Warm Up Problem

- Write a CFG that recognizes $L = \{a^i b^j c^i : i, j \in \mathbb{N}\}$.

CS 241 Lecture 10

Context Free Grammars, Ambiguity, Top Down Parsing

With thanks to Brad Lushman, Troy Vasiga and Kevin Lanctot

Practice

Let $\Sigma = \{a, b\}$. Find a CFG for each of the following:

- $a(a|b)^*b$ (can also write as $a(a+b)^*b$)
- $\{a^n b^n : n \in \mathbb{N}\}$
- Palindromes over $\{a, b, c\}$

Further, for the second language, find a derivation in your grammar of *aaabbb*.

Solution to Second

CFG:

$$S \rightarrow \epsilon \mid aSb$$

Derivation:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$.

A Fundamental Example

Let's consider arithmetic operations over $\Sigma = \{a, b, c, +, -, *, /, (,)\}$. Find a CFG for the following

- L_1: Arithmetic expressions from Σ without parentheses
- L_2: Well-formed [in the CS 245 sense] arithmetic expressions from Σ with balanced parentheses

Also, find a derivation for $a - b$ in the first language and for $(a - b)$ in the second one

Solutions

For L_1: Arithmetic expressions from Σ without parentheses

$$\begin{aligned} S &\rightarrow a \mid b \mid c \mid SRS \\ R &\rightarrow + \mid - \mid * \mid / \end{aligned}$$

For L_2: Arithmetic expressions from Σ with balanced parentheses

$$\begin{aligned} S &\rightarrow a \mid b \mid c \mid (\, SRS \,) \\ R &\rightarrow + \mid - \mid * \mid / \end{aligned}$$

Derivations:

$$S \Rightarrow SRS \Rightarrow aRS \Rightarrow a - S \Rightarrow a - b$$

$$S \Rightarrow (SRS) \Rightarrow (SRb) \Rightarrow (S - b) \Rightarrow (a - b)$$

Note

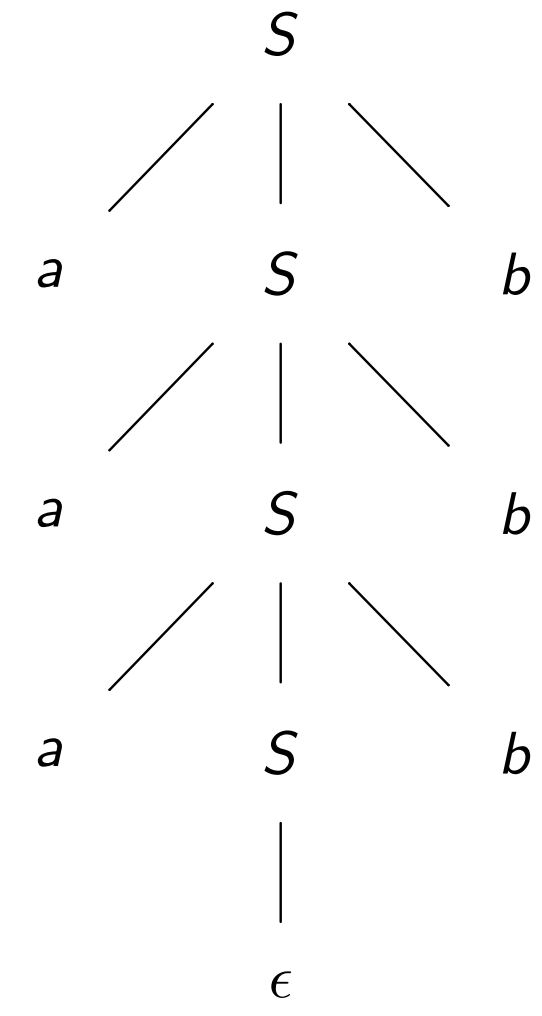
Notice in these two derivations that we had a choice at each step which element of N to replace

$$S \Rightarrow SRS \Rightarrow aRS \Rightarrow a - S \Rightarrow a - b$$
$$S \Rightarrow (SRS) \Rightarrow (SRb) \Rightarrow (S - b) \Rightarrow (a - b)$$

In the first derivation, we chose to do a left derivation, that is, one that always expands from the left first.
In the second derivation, we chose to do a right derivation, that is, one that always expands from the right first.

