Topics in IT 1

Parsing and Pattern Recognition

Week 14

Nullable Rules in
Top-Down Table-Driven Parsers

College of Information Science and Engineering
Ritsumeikan University
review: table-driven parsing

goal:
  • given a grammar, construct a deterministic parser

approach:
  • scan input left-to-right
  • keep the current sentential form of the derivation on a stack
  • repeatedly
    – match a terminal token on the stack with the next input token
    or
    – replace a non-terminal with with one of its productions
  when replacing a non-terminal with one of its productions
    • deterministically choose the correct production
    • based on the non-terminal and the next input token
    • i.e., $N \times t \rightarrow \alpha$

to do this we construct a table: $\text{PREDICT}[N, t] = \alpha$
review: table-driven parsing

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Parser Stack</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREDICT[ Sess, !] = Fact Sess</td>
<td>Sess ! x &lt; y ? x = y</td>
<td>Sess ! x &lt; y ? x = y</td>
</tr>
<tr>
<td>PREDICT[ Fact, !] = ! str</td>
<td>Fact Sess ! x &lt; y ? x = y</td>
<td>Fact Sess ! x &lt; y ? x = y</td>
</tr>
<tr>
<td>PREDICT[ Sess, ?] = ! str</td>
<td>Sess ! x &lt; y ? x = y ? x = y</td>
<td>Sess ! x &lt; y ? x = y</td>
</tr>
<tr>
<td></td>
<td>str Sess ! x &lt; y ? x = y</td>
<td>str ! x &lt; y ? x = y ? x = y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>! x &lt; y ? x = y</td>
</tr>
</tbody>
</table>

Session → Fact Session
Session → Question
Session → (Session) Session
Fact → ! string
a Question → ? string

Session
Fact
Question
Session
Fact
Question

PREDICT[ N, t ] = ! ? ( )

Session
Fact Sess
Ques
(Sess) Sess
Session
Fact
! str
Question
? str
review: table-driven parsing

Let \( t \) stand for a terminal symbol, \( A \) and \( B \) stand for a non-terminals, and \( \alpha \) and \( \beta \) be any sequence of symbols (terminal or non-terminal).

To construct FIRST sets for each non-terminal, we begin with all sets being empty. Then we consider each of the productions in the grammar.

for each production \( A \rightarrow \alpha : \)

\[
\text{add } \text{FIRST}(\alpha) \text{ to } \text{FIRST}(A)
\]

where \( \text{FIRST}(\alpha) \) depends on its initial symbol:

\[
\begin{align*}
\text{if } \alpha &= t \beta \text{ then } \text{FIRST}(\alpha) &= \text{FIRST}(t \beta) = t \\
\text{if } \alpha &= B \beta \text{ then } \text{FIRST}(\alpha) &= \text{FIRST}(B \beta) = \text{FIRST}(B)
\end{align*}
\]

The process is repeated until no more elements can be added to any of the sets.

(Each iteration allows new information to ‘propagate backwards’ to sets computed earlier. For example, in \( A \rightarrow B \beta \), \( \text{FIRST}(B) \) might not have been calculated when it is added to \( \text{FIRST}(A) \).)
Grammar:

\[ G = \begin{align*}
    S & \rightarrow X Y \\
    S & \rightarrow Y X \\
    X & \rightarrow ab \\
    Y & \rightarrow ba
\end{align*} \]

FIRST sets:

<table>
<thead>
<tr>
<th>1st iteration</th>
<th>FIRST(S)</th>
<th>FIRST(X)</th>
<th>FIRST(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
review: table-driven parsing

Grammar:

\[
G = \begin{align*}
S & \rightarrow X Y \\
S & \rightarrow Y X \\
X & \rightarrow ab \\
Y & \rightarrow ba \\
\end{align*}
\]

FIRST sets:

<table>
<thead>
<tr>
<th></th>
<th>FIRST(S)</th>
<th>FIRST(X)</th>
<th>FIRST(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st iteration</td>
<td>(\emptyset)</td>
<td>({a})</td>
<td>({b})</td>
</tr>
<tr>
<td>2nd iteration</td>
<td>(?)</td>
<td>(?)</td>
<td>(?)</td>
</tr>
</tbody>
</table>
review: table-driven parsing

