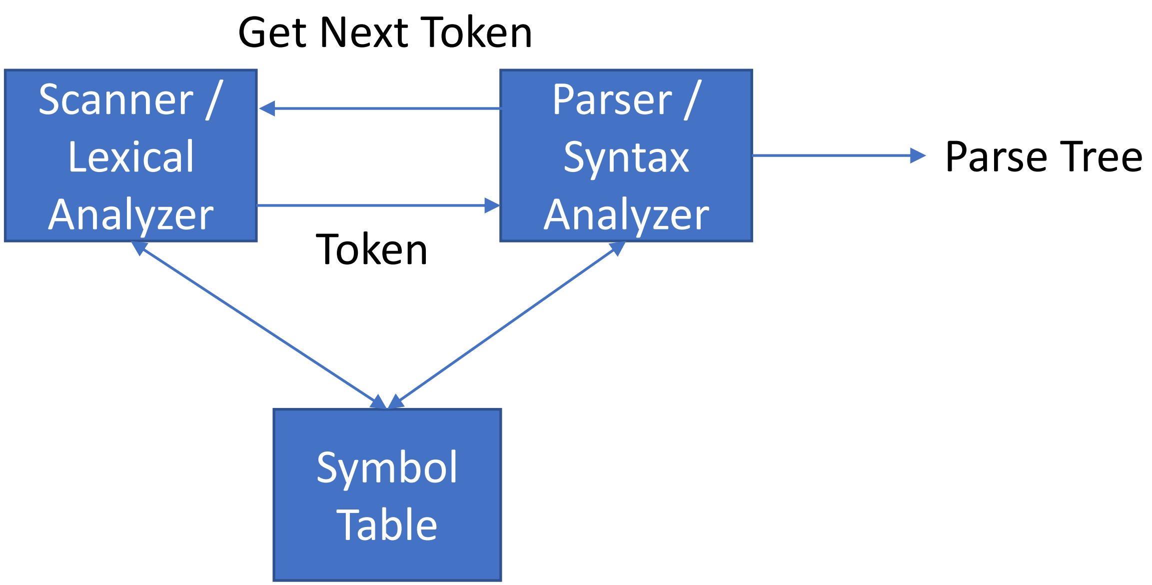
CS406: Compilers

Spring 2022

Week 4: Parsers

Demo

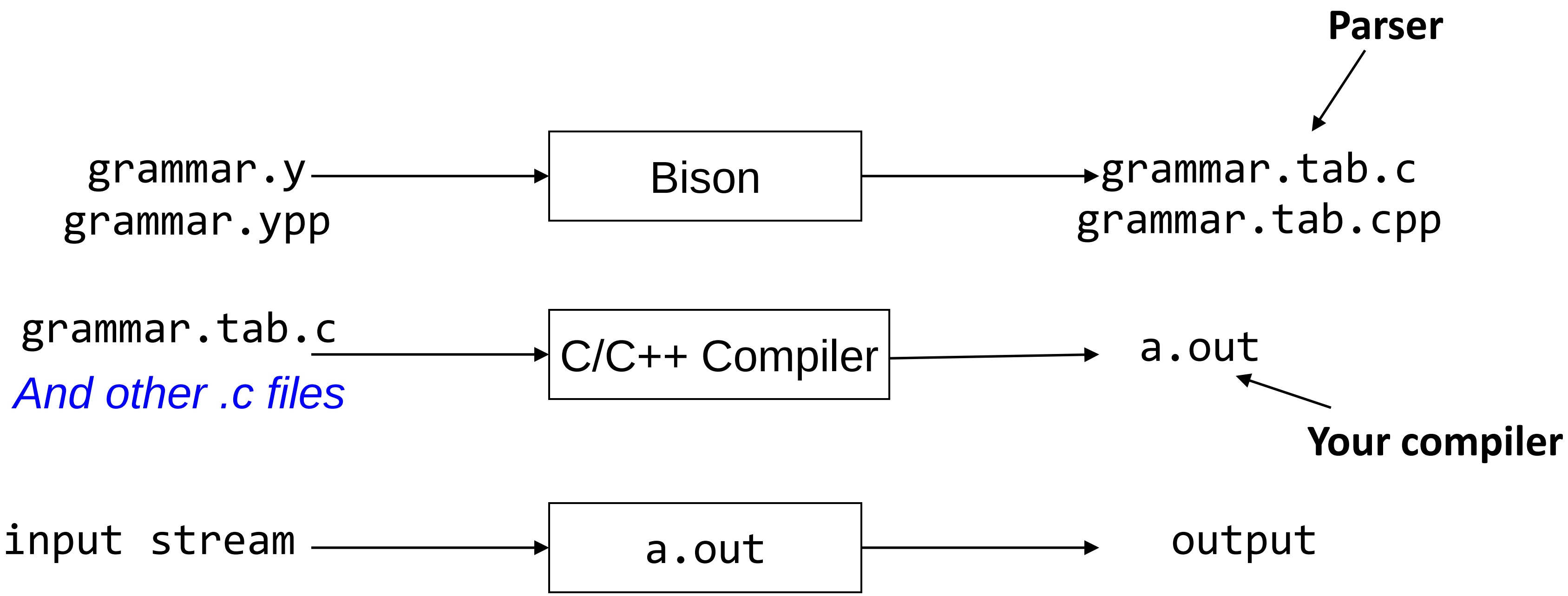
- Parser (in an implementation of a compiler)



