CS 516: COMPILERS

Lecture 11

Topics

- Parsing (finding derivations in a grammar)
  - via Recursive Descent
  - LL(1) Grammars
  - Next week: Shift/Reduce, LR Grammars

Materials

- lec11.zip

CFGs Mathematically

- A Context-free Grammar (CFG) consists of
  - A set of *terminals* (e.g., a token or ε)
  - A set of *nonterminals* (e.g., S and other syntactic variables)
  - A designated nonterminal called the *start symbol*
  - A set of productions: LHS $\longmapsto$ RHS
    - LHS is a nonterminal
    - RHS is a *string* of terminals and nonterminals

- Example: The balanced parentheses language:

$$S \longmapsto (S)S$$

$$S \longmapsto \varepsilon$$

- How many terminals? How many nonterminals? Productions?

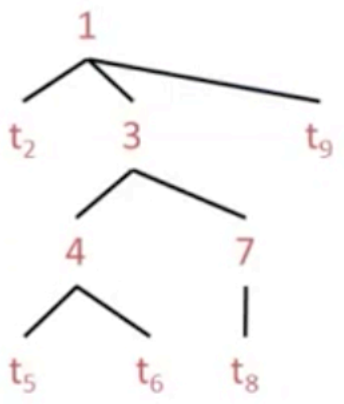
Brute-force parsing

RECURSIVE DESCENT

Recursive Descent Parsing

- The parse tree is constructed
  - From the top
  - From left to right
- Terminals are seen in order of appearance in the token stream:

  $t_2\ t_5\ t_6\ t_8\ t_9$



Recursive Descent Parsing

- Consider the grammar

$$E \longmapsto T \mid T + E$$
$$T \longmapsto \text{int} \mid \text{int} * T \mid (E)$$

- Token stream is: (int_5)
- Start with top-level non-terminal E
  - Try the rules for E in order

Recursive Descent Parsing

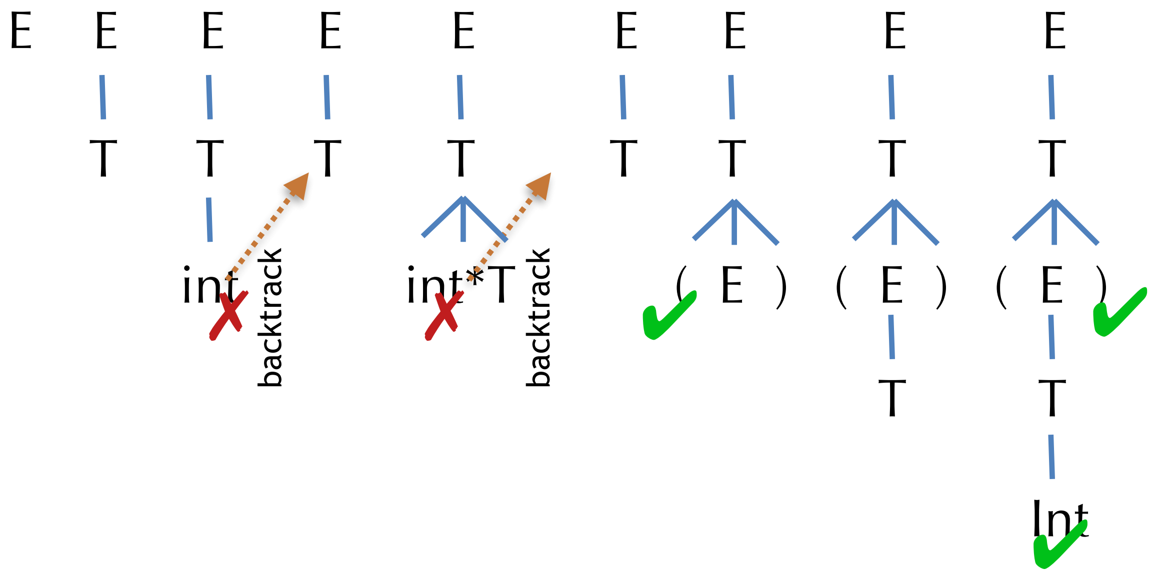
$$E \longmapsto T \mid T + E$$
$$T \longmapsto \text{int} \mid \text{int} * T \mid (E)$$

