Theorem 4.1

The Language ...

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

... is Decidable

Possible Confusion

String
w
Language
DFA ≡ Regular Language

String $\langle B, w\rangle$
Language A_{DFA} ⟹ Not Regular,
Not CFG,
But it is decidable

Proof That

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

is Decidabe.

Provide a TM that decides it.

The TM is given as input $\langle B, w \rangle$ a DFA and a string w.

The TM checks to make sure B is a valid representation.

The TM then simulates B on w.

If B reaches a final state at the end of w,

Then the TM will accept.

Otherwise The TM will reject.

This TM will always halt.

(We could prove that the TM will Always halt, if necessary.)

Theorem 4.2

The Language ...

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

... is Decidable.

Proof:

Construct a TM that takes as input the representation of an NFA B and a candidate string w.

Approach 1: Simulate the NFA on w.

Approach 2:

Convert the NFA to a DFA
This algorithm was described in Chapter 1
We could program a TM to do it.

Simulate the DFA on w.
We could program a TM to do it
[we did that for A_{DFA}]

Accept if the simulation accepts, otherwise reject.

Theorem 4.3

The Language...

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

... is Decidable.

How do we know?

We can build a TM / we can write a program that, given a REG. Expression R and string w as input, will determine whether $\langle R, w\rangle$ is in A_{REX}.

And that algorithm / TM will always halt.

Halting?

Often this is self-evident. Sometimes we ought to Prove it carefully.

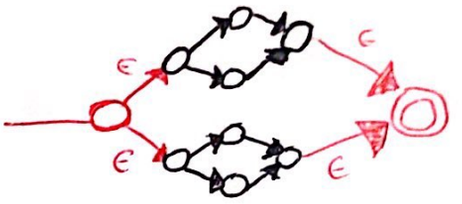
Many Programs can Be Proven to Always halt.
But this does not mean the halting Problem is decidable!

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

Algorithm/TM

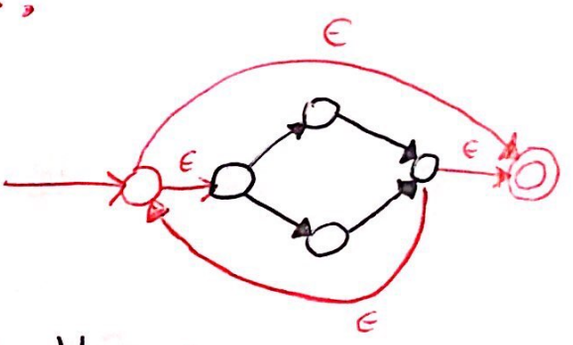
Step 1: Convert R into a NFA, B'. (Recall the algorithm)

$X \mid Y \Longrightarrow$



$X \circ Y \Longrightarrow \ldots$

$X^* \Longrightarrow$



Step 2: Construct the string $\langle B', w\rangle$

Step 3: Use the TM from the last theorem to decide whether this is in

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

Acceptance Testing

Is string w in the Language?

$$A_{DFA} = \{\langle B, w\rangle \mid \ldots\}$$

Emptness Testing

Is the Language empty?

$L = \emptyset$?

$$E_{DFA} = \{\langle B\rangle \mid \ldots\}$$

EQuality Testing

Are Two Languages the same?

$$EQ_{DFA} = \{\langle A, B\rangle \mid \ldots\}$$

Theorem 4.4

The Language

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

... is decidable

Algorithm / TM to decide it:

Given A DFA "A", can we go from the intial state to final state?

If so, then the DFA could generate some string.

⇒ Is any final state reachable from the initial state?

A graph problem:

- Mark the initial state
- Repeat until No new states get marked...
- Mark any state where there is a tranition to it from a marked state
- check to see if any final state got marked

Theorem 4.5

The Language

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

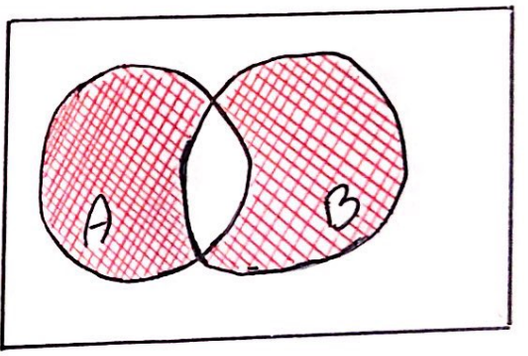
... is decidable

Proof:

Let C be the "Symmetric Difference" between A and B

"Anything in A or B But Not both"

$$C = (A \cap \bar{B}) \cup (\bar{A} \cap B)$$



Note If $A = B$ then the Symmetric Difference will be $\emptyset$.

