Programming Paradigms
Lecture 4: Syntax (Part 3)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2019/2020
Wake-up Exercise

What does the following Perl code print?

```
$b = 0;
sub foo {
    return $b;
}
sub bar {
    local $b = 1;
    return foo();
}
print bar();
```

https://ilias3.uni-stuttgart.de/vote/0ZT9
Wake-up Exercise

What does the following Perl code print?

```perl
$b = 0;
sub foo {
    return $b;
}
sub bar {
    local $b = 1;
    return foo();
}
print bar();
```

Result: 1

https://ilias3.uni-stuttgart.de/vote/0ZT9
Wake-up Exercise

What does the following Perl code print?

```perl
$b = 0;
sub foo { return $b; }
sub bar { local $b = 1; return foo(); }
print bar();
```

Result: 1

Global variable that would be used with static scoping
Wake-up Exercise

What does the following Perl code print?

```perl
$b = 0;
sub foo { 
    return $b;
}
sub bar { 
    local $b = 1;
    return foo();
}
print bar();
```

- But: Perl has dynamic scoping
- Uses last encountered definition of $b

Result: 1

https://ilias3.uni-stuttgart.de/vote/0ZT9
Overview

- Specifying syntax
  - Regular expressions
  - Context-free grammars
- Scanning
- Parsing
  - Top-down parsing
  - Bottom-up parsing
How to Automate This?

To generate an LL(k) parser, need to predict which rule to apply

Compute PREDICT sets for all productions, based on two helpers

- **FIRST(N)**: What terminals come first when expanding non-terminal N?
- **FOLLOW(N)**: What terminals follow after non-terminal N?
PREDICT Sets

PREDICT set for a rule: Which terminals to look for in LL(1) parser

- If next input token is in PREDICT of rule, apply the rule

- Computing the PREDICT set for rule $A \rightarrow \alpha$:
  - If $\epsilon$ in FIRST($A$):
    \[
    \text{PREDICT}(A \rightarrow \alpha) = (\text{FIRST}(\alpha) - \{ \epsilon \}) \cup \text{FOLLOW}(A)
    \]
  - Otherwise:
    \[
    \text{PREDICT}(A \rightarrow \alpha) = \text{FIRST}(\alpha)
    \]
Example

Grammar:

<table>
<thead>
<tr>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>a, b</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
</tr>
</tbody>
</table>

- \[ S \rightarrow a \; B \]
- \[ S \rightarrow b \; C \]
- \[ B \rightarrow b \; b \; C \]
- \[ C \rightarrow c \; c \]
Example

Grammar:

\[
\begin{align*}
S & \rightarrow a \ B \\
S & \rightarrow b \ C \\
B & \rightarrow b \ b \ C \\
C & \rightarrow c \ c \\
\end{align*}
\]

FIRST & FOLLOW

<table>
<thead>
<tr>
<th></th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>a, b</td>
<td>EOF</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>EOF</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Example

Grammar:

\[
\begin{align*}
S & \rightarrow a \ B \\
S & \rightarrow b \ C \\
B & \rightarrow b \ b \ C \\
C & \rightarrow c \ c
\end{align*}
\]

PREDICT:

\[
\begin{align*}
S() & \{ \text{a} \} \\
S() & \{ \text{b} \} \\
B() & \{ \text{b} \} \\
C() & \{ \text{c} \}
\end{align*}
\]

FIRST FOLLOW

\[
\begin{array}{ll}
S & \text{a, b EOF} \\
B & \text{b EOF} \\
C & \text{c EOF}
\end{array}
\]
Example

Grammar:

- \( S \rightarrow a \; B \)
- \( S \rightarrow b \; C \)
- \( B \rightarrow b \; b \; C \)
- \( C \rightarrow c \; c \)

Python Code:

```python
S() {
    if (inputToken == a)
        match(a); B();
    else if (inputToken == b)
        match(b); C();
    else error();
}
B() {
    if (inputToken == b)
        match(b); match(b); C();
    else error();
}
C() {
    if (inputToken == c)
        match(c); match(c);
    else error();
}
```

<table>
<thead>
<tr>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S a, b EOF</td>
<td></td>
</tr>
<tr>
<td>B b EOF</td>
<td></td>
</tr>
<tr>
<td>C c EOF</td>
<td></td>
</tr>
</tbody>
</table>

FIRST FOLLOW

- \( S \rightarrow a, b \) EOF
- \( B \rightarrow b \) EOF
- \( C \rightarrow c \) EOF

PREDICT:

- \( \{ a \} \)
- \( \{ b \} \)
- \( \{ b \} \)
- \( \{ c \} \)
Example

Grammar:

\[
\begin{align*}
S & \rightarrow a \ B \\
S & \rightarrow b \ C \\
B & \rightarrow b \ b \ C \\
C & \rightarrow c \ c
\end{align*}
\]