Grammar:

\[
\begin{align*}
G = \quad & S \rightarrow X \ Y \\
& S \rightarrow Y \ X \\
& X \rightarrow \text{ab} \\
& Y \rightarrow \text{ba}
\end{align*}
\]

FIRST sets:

<table>
<thead>
<tr>
<th></th>
<th>FIRST(S)</th>
<th>FIRST(X)</th>
<th>FIRST(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st iteration</td>
<td>\emptyset</td>
<td>{a}</td>
<td>{b}</td>
</tr>
<tr>
<td>2nd iteration</td>
<td>{a, b}</td>
<td>{a}</td>
<td>{b}</td>
</tr>
<tr>
<td>3rd iteration</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
review: table-driven parsing

Grammar:

\[
G = \begin{align*}
S & \rightarrow X Y \\
S & \rightarrow Y X \\
X & \rightarrow ab \\
Y & \rightarrow ba
\end{align*}
\]

FIRST sets:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>FIRST(S)</th>
<th>FIRST(X)</th>
<th>FIRST(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>$\emptyset$</td>
<td>${a}$</td>
<td>${b}$</td>
</tr>
<tr>
<td>2nd</td>
<td>${ab}$</td>
<td>${a}$</td>
<td>${b}$</td>
</tr>
<tr>
<td>3rd</td>
<td>${ab}$</td>
<td>${a}$</td>
<td>${b}$</td>
</tr>
<tr>
<td>finished!</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
review: table-driven parsing

When expanding a non-terminal \( A \) on top of the parser stack, the PREDICT table tells us which one of \( A \)'s right-hand sides to use when the next input token is some terminal \( t \).

To build PREDICT, consider each production in the grammar.

for each production \( A \rightarrow \alpha \):
  for each terminal symbol \( t \) in \( \text{FIRST}(\alpha) \):
    set \( \text{PREDICT}[A, t] = \alpha \)

\[
\begin{array}{c}
S \rightarrow X Y \\
S \rightarrow Y X \\
X \rightarrow a b \\
Y \rightarrow b a \\
\end{array}
\]

\[
\text{FIRST}(X Y) = \text{FIRST}(X) = \{ a \} \\
\text{FIRST}(Y X) = \text{FIRST}(Y) = \{ b \} \\
\text{FIRST}(a b) = \{ a \} \\
\text{FIRST}(b a) = \{ b \}
\]

PREDICT table for \( G \):

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>X</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Y</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
### review: table-driven parsing

For each production $A \rightarrow \alpha$:

- For each terminal symbol $t$ in $\text{FIRST}(\alpha)$:
  - Set $PREDICT[A, t] = \alpha$

---

**PREDICT table for $G$:**

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$X Y$</td>
<td>$Y X$</td>
</tr>
<tr>
<td>$X$</td>
<td>$a b$</td>
<td></td>
</tr>
<tr>
<td>$Y$</td>
<td></td>
<td>$b a$</td>
</tr>
</tbody>
</table>

---

**FIRST Calculations**:

- $S \rightarrow X Y$  \quad $\text{FIRST}(X Y) = \text{FIRST}(X) = \{ a \}$
- $S \rightarrow Y X$  \quad $\text{FIRST}(Y X) = \text{FIRST}(Y) = \{ b \}$
- $X \rightarrow a b$  \quad $\text{FIRST}(a b) = \{ a \}$
- $Y \rightarrow b a$  \quad $\text{FIRST}(a b) = \{ b \}$
motivating example

example grammar for arithmetic expressions:

\[
\begin{align*}
E & \rightarrow T \ + \ E \\
E & \rightarrow T \\
T & \rightarrow \text{int} \\
T & \rightarrow \text{int} \ * \ T \\
T & \rightarrow ( \ E )
\end{align*}
\]

