LL(1) parsing

Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.

Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.
- Top-down parsers come in two forms: **backtracking parsers** and **predictive parsers**.

Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.
- Top-down parsers come in two forms:
  backtracking parsers and **predictive parsers**.
- A predictive parser attempts to predict the next construction using one or more lookahead tokens.

Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.
- Top-down parsers come in two forms: **backtracking parsers** and **predictive parsers**.
- A predictive parser attempts to predict the next construction using one or more lookahead tokens.
- Two well-known top-down parsing methods are **recursive-descent parsing** and **LL(1) parsing**.

Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.
- Top-down parsers come in two forms: **backtracking parsers** and **predictive parsers**.
- A predictive parser attempts to predict the next construction using one or more lookahead tokens.
- Two well-known top-down parsing methods are **recursive-descent parsing** and **LL(1) parsing**.
- Recursive descent parsing is the most suitable method for a handwritten parser.

LL(1) parsing

- The first "L" in LL(1) refers to the fact that the input is processed from left to right.

LL(1) parsing

- The first "L" in LL(1) refers to the fact that the input is processed from left to right.
- The second "L" refers to the fact that LL(1) parsing determines a leftmost derivation for the input string.

LL(1) parsing

- The first “L” in LL(1) refers to the fact that the input is processed from left to right.
- The second “L” refers to the fact that LL(1) parsing determines a leftmost derivation for the input string.
- The “1” in parentheses implies that LL(1) parsing uses only one symbol of input to predict the next grammar rule that should be used.

LL(1) parsing - Example 1

- We start by considering the following grammar that generates strings of balanced parenthesis:
  $S \rightarrow (\ S \ ) \ S \mid \varepsilon$

LL(1) parsing - Example 1

- We start by considering the following grammar that generates strings of balanced parenthesis:
  $S \rightarrow (\ S \ ) \ S \mid \varepsilon$
- We will assume that \$ marks the bottom of a stack and the end of input.

LL(1) parsing - Example 1

- We start by considering the following grammar that generates strings of balanced parenthesis:
  $S \rightarrow (\; S \;) \; S \mid \varepsilon$
- We will assume that \$ marks the bottom of a stack and the end of input.
- Parsing action of an LL(1) parser:

	Parsing Stack	Input	Action
1	$\$S$	()\$	$S \rightarrow (S)S$
2	$\$S)S($	()\$	match
3	$\$S)S$	)\$	$S \rightarrow \varepsilon$
4	$\$S)$	)\$	match
5	$\$S$	\$	$S \rightarrow \varepsilon$
6	\$	\$	accept

LL(1) parsing - Example 1

- We start by considering the following grammar that generates strings of balanced parenthesis:
  $S \longrightarrow (\ S \ ) \ S \mid \varepsilon$
- We will assume that \$ marks the bottom of a stack and the end of input.
- Parsing action of an LL(1) parser:

	Parsing Stack	Input	Action
1	$\$S$	$()\$$	$S \longrightarrow (S)S$
2	$\$S)S($	$()\$$	match
3	$\$S)S$	$)\$$	$S \longrightarrow \varepsilon$
4	$\$S)$	$)\$$	match
5	$\$S$	$\$$	$S \longrightarrow \varepsilon$
6	$\$$	$\$$	accept

- The LL(1) parsing table $M[N, T]$:

$M[N, T]$	(	)	\$
S	$S \longrightarrow (S)S$	$S \longrightarrow \varepsilon$	$S \longrightarrow \varepsilon$

LL(1) parsing tables

- We use the parsing table to decide which decision should be made if a given nonterminal N is at the top of the parsing stack, based on the current input symbol T.

LL(1) parsing tables

- We use the parsing table to decide which decision should be made if a given nonterminal N is at the top of the parsing stack, based on the current input symbol T.
- We add production choices to the LL(1) parsing table as follows:

LL(1) parsing tables

- We use the parsing table to decide which decision should be made if a given nonterminal N is at the top of the parsing stack, based on the current input symbol T.
- We add production choices to the LL(1) parsing table as follows:
  1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow^* a\beta$, where a is a token, then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.