E

(int)

Recursive Descent Parsing

$E \longmapsto T \mid T + E$
$T \longmapsto \text{int} \mid \text{int} * T \mid (E)$



(int)

Recursive Descent Parsing

```
let stream = Tok array
let next = ref 0

let term tok : bool =
  if stream.(next) == peek() then (next++; true) else false

let E1 () : bool = T ()
let E2 () : bool = T () && term(PLUS) && E ()
let E  () : bool =
  let save = !next in
    (next := save; E1()) || (next := save; E2 ()))

let T1 () : bool = term(INT)
let T2 () : bool = term(INT) && term(TIMES) && T ()
let T3 () : bool = term(OPEN) && E() && term(CLOSE)
let T  () : bool =
  let save = !next in
    (next := save; T1()) || (next := save; T2 ())
                         || (next := save; T3 ()))
```

Can we avoid backtracking?

Searching for derivations.

LL PARSING

LL(1) GRAMMARS

Consider finding left-most derivations

- Look at only one input symbol at a time.

$$S \longmapsto E + S \ \mid \ E$$
$$E \longmapsto \text{number} \mid (\ S\ )$$

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \;|\; E$
$E \longmapsto \text{number} \;|\; (\; S \;)$

Partly-derived String	Parsed/Unparsed Input

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
S	(1 + 2 + (3 + 4)) + 5

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
<u>S</u>	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto$ **<u>E</u>** + S	$\boxed{(}1 + 2 + (3 + 4)) + 5$

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \;|\; E$
$E \longmapsto \text{number} \;|\; (\, S\,)$

Partly-derived String	Parsed/Unparsed Input
S	(1 + 2 + (3 + 4)) + 5
$\longmapsto$ **E** + S	(1 + 2 + (3 + 4)) + 5
$\longmapsto$ (**S**) + S	(1 + 2 + (3 + 4)) + 5

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
S	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto$ **E** + S	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto$ (**S**) + S	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto$ (**E** + S) + S	$(\boxed{1} + 2 + (3 + 4)) + 5$

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
$\underline{\mathbf{S}}$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto \underline{\mathbf{E}} + S$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{S}}) + S$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{E}} + S) + S$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{S}}) + S$	$(1 + \boxed{2} + (3 + 4)) + 5$

Consider finding left-most derivations

- Look at only one input symbol at a time.

S ⟼ E + S | E
E ⟼ number | (S)

Partly-derived String	Parsed/Unparsed Input
S	(1 + 2 + (3 + 4)) + 5
⟼ **E** + S	(1 + 2 + (3 + 4)) + 5
⟼ (**S**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (**E** + S) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + **S**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + **E** + S) + S	(1 + 2 + (3 + 4)) + 5

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
$\underline{\mathbf{S}}$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto \underline{\mathbf{E}} + S$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{S}}) + S$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{E}} + S) + S$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{S}}) + S$	$(1 + \boxed{2} + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{E}} + S) + S$	$(1 + 2 + (3 + 4)) + 5$
$\longmapsto (1 + 2 + \underline{\mathbf{S}}) + S$	$(1 + 2 + (3 + 4)) + 5$

Consider finding left-most derivations

- Look at only one input symbol at a time.

S ⟼ E + S | E
E ⟼ number | (S)

Partly-derived String	Parsed/Unparsed Input
S	(1 + 2 + (3 + 4)) + 5
⟼ **E** + S	(1 + 2 + (3 + 4)) + 5
⟼ (**S**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (**E** + S) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + **S**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + **E** + S) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + 2 + **S**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + 2 + **E**) + S	(1 + 2 + (3 + 4)) + 5

Consider finding left-most derivations

- Look at only one input symbol at a time.