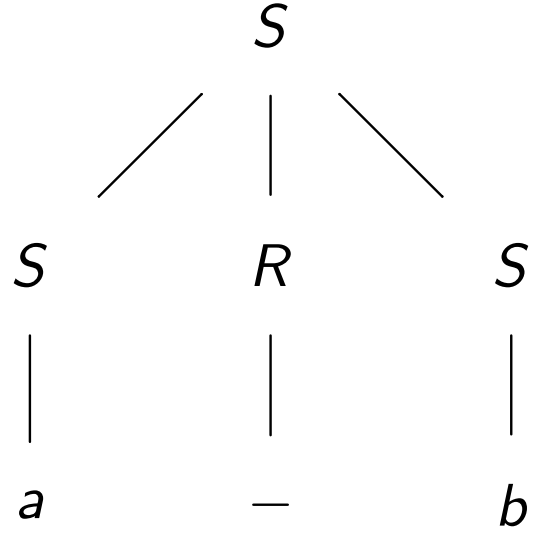
Parse Trees

$aaabbb$ in $S \rightarrow \epsilon | aSb$



$a - b$ in

$$S \rightarrow a \mid b \mid c \mid SRS$$
$$R \rightarrow + \mid - \mid * \mid /$$



Note: To every left (right) most derivation there exists a unique parse tree (and vice versa).

Question

Given a grammar, is it possible that every (left/right) derivation for a string is unique?

Question

Given a grammar, is it possible that every (left/right) derivation for a string is unique?

No! Consider the following two left-most derivations on the next slide for $a - b * c$.

Left-Most Derivations

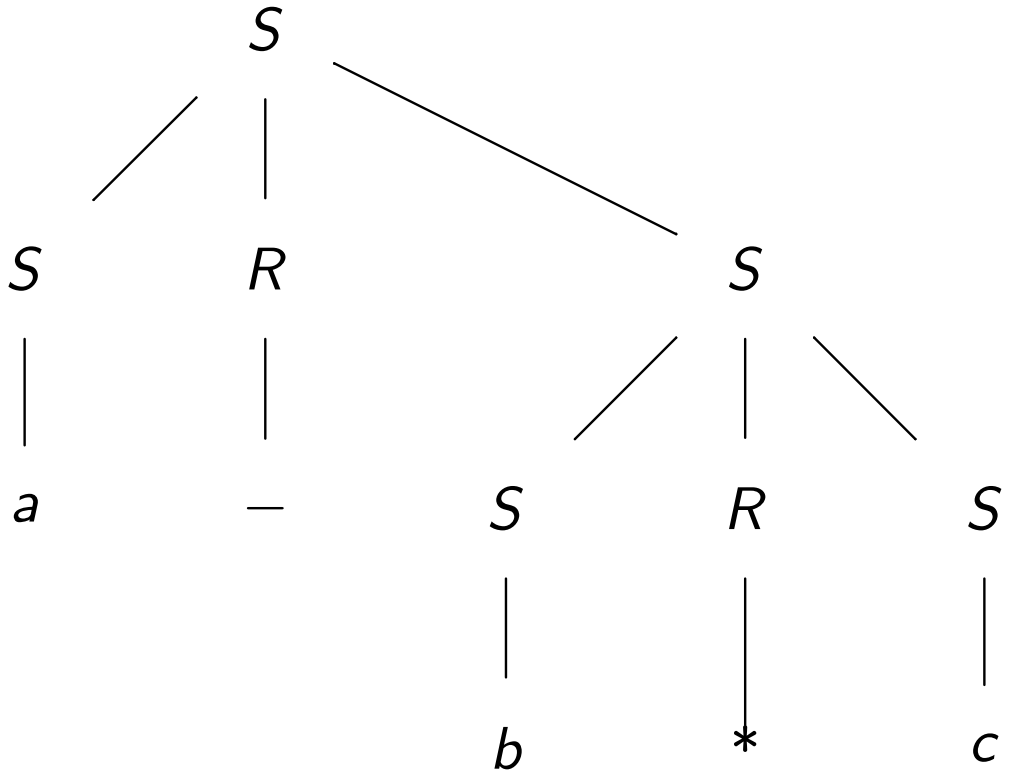
$$
\begin{aligned}
S &\rightarrow a \mid b \mid c \mid SRS \\
R &\rightarrow + \mid - \mid * \mid /
\end{aligned}
$$

