Lecture 17 - Nov. 15

Syntactic Analysis

FIRST Set: Algorithm

Announcements

- **Assignment 3** released

- **Project Milestone 2** meeting signup starting 6pm on Wednesday

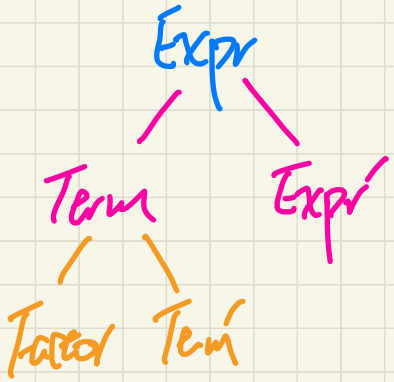
Project: Milestone 2

Milestone 2: Show Additional 5 More Advanced Example Runs [**2%**]

- On **week of November 21** (about 5 weeks after the project is released), your team is required to meet with Jackie and demonstrate:
  - 5 example runs (with no overlap from those in Milestone 1) of your compiler.
  - Though not required, you should aim at showing some of the more advanced features that are outside the above list (see Section 9).
  - The corresponding produced outputs should cover **at least two** control-flow coverage criteria and **at least two** data-flow coverage criteria.
- These example runs are meant to be a clear indication of progress from Mile Stone 1 (e.g., more programming features and coverage criteria supported, more sophisticated scenarios such as nested conditionals).
- **In this meeting, Jackie may suggest specific tasks that your team should complete and will be included in the evaluation of the final submission.**

Assignment 2: Variable Arguments

```java
/**
 * Each ASTNode corresponds to some non-terminal in the
 * context-free grammar in question.
 * @param label name of the non-terminal which this ASTNode represents
 * @param children zero or more child nodes of this ASTNode
 */
public ASTNode(String label, ASTNode ...children) {
    /* Your Task */
}
```




```java
ASTNode root2 =
    new ASTNode("Expr",
        new ASTNode("Term",
            new ASTNode("Factor",
                new ASTNode("a")
            ),
            new ASTNode("Term'",
                new ASTNode("epsilon")
            )
        ),
        new ASTNode("Expr'",
            new ASTNode("+"),
            new ASTNode("Term",
                new ASTNode("Factor",
                    new ASTNode("a")
                ),
                new ASTNode("Term'",
                    new ASTNode("epsilon")
                )
            ),
            new ASTNode("Expr'",
                new ASTNode("epsilon")
            )
        )
    );
```

FIRST Set: Algorithm

$$\textsc{First}(\alpha) = \begin{cases} \{\alpha\} & \text{if } \alpha \in T \\ \{w \mid w \in \Sigma^* \wedge \alpha \overset{*}{\Rightarrow} w\beta \wedge \beta \in (V \cup \Sigma)^*\} & \text{if } \alpha \in V \end{cases}$$

terminals / words — starting symbols — terminal / variable

```
ALGORITHM: GetFirst
  INPUT: CFG G = (V, Σ, R, S)
  T ⊂ Σ* denotes valid terminals
  OUTPUT: First : V ∪ T ∪ {ϵ, eof} ⟶ ℙ(T ∪ {ϵ, eof})
PROCEDURE:
  for α ∈ (T ∪ {eof, ϵ}): First(α) := {α}
  for A ∈ V: First(A) := ∅
  lastFirst := ∅
  while lastFirst ≠ First :
    lastFirst := First
    for A → β1 β2 ... βk ∈ R s.t. ∀βj : βj ∈ (T ∪ V):
      rhs := First(β1) − {ϵ}
      for (i := 1; ϵ ∈ First(βi) ∧ i < k; i++) :
        rhs := rhs ∪ (First(βi+1) − {ϵ})
      if i = k ∧ ϵ ∈ First(βk) then
        rhs := rhs ∪ {ϵ}
      end
      First(A) := First(A) ∪ rhs
```

function

as soon as FIRST map cannot be found exit.