S ⟼ E + S | E
E ⟼ number | (S)

Partly-derived String	Parsed/Unparsed Input
<u>S</u>	(1 + 2 + (3 + 4)) + 5
⟼ **<u>E</u>** + S	(1 + 2 + (3 + 4)) + 5
⟼ (**<u>S</u>**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (**<u>E</u>** + S) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + **<u>S</u>**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + **<u>E</u>** + S) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + 2 + **<u>S</u>**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + 2 + **<u>E</u>**) + S	(1 + 2 + (3 + 4)) + 5
⟼ (1 + 2 + (**<u>S</u>**)) + S	(1 + 2 + (3 + 4)) + 5

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
$\underline{\mathbf{S}}$	(1 + 2 + (3 + 4)) + 5
$\longmapsto \underline{\mathbf{E}} + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (\underline{\mathbf{S}}) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (\underline{\mathbf{E}} + S) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (1 + \underline{\mathbf{S}}) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (1 + \underline{\mathbf{E}} + S) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (1 + 2 + \underline{\mathbf{S}}) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (1 + 2 + \underline{\mathbf{E}}) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (1 + 2 + (\underline{\mathbf{S}})) + S$	(1 + 2 + (3 + 4)) + 5
$\longmapsto (1 + 2 + (\underline{\mathbf{E}} + S)) + S$	(1 + 2 + (3 + 4)) + 5

Consider finding left-most derivations

- Look at only one input symbol at a time.

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

Partly-derived String	Parsed/Unparsed Input
$\underline{\mathbf{S}}$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto \underline{\mathbf{E}} + S$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{S}}) + S$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{E}} + S) + S$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{S}}) + S$	$(1 + \boxed{2} + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{E}} + S) + S$	$(1 + 2 + (3 + 4)) + 5$
$\longmapsto (1 + 2 + \underline{\mathbf{S}}) + S$	$(1 + 2 + (3 + 4)) + 5$
$\longmapsto (1 + 2 + \underline{\mathbf{E}}) + S$	$(1 + 2 + (3 + 4)) + 5$
$\longmapsto (1 + 2 + (\underline{\mathbf{S}})) + S$	$(1 + 2 + (3 + 4)) + 5$
$\longmapsto (1 + 2 + (\underline{\mathbf{E}} + S)) + S$	$(1 + 2 + (3 + 4)) + 5$
. . .	. . .

There is a problem

$S \longmapsto E + S \mid E$
$E \longmapsto \text{number} \mid (S)$

- We want to decide which production to apply based on the look-ahead symbol.
- But, there is a choice:

(1) $\quad\boxed{S \longmapsto E} \longmapsto (S) \longmapsto (E) \longmapsto (1)$

vs.

$(1) + 2$ $\quad\boxed{S \longmapsto E + S} \longmapsto (S) + S \longmapsto (E) + S \longmapsto (1) + S$

$\ldots \longmapsto (1) + E \longmapsto (1) + 2$

- Given the look-ahead symbol: '(' it isn't clear whether to pick $S \longmapsto E$ or $S \longmapsto E + S$ first.

Grammar is the problem

- Not all grammars can be parsed "top-down" with only a single lookahead symbol.
- *Top-down*: starting from the start symbol (root of the parse tree) and going down (eg, recursive descent)

- LL(1) means
  - **L**eft-to-right scanning
  - **L**eft-most derivation,
  - **1** lookahead symbol

- This language isn't "LL(1)"
- Is it LL(k) for some k?

$$S \longmapsto E + S \mid E$$
$$E \longmapsto \text{number} \mid (S)$$

- What can we do?

