Instituto Tecnológico de Costa Rica

San José

Compiladores e Intérpretes

Apuntes de la clase del día viernes 12 de mayo del 2017

Apuntador:

Bryan Steve Jiménez Chacón (2014114175)

Profesor:

Dr. Francisco Torres Rojas

I Semestre 2017

Contenido

Importante:

- ➔ Nos entregaron quices y nota del proyecto.
- ➔ Falta un proyecto más que vale 8 puntos y es para ahorita, tal vez para la penúltima semana
- ➔ Lo más probable es que el último parcial sea el último viernes de clases y el final una semana después.
- ➔ El profe dijo que nos iba a traer donas un día de estos :)

Construcción de tabla de Parsing

Ahora en algún momento llegaremos a poder construir una tabla de Parsing, por ahora hay que saber que hay características en las gramáticas que nos hacen imposible encontrar la tabla de parsing (la que nos caía mágicamente del cielo), entonces tenemos que modificar gramática, pero sin alterar el lenguaje.

- Hay algoritmos para construir tablas de parsing.
- Se requiere cierto preproceso de la gramática.
- El lenguaje asociado no cambia:
    1. Genera las mismas hileras
    2. Ni una hilera más

Recursividad por la izquierda

Este es el primer problema que nos puede aparecer, si no lo resolveremos tendríamos problemas al hacer el parsing

- Una GFG es **recursiva por la izquierda,** si hay al menos una regla de la forma:

$$A \rightarrow A\alpha$$

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

- Debe haber otra regla diferente que la “aterrice”:

$$A \rightarrow \beta$$

- Las hileras generadas son de la forma:

βαααααα...ααα

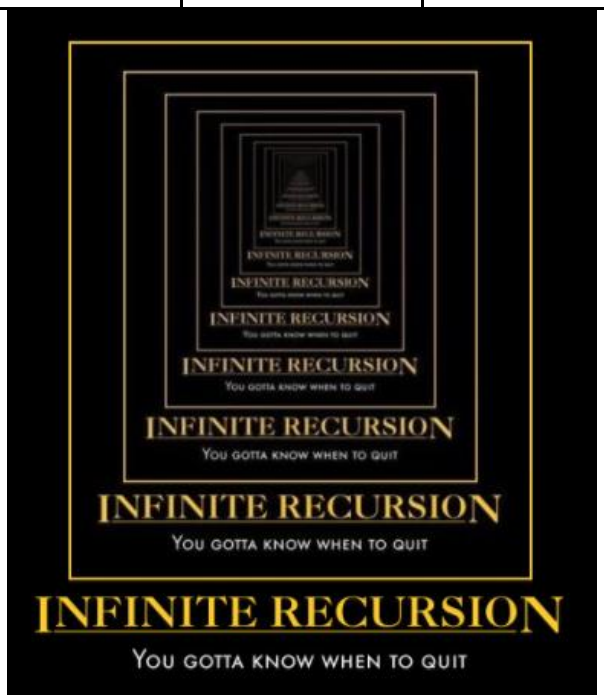
Recursividad por la Izquierda y Parsing

- ¿Qué problema tiene que una gramática sea recursiva por la izquierda?

El problema es que, al expandir con esa regla, la A queda en el mismo lugar en la pila y nuestro algoritmo de parsing patalea, porque nunca avanzó de la A. Se encicla.

- Al aplicar el algoritmo predictivo se podría dar esta situación.

A			A → Aα	



Eliminar Recursividad por la izquierda

Hay que hacer algo para que se sigan generando las hileras pero que no sean recursivas por la izquierda.

