Análisis Semántico

Joaquín Rossi y Soledad Techera

Semántica

- La sintaxis describe la forma de un programa válido, la semántica su significado.
- Permite verificar reglas más allá de la estructura (consistencia de tipos, etc.).
- Proporciona información necesaria para generar el programa objeto.

Aunque ciertas reglas si podrían ser chequeadas en la sintaxis, estas necesitan el uso de gramáticas no libres de contexto, pero tiene una complejidad tan alta que se suele preferir dejar esto para la semántica.

Reglas semánticas

Se dividen en dos tipos:

- Estáticas, controladas en tiempo de complicación por el compilador.
- Dinámicas, controladas en tiempo ejecución por código insertado que verifica que se cumplan, ya que no es posible hacerlo estáticamente.
  - Verificaciones de seguridad: acceso a memoria, división por cero.
  - Verificaciones definidas por el programador: assertions, invariantes, precondiciones, postcondiciones.

En ciertos casos se puede obviar los chequeos dinámicos, si el compilador puede demostrar que nunca ocurre ese error.

Gramáticas de atributos

Para agregar significado a la gramática de un lenguajes se utiliza *atributos* que asocian valores a cada nodo.

A cada producción se le agrega las reglas que definen como estos valores son calculados. Estrictamente estas pueden ser *copy rules* (el atributo es la copia de otro) o *semantic functions calls* (el atributo se computa a partir de otros con una función conocida).

En la práctica se relaja esta definición permitiendo usar pseudocódigo o código del lenguaje en el que se programa el compilador en las reglas.

1.	$E_1 \longrightarrow E_2$ `+` T	□ E_1.val := sum(E_2.val, T.val)
2.	$E_1 \longrightarrow E_2$ `-` T	□ E_1.val := difference(E_2.val, T.val)
3.	$E \longrightarrow T$	□ E.val := T.val
4.	$T_1 \longrightarrow T_2$ `*` F	□ T_1.val := product(T_2.val, F.val)
5.	$T_1 \longrightarrow T_2$ `/` F	□ T_1.val := quotient(T_2.val, F.val)
6.	$T \longrightarrow F$	□ T.val := F.val
7.	$F_1 \longrightarrow$ `-` F_2	□ F_1.val := additive_inverse(F_2.val)
8.	$F \longrightarrow$ `(` E `)`	□ F.val := E.val
9.	$F \longrightarrow$ `const`	□ F.val := const.val

Figure 4.1 **A simple attribute grammar for constant expressions, using the standard arithmetic operations.** Each semantic rule is introduced by a □ sign.

Ejemplos de atributos

- Nombre del archivo, número de línea y número de columna de donde se leyó.
- Lista de errores semánticos encontrados en sus subárboles.
- Para expresiones, su tipo.
- Para sentencias, su código intermedio correspondiente.
- Para identificadores, la referencia a su información en la tabla de símbolos.