$$
\begin{aligned}
S &\Rightarrow SRS \Rightarrow aRS \Rightarrow a - S \Rightarrow a - SRS \\
&\Rightarrow a - bRS \Rightarrow a - b * S \Rightarrow a - b * c \\
S &\Rightarrow SRS \Rightarrow SRSRS \Rightarrow aRSRS \Rightarrow a - SRS \\
&\Rightarrow a - bRS \Rightarrow a - b * S \Rightarrow a - b * c
\end{aligned}
$$

these correspond to different parse trees! Try to draw these!

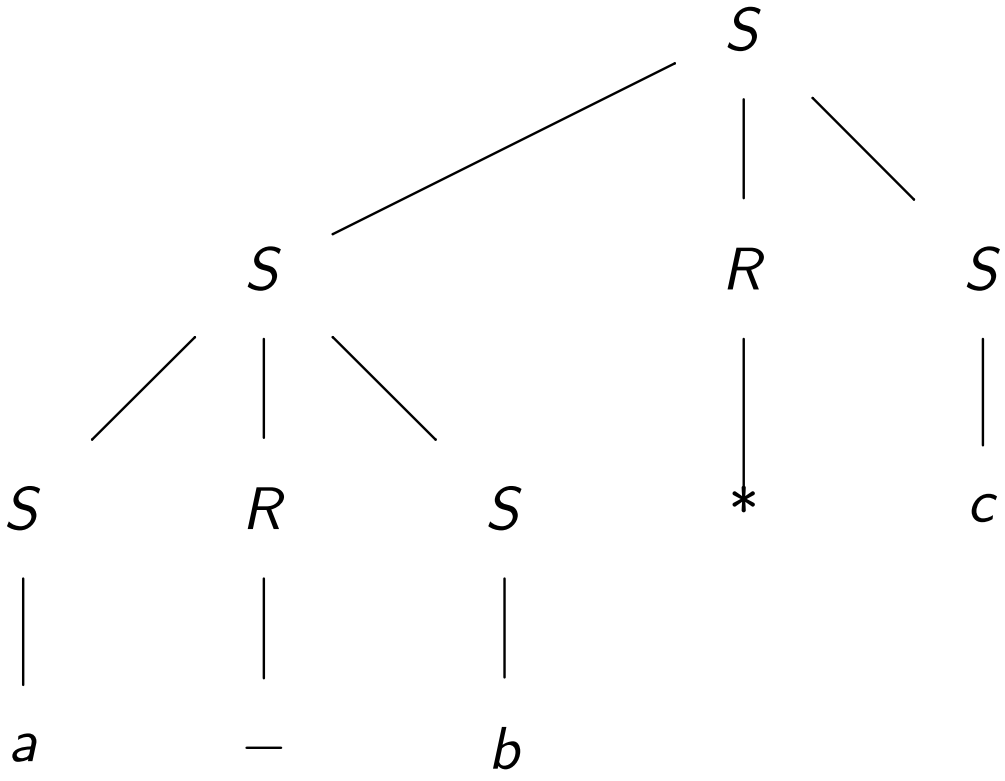
First Parse Tree

$$S \Rightarrow SRS \Rightarrow aRS \Rightarrow a - S \Rightarrow a - SRS$$
$$\Rightarrow a - bRS \Rightarrow a - b * S \Rightarrow a - b * c$$



Second Parse Tree

$$S \Rightarrow SRS \Rightarrow SRSRS \Rightarrow aRSRS \Rightarrow a - SRS$$
$$\Rightarrow a - bRS \Rightarrow a - b * S \Rightarrow a - b * c$$



Ambiguous Grammars

Definition

A grammar for which some word has more than one distinct leftmost derivation/rightmost derivation/parse tree is called **ambiguous**.

The example of:

$$\begin{aligned} S &\rightarrow a \mid b \mid c \mid SRS \\ R &\rightarrow + \mid - \mid * \mid / \end{aligned}$$

was an example of an ambiguous example

Sure But...

- Why do we care about this? Isn't our goal to determine whether or not $w \in L(G)$?

Sure But...