LL(1) parsing tables

- We use the parsing table to decide which decision should be made if a given nonterminal N is at the top of the parsing stack, based on the current input symbol T.
- We add production choices to the LL(1) parsing table as follows:
  1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow^* a\beta$, where a is a token, then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.
  2. If $A \rightarrow \alpha$ is a production choice, and there are derivations $\alpha \Rightarrow^* \varepsilon$ and $S\$ \Rightarrow^* \beta A a \gamma$, where S is the start symbol and a is a token (or \$), then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.

LL(1) parsing tables

- The idea behind these rules are as follows:

LL(1) parsing tables

- The idea behind these rules are as follows:
  1. In rule 1, given a token a in the input, we wish to select a rule $A \to \alpha$ if α can produce an a for matching.

LL(1) parsing tables

- The idea behind these rules are as follows:
  1. In rule 1, given a token a in the input, we wish to select a rule $A \rightarrow \alpha$ if α can produce an a for matching.
  2. In rule 2, if A derives the empty string (via $A \rightarrow \alpha$), and if a is a token that can legally come after A in a derivation, then we want to select $A \rightarrow \alpha$ to make A disappear.

LL(1) parsing tables

- The idea behind these rules are as follows:
  1. In rule 1, given a token *a* in the input, we wish to select a rule $A \rightarrow \alpha$ if α can produce an *a* for matching.
  2. In rule 2, if A derives the empty string (via $A \rightarrow \alpha$), and if *a* is a token that can legally come after A in a derivation, then we want to select $A \rightarrow \alpha$ to make A disappear.
- These rules are difficult to implement directly, so we will develop algorithms involving **first** and **follow sets**.

LL(1) parsing tables

- The idea behind these rules are as follows:
  1. In rule 1, given a token a in the input, we wish to select a rule $A \rightarrow \alpha$ if α can produce an a for matching.
  2. In rule 2, if A derives the empty string (via $A \rightarrow \alpha$), and if a is a token that can legally come after A in a derivation, then we want to select $A \rightarrow \alpha$ to make A disappear.
- These rules are difficult to implement directly, so we will develop algorithms involving **first** and **follow sets**.
- A grammar is an **LL(1) grammar** if the associated LL(1) parsing table has at most one production in each table entry.

First and Follow sets

- Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.

First and Follow sets

- Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.
- Repeat the following two steps for each nonterminal A and each production $A \longrightarrow \alpha$:

First and Follow sets

- Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.
- Repeat the following two steps for each nonterminal A and each production $A \rightarrow \alpha$:
  1. For each token a in First(α), add $A \rightarrow \alpha$ to the entry $M[A, a]$.

First and Follow sets

- Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.
- Repeat the following two steps for each nonterminal A and each production $A \rightarrow \alpha$:
  1. For each token a in First(α), add $A \rightarrow \alpha$ to the entry $M[A, a]$.
  2. If ε is in First(α), for each element a of Follow(A) (where a is a token or a is \$), add $A \rightarrow \alpha$ to $M[A, a]$.

LL(1) parsing table construction with First and Follow sets

- Now consider the grammar $S \rightarrow (\ S\ )\ S \mid \varepsilon$

LL(1) parsing table construction with First and Follow sets

- Now consider the grammar $S \rightarrow (\ S\ )\ S \mid \varepsilon$
- In this case we have that
  First($(S)S$) $= \{\ (\ \}$
  First(ε) $= \{\varepsilon\}$
  Follow(S) $= \{\ ), \$\ \}$

LL(1) parsing table construction with First and Follow sets

- Now consider the grammar $S \rightarrow (\ S \ ) \ S \mid \varepsilon$
- In this case we have that
  First($(S)S$) $= \{ \ (\ \}$
  First(ε) $= \{\varepsilon\}$
  Follow(S) $= \{ \ ), \$ \ \}$
- Thus we get the following LL(1) parsing table:

$M[N,T]$	(	)	\$
S	$S \rightarrow (S)S$	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$

First sets - definition

- Let X be a grammar symbol (a terminal or nonterminal) or ε. Then the set **First**(X) consisting of terminals, and possibly ε, is defined as follows:

First sets - definition

- Let X be a grammar symbol (a terminal or nonterminal) or ε. Then the set **First(X)** consisting of terminals, and possibly ε, is defined as follows:
  1. If X is a terminal or ε, $\text{First}(X) = \{X\}$.