PREDICT:

\[
\begin{align*}
S() \{ & a \\
S() \{ & b \\
B() \{ & b \\
C() \{ & c
\end{align*}
\]

FIRST FOLLOW

\[
\begin{array}{c|c}
S & a, b \text{ EOF} \\
B & b \text{ EOF} \\
C & c \text{ EOF}
\end{array}
\]
Computing the Parse Table

Computing an LL(1) parse table

- Given: PREDICT set of each rule
- Table is a mapping \( M: N \times T \rightarrow \) Production rule or error
- For all productions \( A \rightarrow \alpha \) do
  - For each terminal \( t \) in PREDICT\((A \rightarrow \alpha)\):
    - \( M[A][t] = A \rightarrow \alpha \)
  - Every undefined table entry is an error
**Example: Parse Table**

<table>
<thead>
<tr>
<th>Non-term.</th>
<th>Term.</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
</tbody>
</table>

1) \( S \rightarrow aB \)
2) \( S \rightarrow bC \)
3) \( B \rightarrow b b C \)
4) \( C \rightarrow c c \)
Table-based, Predictive Parsing

stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
  until x is EOF
Table-based, Predictive Parsing

Parse stack: Prediction of what will be seen in the future

```java
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
    x = stack.pop();
    if x is terminal or EOF
        if x == nextToken
            nextToken = lookAhead()
        else error()
    else // x is non-terminal
        if M[x][nextToken] == x -> y1 y2 .. ym
            stack.push(ym); ...; stack.push(y1);
        else error()
until x is EOF
```
Table-based, Predictive Parsing

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
  until x is EOF
```

Read one token after another, always looking only one token ahead
Table-based, Predictive Parsing

Check if expected terminal is indeed the next token

```
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
  until x is EOF
```
Table-based, Predictive Parsing

stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
  x = stack.pop();
  if x is terminal or EOF
    if x == nextToken
      nextToken = lookAhead()
    else error()
  else // x is non-terminal
    if M[x][nextToken] == x -> y1 y2 .. ym
      stack.push(ym); ...; stack.push(y1);
    else error()
  until x is EOF
Table-based, Predictive Parsing

```c
stack.push(EOF); stack.push(startSymbol);
nextToken = lookAhead();
repeat
    x = stack.pop();
    if x is terminal or EOF
        if x == nextToken
            nextToken = lookAhead()
        else error()
    else // x is non-terminal
        if M[x][nextToken] == x -> y1 y2 .. ym
            stack.push(ym); ...; stack.push(y1);
        else error()
    until x is EOF
```

No entry in table:
Raise error
Example: Table-based Parsing

Input: bcc

<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining input</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF, S</td>
<td>b, c, c, EOF</td>
<td>Pop S</td>
</tr>
<tr>
<td>EOF, C, b</td>
<td>b, c, c, EOF</td>
<td>Use rule S → b C</td>
</tr>
<tr>
<td>EOF, c</td>
<td>c, c, EOF</td>
<td>Push C, b</td>
</tr>
<tr>
<td>EOF, c, c</td>
<td>c, c, EOF</td>
<td>Pop b</td>
</tr>
<tr>
<td>EOF, c</td>
<td>c, c, EOF</td>
<td>Read next token</td>
</tr>
<tr>
<td>EOF, c</td>
<td>c, c, EOF</td>
<td>Push c, c</td>
</tr>
<tr>
<td>EOF, c</td>
<td>c, EOF</td>
<td>Pop c</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>Read next token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pop c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read next token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pop EOF</td>
</tr>
</tbody>
</table>
Plan for Today

- Specifying syntax
  - Regular expressions
  - Context-free grammars
- Scanning
- Parsing
  - Top-down parsing
  - Bottom-up parsing
Bottom-up Parsing

- **LR(k) parsers**
  - Left-to-right scanning, Right-most derivation, \( k \) tokens look-ahead

- Difficult to do by hand

- Mostly based on **automatically generated table**
Shift-reduce Algorithm

- **Repeat** until all tokens read and all symbols reduced to start symbol:
  - Shift (i.e., read) input tokens
  - Try to reduce a group of symbols into a single non-terminal
Example: Shift - Reduce Parsing

\[ S \rightarrow a\,T\,Re \]
\[ T \rightarrow T\,bc \mid b \]
\[ R \rightarrow d \]

Steps:
- Shift \( a \), shift \( b \)
- Reduce \( T \rightarrow b \)
- Shift \( b \), shift \( c \)
- Reduce \( T \rightarrow T\,bc \)
- Shift \( d \)
- Reduce \( R \rightarrow d \)
- Shift \( e \)
- Reduce \( S \rightarrow a\,T\,Re \)
Table-based LR Parsing

- **Two tables**
  - **Action table:**
    \[
    \text{state} \times T \rightarrow \text{reduce/shift/accept/error}
    \]
  - **Goto table:**
    \[
    \text{state} \times N \rightarrow \text{state}
    \]
- **Stack of symbol/state pairs**
  - Record of what has been seen in the past
**Example: LR(1) Table**

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>s2</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>s3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Example: LR(1) Table

### Action Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td></td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

16 - 2
**Example: LR(1) Table**

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>s2</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>s3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goto table**
**Example: LR(1) Table**

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td></td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td></td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>r3</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **s** means shift to some state.
Example: LR(1) Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td></td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

r means reduce using some production
Example: LR(1) Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td></td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td>s6</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>r3</td>
<td></td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accept input (i.e., done with parsing)
### Example: LR(1) Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>EOF</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td></td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td></td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.</td>
<td>acc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td></td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No entry means error
Table-based LR(1) Parsing

stack.push(EOF, 0);
nextToken = lookAhead();
repeat
    s = state on top of stack
    if action[s, nextToken] = shift s'
        stack.push(nextToken, s');
        nextToken = lookAhead();
    else if action[s, nextToken] = reduce x -> y1 .. ym
        pop m pairs from stack
        s' = state on top of stack
        push(x, goto[s', x])
    else if action[s, nextToken] = accept
        accept and return
    else error()
Table-based LR(1) Parsing

Stack hold roots of partial trees found so far

