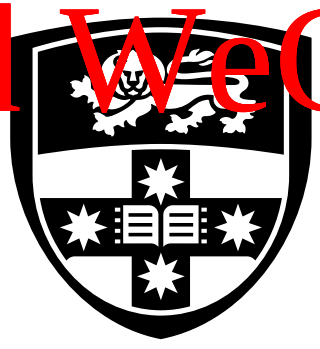
COMP2022: Formal Languages and Logic

2018, Semester 2, Week 8



THE UNIVERSITY OF SYDNEY

Outline

- Revision
-
-
- LL(k) Table-Descent Parsing

Revision

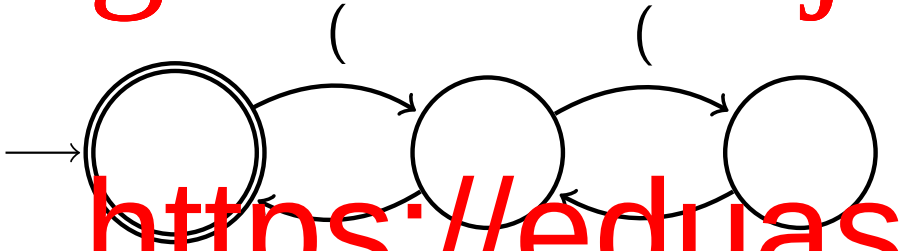
- Generate Context-Free Languages
  - Very important class of languages in CS (compilers, NLP, etc.)
- All rules are in the form $A \quad \alpha$
-
-
- Clean grammars

Next lecture:

- Push-Down Automata
- Parsing

Limits of DFA/NFA

Recall this NFA for balanced parentheses



It accepts strings of balanced parentheses, neste

What if we wanted a depth of 3?

What if we want any level of nesting?

If we added a *stack* to the automata, we cou
nesting

Push Down Automata

- Finite automata *with a stack*

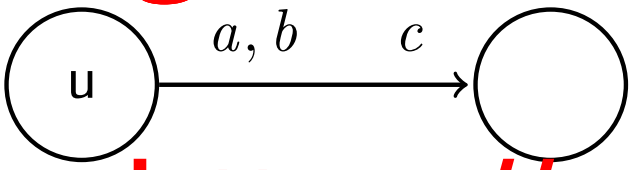
A PD

-
-
- Perform stack operations (new!)

In addition to storing terminals and variables, the stack will accept a special end of string symbol, which we will denote $.

PDA Transitions

PDA Transitions include stack operations



a, b

- Read (and remove) symbol a fro
- Pop symbol b from the top of the stack
- Push c onto the stack

We follow a transition if it is possible to perform these operations

PDA Transitions

We use ε to denote "no operation". For example:

- $\alpha, \varepsilon \to C$
- read (and remove) a from the front of the input

PDA Transitions

We use ε to denote "no operation". For example:

- $a, \varepsilon \to c$
  - read (and remove) a from the front of the input

- $\varepsilon, \varepsilon \to c$
  - do not read anything from the input
  - do not pop anything from the stack
  - push c onto the stack

PDA Transitions

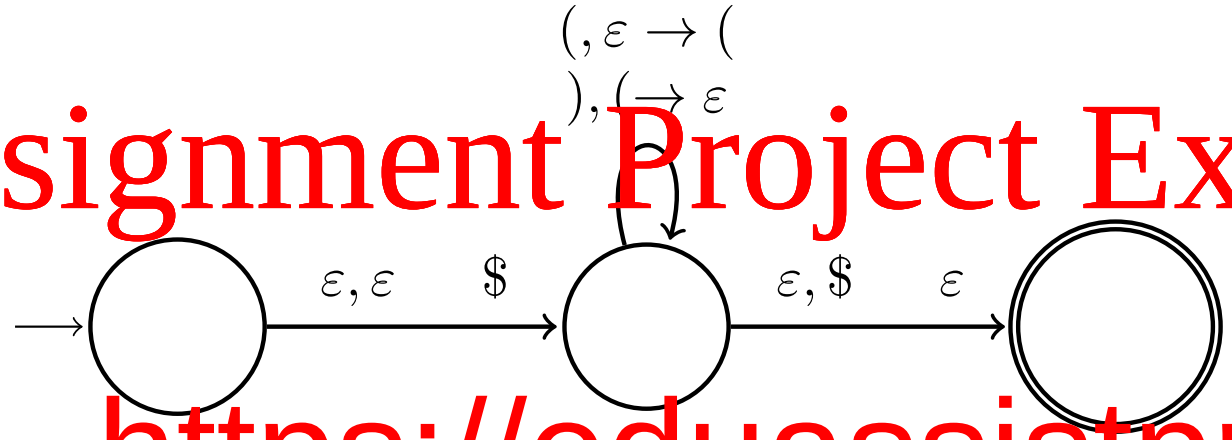
We use ε to denote "no operation". For example:

- $a, \varepsilon \to c$
  - read (and remove) a from the front of the input

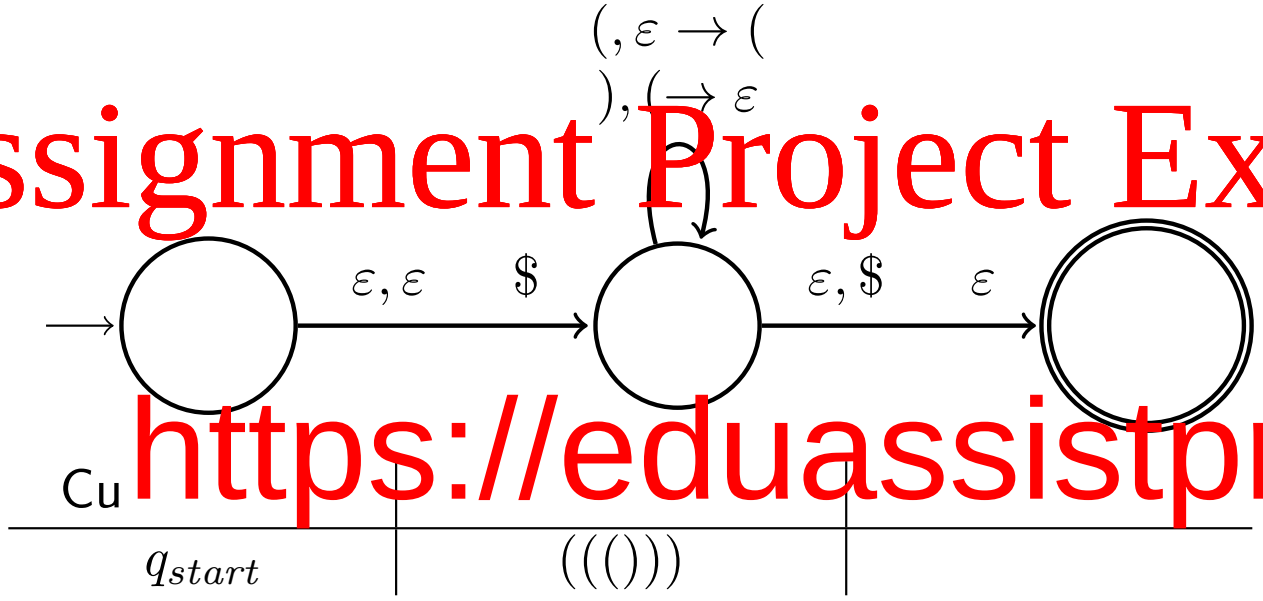
-
  - do not read anything from the input
  - do not pop anything from the stack
  - push c onto the stack