First sets - definition

- Let X be a grammar symbol (a terminal or nonterminal) or ε. Then the set **First(X)** consisting of terminals, and possibly ε, is defined as follows:
  1. If X is a terminal or ε, First(X) $= \{X\}$.
  2. If X is a n nonterminal, then for each production choice $X \rightarrow X_1X_2...X_n$, First($X$) contains First($X_1$)$-\{\varepsilon\}$. If also for some $i < n$, all the sets First(X_1),...,First(X_i) contains ε, then First(X) contains First(X_{i+1})$-\{\varepsilon\}$. If all the sets First(X_1),...,First(X_n) contains ε, then First(X) also contains ε.

First sets - definition continue

- We now define **First(α)** for any string $\alpha = X_1 X_2 ... X_n$ (a string of terminals and nonterminals) as follows:

First sets - definition continue

- We now define **First(α)** for any string $\alpha = X_1X_2...X_n$ (a string of terminals and nonterminals) as follows:
  1. First(α) contains First(X_1)$-\{\varepsilon\}$.

First sets - definition continue

- We now define **First(α)** for any string $\alpha = X_1 X_2 ... X_n$ (a string of terminals and nonterminals) as follows:
  1. First(α) contains First(X_1)$-\{\varepsilon\}$.
  2. For each $i = 2, ..., n$, if First(X_k) contains ε for all $k = 1, ..., i-1$, then First(α) contains First(X_i)$-\{\varepsilon\}$.

First sets - definition continue

- We now define **First(α)** for any string $\alpha = X_1 X_2 ... X_n$ (a string of terminals and nonterminals) as follows:
  1. First(α) contains First(X_1)$-\{\varepsilon\}$.
  2. For each $i = 2, ..., n$, if First(X_k) contains ε for all $k = 1, ..., i-1$, then First(α) contains First(X_i)$-\{\varepsilon\}$.
  3. Finally, if for all $i = 1, ..., n$, First(X_i) contains ε, then First(α) contains ε.

Algorithm for computing First(A) for nonterminals A

- **for** all nonterminals A **do** First(A):=$\{\}$;
  while there are changes to any First(A) **do**
      for each production choice $A \rightarrow X_1 \ldots X_n$ **do**
          $k := 1$; Continue:=true;
          while Continue = true **and** $k <= n$ **do**
              add First(X_k)$-\{\varepsilon\}$ to First(A);
              if ε not in First(X_k) **then** Continue:=false;
              $k := k + 1$;
          if Continue = true **then** add ε to First(A);

Algorithm for computing First(A) for nonterminals A

- **for** all nonterminals A **do** First(A):=$\{\}$;
  while there are changes to any First(A) **do**
  for each production choice $A \rightarrow X_1 \ldots X_n$ **do**
  $k := 1$; Continue:=true;
  while Continue = true **and** $k <= n$ **do**
  add First(X_k)$-\{\varepsilon\}$ to First(A);
  if ε not in First(X_k) **then** Continue:=false;
  $k := k + 1$;
  if Continue = true **then** add ε to First(A);
- Simplified algorithm for First Sets in the absence of ε-productions:

Algorithm for computing First(A) for nonterminals A

- **for** all nonterminals A **do** First(A):=$\{\}$;
  while there are changes to any First(A) **do**
      for each production choice $A \rightarrow X_1 \ldots X_n$ **do**
          $k := 1$; Continue:=true;
          while Continue = true **and** $k <= n$ **do**
              add First(X_k)$-\{\varepsilon\}$ to First(A);
              if ε not in First(X_k) **then** Continue:=false;
              $k := k + 1$;
          if Continue = true **then** add ε to First(A);
- Simplified algorithm for First Sets in the absence of ε-productions:
  for all nonterminals A **do** First(A):=$\{\}$;
  while there are changes to any First(A) **do**
      for each production choice $A \rightarrow X_1 \ldots X_n$ **do**
          add First(X_1) to First(A);