FIRST for terminals

exit when: $\epsilon \notin \text{FIRST}(\beta_i)$ or $i \geq k$

keep calling until FIRST is not nullable

every component of A is nullable ⇒ A is nullable

β_k is nullable

$\beta_1 \ldots \beta_{k-1}$ are nullable

given a valid (non-)terminal, return its FIRST symbols

need to move on to β_2 leading.

stop right here no need to go to β_2

$$A \longrightarrow \beta_1 \; \beta_2 \; \cdots \; \beta_{k-1} \; \beta_k$$

(C1) $\epsilon \in \text{FIRST}(\beta_1)$ (C2) $\epsilon \notin \text{FIRST}(\beta_1)$

Right-Recursive CFG:

0	*Goal*	→	*Expr*	6	*Term′*	→	× *Factor Term′*
1	*Expr*	→	*Term Expr′*	7		|	÷ *Factor Term′*
2	*Expr′*	→	+ *Term Expr′*	8		|	ϵ
3		|	- *Term Expr′*	9	*Factor*	→	(*Expr*)
4		|	ϵ	10		|	num
5	*Term*	→	*Factor Term′*	11		|	name

FIRST Set: Tracing

First choose rules whose RHS starts with a terminal

F, E′, T′, T, E

```
ALGORITHM: GetFirst
  INPUT: CFG G = (V, Σ, R, S)
  T ⊂ Σ* denotes valid terminals
  OUTPUT: FIRST : V ∪ T ∪ {ϵ, eof} ⟶ ℙ(T ∪ {ϵ, eof})
PROCEDURE:
  for α ∈ (T ∪ {eof, ϵ}): FIRST(α) := {α}
  for A ∈ V: FIRST(A) := ∅
  lastFirst := ∅
  while (lastFirst ≠ FIRST):
    lastFirst := FIRST
    for A → β1β2...βk ∈ R s.t. ∀βj : βj ∈ (T ∪ V):
      rhs := FIRST(β1) − {ϵ}
      for (i := 1; ϵ ∈ FIRST(βi) ∧ i < k; i++):
        rhs := rhs ∪ (FIRST(βi+1) − {ϵ})
      if i = k ∧ ϵ ∈ FIRST(βk) then
        rhs := rhs ∪ {ϵ}
      end
      FIRST(A) := FIRST(A) ∪ rhs
```

num	name	+	-	×	÷	(	)	eof	ϵ
num	name	+	-	*	/	(	)	eof	ε

Expr	Expr′	Term	Term′	Factor
∅	∅	∅	∅	∅
	{+}			{(}
	{+,-}			{(, num}
	{+,-,ε}			{(, num, name}

Factor → (Expr)

β1 β2 β3

not executed ∵ FIRST("(") does not contain ε

FIRST Set

```
ALGORITHM: GetFirst
  INPUT: CFG G = (V, Σ, R, S)
  T ⊂ Σ* denotes valid terminals
  OUTPUT: FIRST : V ∪ T ∪ {ϵ, eof} ⟶ ℙ(T ∪ {ϵ, eof})
PROCEDURE:
  for α ∈ (T ∪ {eof, ϵ}): FIRST(α) := {α}
  for A ∈ V: FIRST(A) := ∅
  lastFirst := ∅
  while (lastFirst ≠ FIRST):
    lastFirst := FIRST
    for A → β1 β2 ... βk ∈ R s.t. ∀βj : βj ∈ (T ∪ V):
      rhs := FIRST(β1) − {ϵ}
      for (i := 1; ϵ ∈ FIRST(βi) ∧ i < k; i++):
        rhs := rhs ∪ (FIRST(βi+1) − {ϵ})
      if i = k ∧ ϵ ∈ FIRST(βk) then
        rhs := rhs ∪ {ϵ}
      end
      FIRST(A) := FIRST(A) ∪ rhs
```

Right-Recursive CFG:

0	Goal	→	Expr	6	Term′	→	x Factor Term′
1	Expr	→	Term Expr′	7		|	÷ Factor Term′
2	Expr′	→	+ Term Expr′	8		|	ϵ
3		|	− Term Expr′	9	Factor	→	(Expr)
4		|	ϵ	10		|	num
5	Term	→	Factor Term	11		|	name

nullable

FIRST(Expr′) = {+, −, ε} ∪ FIRST(Term′) ε ∈ FIRST(Term′)

Term′ Factor

β1 β2

Q. Will FIRST(Expr′) change if we add another rule?

Expr′ → Term′ Factor

FIRST(FACTOR)