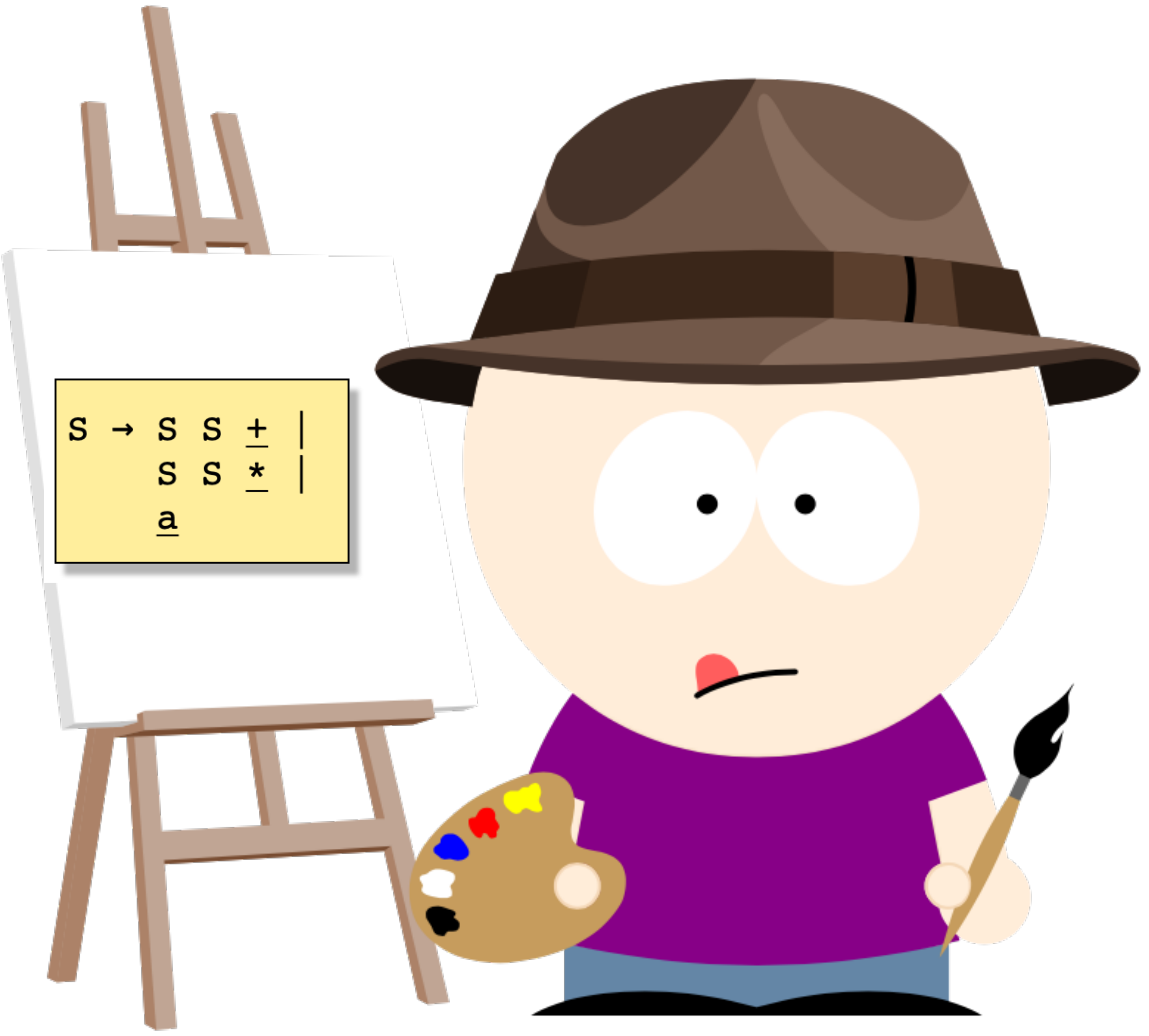
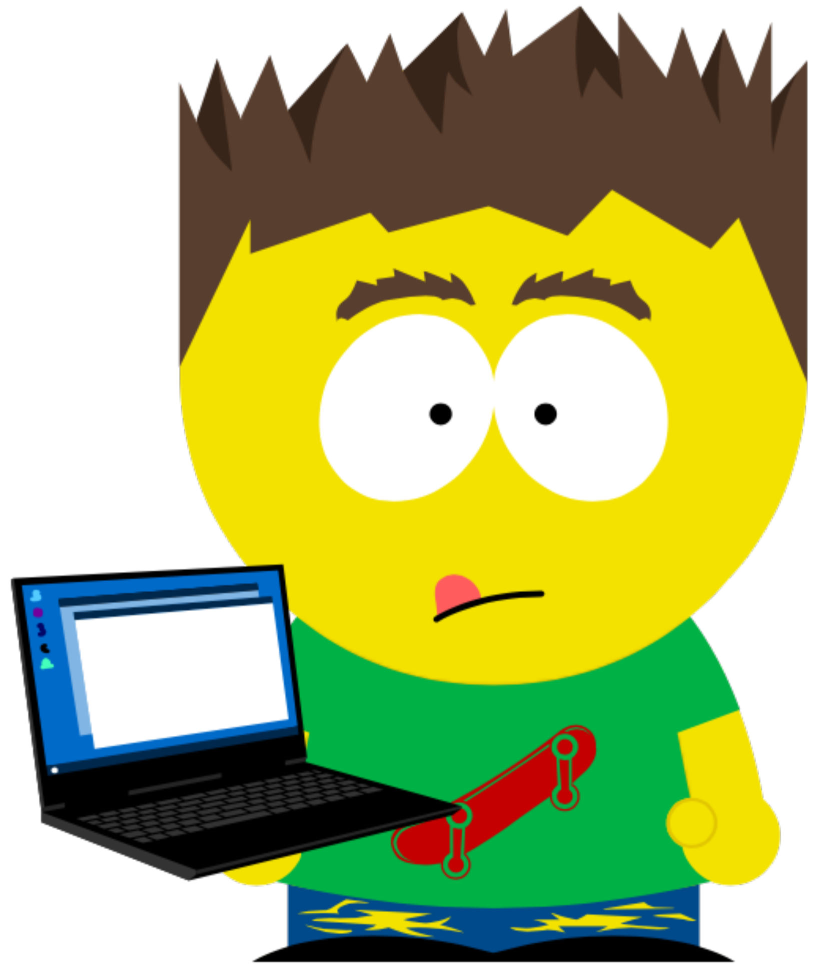