Making a grammar LL(1)

- *Problem:* We can't decide which S production to apply until we see the symbol after the first expression.
- *Solution:* "Left-factor" the grammar. There is a common S prefix for each choice, so add a new non-terminal S′ at the decision point:

$$\begin{array}{l} S \longmapsto E + S \mid E \\ E \longmapsto \text{number} \mid (S) \end{array}$$

- Also need to eliminate left-recursion somehow. Why?
- Consider:

$$\begin{array}{l} S \longmapsto S + E \mid E \\ E \longmapsto \text{number} \mid (S) \end{array}$$

Making a grammar LL(1)

- *Problem:* We can't decide which S production to apply until we see the symbol after the first expression.
- *Solution:* "Left-factor" the grammar. There is a common S prefix for each choice, so add a new non-terminal S′ at the decision point:

$$\begin{array}{l} S \longmapsto E \boxed{+\ S} \mid E \boxed{\varepsilon} \\ E \longmapsto \text{number} \mid (\ S\ ) \end{array} \Longrightarrow \begin{array}{l} S \longmapsto ES' \\ S' \longmapsto \varepsilon \\ S' \longmapsto +\ S \\ E \longmapsto \text{number} \mid (\ S\ ) \end{array}$$

- Also need to eliminate left-recursion somehow. Why?
- Consider:

$$\begin{array}{l} S \longmapsto S + E \mid E \\ E \longmapsto \text{number} \mid (\ S\ ) \end{array}$$

LL(1) Parse of the input string

- Look at only one input symbol at a time.

$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto +\, S$
$E \longmapsto \text{number} \mid (\, S\,)$

Partly-derived String	Parsed/Unparsed Input
$\underline{\mathbf{S}}$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto \underline{\mathbf{E}}\ S'$	$\boxed{(}1 + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{S}})\ S'$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (\underline{\mathbf{E}}\ S')\ S'$	$(\boxed{1} + 2 + (3 + 4)) + 5$
$\longmapsto (1\ \underline{\mathbf{S'}})\ S'$	$(1\ \boxed{+}\ 2 + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{S}})\ S'$	$(1 + \boxed{2} + (3 + 4)) + 5$
$\longmapsto (1 + \underline{\mathbf{E}}\ S')\ S'$	$(1 + \boxed{2} + (3 + 4)) + 5$
$\longmapsto (1 + 2\ \underline{\mathbf{S'}})\ S'$	$(1 + 2\ \boxed{+}\ (3 + 4)) + 5$
$\longmapsto (1 + 2 + \underline{\mathbf{S}})\ S'$	$(1 + 2 + \boxed{(}3 + 4)) + 5$
$\longmapsto (1 + 2 + \underline{\mathbf{E}}\ S')\ S'$	$(1 + 2 + \boxed{(}3 + 4)) + 5$
$\longmapsto (1 + 2 + (\underline{\mathbf{S}})S')\ S'$	$(1 + 2 + (\boxed{3} + 4)) + 5$

Predictive Parsing

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + S$
$E \longmapsto \text{number} \mid (S)$

- Given an **LL(1)** grammar:
  - For a given nonterminal, the *lookahead symbol uniquely determines* the production to apply.
  - Top-down parsing = predictive parsing
  - Driven by a predictive parsing table:
    nonterminal * input token → production

	number	**+**	**(**	**)**	**$ (EOF)**
T	$\longmapsto$ S$		$\longmapsto$S$		
S	$\longmapsto$ E S′		$\longmapsto$E S′		
S′		$\longmapsto$ + S		$\longmapsto$ ε	$\longmapsto$ ε
E	$\longmapsto$ num.		$\longmapsto$ (S)		

- Note: it is convenient to add a special *end-of-file* token $ and a start symbol T (top-level) that requires $.

How do we construct the parse table?

- Consider a given production: $A \longmapsto \gamma$
- We construct two **Sets**:

1. **First(**A**):** The set of all input *tokens* that may appear *first* in strings that can be derived from γ
   – Add the production "$\longmapsto \gamma$" to the entry **(A,token)** for each such token.

