



HARVARD

John A. Paulson
School of Engineering
and Applied Sciences

CS153: Compilers

Lecture 10: LL Parsing

Stephen Chong

<https://www.seas.harvard.edu/courses/cs153>

Contains content from lecture notes by Greg Morrisett

Announcements

- HW3 LLVMlite out
 - Due Oct 15

Today

- LL Parsing
 - Nullable, First, Follow sets
 - Constructing an LL parsing table

LL(k) Parsing

- Our parser combinators backtrack
 - `alt p1 p2 = fun cs -> (p1 cs) @ (p2 cs)`
runs `p1` on `cs`, then backs up and runs `p2` on same input!
 - Inefficient! Tries all possible parses
- Could we somehow know which production to use?
- Basic idea: look at the next k symbols to predict whether we want `p1` or `p2`
- How do we predict which production to use?

FIRST Sets

- Given string γ of terminal and non-terminal symbols $\text{FIRST}(\gamma)$ is set of all terminal symbols that can start a string derived from γ

$$\begin{array}{lll} E \rightarrow T E' & T \rightarrow F T' & F \rightarrow \text{id} \\ E' \rightarrow + T E' & T' \rightarrow * F T' & F \rightarrow \text{num} \\ E' \rightarrow - T E' & T' \rightarrow / F T' & F \rightarrow (E) \\ E' \rightarrow & T' \rightarrow & \end{array}$$

- E.g., $\text{FIRST}(F T') = \{ \text{id}, \text{num}, (\}$
- We can use FIRST sets to determine which production to use!
 - Given nonterminal X , and all its productions
 $X \rightarrow \gamma_1, X \rightarrow \gamma_2, \dots, X \rightarrow \gamma_n,$
if $\text{FIRST}(\gamma_1), \dots, \text{FIRST}(\gamma_n)$ all mutually disjoint,
then next character tells us which production to use

Computing FIRST Sets

- See Appel for algorithm. Intuition here...
- Consider $\text{FIRST}(X Y Z)$
- How do compute it? Do we just need to know $\text{FIRST}(X)$?
- What if X can derive the empty string?
- Then $\text{FIRST}(Y) \subseteq \text{FIRST}(X Y Z)$
- What if Y can also derive the empty string?
- Then $\text{FIRST}(Z) \subseteq \text{FIRST}(X Y Z)$

Computing FIRST, FOLLOW and Nullable

- To compute FIRST sets, we need to compute whether nonterminals can produce empty string
- $\text{FIRST}(\gamma) =$ all terminal symbols that can start a string derived from γ
- $\text{Nullable}(X) = \text{true}$ iff X can derive the empty string
- We will also compute:
 $\text{FOLLOW}(X) =$ all terminals that can immediately follow X
 - i.e., $t \in \text{FOLLOW}(X)$ if there is a derivation containing Xt
- Algorithm iterates computing these until fix point reached
- **Note:** knowing $\text{nullable}(X)$ and $\text{FIRST}(X)$ for all non-terminals X allows us to compute $\text{nullable}(\gamma)$ and $\text{FIRST}(\gamma)$ for arbitrary strings of symbols γ

Example

$S \rightarrow E \text{ eof}$

$E \rightarrow T E'$

$E' \rightarrow + T E'$

$E' \rightarrow - T E'$

$E' \rightarrow$

$T \rightarrow F T'$

$T' \rightarrow * F T'$

$T' \rightarrow / F T'$

$T' \rightarrow$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

	nullable	FIRST	FOLLOW
S	⊥		
E	⊥		
E'	T		
T	⊥		
T'	T		
F	⊥		

X is nullable if there is a production $X \rightarrow \gamma$ where γ is empty, or γ is all nullable nonterminals

T' and E' are nullable!

And, we've finished nullable. Why?