this grammar is not suitable for constructing an LL(1) parsing table

- why not?
- how can we fix it?
motivating example

original

\[
\begin{align*}
E & \rightarrow T + E \\
E & \rightarrow T \\
T & \rightarrow \text{int} \\
T & \rightarrow \text{int} \ast T \\
T & \rightarrow (E)
\end{align*}
\]

left-factored

\[
\begin{align*}
E & \rightarrow T \ X \\
X & \rightarrow + \ E \\
X & \rightarrow \epsilon \\
T & \rightarrow (E) \\
T & \rightarrow \text{int} \ Y \\
Y & \rightarrow \ast \ T \\
Y & \rightarrow \epsilon
\end{align*}
\]

left factoring cured the ‘FIRST conflicts’

- but introduced its own problems

two rules now contain \(\epsilon\)-productions

- non-terminals with \(\epsilon\)-productions are called **nullable**
- they can match input tokens (e.g., \(Y \rightarrow \ast T\)), or
- they can *vanish* from the derivation *without matching anything*
problems with $\epsilon$

\[
S \rightarrow A \ b \\
A \rightarrow a \mid \epsilon
\]

what if the first non-terminal in a production can produce $\epsilon$?

- what should FIRST(S) contain?

how do we even deal with $A$ when parsing “b”?

- what is PREDICT[S,b]?
- what is PREDICT[A,b]?

we need to reconsider the definition of FIRST; e.g.

- if $B$ is nullable ($B \rightarrow^* \epsilon$) then we might have a derivation

  \[
  A \Rightarrow Ba \Rightarrow a
  \]

we have to be able to ignore $B$, if appropriate,

- when constructing FIRST($A$), *and*
- whenever we encounter it on top of the parsing stack
computation of FIRST sets with $\epsilon$

FIRST($A$) is the set of terminals that can be at the start of a string produced by $A$

\[ t \in \text{FIRST}(A) \iff A \to^* t \alpha \]

we will now allow $\epsilon$ into FIRST($A$), iff $A$ can produce the empty string

\[ \epsilon \in \text{FIRST}(A) \iff A \to^* \epsilon \]

formally:

- \[ \text{FIRST}(A) = \{ t \mid A \to^* t \alpha \} \cup \{ \epsilon \mid A \to^* \epsilon \} \]

  (the set of all terminals that can appear at the start of strings produced from $A$, including $\epsilon$ if $A$ can produce the empty string)

fortunately the algorithm to compute this is only slightly more complicated than the case where there are no $\epsilon$-productions
computing FIRST sets

again, we start by placing all productions with an initial terminal into the FIRST set of their left-hand-side non-terminal

1. if $A \rightarrow t\alpha$ then add $t$ to $\text{FIRST}(A)$

next we deal with the $\epsilon$-productions

2. if $A \rightarrow \epsilon$ then add $\epsilon$ to $\text{FIRST}(A)$

then we perform a closure operation until a fixed point is reached:

3. let $\gamma$ be a string of non-terminals $A_i$ such that $\forall i, \epsilon \in \text{FIRST}(A_i)$
   
   (i.e., the entire string $\gamma$ can produce the empty string)

   repeat:
   
   for each production add to $\text{FIRST}(A)$

   $A \rightarrow \gamma$ \hspace{1cm} $\epsilon$

   $A \rightarrow \gamma t \beta$ \hspace{1cm} $t$

   $A \rightarrow \gamma B \beta$ \hspace{1cm} $\text{FIRST}(B) - \{\epsilon\}$

   until no more additions can be made
a notational simplification

given FIRST sets for nonterminals, we can define FIRST* sets for the strings in the right-hand sides of productions:

\[ \text{FIRST}^*(\varepsilon) = \{\varepsilon\} \]
\[ \text{FIRST}^*(t\alpha) = \{t\} \]
\[ \text{FIRST}^*(A\alpha) = \text{FIRST}(A) \quad \text{if} \quad \varepsilon \notin \text{FIRST}(A) \]
\[ \text{FIRST}^*(A\alpha) = (\text{FIRST}(A) - \{\varepsilon\}) \cup \text{FIRST}^*(\alpha) \quad \text{if} \quad \varepsilon \in \text{FIRST}(A) \]