- Why do we care about this? Isn't our goal to determine whether or not $w \in L(G)$?
- As compiler writers we care about where the derivation came from!
- Parse trees give meaning to the string with respect to the grammar.
- Go back to the parse trees. The first example of $a - b * c$ means that $a - (b * c)$ whereas the second means $(a - b) * c$.
- How can we fix this?

Solutions

- Use some sort of precedence heuristics to guide the derivation process (very dependent on grammar).
- Make the grammar unambiguous! This is what we did with L_2

$$
\begin{aligned}
S &\rightarrow a \mid b \mid c \mid (SRS) \\
R &\rightarrow + \mid - \mid * \mid /
\end{aligned}
$$

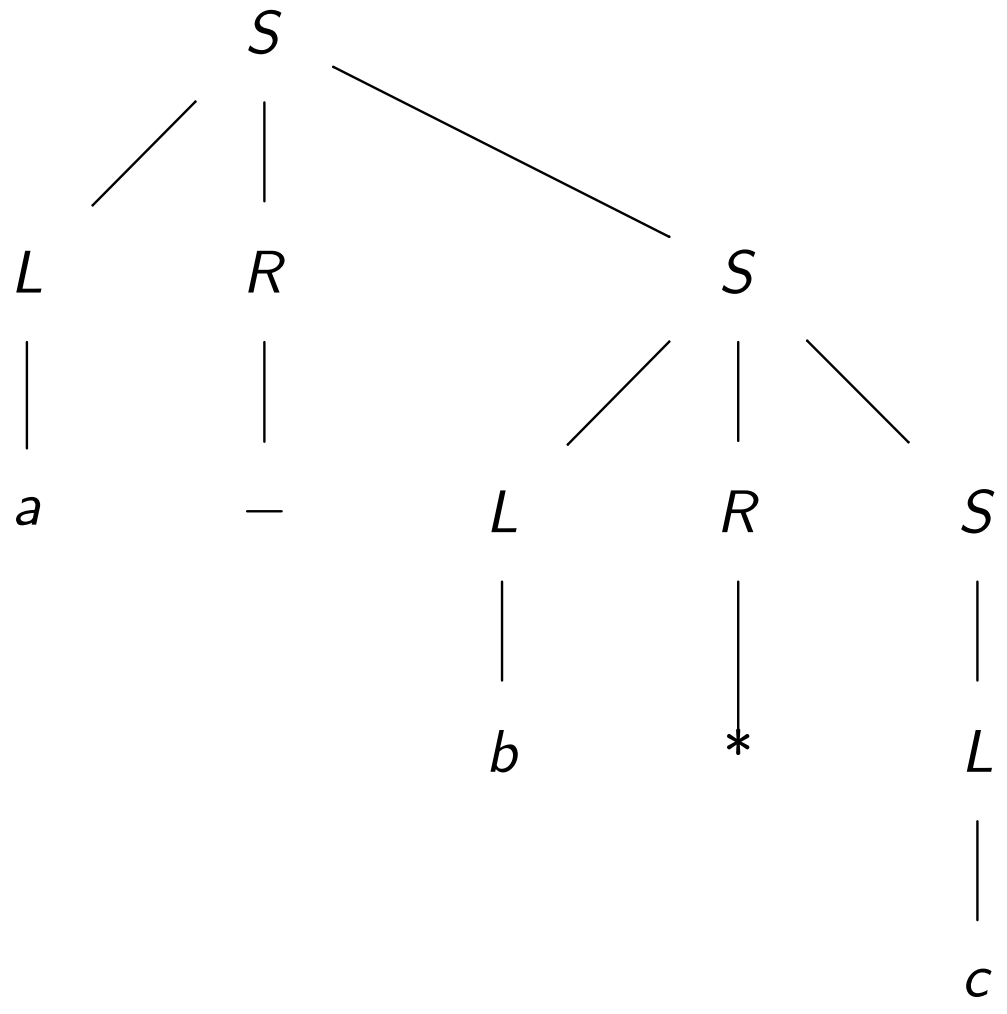
Ambiguity

- Another way to remove ambiguity without parentheses is to really examine how parse trees work.
- In a parse tree, you evaluate the expression in a depth first post-order traversal (Left, Right, Root - Recall A3).
- We also have the other issue that we want to interpret $a - b + c$ as $(a - b) + c$ and NOT as $a - (b + c)$ (that is, we want left associativity).
- We can make a grammar left/right associative by insisting on how the recursion works! (See next slides)