- $\varepsilon, b \to \varepsilon$
  - do not read anything from the input
  - pop b from the top of the stack
  - do not push anything onto the stack

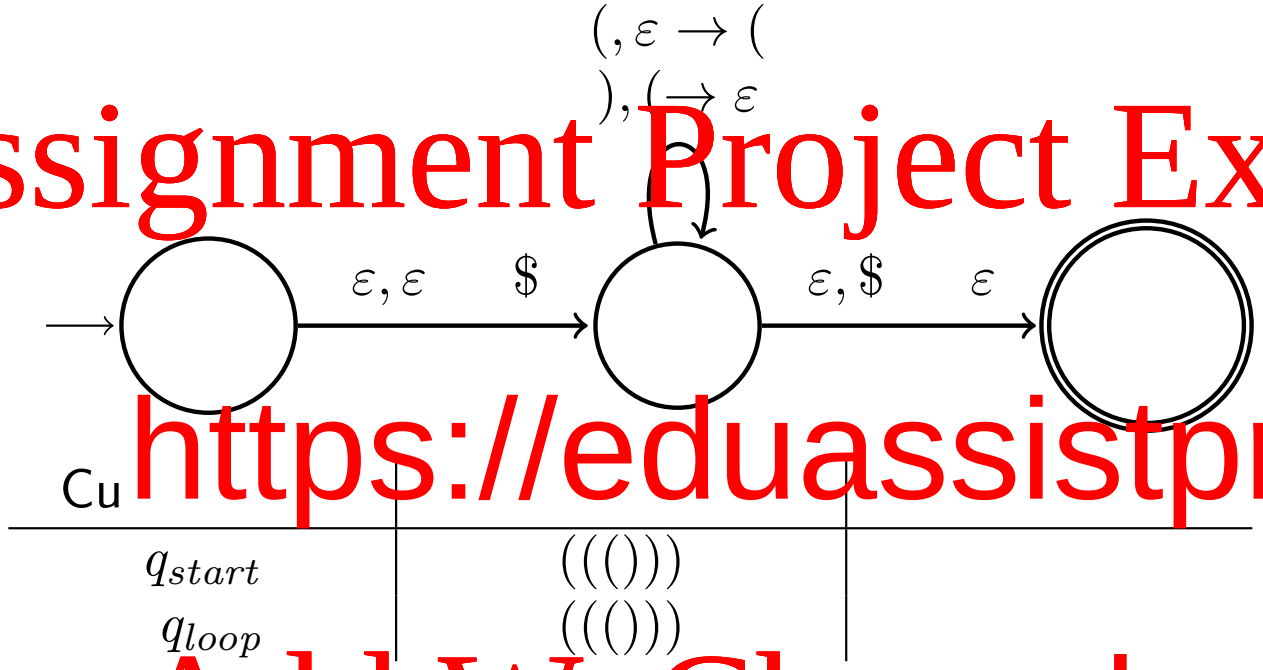
Parsing ((()))



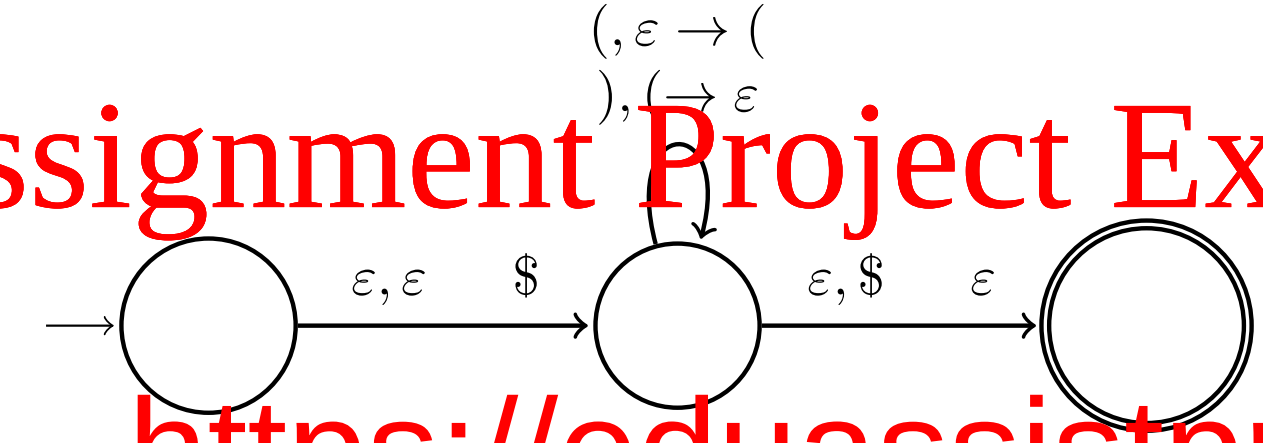
Parsing ((()))



Parsing ((()))