```
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
    s = state on top of stack
    if action[s, nextToken] = shift s'
        stack.push(nextToken, s');
    nextToken = lookAhead();
    else if action[s, nextToken] = reduce x -> y1 .. ym
        pop m pairs from stack
        s' = state on top of stack
        push(x, goto[s', x])
    else if action[s, nextToken] = accept
        accept and return
    else error()
```
Table-based LR(1) Parsing

Reduce partial trees into a non-terminal by applying a rule

```plaintext
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
    s = state on top of stack
    if action[s, nextToken] = shift s'
        stack.push(nextToken, s');
        nextToken = lookAhead();
    else if action[s, nextToken] = reduce x -> y1 .. ym
        pop m pairs from stack
        s' = state on top of stack
        push(x, goto[s', x])
    else if action[s, nextToken] = accept
        accept and return
    else error()
```

Reduce partial trees into a non-terminal by applying a rule
Table-based LR(1) Parsing

```plaintext
stack.push(EOF, 0);
nextToken = lookAhead();
repeat
    s = state on top of stack
    if action[s, nextToken] = shift s'
        stack.push(nextToken, s');
        nextToken = lookAhead();
    else if action[s, nextToken] = reduce x -> y1 .. ym
        pop m pairs from stack
        s' = state on top of stack
        push(x, goto[s', x])
    else if action[s, nextToken] = accept
        accept and return
    else error()
Read another token
```

Table-based LR(1) Parsing

stack.push(EOF, 0);
nextToken = lookAhead();
repeat
   s = state on top of stack
   if action[s, nextToken] = shift s'
      stack.push(nextToken, s');
      nextToken = lookAhead();
   else if action[s, nextToken] = reduce x -> y1 .. ym
      pop m pairs from stack
      s' = state on top of stack
      push(x, goto[s', x])
   else if action[s, nextToken] = accept
      accept and return
   else error()

All subtrees reduced to start symbol
How to Get the Table?

- Using a “characteristic finite-state machine” computed from the grammar
- Details differ for different kinds of LR parsers
  - SLR (simple LR)
  - LALR (look-ahead LR)
  - Full-LR
- Beyond the scope of this course
Quiz: Parsing

Which of these statements is true?

- Recursive descent builds a parse tree from the bottom up.
- The k in LR(k) stands for k tokens look-ahead.
- PREDICT sets are used to compute FIRST and FOLLOW sets.
- The stack of a top-down parser contains the symbols expected in the future.

https://ilias3.uni-stuttgart.de/vote/0ZT9
Quiz: Parsing

Which of these statements is true?

- Recursive descent builds a parse tree from the bottom up.
- The k in LR(k) stands for k tokens look-ahead.
- PREDICT sets are used to compute FIRST and FOLLOW sets.
- The stack of a top-down parser contains the symbols expected in the future.

https://ilias3.uni-stuttgart.de/vote/0ZT9
Plan for Today

- Specifying syntax
  - Regular expressions
  - Context-free grammars

- Scanning

- Parsing
  - Top-down parsing
  - Bottom-up parsing