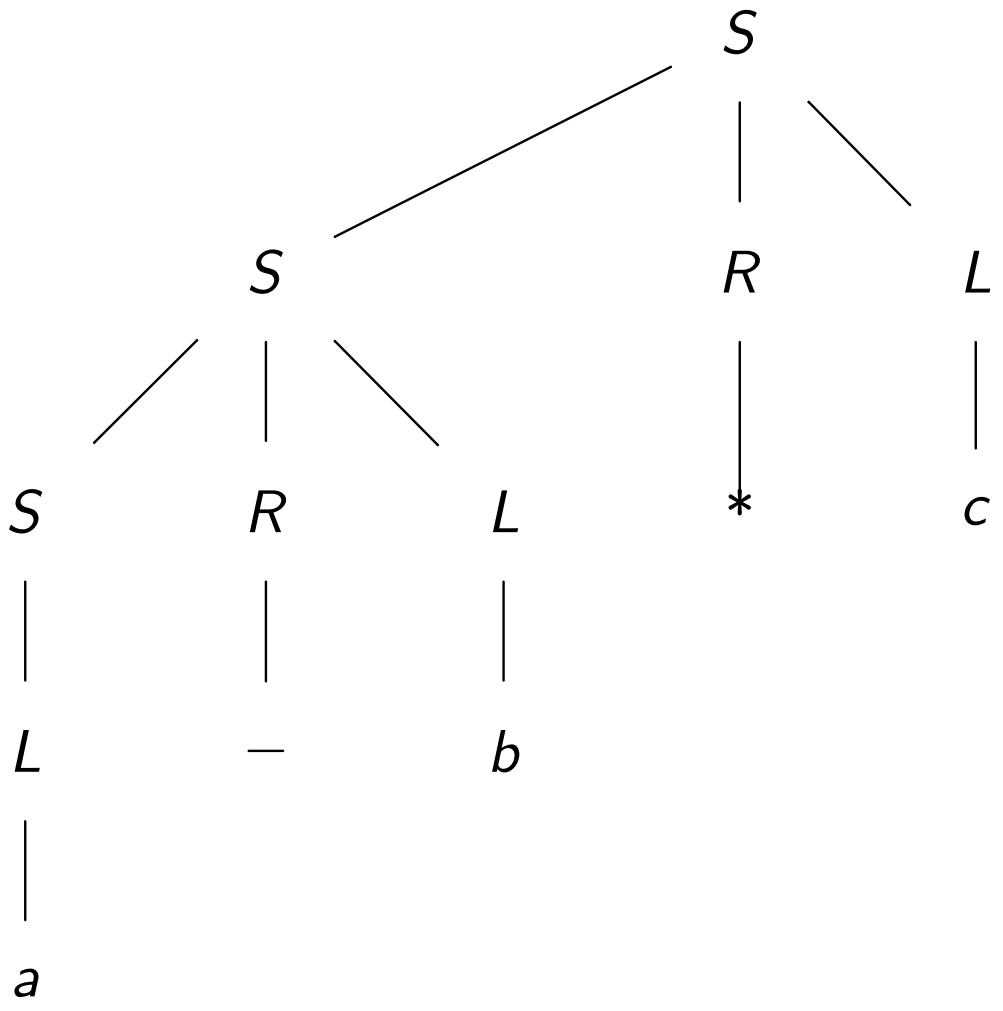
Forcing Right Associative

$$
\begin{aligned}
S &\rightarrow LR\mathbf{S} \mid L \\
L &\rightarrow a \mid b \mid c \\
R &\rightarrow + \mid - \mid * \mid /
\end{aligned}
$$

This forces a right associative grammar (eg. Parse Tree for $a - b * c$ is to the right)



Forcing Left Associative



$S \rightarrow \mathbf{S}RL \mid L$
$L \rightarrow a \mid b \mid c$
$R \rightarrow + \mid - \mid * \mid /$

This forces a left associative grammar (eg. Parse Tree for $a - b * c$ is to the left)

With the above...