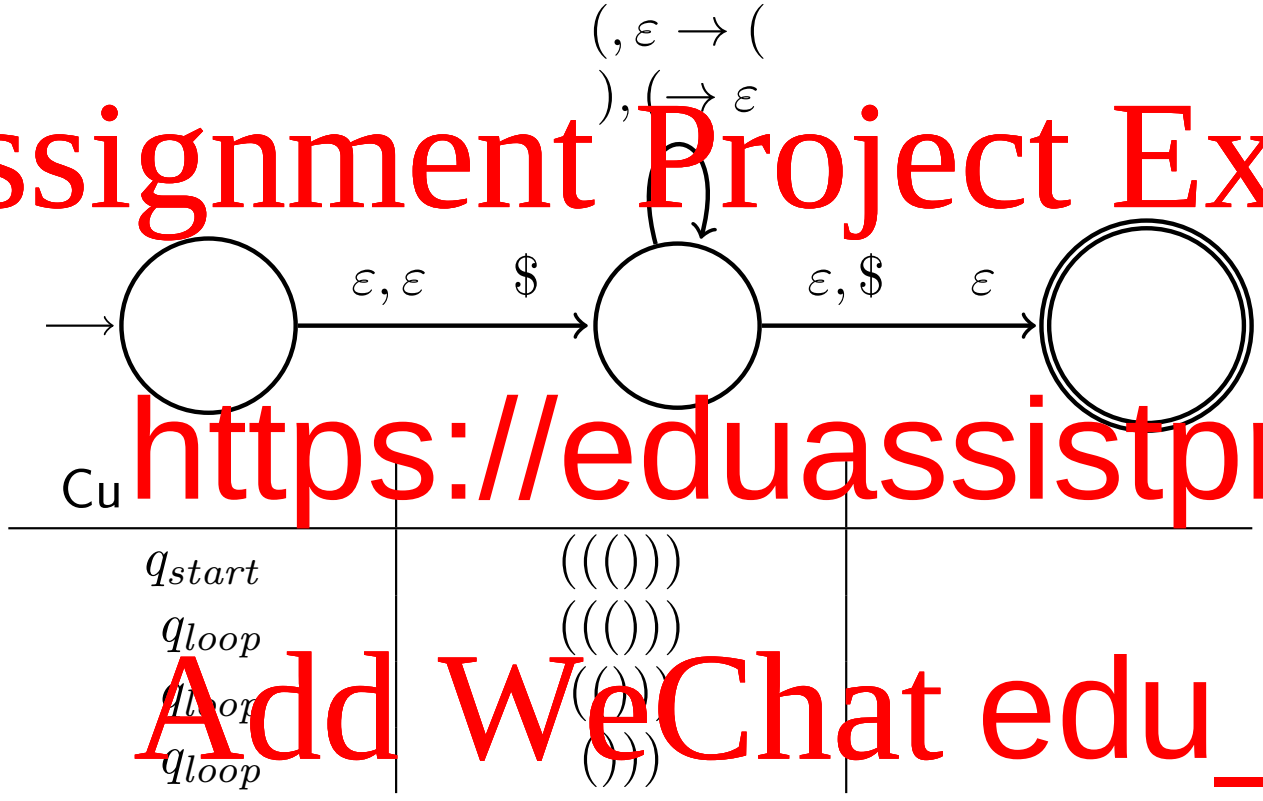
Parsing ((()))



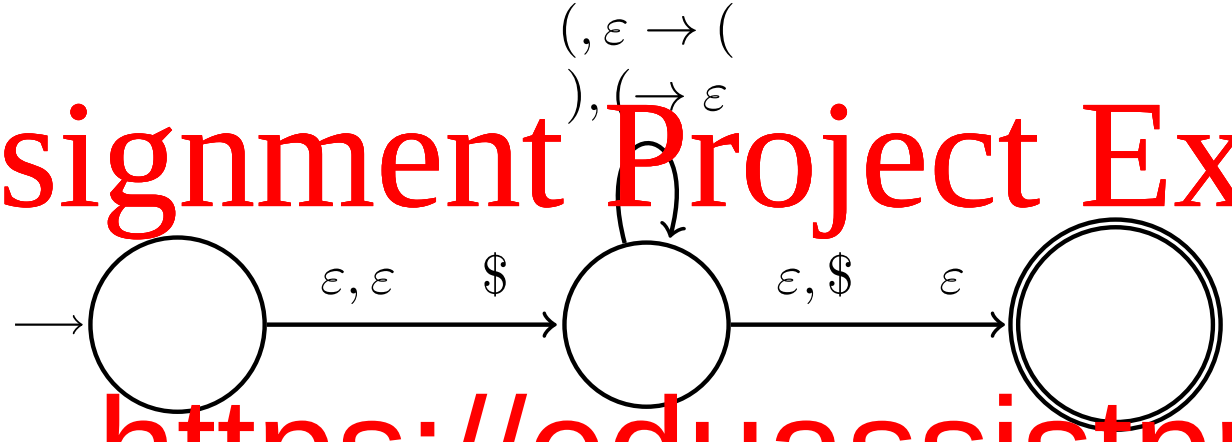
Cu

q_{start}	((()))	
q_{loop}	((()))	
q_{loop}	(())	

Parsing ((()))



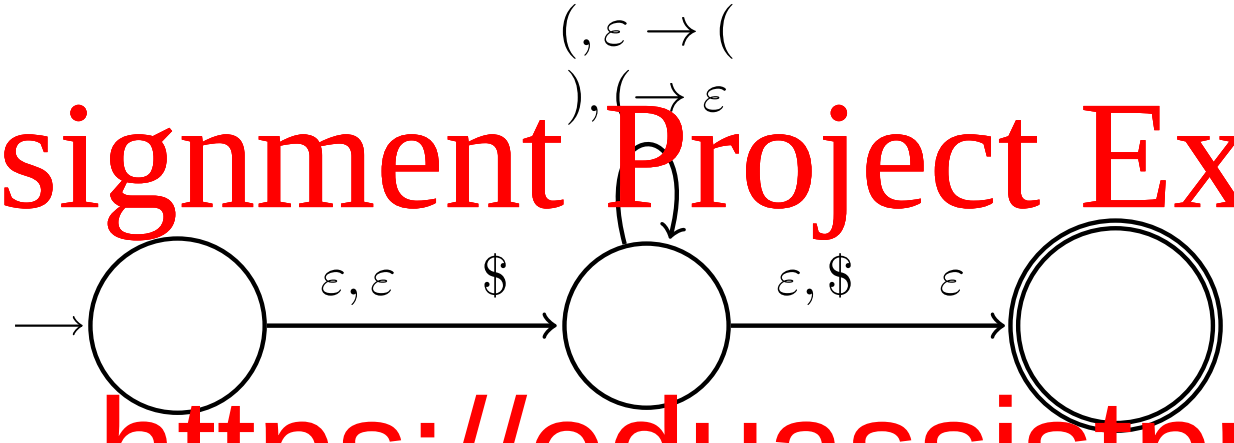
Parsing ((()))



Cu

q_{start}	((()))	
q_{loop}	((()))	
q_{loop}	(())	
q_{loop}	())	
q_{loop}	)))	(((\$

