Tarea No. 5

Importante: Dejar constancia de procedimiento

- Describa (gramática, funcionamiento, objetivo, alcance, etc) las diferentes implementaciones de analizadores sintácticos (LL, SLR, LR, LALR).
- Describa brevemente los elementos necesarios para construir analizadores sintácticos LR(k).
- Discuta la diferencia y/o similitudes existentes entre los analizadores sintácticos SLR, LR y LALR.
- Considere la gramática

  E → E **+** T | T
  T → T F | F
  F → F ***** | **a** | **b**

  Construya la tabla de análisis sintáctico **SLR(1)**

- Considere la gramática

  S → A**a** | **b**A**c** | B**c** | **b**B**a**
  A → **d**
  B → **d**

  Construya la tabla de análisis sintáctico **LR(1)**

ANEXO

LR(0)

Closure()

- Every item in I is also an item in Closure(I)
- If A → α • B β is in Closure(I) and B → • γ is an item, then add B → • γ to Closure(I)
- Repeat until no more new items can be added to Closure(I)

Goto()

- Algorithm for **Goto(I, X)**, where **I** is a set of items and **X** is a grammar symbol: **Goto(I, X)** = Closure({**A → α X • β** | **A → α • X β** in **I**})

Building the DFA states

- Start with the production **<S'> → • <S> $**
- Create the first state to be **closure(<S'> → • <S> $)**
- Pick a state **I**
  - for each **A → α • X β** in **I**
    - find **Goto(I, X)**
    - if **Goto(I, X)** is not already a state, make one
    - Add an edge **X** from state **I** to **goto(I, X)** state
- Repeat until no more additions possible

Creating the parse tables

For each state

- Transition to another state using a **terminal** symbol is a **shift** to that state (*shift to sn*)
- Transition to another state using a **non-terminal** is a **goto** that state (*goto sn*)
- If there is an item **A → α •** in the state do a **reduction** with that production **for all terminals** (*reduce k*)

SLR(1)

first(): For any string **α**, **first(α)** is the set of terminals that begin the string derived from **α**.

- If **a** is a terminal, then **first(a) = {a}**
- If **A → ε** is a production or
  - **A → X_1 X_2... X_k** is a production and **ε ∈ first(X_1)**, …, **first(X_k)**
- then **first(A) = first(A) ∪ {ε}**
- If **A → X_1 X_2... X_i X_{i+1} ... X_k** is a production and
  - **ε ∈ first(X_1)**, …, **first(X_i)** and
  - the terminal **a ∈ first(X_{i+1})**
- then **first(A) = first(A) ∪ {a}**

follow(S): For each non-terminal ***A***, **follow(*A*)** is the set of terminals that can come after ***A*** in some derivation

- **follow(S) = {$}** where S is the start symbol

- If **A → αBβ** is a production then **follow(B) = follow(B) ∪ (first(β) - {ε})**
- If **A → αB** is a production or
  - **A → αBβ** is a production and **ε ∈ first(β)**
- then **follow(B) = follow(B) ∪ follow(A)**

Creating the parse tables

For each state

- Transition to another state using a **terminal** symbol is a **shift** to that state (*shift to sn*)
- Transition to another state using a **non-terminal** is a **goto** that state (*goto sn*)
- If there is an item **A → α •** in the state,
  for all terminals **c ∈ follow(A)**
  do a **reduction** with the production (*reduce k*)

LR(1)

Closure(I)

```
repeat
      for all items [A → α • X β   c] in I
            for any production X → γ
                  for any d ∈ First(βc)
                        I = I ∪ {[X → • γ   d]}
until I does not change
```

Goto(I, X)

```
J = { }
for any item [A → α • X β   c] in I
      J = J ∪ {[A → α X • β   c]}
return Closure(J)
```

Building LR(1) DFA

- Start with the item [**<S'>** → • **<S>** **$**]
- Find the **closure** of the item and make an state
- Pick a state **I**
  - for each item [**A→ α • X β c**] in **I**
    - find **Goto(I, X)**
    - if **Goto(I, X)** is not already a state, make one
    - Add an edge **X** from state **I** to **Goto(I, X)** state
- Repeat until no more additions possible

Creating the parse tables

For each LR(1) DFA state

- Transition to another state using a **terminal** symbol is a **shift** to that state (*shift to sn*)
- Transition to another state using a **non-terminal** is a **goto** that state (*goto sn*)
- If there is an item [**A → α • a**] in the state, do a **reduction** for input symbol **a** with the production **A → α** (*reduce k*)