Bison (YACC)

- Specify the grammar
- Write a lexical analyzer to process input programs and pass the tokens to parser
- Call `yyparse()` from `main`
- Write error-handlers (what happens when the compiler encounters invalid programs?)

Bison (YACC)


Parser
grammar.y
grammar.ypp
Bison
grammar.tab.c
grammar.tab.cpp
grammar.tab.c
And other .c files
C/C++ Compiler
a.out
Your compiler
input stream
a.out
output

Bison (YACC) – Input Format

```
%{
Prologue
%}
Bison declarations
%%
Grammar rules
%%
Epilogue
```

Bison (YACC) – Grammar Rules

```
%{
Prologue
%}
Bison declarations
%%
E: E PLUS E {}
 | INTEGER_LITERAL {}
 ;
%%
Epilogue
```

Bison (YACC) - Prologue

```
%{
Prologue
%}
%token PLUS INTEGER_LITERAL
%left PLUS
%%
E: E PLUS E {}
 | INTEGER_LITERAL {}
 ;
%%
Epilogue
```

Bison (YACC) - Actions

```
%{
Prologue
%}
%token PLUS INTEGER_LITERAL
%left PLUS
%%
E: E PLUS E { $$ = $1 + $3; }
 | INTEGER_LITERAL { $$ = $1; }
 ;
%%
Epilogue
```

Legal C/C++ code

Bison (YACC) – Semantic Values

```
%{
Prologue
%}
%token PLUS INTEGER_LITERAL
%left PLUS
%%           $2      $3
         $1
E: E PLUS E { $$ = $1 + $3; }
 | INTEGER_LITERAL { $$ = $1; }
 ;
%%
Epilogue
```

Bison (YACC) – Helper Functions

```
%{
int yylex();
void yyerror(char *s);
%}
%token PLUS INTEGER_LITERAL
%left PLUS
%%
E: E PLUS E { $$ = $1 + $3; }
 | INTEGER_LITERAL { $$ = $1; }
 ;
%%
Epilogue
```

Bison (YACC) – Helper Functions

```
%{
#include<stdlib.h>
#include<stdio.h>
int yylex();
void yyerror(char const *s);
%}
%token PLUS INTEGER_LITERAL
%left PLUS
%%
E: E PLUS E { $$ = $1 + $3; }
 | INTEGER_LITERAL { $$ = $1; };
%%
void yyerror(char const* s) {
      fprintf(stderr,”%s\n”,s);
      exit(1);
}
```

Bison (YACC) – Integrating

- Recall that terminals are tokens
- Lexer produces tokens
  - How do the parser and lexer have a common understanding of tokens?
  - How should the Lexer return tokens?

```
//grammar.y file
…
%token PLUS INTEGER_LITERAL
%%
E: E PLUS E { $$ = $1 + $3; }
 | INTEGER_LITERAL { $$ = $1; };
%%
…
```

```
bison -d grammar.y
```

grammar.tab.h

```
//scanner.l file
#include"grammar.tab.h"
extern YYSTYPE yylval
%%
\+        {return PLUS;}
[0-9]+    { yylval=atoi(yytext);
          return INTEGER_LITERAL;}
.|\n      {}
%%
…
```

Bison(YACC) - More..

