Semántica Estática

Semántica Estática

- Gramáticas libres de contexto no pueden describir todas las sintaxis de lenguajes de programación. Ej: Referenciar una variable que ya fue declarada o no (consistencia de tipos).
- Estática significa que tiene relevancia durante la compilación, no durante la ejecución.
- Categorías de constructores que son problemas:
  - Libres de contexto (por ejemplo, **tipos** de operandos en expresiones)
    - Dá para usar BNF pura
    - Gramática enorme
  - No libres de contexto (por ejemplo, variables necesitan ser declaradas antes de ser usadas)
    - No tiene como definir apenas con BNF
    - Informaciones compartidas entre etapas de la derivación.

Gramáticas de Atributos

- Una gramática de atributos (AG) es una extensión de una gramática libre de contexto (CFG)
- Valores primarios de gramáticas de atributos:
  - Especificación de semánticas estáticas
  - Compilación (verificación de semánticas estáticas)
- Informaciones adicionales:
  - Atributos (símbolos)
  - Funciones de Computación de Atributos (reglas)
  - Funciones de Predicado (reglas)

Gramáticas de Atributos: Definición

- Una gramática de atributo es una gramática con los siguientes recursos adicionales:
  - Asociado a cada símbolo X de la gramática está un conjunto de atributos.
    - Pueden ser de cualquier tipo soportados por el lenguaje utilizado en la construcción del compilador
    - Números, cadenas de caracteres, valores booleanos, estructuras, etc
  - Asociado a cada regla gramatical está un conjunto de funciones semánticas y un conjunto posiblemente vacío de funciones de predicado sobre los atributos de los símbolos en la regla gramatical.
    - Esas funciones pueden representar cualquier algoritmo
    - Nuevamente, desde que el algoritmo pueda ser implementado en el lenguaje que está siendo usado en la construcción del compilador o interpretador
- Dada una derivación, el orden de aplicación de esas funciones en los atributos es definido por un recorrido en el árbol de análisis.

Gramáticas de Atributos: Definición

- Sea $X_0 \rightarrow X_1 \ldots X_n$, una regla
  - Funciones semánticas de la forma $S(X_0) = f(A(X_1), ..., A(X_n))$ definen atributos sintetizados
  - Funciones de la forma $I(X_j) = f(A(X_0), ..., A(X_n))$, para $i<=j<=n$, definen atributos heredados
  - Atributos intrínsecos son atributos sintetizados de los nodos hoja cuyos valores son determinados fuera del árbol de análisis sintáctico.
  - Funciones de predicados son expresiones lógicas $\{A(X_0),...,A(X_n)\}$ que restringen los valores posibles de atributos.
- Imagine un programa, que recorre el árbol de análisis sintáctico, ejecutando funciones para calcular los atributos y verificando si los predicados continúan verdaderos

Gramáticas de Atributos: Un ejemplo

- Sintaxis
- <assign> → <var> = <expr>
- <expr> → <var> + <var> | <var>
- <var> → A | B | C
- actual_type: sintetizados para <var> y <expr>
- expected_type: heredado para <expr>
- El tipo de una variable (A,B o C) es intrínseco, dado por una tabla externa llenada en la declaración de la variable.

Gramática de Atributos

1. Regla sintáctica: <assign> → <var> = <expr>

   Regla semántica: <expr>.expected_type ← <var>.actual_type

2. Regla sintáctica: <expr> → <var>[2] + <var>[3]

   Regla semántica: <expr>.actual_type ←

   if (<var>[2].actual_type = int) and

   (<var>[3].actual_type = int)