In-Class Exercises

Version: 2026-02-16

Parsing 4–Tying it all Together

Download exercises

1. Go to

   https://ligerlabs.org/compilers.html

2. Download the file

   `parse-4-exercises.zip`

3. Open up a terminal and Unzip the file

```
> unzip parse-4-exercises.zip

> cd parse-4-exercises

> ls
```

Task 1

- Is this grammar LL(1)? Why?

```
1. S → x B A y
2. A → z
3. A → ϵ
4. B → y
5. B → A x
```

```
FIRST(S) = {x}
FIRST(A) = {z,ϵ}
FIRST(B) = {x,y,z}

FOLLOW(S) = {$}
FOLLOW(A) = {x,y}
FOLLOW(B) = {y,z}
```

Task 2

- Is this grammar LL(1)? Why?

```
1. S → x A
2. S → B
3. A → x
4. A → z
5. B → y A
```

```
FIRST(S) = {x,y}
FIRST(A) = {x,z}
FIRST(B) = {y}

FOLLOW(S) = {$}
FOLLOW(A) = {x,y}
FOLLOW(B) = {y,z}
```

Task 3

- Is this grammar LL(1)? Why?

```
1. S → x A
2. A → y
3. A → B
4. B → z
5. B → y w
```

```
FIRST(S) = {x}
FIRST(A) = {y,z}
FIRST(B) = {z,y}
```

Task 4

- Is this grammar LL(1)? Why?

```
1. S → x A B
2. A → y
3. A → ϵ
4. B → y
```

```
FIRST(S) = {x}
FIRST(A) = {y,ϵ}
FIRST(B) = {y}

FOLLOW(S) = {$}
FOLLOW(A) = {y}
FOLLOW(B) = {$}
```