with this, the algorithm to compute FIRST sets can be written:

\begin{verbatim}
repeat
    for all \( A \rightarrow \alpha \),
        add everything in \( \text{FIRST}^*(\alpha) \) to \( \text{FIRST}(A) \)
until no further additions can be made
\end{verbatim}
example

\[ E \rightarrow TX \quad X \rightarrow +E \mid \epsilon \]
\[ T \rightarrow (E) \mid \text{int} \ Y \quad Y \rightarrow *T \mid \epsilon \]

1. if \( A \rightarrow t\alpha \) then add \( t \) to FIRST(A)

\[
\begin{array}{c|cccc}
\text{FIRST:} & E & X & T & Y \\
\hline
 & + & ( \text{int} & * \\
\end{array}
\]

2. if \( A \rightarrow \epsilon \) then add \( \epsilon \) to FIRST(A)

\[
\begin{array}{c|cccc}
\text{FIRST:} & E & X & T & Y \\
\hline
 & + \epsilon & ( \text{int} & * \epsilon \\
\end{array}
\]
example

\[
E \to TX \\
X \to + E | \epsilon \\
T \to (E) | \text{int} Y \\
Y \to * T | \epsilon
\]

3. repeat

for each production \( A \to \alpha \):

add everything in FIRST*(\( \alpha \)) to FIRST(\( A \))

until no more additions can be made

<table>
<thead>
<tr>
<th>FIRST:</th>
<th>E</th>
<th>X</th>
<th>T</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>+ ( \epsilon )</td>
<td>( \text{int}</td>
</tr>
<tr>
<td>FIRST*(TX) = FIRST(T) = { \text{int} }</td>
<td>( \text{int}</td>
<td>\epsilon</td>
<td>( \text{int} )</td>
<td>\epsilon)</td>
</tr>
<tr>
<td>done</td>
<td>( \text{int}</td>
<td>\epsilon</td>
<td>( \text{int} )</td>
<td>\epsilon)</td>
</tr>
</tbody>
</table>

more complications with $\epsilon$

non-terminals can be nullable

- they produce $\epsilon$

consider a grammar

$$S \rightarrow \alpha \ A \ \beta$$
$$A \rightarrow \epsilon \mid \ldots$$

if $A$ is on top of the stack, predict which production to replace it with

- how do we know if we should use production $A \rightarrow \epsilon$?

we must consider what tokens can come after $A$

- i.e., what is in $\text{FIRST}^*(\beta)$

in other words

- $\forall t \in \text{FIRST}^*(\beta) : \text{PREDICT}[A, t] = A \rightarrow \epsilon$

as well as for all other terminals that can follow $A$ in the grammar
FOLLOW sets

with \(\epsilon\)-productions, we may have to ‘look past’ the current nonterminal

the set \(\text{FOLLOW}(A)\) represents the terminals that might come after \(A\):

- every terminal that can follow \(A\) in a derivation:

\[
\text{FOLLOW}(A) = \{ t \mid S \rightarrow^* \alpha At\beta \}
\]

(where \(S\) is the start symbol of the grammar)

like FIRST sets, FOLLOW sets can be constructed mechanically

note we have to make the end-of-sentence marker explicit again

- because we need some input token to predict \(A\) in \(S \rightarrow \alpha A\)

  (where \(S\) is the start rule and \(A \rightarrow^* \epsilon\))

- we will write it as $, giving us: \(S \rightarrow \alpha A $

  (and we will consider $ to be a terminal symbol)
constructing FOLLOW sets

begin with all FOLLOW sets empty

first deal with the end-of-sentence marker

1. if $S$ is the start symbol:
   add $\$\$ to FOLLOW($S$)

then apply a closure operation until the fixed point is found

2. repeat
   for all $A \rightarrow \alpha B \beta$:
   add FIRST*($\beta$) $\setminus \{\epsilon\}$ to FOLLOW($B$), and
   if $\epsilon \in$ FIRST*($\beta$) then add FOLLOW($A$) to FOLLOW($B$)
   until no more additions can be made
### constructing FOLLOW sets

\[
\begin{align*}
E & \rightarrow TX \\
X & \rightarrow +E | \epsilon \\
a & \rightarrow (E) | \text{int} Y \\
T & \rightarrow *T | \epsilon \\
Y & \rightarrow \text{int} Y \\
\end{align*}
\]