First sets example 1

- Consider the simple integer expression grammar:
  exp $\longrightarrow$ *exp addop term* | *term*
  addop $\longrightarrow$ $+$ | $-$
  term $\longrightarrow$ *term mulop factor* | *factor*
  mulop $\longrightarrow$ $*$
  factor $\longrightarrow$ (*exp*) | **number**

First sets example 1

- Consider the simple integer expression grammar:
  exp $\longrightarrow$ *exp addop term* | *term*
  addop $\longrightarrow$ $+$ | $-$
  term $\longrightarrow$ *term mulop factor* | *factor*
  mulop $\longrightarrow$ $*$
  factor $\longrightarrow$ (*exp*) | **number**

-

Grammar rule	Pass 1	Pass 2	Pass 3
exp $\longrightarrow$ *exp addop term*			
exp $\longrightarrow$ *term*			First(*exp*) = {(, **number**}
addop $\longrightarrow$ $+$	First(*addop*) $= \{+\}$		
addop $\longrightarrow$ $-$	First(*addop*) $= \{+, -\}$		
term $\longrightarrow$ *term mulop factor*			
term $\longrightarrow$ *factor*		First(*term*)= {(, **number**}	
mulop $\longrightarrow$ $*$	First(*mulop*) $= \{*\}$		
factor $\longrightarrow$ (*exp*)	First(*factor*) = { (}		
factor $\longrightarrow$ **number**	First(*factor*) = {(, **number**}		

First sets example 1 continue

- Thus:
  $\text{First}(\mathit{exp}) = \{(, \textbf{number}\}$
  $\text{First}(\mathit{term}) = \{(, \textbf{number}\}$
  $\text{First}(\mathit{factor}) = \{(, \textbf{number}\}$
  $\text{First}(\mathit{addop}) = \{+, -\}$
  $\text{First}(\mathit{mulop}) = \{*\}$

First sets example 2

- Consider the grammar:
  statement $\longrightarrow$ *if-stmt* | **other**
  if-stmt $\longrightarrow$ **if** (*exp*) *statement* *else-part*
  else-part $\longrightarrow$ **else** *statement* | ε
  exp $\longrightarrow$ 0 | 1

First sets example 2

- Consider the grammar:
  statement $\longrightarrow$ *if-stmt* | **other**
  if-stmt $\longrightarrow$ **if** (*exp*) *statement else-part*
  else-part $\longrightarrow$ **else** *statement* | ε
  exp $\longrightarrow$ 0 | 1

Grammar rule	Pass 1	Pass 2
statement $\longrightarrow$ *if-stmt*		First(*statement*)= {**if**, **other**}
statement $\longrightarrow$ **other**	First(*statement*) = {**other**}	
if-stmt $\longrightarrow$ **if**(*exp*) *statement else-part*	First(*if-stmt*)= = {**if**}	
else-part $\longrightarrow$ **else** *statement*	First(*else-part*)= = {**else**}	
else-part $\longrightarrow$ ε	First(*else-part*)= = {**else**, ε}	
exp $\longrightarrow$ 0	First(*exp*)={0}	
exp $\longrightarrow$ 1	First(*exp*)={0, 1}	

First sets example 2 continue

- Thus:
  First(*statement*) $= \{\textbf{if}, \textbf{other}\}$
  First(*if-stmt*) $= \{\textbf{if}\}$
  First(*else-part*) $= \{\textbf{else}, \varepsilon\}$
  First(*exp*) $= \{0, 1\}$

Follow sets - definition

- Given a nonterminal A, the follow set **Follow(A)**, consisting of terminals, and possibly \$, is defined as follows:

Follow sets - definition

- Given a nonterminal A, the follow set **Follow(A)**, consisting of terminals, and possibly \$, is defined as follows:
    1. If A is the start symbol, then \$ is in Follow(A).

Follow sets - definition

- Given a nonterminal A, the follow set **Follow(A)**, consisting of terminals, and possibly \$, is defined as follows:
    1. If A is the start symbol, then \$ is in Follow(A).
    2. If there is a production $B \to \alpha A \gamma$, then First(γ)$-\{\varepsilon\}$ is in Follow(A).