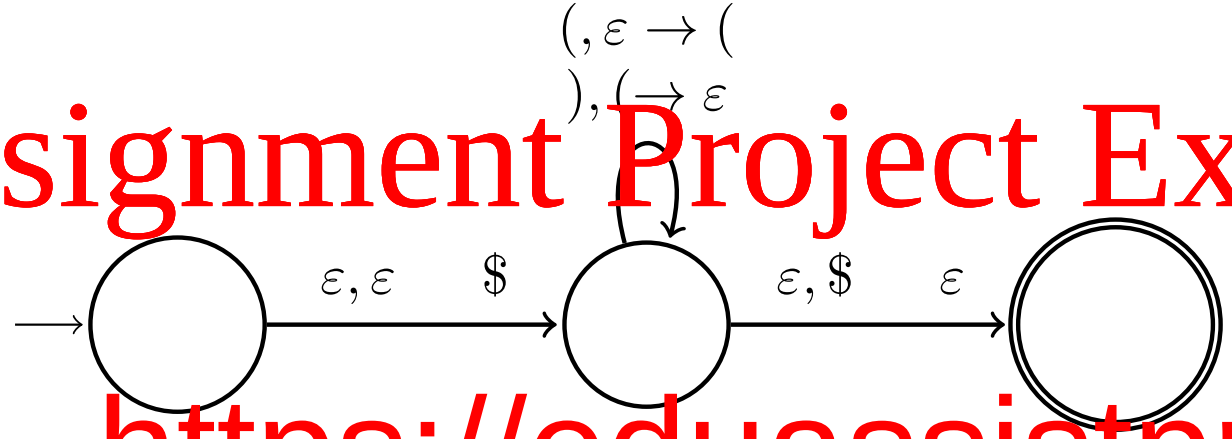
Parsing ((()))



Cu

q_{start}	((()))	
q_{loop}	((()))	
q_{loop}	(())	
q_{loop}	())	
q_{loop}	)))	(((\$
q_{loop}	))	((\$

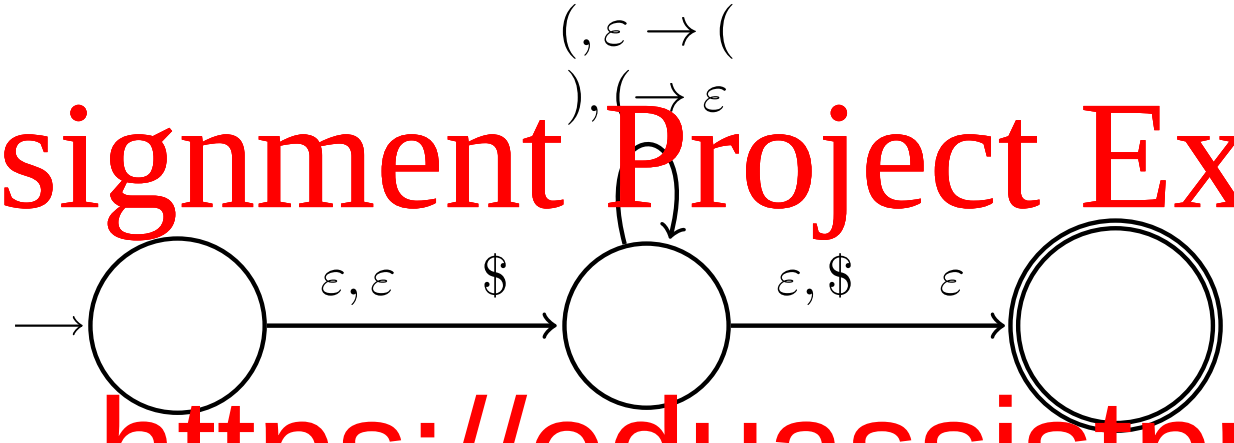
Parsing ((()))



Cu

q_{start}	((()))	
q_{loop}	((()))	
q_{loop}	(())	
q_{loop}	())	
q_{loop}	)))	(((\$
q_{loop}	))	((\$
q_{loop}	)	(\$

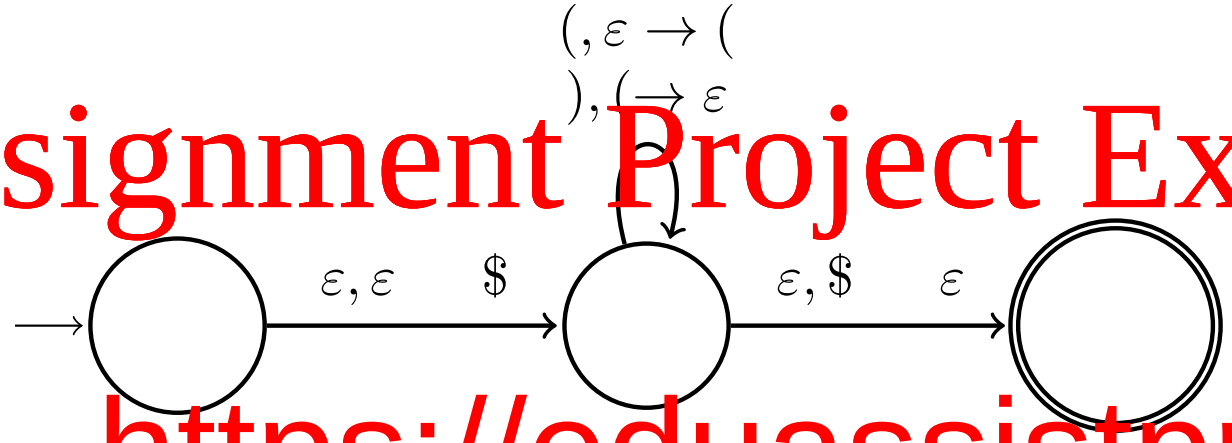
Parsing ((()))



Cu

q_{start}	((()))	
q_{loop}	((()))	
q_{loop}	((()))	
q_{loop}	(())	
q_{loop}	)))	(((\$
q_{loop}	))	((\$
q_{loop}	)	(\$
q_{loop}	ε	\$

Parsing ((()))



Cu		
q_{start}	((()))	
q_{loop}	((()))	
q_{loop}	((()))	
q_{loop}	((()))	
q_{loop}	)))	(((\$
q_{loop}	))	((\$
q_{loop}	)	(\$
q_{loop}	ε	\$
q_{accept}	ε	$\emptyset$

From CFG → PDA → Table Driven parsers

The interesting part of the PDA is the stack and the Q_{loop} state

The stack remembers the part of the string that is yet to be derived.

This c

- https://eduassistpro.github.i
- Output: which rule to use

Non-determinism in the table is a problem: how do which rule to use?