Given...

A = DFA to accept L(A)

B = DFA to accept L(B) ...

Build DFA C to accept the Symmetric Difference.

to combine DFAs.

$\overline{L(A)}$

$L(A) \cup L(B)$

$L(A) \cap L(B)$

Use the TM from previous theorem [E_{DFA} ≡ empty language] to test

Proof:

Construct a TM

Input: $\langle A, B \rangle$ (2 DFA's)

Construct a DFA C to accept

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

Use the previous TM to test whether the Language that C accepts is empty.

Accept if so; Reject otherwise.

Theorem 4.7

The Language

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

is decidable

In other words:

"Given A CFG and a string, we can write a program that will always halt that will tell us Yes/No whether the grammar generates the string"

All grammers are "parsable"

Some ~~kinds~~ kinds of grammers (eg. LL(k) or LR(k) grammers) can be parsed very efficiently. ← $O(N)$

But in general, the parser may take $O(N^3)$ time.

Proof:

Inputs: G = a CFG

w = a string

Step 1: Convert G into Chomsky normal form.

Derivations using CNF grammer

At each step, the length grows by exactly 1

$S \to SS$

$S \to a$

$S \Rightarrow_1 SS \Rightarrow_2 SSS \Rightarrow_3 SSSS \Rightarrow_4 SSSSS$ (N-1 steps; 5)

Plus 1 additional step for each terminal symbol:

$\Rightarrow_1 aSSSS \Rightarrow_2 aaSSS \Rightarrow_3 aaaSS \Rightarrow_4 aaaaS \Rightarrow_5 aaaaa$ (N steps)

∴ every derivation has exactly $2N-1$ steps

Step 2:

Let N be the length of w.

List all derivations of length $2N-1$ (There are only finitely many)

Check each derivation to see if it generates w.

If any derivation generates w then accepts. else rejects.

Theorem 4.8

The language

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

is decidable

To prove E_{CFG} is decidable...

To Prove some Problem is decidable...

Give an algorithm (i.e. a T.M.) verify the algorithm is correct

* Always halts.
* Gives the right answer.

Algorithm

Input: a CFG "G"

A "marking" algorithm:

Mark all terminal symbols

REPEAT

Look for a rule

$A \rightarrow XXXX$

where all symbols on right side have been marked

$B \rightarrow CA$

Mark the non-terminal on the Left Side.

until nothing more can be marked

If the Start Symbol is NOT marked

then accept ($L(G) = \emptyset$)

else Reject ($L(G) \neq \emptyset$)

Ex:

- Consider this grammar
- Which nonterminals can generate a string of terminals?

$S \rightarrow ABCD$

$A \rightarrow BCA$

$A \rightarrow xyz$

$B \rightarrow CA$

$B \rightarrow AB$

$B \rightarrow BBB$ w

$C \rightarrow CB$

$C \rightarrow w$ w

$D \rightarrow DD$

$D \rightarrow BD$

$D \rightarrow DC$

Theorem 4.9

The Language

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

Theorem 4.5 gave an algorithm that decides the analogous Language EQ_{DFA} for finit automata. We used the decision procedure for E_{DFA} to prove that EQ_{DFA} is decidable. Because E_{CFG} also is decidable, you might think that we can use a similar strategy to prove that EQ_{CFG} is decidable. But something is wrong with this idea. The class of context-free Language is Not closed under complementation or intersection, as you proved. In fact, EQ_{CFG} is NOT Decidable. The technique for proving so is presented in coming chapters.

Proof:

Let G be a CFG for A and design a TM M_G that decides A.

We build a copy of G into M_G. It works as follows.

M_G = "On input w:

1) Run TM S on input $\langle G, w\rangle$.
2) if this machine accepts, accept
   if it reject, reject"

Theorem 4.22

A Language L is decidable iff L and $\overline{L}$ are Turing Recognizable.

Definition

A Language is "co-Turing Recognizable" if its complement is Turing Recognizable

Restating the Theorem

A Language is decidable iff it is:

1) Turing Recognizable
2) co-Turing Recognizable

Proof:

Assume $\overline{A_{TM}}$ is Turing Recognizable

If a Language and its complement are both Turing Recognizable, Then the language is decidable

But we know A_{TM} is NOT decidable

Recall:

A_{TM} is Turing Recognizable

A_{TM} is NOT Decidable

Therefore

$\overline{A_{TM}}$ is NOT Turing Recognizable

Theorem 5.1

The Language ...

