LL(1) Parsing

Parsing Techniques

Top-down parsers *(LL(1), recursive descent)*

- Start at the root of the parse tree and grow toward leaves
- Pick a production & try to match the input
- Bad "pick" $\Rightarrow$ may need to backtrack
- Some grammars are backtrack-free *(predictive parsing)*

Bottom-up parsers *(LR(1), operator precedence)*

- Start at the leaves and grow toward root
- As input is consumed, encode possibilities in an internal state
- Start in a state valid for legal first tokens
- Bottom-up parsers handle a large class of grammars

Top-down Parsing

A top-down parser starts with the root of the parse tree

The root node is labeled with the goal symbol of the grammar

Top-down parsing algorithm:

Construct the root node of the parse tree

Repeat until lower fringe of the parse tree matches the input string

1 *At a node labeled A, select a production with A on its lhs and, for each symbol on its rhs, construct the appropriate child*

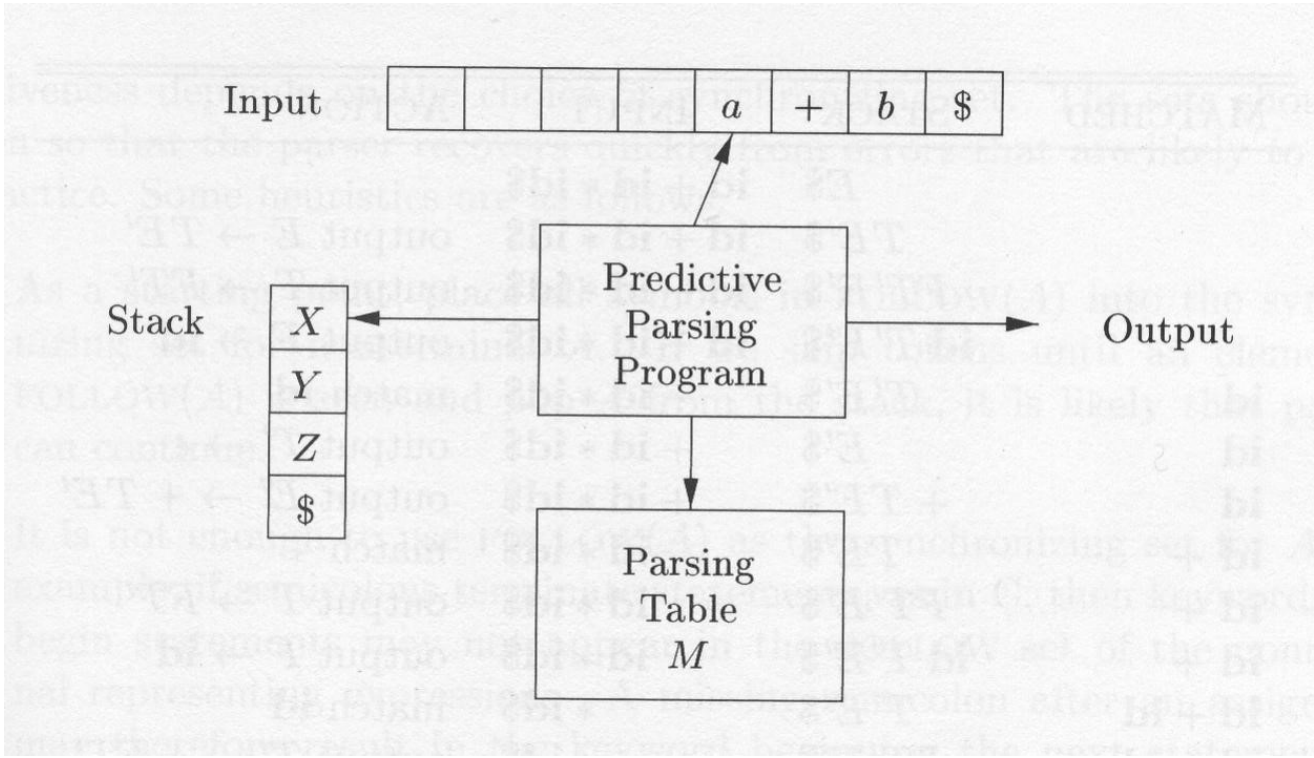
2 *When a terminal symbol is added to the fringe and it doesn't match the fringe, backtrack*

3 *Find the next node to be expanded* *(label $\in$ NT)*

The key is picking the right production in step 1

- *That choice should be guided by the input string*

Model of a table-driven predictive parser



Construction of a predictive parsing table

- The following rules are used to construct the predictive parsing table:

Consider the production **A → α**

- 1. for each terminal a in FIRST(α),
  add A → α to matrix M[A,a]

- 2. if λ is in FIRST(α), then
  for each terminal b in FOLLOW(A),
  add A → α to matrix M[A,b]

LL(1) Parsers (Cont.)

- Given the grammar:
  - E → TE’ 1
  - E’ → +TE’ 2
  - E’ → ϵ 3
  - T → FT’ 4
  - T’ → *FT’ 5
  - T’ → ϵ 6
  - F → (E) 7
  - F → id 8

LL(1) Parsers (Cont.)