- Tenemos reglas:

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

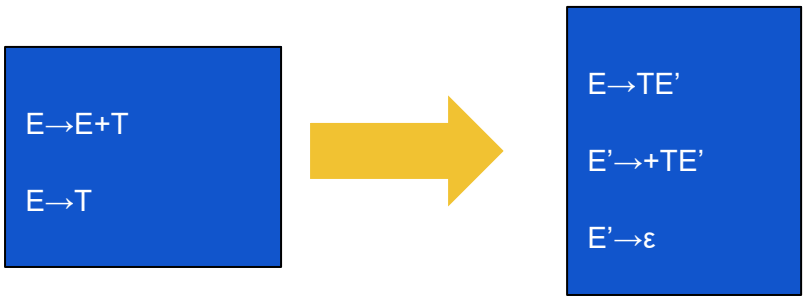
- Si se cambian por:

$$A \rightarrow \beta\ A'$$
$$A' \rightarrow \alpha\ A'$$
$$A' \rightarrow \varepsilon$$

Las tres reglas generan lo mismo que las primeras sin posibilidad de que salga otra cosa, así las hileras generadas no cambian con esta nueva gramática. Esto es como una receta que debemos seguir para que todo nos salga bien

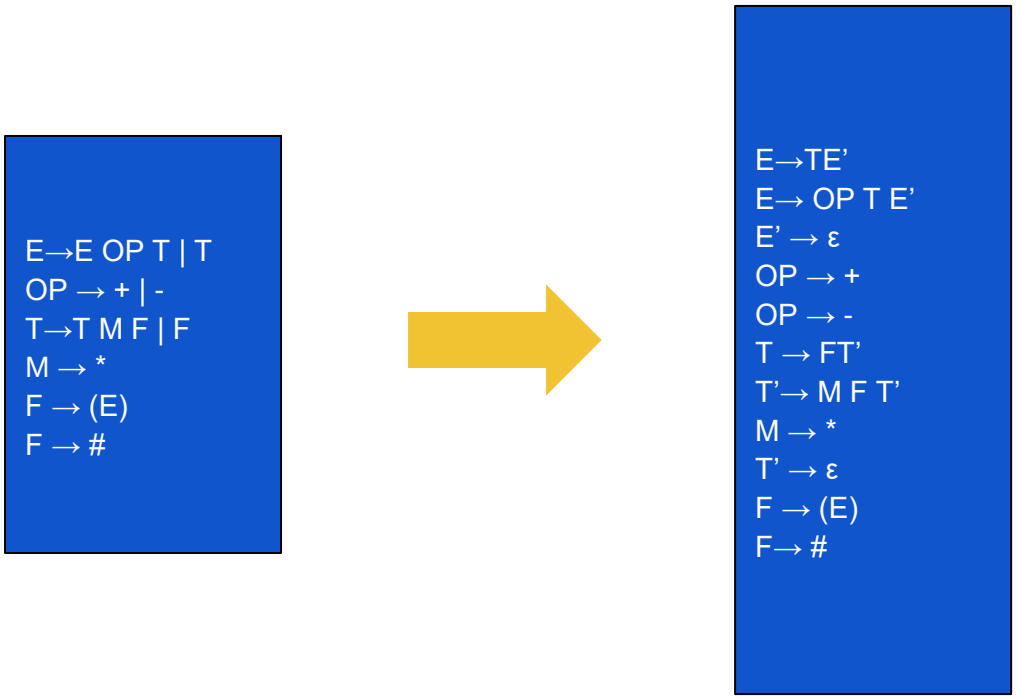
Ejemplo 1

Acá podemos ver que +T es alpha y T es beta por lo tanto convertir la gramática queda de la siguiente manera.



Ejemplo 2

Aunque este ejemplo tiene varias reglas, si nos fijamos bien en realidad solo la primera y la tercera están mal, porque aparece recursividad por la izquierda, entonces las tenemos que corregir con la misma receta.



Factorización Izquierda

Es otra de las cosas que hay que corregir

- Si hay 2 o más reglas de la forma

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

$$A \rightarrow \alpha\ \gamma$$

- Comparten la parte izquierda de su lado derecho… □
- La hilera α es lo más larga posible.

El problema aquí es que si no modificamos la gramática entonces se crea una ambigüedad en la tabla de parsing que nos indicaría que existen dos reglas que se pueden utilizar, para evitar esto hacemos lo siguiente.