Example

$S \rightarrow E \text{ eof}$

$E \rightarrow T E'$

$E' \rightarrow + T E'$

$E' \rightarrow - T E'$

$E' \rightarrow$

$T \rightarrow F T'$

$T' \rightarrow * F T'$

$T' \rightarrow / F T'$

$T' \rightarrow$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

	nullable	FIRST	FOLLOW
S	⊥		
E	⊥		
E'	T	+ -	
T	⊥		
T'	T	* /	
F	⊥	id num (

Given production $X \rightarrow t\gamma$, $t \in \text{FIRST}(X)$

Example

$S \rightarrow E \text{ eof}$

$E \rightarrow T E'$

$E' \rightarrow + T E'$

$E' \rightarrow - T E'$

$E' \rightarrow$

$T \rightarrow F T'$

$T' \rightarrow * F T'$

$T' \rightarrow / F T'$

$T' \rightarrow$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

	nullable	FIRST	FOLLOW
S	⊥	id num (
E	⊥	id num (
E'	⊤	+ -	
T	⊥	id num (
T'	⊤	* /	
F	⊥	id num (

Given production $X \rightarrow \gamma Y \sigma$,

if nullable(γ) then $\text{FIRST}(Y) \subseteq \text{FIRST}(X)$

Repeat until no more changes...

Example

$S \rightarrow E \text{ eof}$

$E \rightarrow T E'$

$E' \rightarrow + T E'$

$E' \rightarrow - T E'$

$E' \rightarrow$

$T \rightarrow F T'$

$T' \rightarrow * F T'$

$T' \rightarrow / F T'$

$T' \rightarrow$

$F \rightarrow \text{id}$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

	nullable	FIRST	FOLLOW
S	\perp	id num (
E	\perp	id num (eof)
E'	\top	+ -	eof)
T	\perp	id num (+ - eof)
T'	\top	* /	+ - eof)
F	\perp	id num (* / + - eof)

Given production $X \rightarrow \gamma Z \delta \sigma$

$\text{FIRST}(\delta) \subseteq \text{FOLLOW}(Z)$

and if δ is nullable then $\text{FIRST}(\sigma) \subseteq \text{FOLLOW}(Z)$

and if $\delta \sigma$ is nullable then $\text{FOLLOW}(X) \subseteq \text{FOLLOW}(Z)$

Predictive Parsing Table

- Make **predictive parsing table** with rows nonterminals, columns terminals
 - Table entries are productions
 - When parsing nonterminal X , and next token is t , entry for X and t will tell us which production to use

$S \rightarrow E \text{ eof}$ $T \rightarrow FT'$
 $E \rightarrow TE'$ $T' \rightarrow * FT'$
 $E' \rightarrow + TE'$ $T' \rightarrow / FT'$
 $E' \rightarrow - TE'$ $T' \rightarrow$
 $E' \rightarrow$

$F \rightarrow \text{id}$
 $F \rightarrow \text{num}$
 $F \rightarrow (E)$

Example

	nullable	FIRST	FOLLOW
S	\perp	id num (
E	\perp	id num (eof)
E'	\top	+ -	eof)
T	\perp	id num (+ - eof)
T'	\top	* /	+ - eof)
F	\perp	id num (* / + - eof)

	id	num	+	-	*	/	()	eof
S	$S \rightarrow E \text{ eof}$	$S \rightarrow E \text{ eof}$					$S \rightarrow E \text{ eof}$		
E	$E \rightarrow TE'$	$E \rightarrow TE'$					$E \rightarrow TE'$		
E'			$E' \rightarrow + TE'$	$E' \rightarrow - TE'$				$E' \rightarrow$	$E' \rightarrow$
T	$T \rightarrow FT'$	$T \rightarrow FT'$					$T \rightarrow FT'$		
T'			$T' \rightarrow$	$T' \rightarrow$	$T' \rightarrow * FT'$	$T' \rightarrow / FT'$		$T' \rightarrow$	$T' \rightarrow$
F	$F \rightarrow \text{id}$	$F \rightarrow \text{num}$					$F \rightarrow (E)$		

For $X \rightarrow \gamma$, add $X \rightarrow \gamma$ to row X column t for all $t \in \text{FIRST}(\gamma)$