Task 5

- Is this grammar LL(1)? Why?

```
1. S → x A B
2. A → y
3. A → ϵ
4. B → z
```

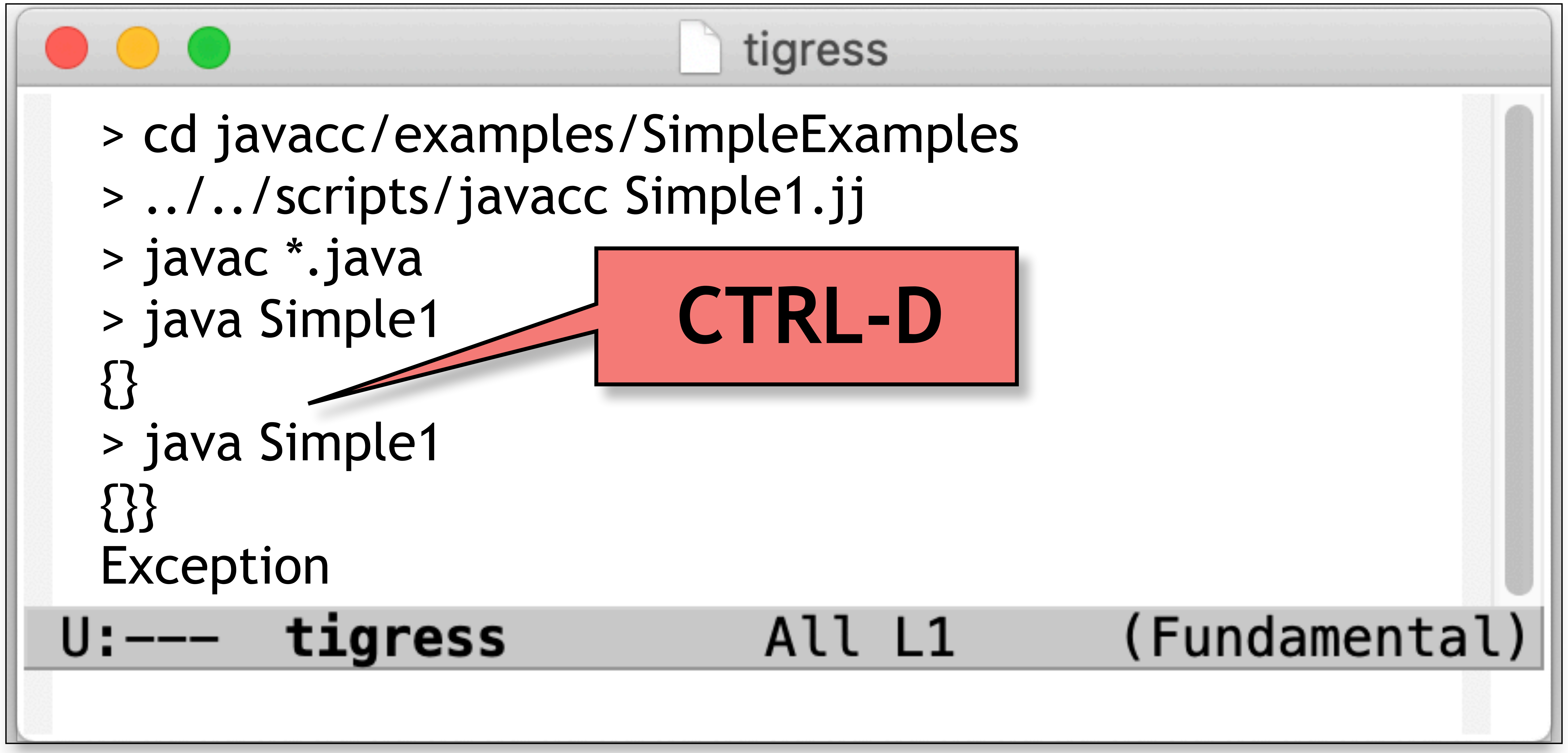
```
FIRST(S) = {x}
FIRST(A) = {y,ϵ}
FIRST(B) = {z}

FOLLOW(S) = {$}
FOLLOW(A) = {z}
FOLLOW(B) = {$}
```

Task 6 (a)

- JavaCC is a popular parser generator for Java. You can read about it here:

  `https://javacc.github.io/javacc/`

- The easiest way to learn how this works is to look at a few examples:



```
> cd javacc/examples/SimpleExamples
> ../../scripts/javacc Simple1.jj
> javac *.java
> java Simple1
{}
> java Simple1
{}}
Exception
U:---  tigress          All L1        (Fundamental)
```

Task 6 (b)

- View `Simple1.jj, Simple2.jj, Simple3.jj`
- How do you
  - define tokens like ":=", "{", ...?
  - define tokens like "BEGIN", "END", ...?
  - define tokens like 123, cow23, ...?
- How do you define syntax?

Task 6 (b)

- View `Simple1.jj, Simple2.jj, Simple3.jj`
- How do you
    - define tokens like ":=", "{", ...?
    - define tokens like "BEGIN", "END", ...?
    - define tokens like 123, cow23, ...?
- How do you define syntax?