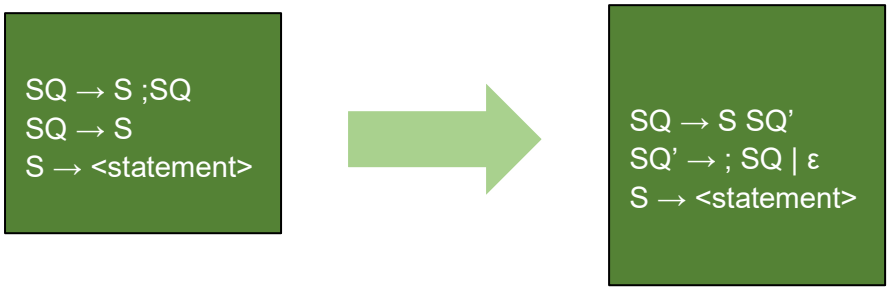
- Las cambiamos por:

$$A \rightarrow \alpha\ A'$$

A' → β | γ

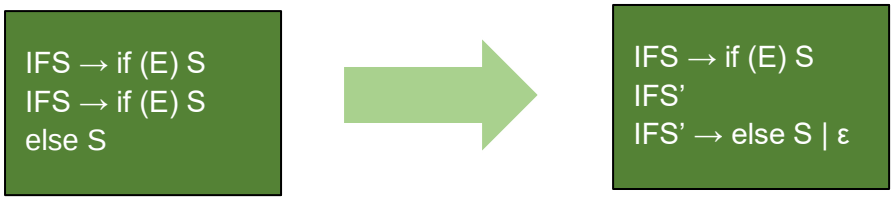
Ejemplo de Factorización Izquierda

En este ejemplo Alpha es S, Beta es ;SQ y Épsilon que no se ve, pero está allí es gamma aplicando la receta queda de la siguiente manera.



Ejemplo de Factorización Izquierda - 2

En este ejemplo alpha es "if(E)S" porque se busca la mayor parte en común de izquierda a derecha del lado derecho, beta es épsilon que está allí pero no se ve y gamma es "else S"



Hay que recordar que es de suma importancia seguir las reglas, ya que el profe nos advirtió que siempre ve gente que quiere ahorrarse pasos y les termina saliendo mal.

Análisis Sintáctico
Cálculo de FIRST()

- Sea G una CFG
- La función FIRST(X) regresa un **conjunto de terminales** y posiblemente ε que indican todos los “inicios” factibles de X bajo la gramática G.

Todavía no definimos X pero es claro que escupe un conjunto con terminales, tokens y épsilon.

Cálculo de FIRST() - cont.

- FIRST(x)
- El argumento x puede ser:
    1. Un terminal
    2. Un No-terminal
    3. Una hilera de terminales y no terminales (puede ser ε)
- El cálculo de FIRST es iterativo hasta que no haya más cambios. Es decir que siempre que detectemos por lo menos un cambio al final de la corrida tenemos que volver a hacerla con los nuevos conjuntos hasta que ya no se den cambios.

Casos Triviales de FIRST(X)

Por dicha existen □

Si X es ε se regresa el conjunto {ε}

- Si X es un terminal se regresa un conjunto que contenga este terminal

1. FIRST(+) = (+)
2. FIRST (id) = {id}

FIRST(X) de un No Terminal

- X es un no terminal
- Por cada regla de la gramática $X \rightarrow X_1X_2X_3...X_n$ haga:

➢ Una FIRST ((X_1) - {ε}) a FIRST(X)

➢ Si FIRST(X_1) contiene a ε

  ➢ Una (FIRST(X_2) - {ε}) a FIRST(X)

  ➢ Si FIRST(X_2) contiene a ε

    ➢ Una (FIRST(X_3) - {ε}) a FIRST(X)

    ➢ Si FIRST(X_3) contiene a ε

      ....

- Si ε pertenece a FIRST(X_1), FIRST(X_2)...FIRST(X_n) entonces se agrega {ε} a FIRST(X)

Esto significa que para encontrar X que es un conjunto entonces vamos a llenarlo con cosas (los conjuntos que nos devuelven los Xn).
Por cada uno hay que calcular, el FIRST(), eso devuelve un conjunto el cual podría contener épsilon, ignoramos el épsilon, y lo demás lo echamos en el conjunto respuesta, sí no tenía épsilon cierro y terminé, pero si sí había un épsilon entonces continuamos haciendo el mismo procedimiento con el siguiente X.

