivalence of two grammars (in the sense of languages generated) is *not* decidable!

Acceptance problems - TMs

- What about Turing-recognizable languages? Are they *also* decidable?

Acceptance problems - TMs

- What about Turing-recognizable languages? Are they *also* decidable?
- If they were, every Turing machine could be converted into an equivalent TM that is guaranteed to halt on every input!

Acceptance problem - TMs

First things first...

Theorem

The language $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$ *is Turing-recognizable.*

Acceptance problem - TMs

First things first...

Theorem

The language $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$ *is Turing-recognizable.*

Proof:

$U =$ On input $\langle M, w\rangle$

1. Simulate M on w.
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*.

Acceptance problem - TMs

First things first...

Theorem

The language $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$ *is Turing-recognizable.*

Proof:

$U =$ On input $\langle M, w\rangle$

1. Simulate M on w.
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*.

U is an example of a *universal Turing machine*!

Acceptance problem - TMs

So what about decidability?

Acceptance problem - TMs

So what about decidability?

Theorem

The language A_{TM} is not decidable.

Acceptance problem - TMs

So what about decidability?

Theorem

The language A_{TM} is not decidable.

Proof:

- Assume it is decidable. Then there exists a decider H that decides A_{TM}. So $H(\langle M, w\rangle) = \mathit{accept}$ iff M accepts w and $H(\langle M, w\rangle) = \mathit{reject}$ iff M fails to accept w.

Acceptance problem - TMs

So what about decidability?

Theorem

The language A_{TM} is not decidable.

Proof:

- Assume it is decidable. Then there exists a decider H that decides A_{TM}. So $H(\langle M, w\rangle) = \mathit{accept}$ iff M accepts w and $H(\langle M, w\rangle) = \mathit{reject}$ iff M fails to accept w.
- Now we construct a new machine D that takes a Turing machine M as input and uses H as a subroutine. In particular, it calls $H(\langle M, \langle M\rangle\rangle)$, i.e., H will tell us whether M accepts or rejects the string $\langle M\rangle$.
- The new machine D will then *reverse* the result, i.e., if H accepts, D rejects and if H rejects, D accepts.

Acceptance problem - TMs

Theorem

The language A_{TM} is not decidable.

Proof:

$$D = \text{On input } \langle M \rangle$$

1. Simulate H on $\langle M, \langle M \rangle \rangle$.
2. If H accepts, *reject*; if H rejects, *accept*.

Acceptance problem - TMs

Theorem

The language A_{TM} is not decidable.

Proof:

$D =$ On input $\langle M \rangle$

1. Simulate H on $\langle M, \langle M \rangle \rangle$.
2. If H accepts, *reject*; if H rejects, *accept*.

- So what is the result of $D(\langle D \rangle)$?

Acceptance problem - TMs

Theorem

The language A_{TM} is not decidable.

Proof:

$D =$ On input $\langle M \rangle$

1. Simulate H on $\langle M, \langle M \rangle \rangle$.
2. If H accepts, *reject*; if H rejects, *accept*.

- So what is the result of $D(\langle D \rangle)$? Remember, $H(\langle M, w \rangle)$ accepts iff $M(w) = accept$.

Acceptance problem - TMs

Theorem

The language A_{TM} is not decidable.

Proof:

$D =$ On input $\langle M \rangle$

1. Simulate H on $\langle M, \langle M \rangle \rangle$.
2. If H accepts, *reject*; if H rejects, *accept*.

- So what is the result of $D(\langle D \rangle)$? Remember, $H(\langle M, w \rangle)$ accepts iff $M(w) = accept$.
- If $D(\langle D \rangle) = reject$, then $H(\langle D, \langle D \rangle \rangle) = accept$, i.e., $D(\langle D \rangle) = accept$. Contradiction!

Acceptance problem - TMs

Theorem

The language A_{TM} is not decidable.

Proof:

$$D = \text{On input } \langle M \rangle$$

1. Simulate H on $\langle M, \langle M \rangle \rangle$.
2. If H accepts, *reject*; if H rejects, *accept*.

- So what is the result of $D(\langle D \rangle)$? Remember, $H(\langle M, w \rangle)$ accepts iff $M(w) = \mathit{accept}$.
- If $D(\langle D \rangle) = \mathit{reject}$, then $H(\langle D, \langle D \rangle \rangle) = \mathit{accept}$, i.e., $D(\langle D \rangle) = \mathit{accept}$. Contradiction!
- If $D(\langle D \rangle) = \mathit{accept}$, then $H(\langle D, \langle D \rangle \rangle) = \mathit{reject}$, i.e., $D(\langle D \rangle) = \mathit{reject}$. Contradiction!

Acceptance problem - TMs

Theorem

The language A_{TM} is not decidable.

Proof:

$D =$ On input $\langle M \rangle$

1. Simulate H on $\langle M, \langle M \rangle \rangle$.
2. If H accepts, *reject*; if H rejects, *accept*.

- So what is the result of $D(\langle D \rangle)$? Remember, $H(\langle M, w \rangle)$ accepts iff $M(w) = accept$.
- If $D(\langle D \rangle) = reject$, then $H(\langle D, \langle D \rangle \rangle) = accept$, i.e., $D(\langle D \rangle) = accept$. Contradiction!
- If $D(\langle D \rangle) = accept$, then $H(\langle D, \langle D \rangle \rangle) = reject$, i.e., $D(\langle D \rangle) = reject$. Contradiction!
- Hence neither D nor H can exist! $\rightarrow A_{TM}$ is undecidable!