Task 6 (c)

- Modify Tiny.jj to handle the extended syntax of the Tiny language.

$$
\begin{array}{rcl}
\text{program} & \rightarrow & \texttt{'BEGIN'}\ \text{stats}\ \texttt{'END'} \\
\text{stats} & \rightarrow & \text{stat stats} \mid \epsilon \\
\text{stat} & \rightarrow & \text{ident}\ \texttt{'='}\ \text{expr}\ \texttt{';'} \\
 & \mid & \texttt{'PRINT'}\ \text{expr}\ \texttt{';'} \\
 & \mid & \texttt{'IF'}\ \text{expr}\ \texttt{'GOTO'}\ \text{int}\texttt{';'} \\
 & \mid & \texttt{'GOTO'}\ \text{int}\ \texttt{';'} \\
 & \mid & \text{int}\ \texttt{':'}\ \texttt{';'} \\
\text{expr} & \rightarrow & \text{expr}\ \texttt{'+'}\ \text{expr} \\
 & \mid & \text{expr}\ \texttt{'-'}\ \text{expr} \\
 & \mid & \text{expr}\ \texttt{'<'}\ \text{expr} \\
 & \mid & \text{ident} \\
 & \mid & \text{int} \\
\text{ident} & \rightarrow & \text{LETTER idp} \\
\text{idp} & \rightarrow & \text{LETTER idp} \mid \text{DIGIT idp} \mid \epsilon \\
\text{int} & \rightarrow & \text{DIGIT intp} \\
\text{intp} & \rightarrow & \text{DIGIT intp} \mid \epsilon
\end{array}
$$

Algorithms

Computing FIRST Sets (no A→ϵ rules)

```
1. FOR each non-terminal A DO
       FIRST($A$) = {}
2. FOR each terminal t DO
       FIRST($t$) = {$t$}
3. REPEAT until no more changes:
       FOR each production $A \rightarrow Y_1 \ldots Y_k$ DO
           FIRST($A$) ∪= FIRST($Y_1$);
```

Computing FIRST Sets (with A→ϵ rules)

```
1.FOR each non-terminal $A$    DO FIRST($A$) = {}
2.FOR each terminal $t$        DO FIRST($t$) = {$t$}
3.FOR each production $A \to \epsilon$ DO FIRST($A$) = {ϵ}
4.REPEAT until no more changes:
    FOR each production $A \to Y_1 \ldots Y_k$ except $A \to \epsilon$ DO
        FIRST($A$) ∪= FIRST($Y_1$) - {ϵ};
        FOR $i$ = 1 to $k-1$ DO
            IF ϵ is in FIRST($Y_1$) ∧ ... ∧ ϵ is in FIRST($Y_i$) THEN
                FIRST($A$) ∪= FIRST($Y_{i+1}$) - {ϵ};
        IF ϵ is in FIRST($Y_1$) ∧ ... ∧ ϵ is in FIRST($Y_k$) THEN
            FIRST($A$) ∪= {ϵ};
```

Computing FOLLOW Sets

```
1.FOR each non-terminal $A$ DO FOLLOW($A$) = {}
2.FOLLOW($S$) = {$\underline{\$}$}
REPEAT until no more changes:
  3.FOR each production $A \rightarrow \alpha B\beta$ DO
      FOLLOW($B$) ∪= (FIRST($\beta$) – {$\epsilon$})
  4.FOR each production $A \rightarrow \alpha B$ DO
      FOLLOW($B$) ∪= FOLLOW($A$)
  5.FOR each production $A \rightarrow \alpha B\beta$ WHERE $\epsilon$ is in FIRST($\beta$) DO
      FOLLOW($B$) ∪= FOLLOW($A$)
```

The LL(1) Property

$$\boxed{\begin{array}{l} A \rightarrow \alpha \\ A \rightarrow \beta \end{array}}$$

A grammar G is LL(1) iff G has these productions and the following conditions hold:

1. α and β don't both derive strings beginning with terminal a.
2. α and β don't both derive the empty string.

$$\boxed{\mathtt{FIRST}(\alpha) \cap \mathtt{FIRST}(\beta) = \varnothing}$$

The LL(1) Property...

$A \rightarrow \alpha$
$A \rightarrow \beta$

3. If $\beta \overset{*}{\Rightarrow} \epsilon$, then α does not derive any string beginning with a terminal in `FOLLOW(A)`.

$$\boxed{\text{if } \epsilon \text{ is in } \mathtt{FIRST}(\beta)\text{, then } \mathtt{FIRST}(\alpha) \cap \mathtt{FOLLOW(A)} = \varnothing}$$

4. If $\alpha \overset{*}{\Rightarrow} \epsilon$, then β does not derive any string beginning with a terminal in `FOLLOW(A)`.

$$\boxed{\text{if } \epsilon \text{ is in } \mathtt{FIRST}(\alpha)\text{, then } \mathtt{FIRST}(\beta) \cap \mathtt{FOLLOW(A)} = \varnothing}$$

collberg.cs.arizona.edu

ligerlabs.org

Supported by

NSF SATC/TTP-1525820

SATC/EDU-2029632