Try to construct a deterministic table whenever possible

- Not all CFG have an equivalent *deterministic* PDA

Parsing bcc with a table-driven parser

Remaining input	Stack
$bcc\$$	$S\$$

$S \to BC$

$S \to a$

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack
$bcc\$$	$S\$$
$bcc\$$	$BC\$$

$S \to BC$

$S \to a$

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack
$bcc\$$	$S\$$
$bcc\$$	$BC\$$

$S \to BC$

$S \to a$

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$
$cc\$$	$cC\$$	

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$
$cc\$$	$cC\$$	
$c\$$	$C\$$	

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$
$cc\$$	$cC\$$	
$c\$$	$C\$$	
$c\$$	$C\$$	

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$
$cc\$$	$cC\$$	
$c\$$	$C\$$	
$c\$$	$C\$$	
$\$$	$C\$$	

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	
		$S \to a$
$cc\$$	$cC\$$	
$c\$$	$C\$$	
$c\$$	$C\$$	
$\$$	$C\$$	
$\$$	$\$$	

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing bcc with a table-driven parser

Remaining input	Stack	
$bcc\$$	$S\$$	$S \to BC$
$bcc\$$	$BC\$$	$S \to a$
$cc\$$	$cC\$$	
$c\$$	$C\$$	
$c\$$	$C\$$	
$\$$	$C\$$	
$\$$	$\$$	
ACCEPTED		

S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Grammars are fundamental for constructing parsers.

Top-down parsing:

Grammars are fundamental for constructing parsers.

Top-down parsing: A parse tree is constructed from the root, proceeding downward towards the leaves

-
-
  - **L**eft-to-right, **L**eftmost derivation

Grammars are fundamental for constructing parsers.

Top-down parsing: A parse tree is constructed from the root, proceeding downward towards the leaves

-
-
  - **L**eft-to-right, **L**eftmost derivation

Parsers should be efficient, so determinism is important.

LL(1) parsing

LL(1) grammar

- L: Left to right scanning of input
-

Deterministic derivation by looking ahead

Using less lookahead symbols is usually more efficient

Lookahead symbols

Suppose we have a grammar with two rules for S:

$$S \to XY \mid YZ \qquad \text{(other rules not shown)}$$

How d

Suppose XY can only derive strings which sta Z
can only derive strings which start with

Then if we look ahead one symbol, we know which rul

- to derive abc we must choose $S \Rightarrow XY$
- to derive cab we must choose $S \Rightarrow YZ$

Table-driven LL(1) parsing

In a PDA: the stack contains the right hand side of the rules for a leftm

In a *descent table-driven parser*: Given th variable on top of the stack, the table specifies which rule to use.

Descent table-driven LL(1) parser


```
loop
    T = symbol on top of the stack

        else error
    else if P[T,c] = α is define
        pop T and push α onto th
                //(in reverse order)
    else error
endloop
```

Rules starting with non-terminals

$$S \to A \mid B$$

The productions for A and B *seem* LL
which rule to use when deriving from

We look at the possible symbols which can start stri
after $S \to A$, or after $S \to B$.

The set of symbols which could start any string derived from α is called the FIRST set of α

FIRST and FOLLOW sets

In order to fill in the entries of the table-drived parser, we need to compute some FIRST and FOLLOW sets.

FIR deriv production of G (i.e. the *right hand side*

$FOLLOW(V)$ is the set of all terminals which w the variable V at any stage of the derivation. Needed whenever V can derive ε.

Table construction: FIRST sets

If α is a string, then $FIRST(\alpha)$ is the set of terminals which can begin strings derived from α. Iff $\alpha \stackrel{+}{\Rightarrow} \varepsilon$ then $\varepsilon \in FIRST(\alpha)$.

Cons g of term

Table construction: FIRST sets

If α is a string, then $FIRST(\alpha)$ is the set of terminals which can begin strings derived from α. Iff $\alpha \stackrel{+}{\Rightarrow} \varepsilon$ then $\varepsilon \in FIRST(\alpha)$.

Cons g of term

1.

Table construction: FIRST sets

If α is a string, then $FIRST(\alpha)$ is the set of terminals which can begin strings derived from α. If $\alpha \stackrel{+}{\Rightarrow} \varepsilon$ then $\varepsilon \in FIRST(\alpha)$.

Cons g of term

1. $FIRST(a) = \{a\}$
2. $FIRST(a\alpha) = FIRST(a) = \{a\}$

Table construction: FIRST sets

If α is a string, then $FIRST(\alpha)$ is the set of terminals which can begin strings derived from α. If $\alpha \stackrel{*}{\Rightarrow} \varepsilon$ then $\varepsilon \in FIRST(\alpha)$.

Cons g of term

1. https://eduassistpro.github.i
2. $FIRST(a\alpha) = FIRST(a) = \{a\}$
3. If $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ then $FIRST(A) = FIRST(\alpha_1) \cup \ldots \cup$ Add WeChat edu_assist_pr

Table construction: FIRST sets

If α is a string, then $FIRST(\alpha)$ is the set of terminals which can begin strings derived from α. If $\alpha \Rightarrow^* \varepsilon$ then $\varepsilon \in FIRST(\alpha)$.

Cons g of term

1.
2. $FIRST(a\alpha) = FIRST(a) = \{a\}$
3. If $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ then $FIRST(A) = FIRST(\alpha_1) \cup \ldots \cup$
4. If $\alpha \neq \varepsilon$ then
   - If $\varepsilon \notin FIRST(A)$ then $FIRST(A\alpha) = FIRST(A)$
   - If $\varepsilon \in FIRST(A)$ then $FIRST(A\alpha) = FIRST(A) \setminus \{\varepsilon\} \cup FIRST(\alpha)$

Examples calculating FIRST sets

Examples calculating FIRST sets

$B \quad (B)B \quad \varepsilon$

F ule 2

$FIRST(B) =$

Examples calculating FIRST sets

$$B \quad (B)B \quad \varepsilon$$

F ule 2

$$FIRST(B) = FIRST(\text{“}(B)$$

$$=$$

Examples calculating FIRST sets

$$B \quad (B)B \quad \varepsilon$$

F ule 2

$$FIRST(B) = FIRST(\text{“}(B)$$

$$= FIRST(\text{“}(B)$$

$$=$$

Examples calculating FIRST sets

$$B \quad (B)B \quad \varepsilon$$

F ule 2

$$FIRST(B) = FIRST(\text{“}(B)$$
$$= FIRST(\text{“}(B)$$
$$= \{(\} \cup \{\varepsilon\}$$
$$= \{(, \varepsilon\}$$