FIRST: \( E \quad X \quad T \quad Y \)

\( (\text{int} \quad + \epsilon \quad (\text{int} \quad * \epsilon) \)

1. if \( S \) is the start symbol : add \( \$ \) to FOLLOW(\( S \))

<table>
<thead>
<tr>
<th>FOLLOW:</th>
<th>E</th>
<th>X</th>
<th>T</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>( $ )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
constructing FOLLOW sets

E → TX
X → +E | ε
aT → (E) | int Y
Y → *T | ε

FIRST: E X T Y
( int + ε ( int * ε

2. repeat until no more changes:

for all \( A → αBβ \):

add \( \text{FIRST}^*(β) - \{ε\} \) to \( \text{FOLLOW}(B) \), and

if \( ε \in \text{FIRST}^*(β) \) then add \( \text{FOLLOW}(A) \) to \( \text{FOLLOW}(B) \)

<table>
<thead>
<tr>
<th>FOLLOW:</th>
<th>E</th>
<th>X</th>
<th>T</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → TX</td>
<td>( \text{FIRST}^*(X) = {+ε} )</td>
<td>( T+={+} )</td>
<td>( T+=E )</td>
<td>( X+=E )</td>
</tr>
<tr>
<td>( ε \in \text{FIRST}^*(X) )</td>
<td>( ε \in \text{FIRST}^*(X) )</td>
<td>( ε \in \text{FIRST}^*(X) )</td>
<td>( ε \in \text{FIRST}^*(X) )</td>
<td>( ε \in \text{FIRST}^*(X) )</td>
</tr>
<tr>
<td>( T → (E) )</td>
<td>( \text{FIRST}^*(()) = {} )</td>
<td>( E+={} )</td>
<td>( Y+=T )</td>
<td>( Y+=T )</td>
</tr>
<tr>
<td>( T → \text{int } Y )</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
</tr>
<tr>
<td>( E → TX )</td>
<td>( \epsilon \in \text{FIRST}^*(X) )</td>
<td>( \epsilon \in \text{FIRST}^*(X) )</td>
<td>( \epsilon \in \text{FIRST}^*(X) )</td>
<td>( \epsilon \in \text{FIRST}^*(X) )</td>
</tr>
<tr>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
</tr>
<tr>
<td>( T → \text{int } Y )</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
</tr>
<tr>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
</tr>
<tr>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
<td>( \epsilon \in \text{FIRST}^*()</td>
</tr>
<tr>
<td>done!</td>
<td>( $ )</td>
<td>( $ )</td>
<td>( $ )</td>
<td>( $ )</td>
</tr>
</tbody>
</table>


constructing PREDICT with $\epsilon$

we now have:

- the original grammar
  
  $$
  \begin{align*}
  E & \rightarrow T X \\
  X & \rightarrow + E \mid \epsilon \\
  T & \rightarrow ( E ) \mid \text{int} \ Y \\
  Y & \rightarrow \ast T \mid \epsilon
  \end{align*}
  $$

- the FIRST sets, with $\epsilon$

  \[
  \text{FIRST:} \begin{array}{llll}
  \hline
  & E & X & T & Y \\
  \hline
  & ( \text{int} & + & \epsilon & ( \text{int} & \ast & \epsilon \\
  \hline
  \end{array}
  \]

- the FOLLOW sets, with $\$$

  \[
  \text{FOLLOW:} \begin{array}{llllll}
  \hline
  & E & X & T & Y \\
  \hline
  & \$ & \$ & + & \$ & + & \$ \\
  \hline
  \end{array}
  \]

we can now use

- FIRST to construct PREDICT for *non-nullable* productions
- FOLLOW to construct PREDICT for *nullable* productions
constructing PREDICT with $\epsilon$