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine and } M \text{ halts on input } w \}$$

is UNDECIDABLE

Proof:

- Assume it is decidable (There is a TM R that decides $HALT_{TM}$)
- Use R to build another TM, S, that decides A_{TM}
  - * Reduce A_{TM} to $HALT_{TM}$ *
- But A_{TM} is undecidable.
  - * Contradiction *

$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts } w \}$

"S"

To Decide A_{TM} ...
Given M and some input w,
If M accepts w
Then ⟹ accept.
If M ~~rejects~~ w OR Loops
Then ⟹ rejects.

* Deciders can never Loop *

Note: "S" is Proven NOT to Exist.

R = A Turing Machine that decides
$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on } w \}$

"R"

To Decide $HALT_{TM}$
Given M ~~and accepts~~ some input ~~or rejects~~ w,
If M accepts OR rejects w ...
Then ⟹ accepts
If M Loops ...
Then ⟹ rejects

Here is Algorithm S:

Input: $\langle M, w \rangle$

- Run R on $\langle M, w \rangle$ to see if M halts or loops
- If R accepts, it means M halts. * and it will loop *
- Simulate / Run M on input w. when M halts ...

If M accepts, Then ⟹ accepts
If M rejects, Then ⟹ rejects

So "S" is a decider for A_{TM}.
Contradiction:
A_{TM} is undecidable.

Theorem 5.2

Does a TM accept any string? — undecidable

The Language ...

$$E_{TM} = \{ \langle M \rangle \mid M \text{ is a Turing Machine and } L(M) = \emptyset \}$$

is undecidable.

Proof:

- Assume that R decides E_{TM}.
- Use it to construct S, a decider for A_{TM} * we will reduce A_{TM} to E_{TM} *
- Contradiction.

Step 1:

- Modify M a ~~little~~ bit.
- Call the new machine M'.

M' (Assume: w is a constant and x is the input)

- will reject all strings that do NOT match w.
- May or may not accept w.

$$L(M') = \{w\} \text{ or } L(M') = \emptyset$$

Algorithm for M'

Input: x
if $x \neq w$ then reject
otherwise
Simulate M on x.
if M accepts, then accept.
if M rejects, then reject.

Step 2:

Use R to decide whether L(M') is empty or not.

R accepts $\Rightarrow$ L(M') is empty $\Rightarrow$ M does NOT accept w.

R rejects $\Rightarrow L(M') = \{w\} \Rightarrow$ M accept w.

Step 3:

We now have decided A_{TM}

Contradiction (E_{TM} is undecidable)

$\Leftarrow$ other 2 steps

Goal:

- Construct an Algorithm(S) which is given $\langle M, w \rangle$ and decides whether M accepts w

Algorithm for S: (to decide A_{TM})
input M, w
Step 1: Construct M'.
Take M
add a test (x=w)
in front of the initial State
Then pass control to M

*** Note: ***
$L(M') = \{w\}$ if M accepts w
$L(M') = \emptyset$ otherwise.

Theorem 5.3

The Language ...

$$Regular_{TM} = \{ \langle M \rangle \mid \begin{array}{l} M \text{ is Turing Machine and} \\ L(M) \text{ is a Regular Language.} \end{array} \}$$

is undecidable

Proof:

- Assume that R decides $Regular_{TM}$
- Use it to ~~construct~~ Construct S, a decider for A_{TM}

S = "On input $\langle M, w \rangle$, where M is a TM and w is a string

1- Construct the following TM M_2.

M_2 { On input x
- If x has the form $0^n 1^n$ then accept
  If x does NOT have this form, run M on input w and if accepts then accept w.

2- Run R on input $\langle M_2 \rangle$

3- If R accepts then accepts.
   If R rejects then rejects.

Theorem 5.4

The Language ...

$$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are Turing Machine and } L(M_1) = L(M_2)\}$$

is Undecidable.

Proof:

- Assume that R decides EQ_{TM}
- Use it to construct S, a decider for E_{TM}

S { $E_{TM} = \{\langle M\rangle \mid M \text{ is a Turing Machine}\}$

To decide E_{TM} ...

- Run R on input $\langle M, M_1\rangle$ where is TM that reject all inputs
- If R accepts then accepts.
  If R rejects then rejects.

- If R decides EQ_{TM}, S decides E_{TM}.

But E_{TM} is Undecidable by Theorem 5.2,

So EQ_{TM} # also must by Undecidable.