Follow sets - definition

- Given a nonterminal A, the follow set **Follow(A)**, consisting of terminals, and possibly \$, is defined as follows:
  1. If A is the start symbol, then \$ is in Follow(A).
  2. If there is a production $B \to \alpha A \gamma$, then First(γ)$-\{\varepsilon\}$ is in Follow(A).
  3. If there is a production $B \to \alpha A \gamma$ such that ε is in First(γ), then Follow(A) contains Follow(B).

Algorithm for the computation of Follow Sets

- Follow(start-symbol):= $\{\$\}$;
  for all nonterminals $A \neq$ start-symbol **do** Follow(A):=$\{\}$;
  while there are changes to any Follow sets **do**
      for each production $A \longrightarrow X_1 \ldots X_n$ **do**
          for each X_i that is a nonterminal **do**
              add First($X_{i+1} \ldots X_n$)$-\{\varepsilon\}$ to Follow(X_i)
              (* Note: if $i = n$, then $X_{i+1} \ldots X_n = \varepsilon$ *)
              if ε is in First($X_{i+1} \ldots X_n$) **then**
                  add Follow(A) to Follow(X_i)

Follow sets example 1

- We consider again the grammar:

  $exp \rightarrow exp\ addop\ term \mid term$

  $addop \rightarrow + \mid -$

  $term \rightarrow term\ mulop\ factor \mid factor$

  $mulop \rightarrow *$

  $factor \rightarrow (\ exp\ ) \mid \textbf{number}$

Follow sets example 1

- We consider again the grammar:
  $exp \rightarrow exp\ addop\ term \mid term$
  $addop \rightarrow + \mid -$
  $term \rightarrow term\ mulop\ factor \mid factor$
  $mulop \rightarrow *$
  $factor \rightarrow (\ exp\ ) \mid \textbf{number}$
- Recall that:
  $\text{First}(exp) = \{(, \textbf{number}\}$
  $\text{First}(term) = \{(, \textbf{number}\}$
  $\text{First}(factor) = \{(, \textbf{number}\}$
  $\text{First}(addop) = \{+, -\}$
  $\text{First}(mulop) = \{*\}$

Follow sets example 1 continue

- In the computation of the Follow sets for the grammar we omit the four grammar rule choices that have no possibility of affecting the computation.

Follow sets example 1 continue

- In the computation of the Follow sets for the grammar we omit the four grammar rule choices that have no possibility of affecting the computation.

Grammar rule	Pass 1	Pass 2
exp $\longrightarrow$ *exp addop term*	Follow(*exp*)= $\{\$, +, -\}$ Follow(*addop*)= $\{($, **number**$\}$ Follow(*term*)= $\{\$, +, -\}$	Follow(*term*)= $\{\$, +, -, *,)\}$
exp $\longrightarrow$ *term*		
term $\longrightarrow$ *term mulop factor*	Follow(*term*)= $\{\$, +, -, *\}$ Follow(*mulop*)= $\{($, **number**$\}$ Follow(*factor*) = $\{\$, +, -, *\}$	Follow(*factor*)= $\{\$, +, -, *,)\}$
term $\longrightarrow$ *factor*		
factor $\longrightarrow$ (*exp*)	Follow(*exp*) $= \{ \$, +, -,)\}$	

Follow sets example 1 continue

- Thus:
  $\text{Follow}(\mathit{exp}) = \{\$, +, -,)\}$
  $\text{Follow}(\mathit{term}) = \{\$, +, -, *,)\}$
  $\text{Follow}(\mathit{factor}) = \{\$, +, -, *,)\}$
  $\text{Follow}(\mathit{addop}) = \{(, \textbf{number}\}$
  $\text{Follow}(\mathit{mulop}) = \{(, \textbf{number}\}$

LL(1) parsing example

- $\mathit{statement} \rightarrow \mathit{if\text{-}stmt} \mid \textbf{other}$
  $\mathit{if\text{-}stmt} \rightarrow \textbf{if}\ (\ \mathit{exp}\ )\ \mathit{statement}\ \mathit{else\text{-}part}$
  $\mathit{else\text{-}part} \rightarrow \textbf{else}\ \mathit{statement} \mid \varepsilon$
  $\mathit{exp} \rightarrow 0 \mid 1$