For $X \rightarrow \gamma$, if γ is nullable, add $X \rightarrow \gamma$ to row X column t for all $t \in \text{FOLLOW}(X)$

Example

	id	num	+	-	*	/	()	eof
S	$S \rightarrow E \text{ eof}$	$S \rightarrow E \text{ eof}$					$S \rightarrow E \text{ eof}$		
E	$E \rightarrow T E'$	$E \rightarrow T E'$					$E \rightarrow T E'$		
E'			$E' \rightarrow + T E'$	$E' \rightarrow - T E'$				$E' \rightarrow$	$E' \rightarrow$
T	$T \rightarrow F T'$	$T \rightarrow F T'$					$T \rightarrow F T'$		
T'			$T' \rightarrow$	$T' \rightarrow$	$T' \rightarrow * F T'$	$T' \rightarrow / F T'$		$T' \rightarrow$	$T' \rightarrow$
F	$F \rightarrow \text{id}$	$F \rightarrow \text{num}$					$F \rightarrow (E)$		

- If each cell contains at most one production, parsing is predictive!
 - Table tells us exactly which production to apply

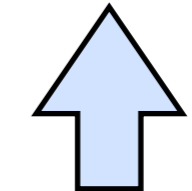
Example

	id	num	+	-	*	/	()	eof
S	$S \rightarrow E \text{ eof}$	$S \rightarrow E \text{ eof}$					$S \rightarrow E \text{ eof}$		
E	$E \rightarrow T E'$	$E \rightarrow T E'$					$E \rightarrow T E'$		
E'			$E' \rightarrow + T E'$	$E' \rightarrow - T E'$				$E' \rightarrow$	$E' \rightarrow$
T	$T \rightarrow F T'$	$T \rightarrow F T'$					$T \rightarrow F T'$		
T'			$T' \rightarrow$	$T' \rightarrow$	$T' \rightarrow * F T'$	$T' \rightarrow / F T'$		$T' \rightarrow$	$T' \rightarrow$
F	$F \rightarrow \text{id}$	$F \rightarrow \text{num}$					$F \rightarrow (E)$		

S
 $E \text{ eof}$
 $T E' \text{ eof}$
 $F T' E' \text{ eof}$
 $(E) T' E' \text{ eof}$

Parse S , next token is (, use $S \rightarrow E \text{ eof}$
 Parse E , next token is (, use $E \rightarrow T E'$
 Parse T , next token is (, use $T \rightarrow F T'$
 Parse F , next token is (, use $F \rightarrow (E)$

(foo + 7) eof



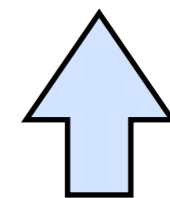
Example

	id	num	+	-	*	/	()	eof
<i>S</i>	$S \rightarrow E \text{ eof}$	$S \rightarrow E \text{ eof}$					$S \rightarrow E \text{ eof}$		
<i>E</i>	$E \rightarrow T E'$	$E \rightarrow T E'$					$E \rightarrow T E'$		
<i>E'</i>			$E' \rightarrow + T E'$	$E' \rightarrow - T E'$				$E' \rightarrow$	$E' \rightarrow$
<i>T</i>	$T \rightarrow F T'$	$T \rightarrow F T'$					$T \rightarrow F T'$		
<i>T'</i>			$T' \rightarrow$	$T' \rightarrow$	$T' \rightarrow * F T'$	$T' \rightarrow / F T'$		$T' \rightarrow$	$T' \rightarrow$
<i>F</i>	$F \rightarrow \text{id}$	$F \rightarrow \text{num}$					$F \rightarrow (E)$		

S
 $E \text{ eof}$
 $T E' \text{ eof}$
 $F T' E' \text{ eof}$
 $(E) T' E' \text{ eof}$
 $(T E') T' E' \text{ eof}$
 $(F T' E') T' E' \text{ eof}$