We can use the above to create a grammar in a similar way that follows BEDMAS rules more closely by making $*, /$ appear further down the tree (hence evaluated first!):

$$
\begin{aligned}
S &\rightarrow SPT \mid T \\
T &\rightarrow TRF \mid F \\
F &\rightarrow a \mid b \mid c \mid (S) \\
P &\rightarrow + \mid - \\
R &\rightarrow * \mid /
\end{aligned}
$$

Note that T will always appear as a child first and hence get evaluated first!

Exercise: Find a derivation of $a - b * c$ and then give the parse tree.

Some Questions

- If L is a context-free language, is there always an unambiguous grammar such that $L(G) = L$?

Some Questions

- If L is a context-free language, is there always an unambiguous grammar such that $L(G) = L$?
- No! First proven in 1961 by Rohit Parikh. From Wikipedia: An example of an inherently ambiguous language is the union of $\{a^n b^m c^m d^n : n, m > 0\}$ with $\{a^n b^n c^m d^m : n, m > 0\}$. This set is context-free, since the union of two context-free languages is always context-free. But Hopcroft and Ulman in 1979 give a proof that there is no way to unambiguously parse strings in the (non-context-free) common subset $\{a^n b^n c^n d^n : n > 0\}$

Decidability of Ambiguous Grammars

- Can we write a computer program to recognize whether or not a grammar is ambiguous?

Decidability of Ambiguous Grammars

- Can we write a computer program to recognize whether or not a grammar is ambiguous?
- No! Most textbooks (see for example Hopcroft, John; Motwani, Rajeev; Ullman, Jeffrey (2001). Introduction to automata theory, languages, and computation Theorem 9.20, pp. 405-406) reduce to the undecidability of Post's Correspondence Problem.
- Original proofs are due to Cantor (1962), Floyd (1962), and Chomsky and Schtzenberger (1963).

Yikes!

- What about an easier problem of given two CFGs G_1 and G_2, determine whether or not $L(G_1) = L(G_2)$.

Yikes!

- What about an easier problem of given two CFGs G_1 and G_2, determine whether or not $L(G_1) = L(G_2)$.
- Maybe even easier what about determining whether or not $L(G_1) \cap L(G_2) = \emptyset$?

Yikes!

- What about an easier problem of given two CFGs G_1 and G_2, determine whether or not $L(G_1) = L(G_2)$.
- Maybe even easier what about determining whether or not $L(G_1) \cap L(G_2) = \emptyset$?
- Still undecidable!

What We Can Answer

- Regular languages corresponded to machines, namely DFAs.
- Is there a machine correspondence for CFGs?

What We Can Answer

- Regular languages corresponded to machines, namely DFAs.
- Is there a machine correspondence for CFGs?
- Yes! The correspondence is with a **pushdown automaton**. These machines are [informally] DFAs with an additional stack that we can process in LIFO order.
- While useful, recall we don't just need to know whether or not $w \in L(G)$ but we also need the **derivation**!
- The problem of finding the derivation is called **parsing**.

Formally

Given: a CFG $G = (N, \Sigma, P, S)$ and a terminal string $w \in \Sigma^*$
Find: The derivation, that is, the steps such that $S \Rightarrow ... \Rightarrow w$ or prove that $w \notin L(G)$.

How Do We Find This Derivation?

Two ideas:

- Forwards (Top-Down Parsing). Start with S and then try to get to w (eg. LL(1))
- Backwards (Bottom-Up Parsing) Start with w and figure out how we could have gotten to w. (Eg. LR(0) and SLR(1))
- Neither seems like a great option...

Top-Down Parsing

- We will start with S and store intermediate derivations in a stack and then match characters to w.
- Every time we pop from the stack, we will have that consumed input + reverse of stack is equal to a intermediate step in our derivation, that is, such a step is an α_i where $S \Rightarrow ... \Rightarrow \alpha_i \Rightarrow ... \Rightarrow w$
- In these procedures, we will actually augment our grammar to include $\vdash$ and $\dashv$ symbolizing the beginning and end of the file respectively. We also include a new start state S' to begin our parsing.
- Thus, our original CFG $G = (N, \Sigma, P, S)$ becomes $G' = (N', \Sigma', P', S')$ where

$$N' = N \cup \{S'\}$$
$$\Sigma' = \Sigma \cup \{\vdash, \dashv\}$$
$$P' = P \cup \{S' \rightarrow \vdash S \dashv\}$$