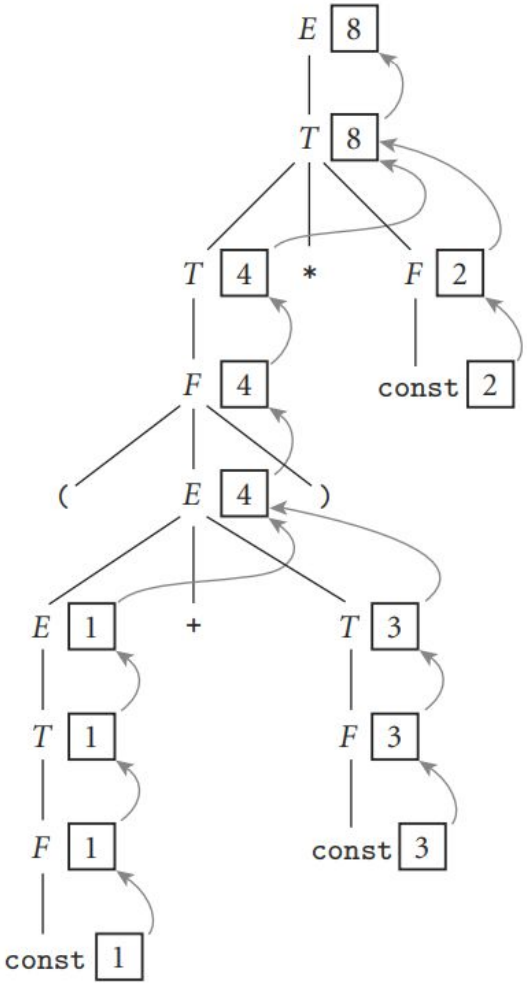
Evaluación de atributos

Se le llama al proceso de anotación o decoración del árbol de parseo, donde al terminar cada nodo contiene un valor para cada uno de sus atributos.

Esto depende de como se definen los atributos:

- Sintetizados: calculados en producciones en las que están a la izquierda, sintetizando los valores de sus hijos. Si todos los atributos son sintetizados la gramática se llama *S-atribuida*.
- Heredados: calculados en producciones en las que están a la derecha, permiten que información pase de padre a hijo.

Los tokens tienen atributos intrínsecos que son sintetizados por el scanner por lo que no se les define reglas.



expr ⟶ `const` *expr_tail*

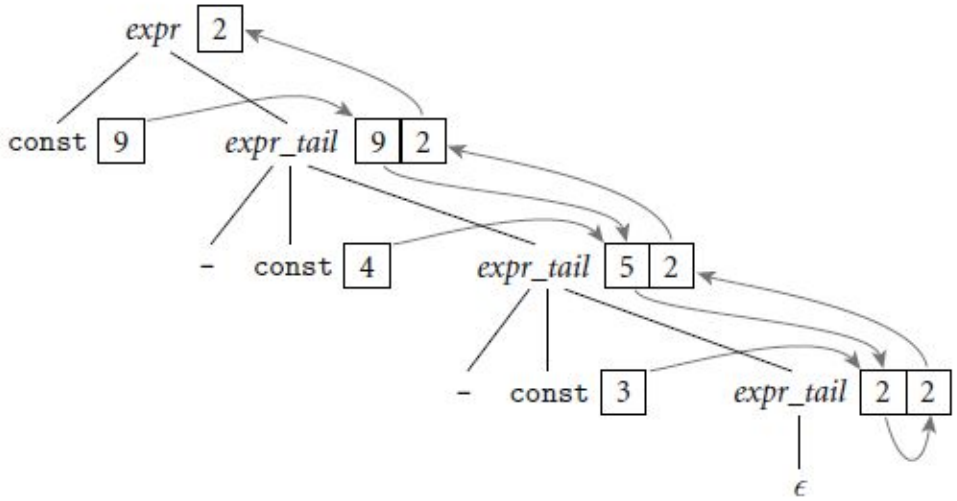
- expr_tail.st := const.val
- expr.val := expr_tail.val

$expr\_tail_1$ ⟶ `-` `const` $expr\_tail_2$

- $expr\_tail_2$.st := $expr\_tail_1$.st − const.val
- $expr\_tail_1$.val := $expr\_tail_2$.val

expr_tail ⟶ ϵ

- expr_tail.val := expr_tail.st



Flujo de atributos

La definición de la gramática es *declarativa*, no especifica en qué orden se deben calcular. Solo se tiene que antes de ejecutar una regla que calcula un atributo, todos los atributos que utiliza en su lado derecho deben haber sido ya calculados.

Una gramática de atributos es *bien definida* si sus reglas determinan un conjunto único de valores para los atributos de cada árbol de parseo posible.

Se le llama *acíclica* si nunca conduce a un árbol en el que hay ciclos en el grafo de flujo de atributos, es decir, que un atributo nunca depende transitivamente de sí mismo.

Esquema de traducción

Es el algoritmo que decora el árbol de parseo invocando las reglas de una gramática de atributos de forma consistente al flujo de atributos del árbol.