Examples calculating FIRST sets

$$S \rightarrow BC \mid a$$
$$B \rightarrow bB \mid \varepsilon$$

We w
othe

$FIRST(\varepsilon) =$

Examples calculating FIRST sets

$$S \rightarrow BC \mid a$$
$$B \rightarrow bB \mid \varepsilon$$

We w
othe

$$FIRST(\varepsilon) = \{\varepsilon\}$$

$$FIRST(B) =$$

Examples calculating FIRST sets

$$S \rightarrow BC \mid a$$
$$B \rightarrow bB \mid \varepsilon$$

We w
othe

$$FIRST(\varepsilon) = \{\varepsilon\}$$

$$FIRST(B) = FIRST(bB) \cup FIR$$

$$=$$

Examples calculating FIRST sets

$$S \rightarrow BC \mid a$$
$$B \rightarrow bB \mid \varepsilon$$

We w
othe

$$FIRST(\varepsilon) = \{\varepsilon\}$$

$$FIRST(B) = FIRST(bB) \cup FIR$$
$$= \{b, \varepsilon\}$$

$$FIRST(C) =$$

Examples calculating FIRST sets

$$S \rightarrow BC \mid a$$
$$B \rightarrow bB \mid \varepsilon$$

We w
othe

$$FIRST(\varepsilon) = \{\varepsilon\}$$
$$FIRST(B) = FIRST(bB) \cup FIR$$
$$= \{b, \varepsilon\}$$
$$FIRST(C) = \{c, \varepsilon\}$$
$$FIRST(BC) =$$

similarly

Examples calculating FIRST sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

We w
othe

$$FIRST(\varepsilon) = \{\varepsilon\}$$
$$FIRST(B) = FIRST(bB) \cup FIR$$
$$= \{b, \varepsilon\}$$
$$FIRST(C) = \{c, \varepsilon\} \quad \text{similarly}$$
$$FIRST(BC) = FIRST(B) \setminus \{\varepsilon\} \cup FIRST(C) \quad \text{rule 4b}$$
$$=$$

Examples calculating FIRST sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

We w
othe

$$FIRST(\varepsilon) = \{\varepsilon\}$$
$$FIRST(B) = FIRST(bB) \cup FIR$$
$$= \{b, \varepsilon\}$$
$$FIRST(C) = \{c, \varepsilon\} \quad \text{similarly}$$
$$FIRST(BC) = FIRST(B) \setminus \{\varepsilon\} \cup FIRST(C) \quad \text{rule 4b}$$
$$= \{b\} \cup \{c, \varepsilon\}$$
$$= \{b, c, \varepsilon\}$$

When rules can yield ε

Consider the grammar

$$S \to AB$$
$$A \to aA \mid \varepsilon$$

A ca

Supp bc

$S \Rightarrow AB$. How does the parser know which produ

next, since b is not in $FIRST(A)$?

When rules can yield ε

Consider the grammar

$$S \to AB$$
$$A \to aA \mid \varepsilon$$

A ca

bc

Supp

$S \Rightarrow AB$. How does the parser know which produ

next, since b is not in $FIRST(A)$?

b is not in $FIRST(aA)$ or $FIRST(\varepsilon)$, b

using $A \to \varepsilon$ will give us $AB \Rightarrow B \Rightarrow bB \Rightarrow bc$

When rules can yield ε

Consider the grammar

$$S \to AB$$
$$A \to aA \mid \varepsilon$$

A ca bc

Supp

$S \Rightarrow AB$. How does the parser know which produ

next, since b is not in $FIRST(A)$?

b is not in $FIRST(aA)$ or $FIRST(\varepsilon)$, b

using $A \to \varepsilon$ will give us $AB \Rightarrow B \Rightarrow bB \Rightarrow bc$

When a variable can derive ε, we need to look at the terminal symbols which can begin strings which could FOLLOW that variable in an derivation. These are called the FOLLOW sets.

Table construction: FOLLOW sets

Definition:

If A is a variable, then $FOLLOW(A)$ is the set of terminals which can be derived from any string which can appear immediately to the right of A in some stage of the derivation.

Cons

Whe s:

Table construction: FOLLOW sets

Definition:

If A is a variable, then $FOLLOW(A)$ is the set of terminals which can be derived from any string which can appear immediately to the right of A in some stage of the derivation.

Cons

Whe s:

1.

Table construction: FOLLOW sets

Definition:

If A is a variable, then $FOLLOW(A)$ is the set of terminals which can be derived from any string which can appear immediately to the right of A in some stage of the derivation.

Cons

Whe s:

1.

2. If $X \to \alpha Y$ then $FOLLOW(X)$
   (i.e. anything that can follow X can follo

Table construction: FOLLOW sets

Definition:

If A is a variable, then $FOLLOW(A)$ is the set of terminals which can be derived from any string which can appear immediately to the right of A in some stage of the derivation.

Cons

Whe s:

1.

2. If $X \to \alpha Y$ then $FOLLOW(X)$
   (i.e. anything that can follow X can follo
3. If $X \to \alpha Y \beta$ then $FIRST(\beta) \setminus \{$
   (i.e. any terminal which can start β can follow Y)

Table construction: FOLLOW sets

Definition:

If A is a variable, then $FOLLOW(A)$ is the set of terminals which can be derived from any string which can appear immediately to the right of A in some stage of the derivation.

Cons

Whe s:

1.
2. If $X \to \alpha Y$ then $FOLLOW(X)$
   (i.e. anything that can follow X can follo
3. If $X \to \alpha Y \beta$ then $FIRST(\beta) \setminus \{$
   (i.e. any terminal which can start β can follow Y)
4. If $X \to \alpha Y \beta$, $\varepsilon \in FIRST(\beta)$ then
   $FOLLOW(X) \subset FOLLOW(Y)$
   (i.e. if X can derive a string ending in Y, anything that follows X can follow Y)

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

$FOLLOW(C) =$

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

$$FOLLOW(C) = FOLLOW(S)$$

Examples calculating FOLLOW sets

$$S \rightarrow BC \mid a$$
$$B \rightarrow bB \mid \varepsilon$$

$FOLLOW(C) = FOLLOW(S)$

$= \{\$\}$

$FOLLOW(B) =$

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

$FOLLOW(C) = FOLLOW(S)$

$= \{\$\}$

$FOLLOW(B) = FIRST(C) \setminus \{\varepsilon$

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

$FOLLOW(C) = FOLLOW(S)$

$= \{\$\}$

$FOLLOW(B) = FIRST(C) \setminus \{\varepsilon$

$\cup\ FOLLOW(C)$ rule 4

$=$

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

$$FOLLOW(C) = FOLLOW(S)$$
$$= \{\$\}$$

$$FOLLOW(B) = FIRST(C) \setminus \{\varepsilon$$
$$\cup\, FOLLOW(C) \qquad \text{rule 4}$$
$$= \{c\} \cup \{\$\}$$
$$=$$

Examples calculating FOLLOW sets

$$S \to BC \mid a$$
$$B \to bB \mid \varepsilon$$

$$FOLLOW(C) = FOLLOW(S)$$
$$= \{\$\}$$

$$FOLLOW(B) = FIRST(C) \setminus \{\varepsilon$$
$$\cup\, FOLLOW(C) \qquad \text{rule 4}$$
$$= \{c\} \cup \{\$\}$$
$$= \{c, \$\}$$