Parse S , next token is (, use $S \rightarrow E \text{ eof}$
 Parse E , next token is (, use $E \rightarrow T E'$
 Parse T , next token is (, use $T \rightarrow F T'$
 Parse F , next token is (, use $F \rightarrow (E)$
 Parse E , next token is id , use $E \rightarrow T E'$
 Parse T , next token is id , use $T \rightarrow F T'$
 Parse F , next token is id , use $F \rightarrow \text{id}$

(foo + 7) eof



Example

	id	num	+	-	*	/	()	eof
S	$S \rightarrow E \text{ eof}$	$S \rightarrow E \text{ eof}$					$S \rightarrow E \text{ eof}$		
E	$E \rightarrow T E'$	$E \rightarrow T E'$					$E \rightarrow T E'$		
E'			$E' \rightarrow + T E'$	$E' \rightarrow - T E'$				$E' \rightarrow$	$E' \rightarrow$
T	$T \rightarrow F T'$	$T \rightarrow F T'$					$T \rightarrow F T'$		
T'			$T' \rightarrow$	$T' \rightarrow$	$T' \rightarrow * F T'$	$T' \rightarrow / F T'$		$T' \rightarrow$	$T' \rightarrow$
F	$F \rightarrow \text{id}$	$F \rightarrow \text{num}$					$F \rightarrow (E)$		

$(F T' E') T' E' \text{ eof}$

$(\text{id } T' E') T' E' \text{ eof}$

$(\text{id } E') T' E' \text{ eof}$

$(\text{id } + T E') T' E' \text{ eof}$

$(\text{id } + F T' E') T' E' \text{ eof}$

Parse F , next token is id , use $F \rightarrow \text{id}$

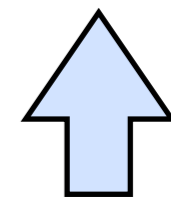
Parse T' , next token is $+$, use $T' \rightarrow$

Parse E' , next token is $+$, use $E' \rightarrow + T E'$

Parse T , next token is num , use $T \rightarrow F T'$

Parse F , next token is num , use $F \rightarrow \text{num}$

$(\text{foo } + 7) \text{ eof}$



Example

	id	num	+	-	*	/	()	eof
S	$S \rightarrow E \text{ eof}$	$S \rightarrow E \text{ eof}$					$S \rightarrow E \text{ eof}$		
E	$E \rightarrow T E'$	$E \rightarrow T E'$					$E \rightarrow T E'$		
E'			$E' \rightarrow + T E'$	$E' \rightarrow - T E'$				$E' \rightarrow$	$E' \rightarrow$
T	$T \rightarrow F T'$	$T \rightarrow F T'$					$T \rightarrow F T'$		
T'			$T' \rightarrow$	$T' \rightarrow$	$T' \rightarrow * F T'$	$T' \rightarrow / F T'$		$T' \rightarrow$	$T' \rightarrow$
F	$F \rightarrow \text{id}$	$F \rightarrow \text{num}$					$F \rightarrow (E)$		

(id + $F T' E'$) $T' E'$ eof Parse F , next token is num, use $F \rightarrow \text{num}$

(id + num $T' E'$) $T' E'$ eof Parse T' , next token is), use $T' \rightarrow$

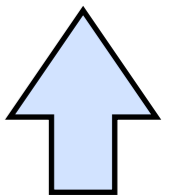
(id + num E') $T' E'$ eof Parse E' , next token is), use $E' \rightarrow$