Cuando se hacen pases repetidos sobre un árbol, invocando funciones semánticas cuyos argumentos ya han sido definidos y deteniéndose cuando los valores no cambian, estamos hablando de un esquema de traducción *olvidadizo* o *no consciente.*

Para gramáticas acíclicas se puede lograr un mejor rendimiento mediante un esquema dinámico, en el cual se ajusta el orden de evaluación a la estructura de un árbol de análisis dado.

Cuando los atributos se pueden evaluar visitando los nodos del árbol de parseo en una única recorrida en profundidad de izquierda a derecha estamos hablando de una gramática *L-atribuida*.

Teniendo en cuenta que un atributo A.s *depende* de un atributo B.t cuando B.t es utilizado por una función semántica de A.s, podemos definir una gramática L-atribuida en dos reglas:

1) Cada atributo sintetizado del lado izquierdo depende únicamente de los atributos heredados de ese símbolo o de atributos de los símbolos en el lado derecho de la producción
2) Cada atributo heredado de un símbolo en el lado derecho depende solo de los atributos heredados del símbolo en el lado izquierdo o de atributos de los símbolos a su izquierda en el lado derecho.

Las gramáticas *S-atribuidas* son un tipo de *L-atribuidas.*

- Las *S-atribuidas* se pueden ir ejecutando mientras se realiza un parseo LR.
- Las *L-atribuidas* se pueden ir ejecutando mientras se realiza un parseo LL.

A los compiladores que mezclan el parseo con el análisis semántico se los llama *single pass*.

Si esto no se hace, lo más común es generar otro árbol a partir del de parseo (árbol de sintaxis) que solo contenga los aspectos relevantes para la semántica, para ser luego analizado.

Rutinas de acción

Son herramientas automáticas que construyen un analizador semántico para una gramática de atributos.

Una rutina de acción es una función semántica que el redactor de la gramática instruye al compilador para que ejecute en determinado punto del análisis.

Una rutina al principio del lado derecho se invocará tan pronto como el analizador prediga la producción. Mientras que una rutina que se encuentra en el medio del lado derecho se llamará cuando el analizador haya emparejado el símbolo de la izquierda.

$E \longrightarrow T$ { TT.st := T.ptr } TT { E.ptr := TT.ptr }

$TT_1 \longrightarrow$ + T { TT_2.st := make_bin_op("+", TT_1.st, T.ptr) } TT_2 { TT_1.ptr := TT_2.ptr }

$TT_1 \longrightarrow$ - T { TT_2.st := make_bin_op("−", TT_1.st, T.ptr) } TT_2 { TT_1.ptr := TT_2.ptr }

$TT \longrightarrow \epsilon$ { TT.ptr := TT.st }

$T \longrightarrow F$ { FT.st := F.ptr } FT { T.ptr := FT.ptr }

$FT_1 \longrightarrow$ * F { FT_2.st := make_bin_op("×", FT_1.st, F.ptr) } FT_2 { FT_1.ptr := FT_2.ptr }

$FT_1 \longrightarrow$ / F { FT_2.st := make_bin_op("÷", FT_1.st, F.ptr) } FT_2 { FT_1.ptr := FT_2.ptr }

$FT \longrightarrow \epsilon$ { FT.ptr := FT.st }

$F_1 \longrightarrow$ - F_2 { F_1.ptr := make_un_op("+/_", F_2.ptr) }

$F \longrightarrow$ (E) { F.ptr := E.ptr }

$F \longrightarrow$ `const` { F.ptr := make_leaf(const.ptr) }

Manejo de almacenamiento para atributos

Los atributos son almacenados en un stack específico.

En el caso de gramáticas S-atribuidas con parser bottom-up, el stack de atributos es un espejo del stack de parseo, donde el driver del parser es el encargado de colocar los atributos y retirarlos del stack.

Para gramáticas L-atribuidas con parser top-down también se puede usar el stack de atributos, pero de una manera más compleja. La otra opción es que cada regla reserve y libere memoria explícitamente para cada registro de atributos.