Examples calculating FOLLOW sets

Examples calculating FOLLOW sets

$\cup\, FIRST(\text{“})$

Examples calculating FOLLOW sets

$$\cup\, FIRST(\text{“})$$
$$= \{\$\} \cup \{)\}$$
$$= \{\$,)\}$$

Constructing the parse table

Rows: one for each variable of the grammar

Colu

strin

Steps to fill the table T:

1. If there is a rule $R \to \alpha$ with $a \in$ in $T[R, a]$
2. If there is a rule $R \to \alpha$ with $\varepsilon \in FIRST(\alpha)$ and $a \in FOLLOW(R)$, then put α in $T[R, a]$

Example

$F \to \alpha$	$FIRST(\alpha)$	$FOLLOW(F)$ if $\varepsilon \in FIRST(\alpha)$
$S \to BC$		
$S \to a$		
$B \quad bB$		
B		
C		
C		

Parse table:

	a	b	c	\$
S				
B				
C				

Example

$F \to \alpha$	$FIRST(\alpha)$	$FOLLOW(F)$ if $\varepsilon \in FIRST(\alpha)$
$S \to BC$	$FIRST(BC) = \{b, c, a\}$	
$S \to a$	$FIRST(a) = \{a\}$	
$B \quad bB$	$FIRST(bB) = \quad b$	
B		
C		
C		

Parse table:

	a	b	c	\$
S				
B				
C				

Example

$F \to \alpha$	$FIRST(\alpha)$	$FOLLOW(F)$ if $\varepsilon \in FIRST(\alpha)$
$S \to BC$	$FIRST(BC) = \{b, c, a\}$	$FOLLOW(S) = \{\$\}$
$S \to a$	$FIRST(a) = \{a\}$	
$B \quad bB$	$FIRST(bB) = \quad b$	
B		$c, \$\}$
C		
C		$\$\}$

Parse table:

	a	b	c	$\$$
S				
B				
C				

Example

$F \to \alpha$	$FIRST(\alpha)$	$FOLLOW(F)$ if $\varepsilon \in FIRST(\alpha)$
$S \to BC$	$FIRST(BC) = \{b, c, a\}$	$FOLLOW(S) = \{\$\}$
$S \to a$	$FIRST(a) = \{a\}$	
$B \quad bB$	$FIRST(bB) = \quad b$	
B		
C		$c, \$\}$
C		$\$\}$

Parse table:

	a	b	c	$\$$
S	a	BC	BC	BC
B		bB	ε	ε
C			cC	ε

Parsing with a parsing table

1. Append the end of input marker $ to the input and push $ to the stack
2. Put the *start variable* on the stack and scan the *first token*
3.

rent

3.2 else *if the top of the stack is a terminal symb* t to a. If they match, pop the stack and scan th Otherwise reject the input.

3.3 else *if the top of the stack and the token are b* accept the input (the stack is empty and we have used all the input.)

3.4 else reject the input (the stack is empty but there is unread input.)

Parsing bcc

Remaining input	Stack

				$\$$
				BC
		bB	ε	ε
				ε

Parsing bcc

Remaining input	Stack
$bcc\$$	$C\$$
$bcc\$$	$BC\$$
bcc	
$cc\$$	
$cc\$$	
$cc\$$	$cC\$$
$c\$$	$C\$$
$c\$$	$cC\$$
$\$$	$C\$$
$\$$	$\$$
ACCEPTED	

				$\$$
				BC
		bB	ε	ε
				ε

Parsing *bcbc*

Remaining input	Stack

Parsing $bcbc$

Remaining input	Stack
$bcbc\$$	$S\$$
bcb	
bcb	
cbc	
$cbc\$$	$C\$$
$cbc\$$	$cC\$$
$bc\$$	$C\$$
REJECTED	

LL(k) parsing

LL(k) grammar

- L: Left to right scanning of input
-
-

Deterministic derivation by looking ahead

Using less lookahead symbols is usually more efficient

LL(1) grammar: example

$$L = \{a^n bc^n \mid n \geq 0\}$$

$$S \to aSc \mid b$$

This g
rule le

Each step of the derivation can be deterministical
examining the current symbol (1 lookahead sym

- If the remaining input starts with
- If the remaining input starts with b, use $S \to b$
- (If it starts with anything else, no derivation exists)

LL(1) grammar: example

$L = \{a^n bc^n \mid n \geq 0\}$

$S \quad aSc \quad b$

Deri

$S \Rightarrow aSc$ Use $S \to aSc$ beca

$\Rightarrow aaScc$ Use $S \to aSc$ bec

$\Rightarrow aabcc$ Use $S \to b$ because bcc begins with b

LL(2) grammar: example

$$L = \{a^m b^n c \mid n \geq 0\}$$

$$S \to AB$$

Each s

examining the current symbol and the next one (2 lo
symbols). e.g. if we need to replace a variable

- If the remaining input starts with
- If the remaining input starts with ab or ac, use $A \to a$
- (If it starts with anything else, no derivation exists)

LL(2) grammar: example

$S \to AB$

$A \to aA \mid a$

Deri

$S \Rightarrow AB$

$\Rightarrow aAB$ Use $A \to aA$ beca

$\Rightarrow aaB$ Use $A \to a$ bec

$\Rightarrow aabB$ Use $B \to bB$ because bbc begins with b

$\Rightarrow aabbB$ Use $B \to bB$ because bc begins with b

$\Rightarrow aabbc$ Use $B \to c$ because c begins with c

non-LL(k) grammar: example

$$S \to aS \mid T$$

Not L
prod $S \to aS$ $S \to T$ gs
starting with a)

Not LL(2): the input aa is not enough eith

Not LL(k): we need to know how many b's there are. For any k we can choose $n > k$ such that $a^n b^n \in L(G)$ but we would need to lookahead $2n > k$ symbols to decide which rule to use first.

Identify if a grammar is LL(1)

Recall that a grammar is LL(1) if it is sufficient to look at the next symbol to determine which rule to follow next.

i.e. If ev

then t

More formally, a grammar is LL(1) iff for every vari

- Let $A \to \alpha_1 \mid \ldots \mid \alpha_n$ be the producti
- Let $X_i = FIRST(\alpha_i)$ if $\varepsilon \notin FIR$
- Let $X_i = FIRST(\alpha_i) \cup FOLLOW(A)$ otherwise
- Then $X_i \cap X_j = \emptyset$ for all $i \neq j$

Transforming non-LL(1) grammars

When a grammar is not LL(1) we try to find an equivalent grammar which is, by applying the following techniques:

- Left factoring
-

Transforming non-LL(1) grammars

When a grammar is not LL(1) we try to find an equivalent grammar which is, by applying the following techniques:

- Left factoring
-

Reca

language

Transforming non-LL(1) grammars

When a grammar is not LL(1) we try to find an equivalent grammar which is, by applying the following techniques:

- Left factoring
-

Reca
language

Such a grammar does not always exist. For exampl
grammar exists for the language
$\{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$

Left factoring: why?

