Martes 7 de Noviembre

martes, 7 de noviembre de 2023 11:35

En muchos casos, al intentar generar código para E^1 and E^2, E^1 or E^2 y not E, surge el siguiente problema:
Cuando se debe ejecutar la reducción con la producción E-> E^1 and E^2, si E^1 es verdadera, entonces se deben llenar las cuádruplas de E^1.V con el valor de la cuádrupla que comienza el código para E^2, pero no se conoce la dirección de la primera instrucción de E^2. Una solución para eso es agregar una variable "marcadora" , cuyo único fin es guardar la dirección de la cuádrupla que comienza el código de E^2.
Se modifica la producción de la siguiente forma:

E-> E^1 and ME^2
M-> epsilon

La traducción de M es M.quad
Además, usaremos los procedimientos:

- Makelist(i): crea una lista y la inicializa con la cuádrupla i
- Merge(p1,p2): concatena las listas apuntadas por p1 y p2 y retorna un puntero a una nueva lista
- Backpatch(p,i): llena los objetivos de las cuádruplas en la lista p, con el valor i

Reglas Semánticas

E-> E^1 or ME^2	{backpatch(E^1.F, M.quad) E.V = merge(E^1.V, E^2.M) E.F = E^2.F}
E-> E^1 and ME^2	{backpatch(E^1.V,M.quad)

	E.F = merge(E^1.F, E^2.F) E.V = E^2.V}
E-> not E^1	{E.V = E^1.F E.F = E^1.V
E-> (E^1)	{E.V = E^1.V E.F = E^1.F}
E-> id	{E.V = makelist(nextquad) E.F = makelist(nextquad+1) Gen(if id.place goto _) Gen(goto_)}
E-> id^1 oprel id^2	{E.V = makelist(nextquad) E.F = makelist(nextquad + 1) Gen(if id^1.place oprel id^2.place goto _) Gen(goto _)
M-> epsilon	{M.quad = nextquad}

Falta el árbol

Ejemplo: P<Q or (R<S and T<U)

Sea inicialmente nextquad = 10

10) If P<Q goto _
11) Goto 12
12) If R<S goto 14
13) Goto _
14) If T<V goto _
15) Goto _
16)

Ejercicio: Escriba las acciones semánticas para generar código

intermedio para las siguientes instrucciones:

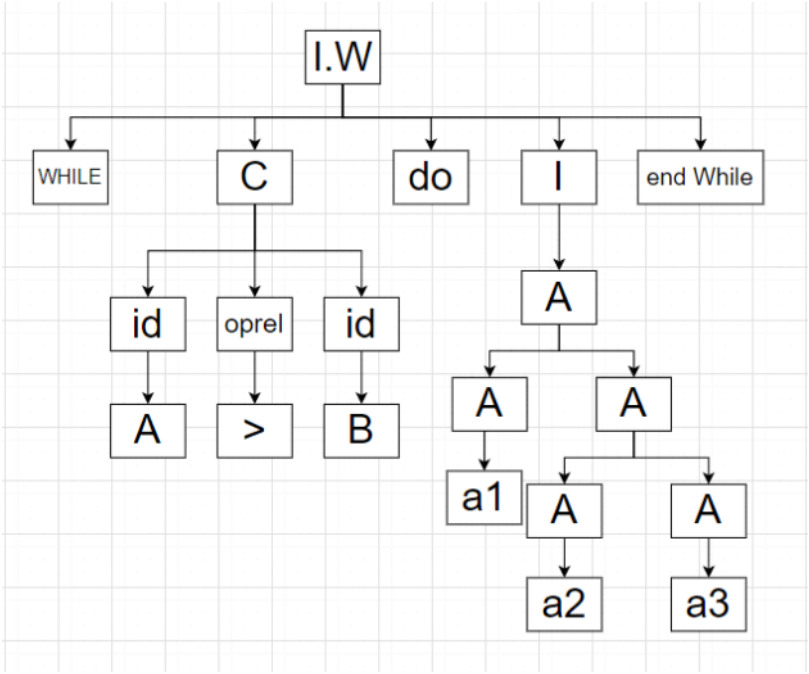
```
While id oprel id do
    a1
    a2
    .
    .
    .
    an
End While
```

Considere que ai es una instrucción cualquiera.

1) Dar un caso particular:

```
While A>B do
    a1
    a2
    a3
End While
```

2) Ver el C.I generado
   1. If A>B goto 3
   2. Goto _
   3. A1
   4. A2
   5. A3
   6. Goto _
   7.
3) Determinar que direcciones necesito
4) Construir el árbol (gramático)



5) Asociar a cada producción de la gramática las acciones semánticas para generar el C.I

C-> id^1 oprel id^2	{C.V = makelist(nextquad) C.F = makelist(nextquad + 1) Gen(if id^1.place oprel id^2.place goto nextquad + 2) Gen(goto _)
A-> ai	{Gen(ai)}
A-> AA	
I-> A	{I.V = makelist(nextquad) I.F = makelist(nextquad + 1) Gen(goto _)}
I.W-> While C do I end While	{backpatch(C.F, I.F) Backpatch(I.V, C.V)}

Tarea: lo mismo para

While id oprel id do

Id = id + id + id + … + id

End While