- %union
- %define
- error

- [Reference: Top (Bison 3.8.1) (gnu.org)]

Top-down Parsing

- Idea: we know sentence has to start with initial symbol
- Build up partial derivations by *predicting* what rules are used to expand non-terminals
  - Often called *predictive parsers*
- If partial derivation has terminal characters, *match* them from the input stream

Top-down Parsing

- Also called recursive-descent parsing
- Equivalent to finding the left-derivation for an input string
  - Recall: expand the leftmost non-terminal in a parse tree
  - Expand the parse tree in pre-order i.e., identify parent nodes before children

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

Step	Input string	Parse tree
1	cad	S

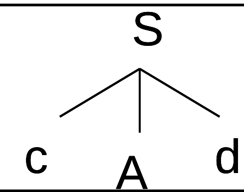
String: **cad**

Start with S

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

Step	Input string	Parse tree
1	cad	S
2	cad	

String: **cad**

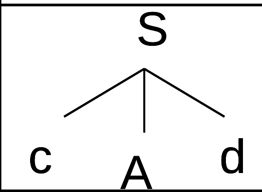
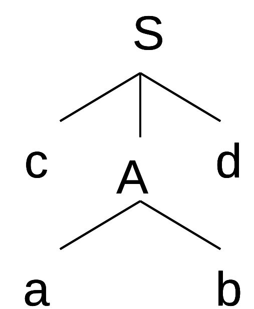
Predict rule 1

Top-down Parsing

: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

String: **cad**

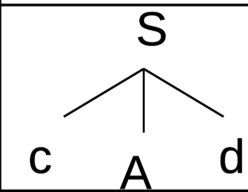
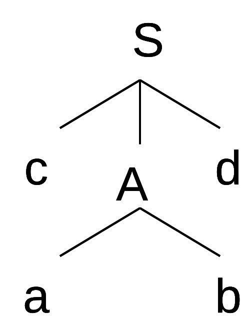
Step	Input string	Parse tree
1	cad	S
2	cad	
3	cad	

Predict rule 2

Top-down Parsing

↑: next symbol to be read

```
1:  S  ->  cAd
2:  A  ->  ab
3:         |  a
```

Step	Input string	Parse tree
1	cad	S
2	cad	
3	cad	

String: **cad**

No more non terminals!

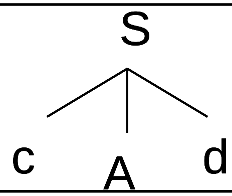
String doesn't match.

Backtrack.

Top-down Parsing

: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:     | a
```

Step	Input string	Parse tree
1	cad	S
2	cad	

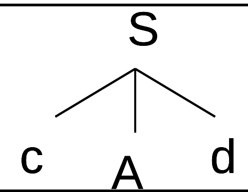
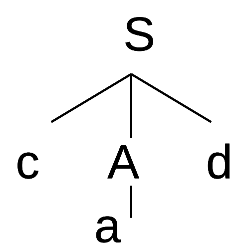
String: **cad**

Top-down Parsing

: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

String: **cad**

Step	Input string	Parse tree
1	cad	S
2	cad	
4	cad	

Predict rule 3

Top-down Parsing – Table-driven Approach

```
1: S -> F
2: S -> (S + F)
3: F -> a
```

string: (a+a)

string': (a+a)$

	(	)	a	+	$
S	2	-	1	-	-
F	-	-	3	-	-

Assume that the table is given.

Top-down Parsing – Table-driven Approach