2. **Follow(**A**):** If γ can derive **ε** (the empty string), then we construct the set of all input tokens that may *follow* the nonterminal A in the grammar.
   – Add the production "$\longmapsto \gamma$" to the entry **(A, token)** for each such token.

- Note: if there are two different productions for a given entry, the grammar is not LL(1)

Example

- First(T) = First(S)
- First(S) = First(E)
- First(S′) = { '+' }
- First(E) = { number, '(' }

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + \, S$
$E \longmapsto \text{number} \mid (\; S \;)$

	number	+	(	)	$ (EOF)
T					
S					
S′					
E					

Example

- First(T) = First(S) = { number, ‘(‘ }
- First(S) = First(E) = { number, ‘(‘ }
- First(S′) = { ‘+′ }
- First(E) = { number, ‘(‘ }

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + S$
$E \longmapsto \text{number} \mid (S)$

	number	+	(	)	$ (EOF)
T					
S					
S′					
E					

Example

- First(T) = First(S) = { number, ‘(‘ }
- First(S) = First(E) = { number, ‘(‘ }
- First(S′) = { ‘+’ }
- First(E) = { number, ‘(‘ }

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + S$
$E \longmapsto \text{number} \mid (S)$

	number	+	(	)	$ (EOF)
T	$\longmapsto S\$$		$\longmapsto S\$$		
S					
S′					
E					

Example

- $\text{First}(T) = \text{First}(S) = \{\ \text{number},\ \text{‘(’}\ \}$
- $\text{First}(S) = \text{First}(E) = \{\ \text{number},\ \text{‘(’}\ \}$
- $\text{First}(S') = \{\ \text{‘+’}\ \}$
- $\text{First}(E) = \{\ \text{number},\ \text{‘(’}\ \}$

$$
\begin{aligned}
T &\longmapsto S\$ \\
S &\longmapsto ES' \\
S' &\longmapsto \varepsilon \\
S' &\longmapsto +\ S \\
E &\longmapsto \text{number} \mid (\ S\ )
\end{aligned}
$$

	number	**+**	**(**	**)**	**\$ (EOF)**
T	$\longmapsto S\$$		$\longmapsto S\$$		
S	$\longmapsto E\ S'$		$\longmapsto E\ S'$		
S′					
E					

Example

- First(T) = First(S) = { number, '(' }
- First(S) = First(E) = { number, '(' }
- First(S′) = { '+' }
- First(E) = { number, '(' }

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + S$
$E \longmapsto \text{number} \mid (\; S \;)$

	number	+	(	)	$ (EOF)
T	$\longmapsto S\$$		$\longmapsto S\$$		
S	$\longmapsto E\ S'$		$\longmapsto E\ S'$		
S′		$\longmapsto + S$			
E					

Example

- First(T) = First(S) = { number, '(' }
- First(S) = First(E) = { number, '(' }
- First(S′) = { '+' }
- First(E) = { number, '(' }

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + S$
$E \longmapsto \text{number} \mid (\ S\ )$

	number	+	(	)	$ (EOF)
T	⟼ S$		⟼S$		
S	⟼ E S′		⟼E S′		
S′		⟼ + S			
E	⟼ num.		⟼ (S)		

Example

- First(T) = First(S) = { number, '(' }
- First(S) = First(E) = { number, '(' }
- First(S′) = { '+' }
- First(E) = { number, '(' }

- Follow(S′) = Follow(S)
- Follow(S) = { \$, ')' } ∪ Follow(S′)

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + S$
$E \longmapsto \text{number} \mid (S)$

	number	+	(	)	\$ (EOF)
T	$\longmapsto S\$$		$\longmapsto S\$$		
S	$\longmapsto E\ S'$		$\longmapsto E\ S'$		
S′		$\longmapsto + S$			
E	$\longmapsto$ num.		$\longmapsto (S)$		

Example

- First(T) = First(S) = { number, ‘(‘ }
- First(S) = First(E) = { number, ‘(‘ }
- First(S′) = { ‘+′ }
- First(E) = { number, ‘(‘ }

- Follow(S′) = Follow(S)
- Follow(S) = { \$, ‘)′ } ∪ Follow(S′)

Note: we want the *least* solution to this system of set equations… a *fixpoint* computation. More on these later in the course.