LL(1) parsing example

- $\textit{statement} \rightarrow \textit{if-stmt} \mid \textbf{other}$
  $\textit{if-stmt} \rightarrow \textbf{if} \ (\ \textit{exp} \ ) \ \textit{statement} \ \textit{else-part}$
  $\textit{else-part} \rightarrow \textbf{else} \ \textit{statement} \mid \varepsilon$
  $\textit{exp} \rightarrow 0 \mid 1$
- Recall that:
  $\text{First}(\textit{statement}) = \{\textbf{if}, \textbf{other}\}$
  $\text{First}(\textit{if-stmt}) = \{\textbf{if}\}$
  $\text{First}(\textit{else-part}) = \{\textbf{else}, \varepsilon\}$
  $\text{First}(\textit{exp}) = \{0, 1\}$

LL(1) parsing example

- $\textit{statement} \rightarrow \textit{if-stmt} \mid \textbf{other}$
  $\textit{if-stmt} \rightarrow \textbf{if} \ (\ \textit{exp} \ ) \ \textit{statement} \ \textit{else-part}$
  $\textit{else-part} \rightarrow \textbf{else} \ \textit{statement} \mid \varepsilon$
  $\textit{exp} \rightarrow 0 \mid 1$
- Recall that:
  $\text{First}(\textit{statement}) = \{\textbf{if}, \textbf{other}\}$
  $\text{First}(\textit{if-stmt}) = \{\textbf{if}\}$
  $\text{First}(\textit{else-part}) = \{\textbf{else}, \varepsilon\}$
  $\text{First}(\textit{exp}) = \{0, 1\}$
- One can verify that:
  $\text{Follow}(\textit{statement}) = \{\$, \textbf{else}\}$
  $\text{Follow}(\textit{if-stmt}) = \{\$, \textbf{else}\}$
  $\text{Follow}(\textit{else-part}) = \{\$, \textbf{else}\}$
  $\text{Follow}(\textit{exp}) = \{ \ ) \ \}$

LL(1) parsing example continue

- Recall that the LL(1) parsing table $M[N, T]$ is constructed by repeating the following two steps for each nonterminal A and each production $A \rightarrow \alpha$:

LL(1) parsing example continue

- Recall that the LL(1) parsing table $M[N, T]$ is constructed by repeating the following two steps for each nonterminal A and each production $A \rightarrow \alpha$:
  1. For each token a in First(α), add $A \rightarrow \alpha$ to the entry $M[A, a]$.

LL(1) parsing example continue

- Recall that the LL(1) parsing table $M[N, T]$ is constructed by repeating the following two steps for each nonterminal A and each production $A \rightarrow \alpha$:
  1. For each token a in First(α), add $A \rightarrow \alpha$ to the entry $M[A, a]$.
  2. If ε is in First(α), for each element a of Follow(A) (where a is a token or a is \$), add $A \rightarrow \alpha$ to $M[A, a]$.

LL(1) parsing example continue

- Using the procedure on the previous slide, we obtain the following table:

LL(1) parsing example continue

- Using the procedure on the previous slide, we obtain the following table:

$M[N, T]$	**if**	**other**	**else**	0	1	\$
statement	*statement* $\rightarrow$ *if-stmt*	*statement* $\rightarrow$ **other**				
if-stmt	*if-stmt* $\rightarrow$ **if** (*exp*) *statement* *else-part*					
else-part			*else-part* $\rightarrow$ **else** *statement* *else-part* $\rightarrow \varepsilon$			*else-part* $\rightarrow \varepsilon$
exp				*exp* $\rightarrow$ 0	*exp* $\rightarrow$ 1	

LL(1) parsing example continue

- We notice, that as expected, this grammar is not LL(1), since the entry M[*else-part*,**else**] contains two entries, corresponding to the dangling else ambiguity.

