Tarea Compiladores - S11 02

Stephano Wurttele

June 2020

Ejercicio 1

Defina una gramática CFG (similar a la de expresión de constantes en lenguaje C) que considere un numero de base 8 si el numeral empieza con un digito 0, y considere base 10 si el numeral empieza con cualquier otro dígito.

1. Escriba la gramática con atributos correspondiente.
2. Analice qué tipo de atributos que posee.
3. Plantee el algoritmo para evaluar sus atributos.

Gramática CFG

Para el problema propuesto, se ha de definir una Gramática Libre de Contexto que tenga como primera entrada la diferencia entre empezar con un 0 o no hacerlo, y luego leer el número completo. Este dígito)o ausencia de este) actuará como determinante del atributo base del número. Con ello, generamos la siguiente gramática:

$$\textit{número} \rightarrow \textit{base dígitos num}$$

$$\textit{base} \rightarrow 0 \mid \epsilon$$

$$\textit{num} \rightarrow \textit{num dígito} \mid \textit{dígito}$$

$$\textit{dígito} \rightarrow \textit{dígitos} \mid 0$$

$$\textit{dígitos} \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Los atributos que se manejaran en la gramática definida son **base**, que guardará cuál es la base de la expresión, y **val**, que guardará el valor numérico de la expresión.

Análisis de atributos

Para realizar el análisis de atributos correspondiente se hará una distinción entre atributos heredados o sintetizados de acuerdo a la regla en la que se encuentran.

El atributo **base** se utiliza en las reglas *número* y *base*. Para cada regla, se puede decir lo siguiente:

- *número*: El atributo **base** en esta regla tiene una dependencia de hijo a padre desde la regla *base*. Por lo tanto, este atributo es sintetizado.
- *base*: En esta regla, *base* se define directamente a partir de los tokens "0" o ϵ con una función definida como "isOctal" que recibe un argumento, el caracter, y devuelve *true* si ese valor es "0", o *false* si ese valor es "ϵ". Por lo tanto, el atributo no tiene dependencias de ningún tipo.

El atributo **val** se utiliza en las reglas *número*, *num* y *dígito*. Para cada regla, se puede decir lo siguiente:

- *número*: El atributo **val** en esta regla tiene dos dependencias de hijo a padre desde la regla *num* y la regla *dígitos*. Por lo tanto, este atributo es sintetizado.
- *num*: El atributo **val** en esta regla tiene dos dependencias de hijo a padre desde las reglas *num* y *dígito*, obteniendo e valor de ambos y concatenándolos o simplemente tomando el valor dependiendo del caso que se tome. Dado que en ambas opciones de transición se tienen dependencias de este tipo, se dice que este atributo es sintetizado.
- *dígito*: En esta regla, *val* se define directamente a partir del token "0" o tiene una dependencia de hijo a padre desde la regla *dígitos*. Por lo tanto, este atributo es sintetizado.
- *dígitos*: En esta regla, *val* se define directamente a partir de los token caracteres de dígitos. Por lo tanto, el atributo no tiene dependencias de ningún tipo.

Esta implementación no requiere de atributos con dependencias de padre a hijo o de hermano a hermano, por lo que no tenemos atributos heredados.

Algoritmo para evaluar atributos

```
procedure Eval(T: treecode);
begin
    case clasenodo de T:
        numero:
            Eval(primer hijo de T)
            asignar base de primer hijo de T a T.base
            Eval(segundo hijo de T)
            Eval(tercer hijo de T)
            valF.val := numVal(T.base, concatenar(digitos,num))
            if valF = false:
                T.val := error
            else:
                T.val := valF
            endif
        base:
            base.base := definirBase(hijo de T)
        num:
            if hijo derecho de T != nil:
                Eval(hijo derecho de T)
                Eval(hijo izquierdo de T)
                T.val = concatenar(valor_izquierdo, valor_derecho)
            else:
                T.val := Eval(hijo izquierdo de T)
            endif
        digito:
            if hijo de T = 0:
                T.val = 0
            else:
                T := Eval(hijo de T)
            endif
        digitos:
            T.val =
            numval(hijo de T);
    end case;
end Eval;
```

Listing 1: Algoritmo

Ejercicio 2

Dada la gramática:

$$exp \rightarrow \ exp/exp \mid num \mid num.num$$

Genere las reglas semánticas, sabiendo que se necesitan tres atributos:

- Un atributo booleano sintetizado isFloat, que indica si alguna parte de una exp tiene un valor de punto flotante
- Un atributo heredado etype, con dos valores int y float que da el tipo de cada subexpresión y que depende de isFloat
- El val calculado de cada subexpresión, que depende del etype heredado

Diagrame el árbol semántico e indique en que orden se calculan los atributos para la expresión 5/2/2.0

Reglas semánticas

Regla gramatical	Reglas Semánticas
$exp_1 \rightarrow exp_2 \ / \ exp_3$	$exp_1.isFloat = (exp_2.isFloat \ or \ ex_3.isFloat) \ ? \ true : false$ $exp_2.etype = exp_1.etype$ $exp_3.etype = exp_1.etype$ $exp_1.val = exp_1.isFloat \ ? \ exp_2/exp_3 : exp_2//exp_3$
$exp \rightarrow num$	$exp.val = num.val$ $exp.isFloat = false$ $if(exp.etype == float)\{exp.val = float(exp.val)\}$
$exp \rightarrow num.num$	$exp.val = num.num.val$ $exp.isFloat = true$

Desarrollo de cadenas