T ⟼ S\$
S ⟼ ES′
S′ ⟼ ε
S′ ⟼ + S
E ⟼ number | (S)

	number	+	(	)	\$ (EOF)
T	⟼ S\$		⟼S\$		
S	⟼ E S′		⟼E S′		
S′		⟼ + S			
E	⟼ num.		⟼ (S)		

Example

- $\text{First}(T) = \text{First}(S) = \{$ number, ‘(‘ $\}$
- $\text{First}(S) = \text{First}(E) = \{$ number, ‘(‘ $\}$
- $\text{First}(S') = \{$ ‘+’ $\}$
- $\text{First}(E) = \{$ number, ‘(‘ $\}$

- $\text{Follow}(S') = \text{Follow}(S)$
- $\text{Follow}(S) = \{ \$,$ ‘)’ $\} \cup \text{Follow}(S')$

Note: we want the *least* solution to this system of set equations… a *fixpoint* computation. More on these later in the course.

$T \longmapsto S\$$
$S \longmapsto ES'$
$S' \longmapsto \varepsilon$
$S' \longmapsto + \, S$
$E \longmapsto$ number $|$ (S)

	number	+	(	)	$ (EOF)
T	$\longmapsto S\$$		$\longmapsto S\$$		
S	$\longmapsto E\ S'$		$\longmapsto E\ S'$		
S′		$\longmapsto + S$		$\longmapsto \varepsilon$	$\longmapsto \varepsilon$
E	$\longmapsto$ num.		$\longmapsto$ (S)		

Converting the table to code

- Define n mutually recursive functions
  - one for each nonterminal A: parse_A
  - The type of parse_A is `unit -> ast` if A is *not* an auxiliary nonterminal
  - Parse functions for auxiliary nonterminals (e.g. S′) take extra ast's as inputs, one for each nonterminal in the "factored" prefix.

- Each function "peeks" at the lookahead token and then follows the production rule in the corresponding entry.
  - Consume terminal tokens from the input stream
  - Call parse_X to create sub-tree for nonterminal X
  - If the rule ends in an auxiliary nonterminal, call it with appropriate ast's. (The auxiliary rule is responsible for creating the ast after looking at more input.)
  - Otherwise, this function builds the ast tree itself and returns it.

Predictive Parsing
Improves over "brute-force" Recursive Descent by peeking ahead

	number	+	(	)	$ (EOF)
T	$\longmapsto$ S$		$\longmapsto$S$		
S	$\longmapsto$ E S′		$\longmapsto$E S′		
S′		$\longmapsto$ + S		$\longmapsto$ ε	$\longmapsto$ ε
E	$\longmapsto$ num.		$\longmapsto$ (S)		

Hand-generated LL(1) code for the table above.

DEMO: PARSER.ML

LL(1) Summary

- Top-down parsing that finds the leftmost derivation.
- Language Grammar ⇒ recursive descent parser
- Language Grammar ⇒ LL(1) grammar ⇒ prediction table ⇒ recursive-descent **predictive** parser
- Notes:
  - With prediction table, don't need backtracking
  - Can extend to LL(k) (it just makes the table bigger)
    - Some **k** that allows the recursive descent parser to decide production by looking at the next **k** input tokens
  - LL(k) exclude ambiguous grammars and left-recursion
- Problems:
  - Grammar must be LL(1)
  - Grammar cannot be left recursive (parser functions will loop!)
- Is there a better way?