LL(1) parsing example continue

- We notice, that as expected, this grammar is not LL(1), since the entry M[*else-part*,**else**] contains two entries, corresponding to the dangling else ambiguity.
- We could apply the disambiguating rule that would always prefer the rule that generates the current lookahead token over any other (this corresponds to the most closely nested disambiguating rule), and thus the production *else-part* $\rightarrow$ **else** *statement*

LL(1) parsing example continue

- We notice, that as expected, this grammar is not LL(1), since the entry M[*else-part*,**else**] contains two entries, corresponding to the dangling else ambiguity.
- We could apply the disambiguating rule that would always prefer the rule that generates the current lookahead token over any other (this corresponds to the most closely nested disambiguating rule), and thus the production *else-part* $\longrightarrow$ **else** *statement*
- We now show the LL(1) parsing actions for the string **if** (0) **if**(1) **other else other**

LL(1) parsing example continue

- We notice, that as expected, this grammar is not LL(1), since the entry M[*else-part*,**else**] contains two entries, corresponding to the dangling else ambiguity.
- We could apply the disambiguating rule that would always prefer the rule that generates the current lookahead token over any other (this corresponds to the most closely nested disambiguating rule), and thus the production
  else-part $\longrightarrow$ **else** *statement*
- We now show the LL(1) parsing actions for the string
  if (0) **if**(1) **other else other**
- We use the following abbreviations:
  statement = *S*
  if-stmt = *I*
  else-part = *L*
  exp = *E*
  if = **i**
  else = **e**
  other = **o**

LL(1) parsing example continue

Parsing stack	Input	Action
$\$S$	**i** (0) **i** (1) **o e o**\$	$S \longrightarrow I$
$\$I$	**i** (0) **i** (1) **o e o**\$	$I \longrightarrow$ **i** (E) S L
$\$LS)E($**i**	**i** (0) **i** (1) **o e o**\$	match
$\$LS)E($	(0) **i** (1) **o e o**\$	match
$\$LS)E$	0) **i** (1) **o e o**\$	$E \longrightarrow 0$
$\$LS)0$	0) **i** (1) **o e o**\$	match
$\$LS)$	) **i** (1) **o e o**\$	match
$\$LS$	**i** (1) **o e o**\$	$S \longrightarrow I$
$\$LI$	**i** (1) **o e o**\$	$I \longrightarrow$ **i** (E) S L
$\$LLS)E($**i**	**i** (1) **o e o**\$	match
$\$LLS)E($	(1) **o e o**\$	match
$\$LLS)E$	1) **o e o**\$	$E \longrightarrow 1$
$\$LLS)1$	1) **o e o**\$	match
$\$LLS)$	) **o e o**\$	match
$\$LLS$	**o e o**\$	$S \longrightarrow$ **o**
$\$LL$**o**	**o e o**\$	match
$\$LL$	**e o**\$	$L \longrightarrow$ **e** S
$\$LS$**e**	**e o**\$	match
$\$LS$	**o**\$	$S \longrightarrow$ **o**
$\$L$**o**	**o**\$	match
$\$L$	\$	$L \longrightarrow \varepsilon$
\$	\$	accept

LL(1) parsing in JFLAP

- Finally we consider the LL(1) parsing example discussed in the JFLAP tutorial at http://www.jflap.org/tutorial/

LL(1) parsing in JFLAP

- Finally we consider the LL(1) parsing example discussed in the JFLAP tutorial at http://www.jflap.org/tutorial/
- We use JFLAP to calculate First and Follow sets and the LL(1) parse table for the grammar:

  $S \rightarrow a\ A\ B\ b$

  $A \rightarrow a\ A\ c$

  $A \rightarrow \lambda$

  $B \rightarrow b\ B$

  $B \rightarrow c$

LL(1) parsing in JFLAP

- Finally we consider the LL(1) parsing example discussed in the JFLAP tutorial at http://www.jflap.org/tutorial/
- We use JFLAP to calculate First and Follow sets and the LL(1) parse table for the grammar:
  $S \rightarrow a\ A\ B\ b$
  $A \rightarrow a\ A\ c$
  $A \rightarrow \lambda$
  $B \rightarrow b\ B$
  $B \rightarrow c$
- Now we use JFLAP to parse *aacbbcb*