Recuerden que estamos haciendo conjuntos así que…

Y también no hay que olvidar que en X solo se agrega épsilon si toooodos los Xn tienen épsilon. El épsilon significa que esa x podría desaparecer entonces hay que seguir Y Todo este proceso es para UNA regla. Hay que hacerlo para todas las reglas.

FIRST(X) de una hilera

- X es una hilera $X_1X_2X_3...X_n$
- Haga:
  - ➢ Una (FIRST(X_1) - {ε}) a FIRST(X)
  - ➢ Si FIRST(X_1) contiene a ε
    - Una (FIRST(X_2) - {ε}) a FIRST(X)
    - Si FIRST(X_2) contiene a ε
      - ➢ Una (FIRST(X_3) - {ε}) a FIRST(X)

➢ Si FIRST(X_3) contiene a ε

- Si ε pertenece a FIRST(X_1), FIRST(X_2)…, FIRST(X_n) entonces se agrega {ε} a FIRST(X)

Ejemplo 1 de FIRST()

- Calcular los FIRST() de todos los **No terminales** de la siguiente gramática
- Al inicio todos están vacíos
- Se recorren todas las reglas hasta que no haya cambios

E → E OP T
E → T
OP → +
OP → -
T → T M F
T → F
M → *
F → (E)
F → #

	FIRST()			
No Terminal	**Inicio**	**Pasada 1**	**Pasada 2**	**Pasada 3**
E	**Ø**	**Ø**	**Ø**	**{ (, # }**
OP	**Ø**	**{ +, - }**	**{ +, - }**	**{ +, - }**
T	**Ø**	**Ø**	**{ (, # }**	**{ (, # }**
M	**Ø**	**{ * }**	**{ * }**	**{ * }**
F	**Ø**	**{ (, # }**	**{ (, # }**	**{ (, # }**

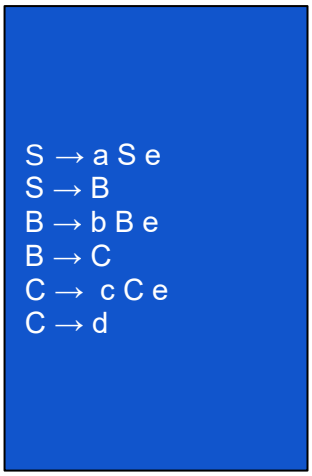
.

Ejemplo 2 de FIRST()

E → T E'
E' → OP T E'
E' → ε
OP → +
OP → -
T → F T'
T → M F T'
T' → ε
M → *
F → (E)
F → #

	FIRST()			
No Terminal	**Inicio**	**Pasada 1**	**Pasada 2**	**Pasada 3**
E	Ø	Ø	Ø	{ (, # }
E'	Ø	{ ε }	{ ε, +, - }	{ ε, +, - }
OP	Ø	{ +, - }	{ +, - }	{ +, - }
T	Ø	Ø	{ (, # }	{ (, # }
T'	Ø	{ ε }	{ ε, * }	{ ε, * }
M	Ø	{ * }	{ * }	{ * }
F	Ø	{ (, # }	{ (, # }	{ (, # }

Ejemplo 3 de FIRST()



Seguimos los mismos pasos de la receta y al final obtenemos lo siguiente.

	FIRST()	
No Terminal	**Inicio**	**Resultado**
S	**Ø**	**{ a, b, c, d }**
B	**Ø**	**{ b, c, d }**
C	**Ø**	**{c, d}**

Ejemplo 4 de FIRST()

S → A B c
A → a
B → b
A → ε
B→ ε

	FIRST()	
No Terminal	**Inicio**	**Resultado**
S	**Ø**	**{ a, b, c }**
A	**Ø**	**{ a, ε }**
B	**Ø**	**{ b, ε }**

Recordemos que cuando hacemos la pasada y ya no hay cambios en ningún conjunto podemos asegurar que ya hemos terminado.

CONTINUARÁ...