FIRST(F) = FIRST(T) = FIRST(E) = { （, id }

FIRST(E’) = {＋, λ}

FIRST(T’) = { * , λ}

FOLLOW(E) = FOLLOW(E’) = {） , $}

FOLLOW(T) = FOLLOW(T’) = {＋, ） , $}

FOLLOW(F) = {＋, ＊, ） , $}

LL(1) Parsers (Cont.)

Non-terminal	Input symbols					
	Id	+	*	(	)	$
E	E → TE’			E → TE’		
E’		E’ → +TE’			E’ → ϵ	E’ → ϵ
T	T → FT’			T → FT’		
T’		T’ → ϵ	T’ → *FT’		T’ → ϵ	T’ → ϵ
F	F → id			F → (E)		

LL(1) Parsers (Cont.)

Stack	Input	Output
$E	id + id * id $	
$E'T	id + id * id $	E → TE'
$E'T'F	id + id * id $	T → FT'
$E'T'id	id + id * id $	F → id
$E'T'	+ id * id $	match id
$E'	+ id * id $	T' → λ
$E'T+	+ id * id $	E' → +TE'

LL(1) Parsers (Cont.)

Stack	Input	Output
$E'T	id * id $	match +
$E'T'F	id * id $	T → FT'
$E'T'id	id * id $	F → id
$E'T'	* id $	match id
$E'T'F*	* id $	T' → *FT'
$E'T'F	id $	match *
$E'T'id	id $	F → id
$E'T'	$	match id
$E'	$	T' → λ
$	$	E' → λ

A Grammar G is LL(1) if

$A \rightarrow \alpha \mid \beta$ are two distinct productions of grammar G, G is LL(1) if the following 3 conditions hold:

1. FIRST(α) cannot contain any terminal in FIRST(β).
2. At most one of α and β can derive λ.
3. if $\beta \rightarrow^* \lambda$, FIRST(α) cannot contain any terminal in FOLLOW(A).
   if $\alpha \rightarrow^* \lambda$, FIRST(β) cannot contain any terminal in FOLLOW(A).

Homework 1

Construct the LL(1) table for the following grammar:

1. Expr → - Expr
2. Expr → (Expr)
3. Expr → Var ExprTail
4. ExprTail → - Expr
5. ExprTail → λ
6. Var → id VarTail
7. VarTail → (Expr)
8. VarTail → λ

Homework 1 Solution

First(Expr) = {-, (, id}

First(ExprTail) = {-, λ}

First (Var) = { id}

First (VarTail) = { (, λ}

Follow (Expr) = Follow (ExprTail) = {$,) }

Follow (Var) = {$,), -}

Follow (VarTail) = {$,), -}

Homework 1 Solution (Cont.)

Non-Terminal	Input Symbol				
	-	(	id	)	$
Expr	1	2	3		
ExprTail	4			5	5
Var			6		
VarTail	8	7		8	8

Homework 2

- Given the grammar:
  - S → i E t S S’ | a
  - S’ → e S | λ
  - E → b

  - 1. Find the first set and follow set.
  - 2. Build the parsing table.

Homework 2 Solution

First(S) = {i, a}

First(S') = {e, λ}

First (E) = {b}

Follow (S) = Follow (S') = {$, e}

Follow (E) = {t}

Homework 2 Solution (Cont.)

Non-Terminal	Input Symbol					
	a	b	e	i	t	$
S	2			1		
S'			3/4			4
E		5				

As First(S') contains λ and Follow (S') = {$, e} So rule 4 is added to e, $.
3/4 (rule 3 or 4) means an error. This is not LL(1) grammar.

Example:

Grammar

E→TE'

E'→+TE' | ë

T→ FT'

T'→*FT' | ë

F→(E) | id

(ë stands for epsilon)

First and Follow

Symbol	FIRST	FOLLOW
E	(,id	$,)
E’	+,ë	$,)
T	(,id	+,$,)
T’	*,ë	+,$,)
F	(,id	*,+,$,)

Building the table

	Id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ë	E′→ë
T	T→FT′			T→FT′		
T′		T′→ë	T′→*FT′		T′→ë	T′→ë
F	F→id				F→(E)	

Input=id+id*id

Stack	Input buffer
$E	*id+id*id$*
$E'T'	*Id+id*id$*
$E'T'F	*Id+id*id$*
$E'T'id	*Id+id*id$*
$E'T'	*+id*id$*
$E'	*+id*id$*
$E'T+	*+id*id$*
$E'T	*id*id$*

Stack	Input Buffer
$E'T'F	*id*id$*
$E'T'id	*id*id$*
$E'T'	**id$*
*$E'T'F**	**id$*
$E'T'F	*id$*
$E'T'id	*id$*
$E'T'	*$*
$E'	*$*
$	Accepted

Thus, we can easily construct an LL parse with 1 lookahead. Since one look ahead is involved, we also call it an LL(1) parser.

There are grammars which may requite LL(k) parsing.

For e.g. look at next example…..