(id + num) $T' E'$ eof Parse T' , next token is eof, use $T' \rightarrow$

(id + num) E' eof Parse E' , next token is eof, use $E' \rightarrow$

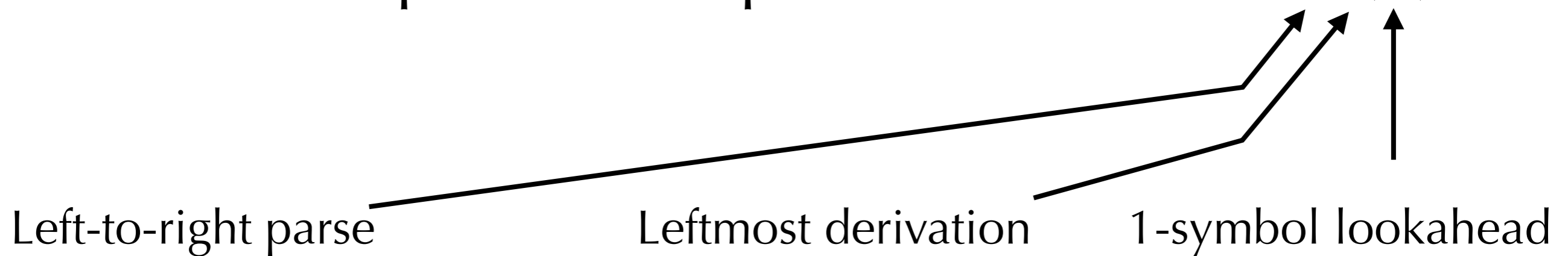
(id + num) eof

(foo + 7) eof



LL(1), LL(k), LL(*)

- Grammars whose predictive parsing table contain at most one production per cell are called LL(1)



i.e., go through token stream
from left to right.
(Almost all parsers do this)

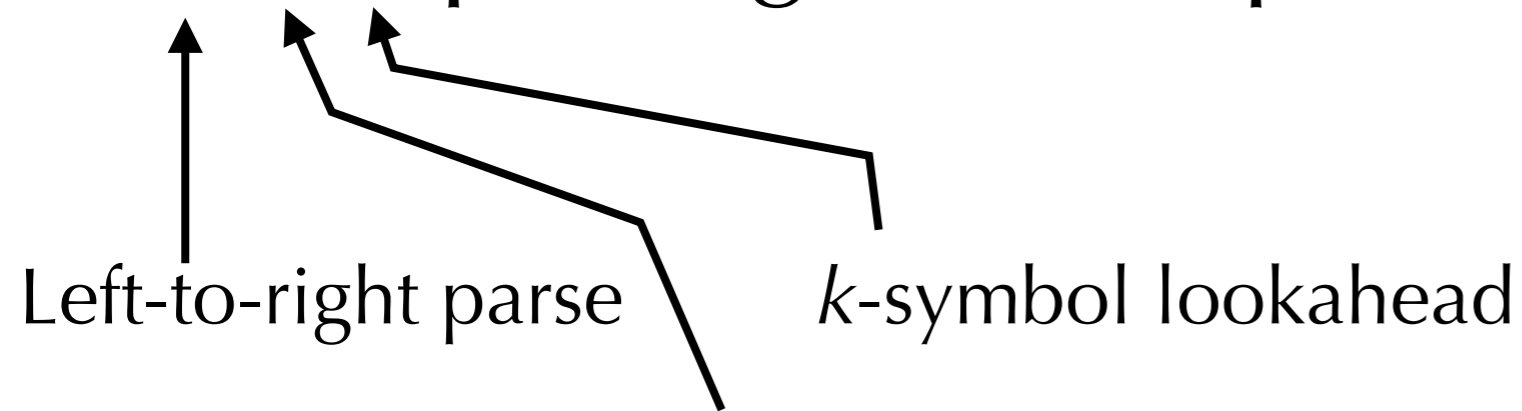
Derivation expands the
leftmost non-terminal

LL(1), LL(k), LL(*)

- Grammars whose predictive parsing table contain at most one production per cell are called LL(1)
- Can be generalized to LL(2), LL(3), etc.
 - Columns of predictive parsing table have k tokens
 - FIRST(X) generalized to FIRST- k (X)
- An LL(*) grammar can determine next production using finite (but maybe unbounded) lookahead
- An ambiguous grammar is not LL(k) for any k , or even LL(*)
 - Why?

LR(k)

- What if grammar is unambiguous but not LL(k)?
- LR(k) parsing is more powerful technique



Rightmost derivation

Derivation expands the
rightmost non-terminal

(Constructs derivation in
reverse order!)