```
1: S -> F
2: S -> (S + F)
3: F -> a
```

string: (a+a)

string': (a+a)$

	(	)	a	+	$
S	2	-	1	-	-
F	-	-	3	-	-

Assume that the table is given.

- Table-driven (Parse Table) approach doesn't require backtracking

But how do we construct such a table?

Important Concepts: First Sets and Follow Sets

First and follow sets

- First(α): the set of terminals (and/or λ) that begin all strings that can be derived from α
    - First(A) = {x, y, λ}
    - First(xaA) = {x}
    - First (AB) = {x, y, b}
- Follow(A): the set of terminals (and/or $, but no λs) that can appear immediately after A in some partial derivation
    - Follow(A) = {b}

$S \rightarrow A\ B\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow \lambda$

$B \rightarrow b$

Slide courtesy: Milind Kulkarni

First and follow sets

- $\text{First}(\alpha) = \{a \in V_t \mid \alpha \Rightarrow^* a\beta\} \cup \{\lambda \mid \text{if } \alpha \Rightarrow^* \lambda\}$
- $\text{Follow}(A) = \{a \in V_t \mid S \Rightarrow^+ \ldots Aa \ldots\} \cup \{\$ \mid \text{if } S \Rightarrow^+ \ldots A\ \$\}$

S:	start symbol
a:	a terminal symbol
A:	a non-terminal symbol
α,β:	a string composed of terminals and non-terminals (typically, α is the RHS of a production

$\Rightarrow$:	derived in 1 step
$\Rightarrow^*$:	derived in 0 or more steps
$\Rightarrow^+$:	derived in 1 or more steps

Computing first sets

- Terminal: First(a) = {a}
- Non-terminal: First(A)
  - Look at all productions for A

    $A \rightarrow X_1X_2 \ldots X_k$
  - $\text{First}(A) \supseteq (\text{First}(X_1) - \lambda)$
  - If $\lambda \in \text{First}(X_1)$, $\text{First}(A) \supseteq (\text{First}(X_2) - \lambda)$
  - If λ is in $\text{First}(X_i)$ for all i, then $\lambda \in \text{First}(A)$
- Computing First(α): similar procedure to computing First(A)

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

special "end of input" symbol

- A sentence in the grammar:

  x a c c $

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: S

A simple example

S → A B c $

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: A B c $

Predict rule

A simple example

$S \rightarrow A\ B\ c\ \$$

Choose based on *first set* of rules

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a A B c $

Predict rule *based on next token*

Slide courtesy: Milind Kulkarni

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  $x\ a\ c\ c\ \$$

Current derivation: $x\ a\ A\ B\ c\ \$$

Match token

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a A B c $

Match token

A simple example

$S \rightarrow A\ B\ c\ \$$

Choose based on *first set* of rules

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a c B c $

Predict rule *based on next token*

Slide courtesy: Milind Kulkarni

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  $x\ a\ c\ c\ \$$

Current derivation: $x\ a\ c\ B\ c\ \$$

Match token

A simple example

$\boxed{S \rightarrow A\ B\ c\ \$}$

$A \rightarrow x\ a\ A$

Choose based on *follow set*

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$\boxed{B \rightarrow b}$

$\boxed{B \rightarrow \lambda}$

- A sentence in the grammar:

  x a c c $

Current derivation: x a c λ c $

Predict rule *based on next token*

Slide courtesy: Milind Kulkarni

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a c c $

$\boxed{\text{Match token}}$

Slide courtesy: Milind Kulkarni

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a c c $

Match token

Towards parser generators

- Key problem: as we read the source program, we need to decide what productions to use
- Step 1: find the tokens that can tell which production P (of the form $A \rightarrow X_1X_2 \ldots X_m$) applies

$$\mathrm{Predict}(P) =$$

$$\begin{cases} \mathrm{First}(X_1 \ldots X_m) & \text{if } \lambda \notin \mathrm{First}(X_1 \ldots X_m) \\ (\mathrm{First}(X_1 \ldots X_m) - \lambda) \cup \mathrm{Follow}(A) & \text{otherwise} \end{cases}$$

- If next token is in Predict(P), then we should choose this production

First Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
first (S) = { ? }
```

Think of all possible strings derivable from `S`.
Get the first terminal symbol in those strings
or λ if `S` derives λ

First Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
first (S) = { x, y, c }
```

First Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

first (S) = { x, y, c }
first (A) = {  ?  }
```

First Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

first (S) = { x, y, c }
first (A) = { x, y, c }
```

First Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
first (S) = { x, y, c }
first (A) = { x, y, c }
first (B) = {  ?  }
```

First Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
first (S) = { x, y, c }
first (A) = { x, y, c }
first (B) = { b, λ }
```

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

`follow (S) = { ? }`

Think of all strings **possible in the language** having the form `..Sa..` Get the following terminal symbol a after S in those strings or `$` if you get a string `..S$`

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

follow (S) = {  }
```

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

follow (S) = {   }
follow (A) = { ? }
```

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

follow (S) = {  }
follow (A) = { b, c }        e.g. xaAbc$, xaAc$
```

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
follow (S) = {   }
follow (A) = { b, c }        e.g. xaAbc$, xaAc$
```

What happens when you consider: `A -> xaA or A -> yaA ?`

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
follow (S) = {   }
follow (A) = { b, c }        e.g. xaAbc$, xaAc$
```

What happens when you consider: `A -> xaA or A -> yaA ?`

- You will get string of the form `A=>⁺ (xa)+A`
- But we need strings of the form: `..Aa..` or `..Ab.` or `..Ac..`

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
follow (S) = {   }
follow (A) = { b, c }
follow (B) = { ? }
```

Follow Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

```
follow (S) = {   }
follow (A) = { b, c }
follow (B) = { c }
```

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

$$\text{Predict}(P) = \begin{cases} \text{First}(X_1 \ldots X_m) & \text{if } \lambda \notin \text{First}(X_1 \ldots X_m) \\ (\text{First}(X_1 \ldots X_m) - \lambda) \cup \text{Follow}(A) & \text{otherwise} \end{cases}$$

Predict (1) = { ? }   = First(ABc$) if λ ∉ First(ABc$)

Predict Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

	x	y	a	b	c	$
S	1	1			1	
A						
B						

Predict (1) = { x, y, c }

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A						
B						

Predict (1) = { x, y, c }

Predict (2) = { ? } = First(xaA) if $\lambda \notin$ First(xaA)

Predict Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

	x	y	a	b	c	$
S	1	1			1	
A	2					
B						

```
Predict (1) = { x, y, c }
Predict (2) = { x }
```

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2					
B						

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { ? }   = First(yaA) if $\lambda \notin$ First(yaA)

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2	3				
B						

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2	3				
B						

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { ? } = First(c) if $\lambda \notin$ First(c)

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B						

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { c }

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B						

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { c }
Predict (5) = { ? }

= First(b) if $\lambda \notin$ First(b)

Predict Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B				5		

```
Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { c }
Predict (5) = { b }
```

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B				5		

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { c }
Predict (5) = { b }
Predict (6) = { ? }

$$\text{Predict}(P) = \begin{cases} \text{First}(X_1 \ldots X_m) & \text{if } \lambda \notin \text{First}(X_1 \ldots X_m) \\ (\text{First}(X_1 \ldots X_m) - \lambda) \cup \text{Follow}(A) & \text{otherwise} \end{cases}$$

= First(λ) ?

Predict Set - Example

1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B				5		

Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { c }
Predict (5) = { b }
Predict (6) = { ? }

$$\text{Predict}(P) = \begin{cases} \text{First}(X_1 \ldots X_m) & \text{if } \lambda \notin \text{First}(X_1 \ldots X_m) \\ \boxed{(\text{First}(X_1 \ldots X_m) - \lambda) \cup \text{Follow}(A) \quad \text{otherwise}} \end{cases}$$

= ~~First(λ)~~ ? Follow(B)

Predict Set - Example

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B				5	6	

```
Predict (1) = { x, y, c }
Predict (2) = { x }
Predict (3) = { y }
Predict (4) = { c }
Predict (5) = { b }
Predict (6) = { c }
```

Computing Parse-Table

```
1) S -> ABc$
2) A -> xaA
3) A -> yaA
4) A -> c
5) B -> b
6) B -> λ
```

	x	y	a	b	c	$
S	1	1			1	
A	2	3			4	
B				5	6	

```
first (S) = {x, y, c}
first (A) = {x, y, c}
first(B) = {b, λ}
```

```
follow (S) = {}
follow (A) = {b, c}
follow(B) = {c}
```

```
P(1) = {x,y,c}
P(2) = {x}
P(3) = {y}
P(4) = {c}
P(5) = {b}
P(6) = {c}
```