$E \rightarrow TX$

$X \rightarrow +E | \epsilon$

$T \rightarrow (E) | \text{int} Y$

$Y \rightarrow *T | \epsilon$

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>X</th>
<th>T</th>
<th>Y</th>
</tr>
</thead>
</table>
| FIRST | ( int | + $\epsilon$ | ( int | * $\epsilon$
| FOLLOW| $\$ ) | $\$ ) | + ) $\$ | + ) $\$

1. for all productions $A \rightarrow \alpha$:

$\forall t \in \text{FIRST}^*(\alpha)$:

set $\text{PREDICT}[A,t] = \alpha$

<table>
<thead>
<tr>
<th>PREDICT</th>
<th>int</th>
<th>*</th>
<th>+</th>
<th>(</th>
<th>$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>T X</td>
<td>T X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>+ E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>int</td>
<td>Y</td>
<td></td>
<td>( E</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>* T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*note that $\epsilon$ is not a terminal symbol!*
constructing PREDICT with $\epsilon$

\[
\begin{align*}
E & \rightarrow TX \\
X & \rightarrow +E | \epsilon \\
T & \rightarrow (E) | \text{int}Y \\
Y & \rightarrow *T | \epsilon
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>$E$</th>
<th>$X$</th>
<th>$T$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>(int $+\epsilon$)</td>
<td>(int $*\epsilon$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOLLOW</td>
<td>$$)$</td>
<td>$$)$</td>
<td>$+$</td>
<td>$$)$</td>
</tr>
</tbody>
</table>

2. for all productions $A \rightarrow \alpha$:

if $\epsilon \in \text{FIRST}^*(\alpha)$:

then $\forall t \in \text{FOLLOW}(A)$:

set $\text{PREDICT}[A,t] = \alpha$

<table>
<thead>
<tr>
<th>PREDICT</th>
<th>int</th>
<th>$*$</th>
<th>$+$</th>
<th>(</th>
<th>$$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>$T$ $X$</td>
<td>$T$ $X$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X$</td>
<td>$+$ $E$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>int $Y$</td>
<td>(</td>
<td>$E$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Y$</td>
<td>$*$ $T$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td></td>
</tr>
</tbody>
</table>

*note that $\$ is treated as a terminal symbol!*
## parsing with $\epsilon$ predictions

<table>
<thead>
<tr>
<th>PREDICT</th>
<th>int * + ( ) $</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>TX TX</td>
</tr>
<tr>
<td>X</td>
<td>+ E $\epsilon$ $\epsilon$ $\epsilon$</td>
</tr>
<tr>
<td>T</td>
<td>int Y ( E )</td>
</tr>
<tr>
<td>Y</td>
<td>* T $\epsilon$ $\epsilon$ $\epsilon$</td>
</tr>
</tbody>
</table>

### stack | input | action
--- | --- | ---
E $ int * int $ | $ E \rightarrow [E, int] = TX $ |
T X $ int * int $ | $ T \rightarrow [T, int] = int Y $ |
int Y X $ int * int $ | match |
Y X $ * int $ | $ Y \rightarrow [Y, \ast] = \ast T $ |
* T X $ * int $ | match |
T X $ int $ | $ T \rightarrow [T, int] = int Y $ |
int Y X $ int $ | match |
Y X $ $ | $ Y \rightarrow [Y, \$] = \epsilon $ |
X $ $ | $ X \rightarrow [X, \$] = \epsilon $ |
$ $ | match & accept |
FIRST set summary