Consider the grammar fragment $S \rightarrow abcC \mid abdD$

- The two rules both start with the same prefix ab
-
-

Left factoring: why?

Consider the grammar fragment $S \to abcC \mid abdD$

- The two rules both start with the same prefix ab
-
-

We ca ab

grammar, where B is a new variable:

$$S \to abB$$

$$B \to cC \mid dD$$

This grammar fragment is equivalent, but is LL(1)

Left Factoring example

Ambiguous grammar:

After factorisation

$S \quad abB$

	a	b	c	d
S	abC abD			
...	...	...	...	...

S				
B				
...	...	...	...	...

Left factoring: definition

If a string w appears on the left of several rules for a variable A:

$$A \quad wX_1 \quad \ldots \quad wX_1$$

The

$$A \to wA'$$
$$A' \to X_1 \ldots$$

Any other rules produced by A are unaffected.

Recursion (from last week)

If a variable X can generate a string containing X itself, then it is recursive

- $\stackrel{+}{\Rightarrow} X\beta$
- $\stackrel{+}{\Rightarrow} \alpha X$
-

A grammar is recursive if any of its variables is recursi

A grammar for an infinite language must contain at least one recursive variable

Eliminate left recursion: why?

Consider this simple grammar:

$$A \rightarrow c$$

$$FIRST(Ab) =$$

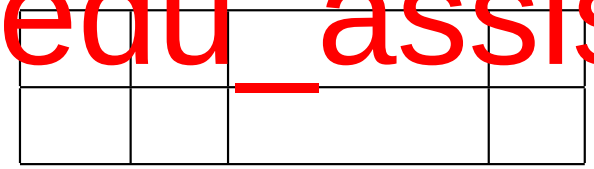
Eliminate left recursion: why?

Consider this simple grammar:

$$A \rightarrow c$$

$$FIRST(Ab) =$$

If we try to construct the parse table:



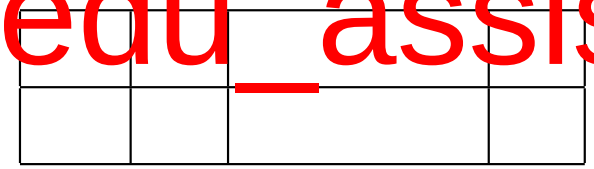
Eliminate left recursion: why?

Consider this simple grammar:

$$A \rightarrow c$$

$$FIRST(Ab) =$$

If we try to construct the parse table:



The base cases for the recursion must have FIRST sets which intersect with the left recursive rule!

Eliminating left recursion

Let α, β be arbitrary strings of terminals and/or variables.

Let A be a variable, and R a new variable

If A

It can be replaced with:

$$A \to \beta R$$
$$R \to \alpha R \mid$$

Eliminating left recursion

Let α, β be arbitrary strings of terminals and/or variables.

Let A be a variable, and R a new variable

If A

It can be replaced with:

$$A \to \beta R$$
$$R \to \alpha R \mid$$

What do the parse trees look like for $\beta\alpha\alpha\alpha$ using the original and transformed grammar?

Simple example

Simple example

Simple example

$$A \to c$$

$$A \to cR$$

$$R \to bR \mid$$

Simple example

$$A \to c$$

$$A \to cR$$

$$R \to bR \mid$$

	b	c	\$
A		cR	
R	bR		ε

Complex example

$$E \rightarrow E + T$$

$$E \quad E \quad T$$

Then $\alpha =$

Complex example

$$E \rightarrow E + T$$

$$E \quad E \quad T$$

Then $\alpha = +T \mid -T$, $\beta =$

Complex example

$$E \rightarrow E + T$$

$$E \quad E \quad T$$

Then $\alpha = +T \mid -T$, $\beta = T$, which give

$$E \rightarrow TR$$

$$R \rightarrow +TR \mid -TR \mid \varepsilon$$

$$T \rightarrow a \mid b \mid c$$

Complex example

$E \mapsto TR$

$R \quad +TR \quad TR \quad \varepsilon$

Complex example

$$E \mapsto TR$$

$$R \quad +TR \quad TR \quad \varepsilon$$

FIR

$FIRST(+TR) =$

Complex example

$$E \mapsto TR$$

$$R \quad +TR \quad TR \quad \varepsilon$$

FIR

$FIRST(+TR) = \{+\}$

$FIRST(-TR) =$

Complex example

$$E \mapsto TR$$

$$R \quad +TR \quad TR \quad \varepsilon$$

FIR

$FIRST(+TR) = \{+\}$

$FIRST(-TR) = \{-\}$

Because $\varepsilon \in FIRST(\varepsilon)$, we calculate

Complex example

$$E \mapsto TR$$

$$R \quad +TR \quad TR \quad \varepsilon$$

FIR

$FIRST(+TR) = \{+\}$

$FIRST(-TR) = \{-\}$

Because $\varepsilon \in FIRST(\varepsilon)$, we calculate

Complex example

$$E \mapsto TR$$

$$R \quad +TR \quad TR \quad \varepsilon$$

FIR

$FIRST(+TR) = \{+\}$

$FIRST(-TR) = \{-\}$

Because $\varepsilon \in FIRST(\varepsilon)$, we calculate

	a	b	c	$+$	$-$	
E	TR	TR	TR			
R				$+TR$	$-TR$	ε
T	a	b	c			

Proving that a grammar is *not* LL(1)

It is sufficient to show any one of the following:

-
- The grammar needs left factoring

- The first sets of the production rules for a varia
  disjoint

Typical exam question

Consider the grammar G:

$$S \quad ST \quad ab$$

Show that the grammar G is not LL(1)

Transform G to obtain a grammar G'

Give the LL(1) parse table for G'

Push-down Automata

- "NFA with a stack"
- CFG to PDA construction method
- Recognises the set of CFL
- Non-deterministic PDA are *more powerful* than D-PDA

Pars

- Not all CFG are LL(k)
- $FIRST(\alpha)$: set of terminals (or ε from α
- $FOLLOW(X)$: the set of terminals (or \$) w strings following X in a derivation
- How to build a parse table for an LL(1) CFG
- How to parse a string using an LL(1) parse table

Announcements

- Assignment 2
  - Will be on this topic (parsing)
  - Due Sunday 14th October (end of week 10)

-

- Alternative tutorials for COMP2022
  - Tuesday 3pm in ABS 3100
  - Wednesday 9am in ABS 3090
    - Select a session here:
      `https://edstem.org/courses/2892/sway/`
- COMP2922: normal tutorial covered in advanced session