FIRST\( (X) = \{ t \mid X \rightarrow t\alpha \} \cup \{ \epsilon \mid X \rightarrow \epsilon \}\)

for right-hand sides of productions, define

\[
\begin{align*}
\text{FIRST}^*(\epsilon) &= \{ \epsilon \} \\
\text{FIRST}^*(t\alpha) &= \{ t \} \\
\text{FIRST}^*(A\alpha) &= \text{FIRST}(A) & \text{if } \epsilon \notin \text{FIRST}(A) \\
\text{FIRST}^*(A\alpha) &= (\text{FIRST}(A) - \{ \epsilon \}) \cup \text{FIRST}^*(\alpha) & \text{if } \epsilon \in \text{FIRST}(A)
\end{align*}
\]

then

- for all productions \( A \rightarrow t\alpha \), \( \text{FIRST}(A) = \{ t \} \)
- for all productions \( A \rightarrow \epsilon \), add \( \epsilon \) to \( \text{FIRST}(A) \)
- repeat
  - for all \( A \rightarrow \alpha \), set \( \text{FIRST}(A) = \text{FIRST}(A) \cup \text{FIRST}^*(\alpha) \)
  
until no further additions can be made
FOLLOW set summary

FOLLOW($X$) = \{ $t$ | $S \rightarrow \alpha X t \beta$ \}

augment the grammar with an end-of-sentence marker $\$

then

- add $\$ to FOLLOW($S$), where $S$ is the start symbol
- repeat
  - for all $A \rightarrow \alpha B \beta$:
    - add FIRST*($\beta$) − \{$\epsilon$\} to FOLLOW($B$), and
    - if $\epsilon \in$ FIRST*($\beta$) then add FOLLOW($A$) to FOLLOW($B$)
  - until no more additions can be made
given \textsc{first}(A) and \textsc{follow}(A)

- For all $A \rightarrow \alpha$,
  \[
  \forall t \in \textsc{first}^* (\alpha), \text{ set } \text{PREDICT}[A,t] = \alpha
  \]
  if $\epsilon \in \textsc{first}^* (\alpha)$, then
  for all $t \in \textsc{follow}(A)$, set $T[A,t] = \alpha$

taking some liberties with notation, we can write the entire process in three lines...
LL(1) parsing summary

to construct a predictive parsing table $T$, for a context-free grammar $G$, with start rule $S$ and end-of-input symbol $\$.$

$$\text{let } \text{FIRST}(X) = \{ t \mid X \to t\alpha \} \cup \{ \epsilon \mid X \to \epsilon \}$$

$$\text{let } \text{FOLLOW}(X) = \{ t \mid S \to \alpha X t \beta \}$$

$$\forall A \rightarrow \alpha, \forall t \in \text{FIRST}^*(\alpha \times \text{FOLLOW}(A)), T[A, t] = \alpha$$

(where $\alpha \times \sigma$ is the set of sequences obtained when each element of $\beta$ is appended to the sequence $\alpha$)

attempting to add more than one entry to any $T[A, t]$ means

- the grammar is not left-factored
- the grammar is left-recursive
- the grammar is ambiguous
- the grammar is otherwise not LL(1)

many programming languages are LL(1) by design
homework

construct the FIRST / FOLLOW sets and the PREDICT table for:

E → T Y
T → [0-9] ← treat this as a single token
T → ( E )
Y → + E
Y → ε

prepare for a short almost-end-of-semester test

● everything we have covered in the course
● up to and including today’s class
**glossary**

**FOLLOW set** — a set of terminal symbols associated with a non-terminal symbol $N$. When parsing a grammatical sentence, any of the symbols in the FOLLOW($N$) set can appear immediately after $N$ in a sentential form.

**non-nullable** — a production that produces at least one terminal symbol.

**nullable** — a production that can produce no symbols, i.e., that can produce $\epsilon$. 
questionnaires

orange questionnaires:

• these are secret: I will never see your comments
• be honest, and not afraid to sign the form

white questionnaires:

• these are for me: I will read and consider all your comments
• you do not have to sign the form
• please use this form to tell me how to improve this course
  – was it too hard? was it too easy? what did you like? dislike?
  – do you want more theory? more practical examples?
  – should there be more practice of algorithms during class?
  – are there relevant topics you thought we did not cover?
  – how can this course be improved?
• is there any topic you wish was taught in this college, but is not?

thank you!