   then int

   else real

   end if

   Predicado: <expr>.actual_type == <expr>.expected_type

3. Regla sintáctica: <expr> → <var>

   Regla semántica: <expr>.actual_type ← <var>.actual_type

   Predicado: <expr>.actual_type == <expr>.expected_type

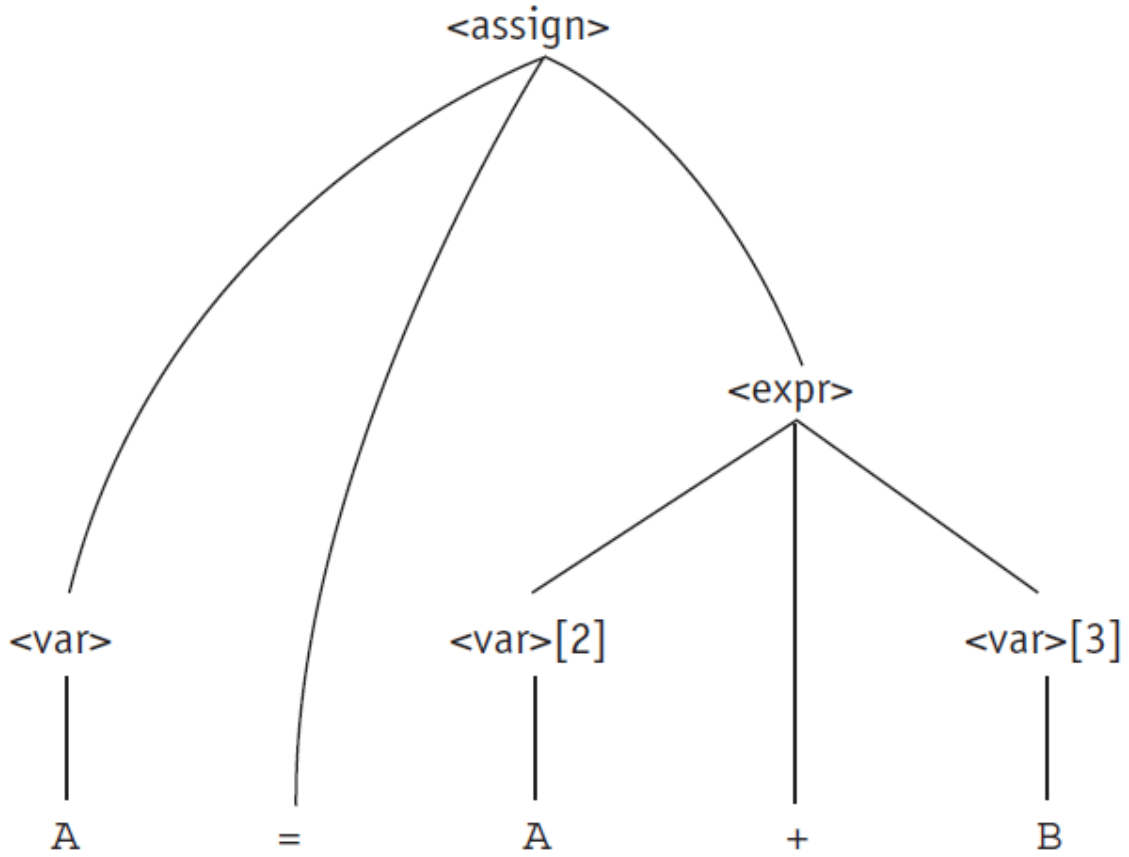
4. Regla sintáctica: <var> → A | B | C

   Regla semántica: <var>.actual_type ← look-up(<var>.string)

La función lookup busca un dado nombre de variable en la tabla de símbolos y retorna el tipo de esa variable.

Gramática de Atributos

- Árbol para A=A+B



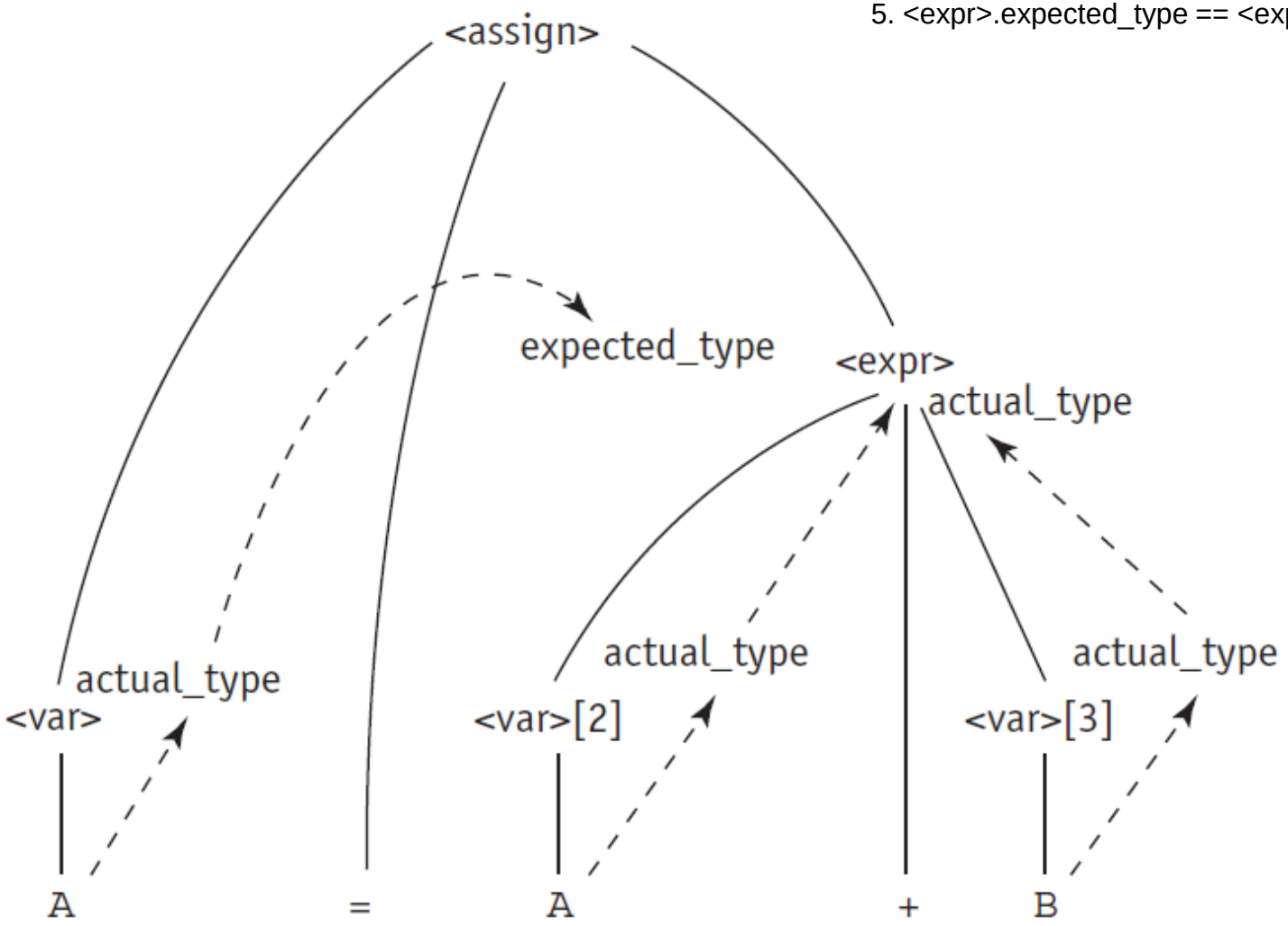
Gramática de Atributos

Cálculo de valores

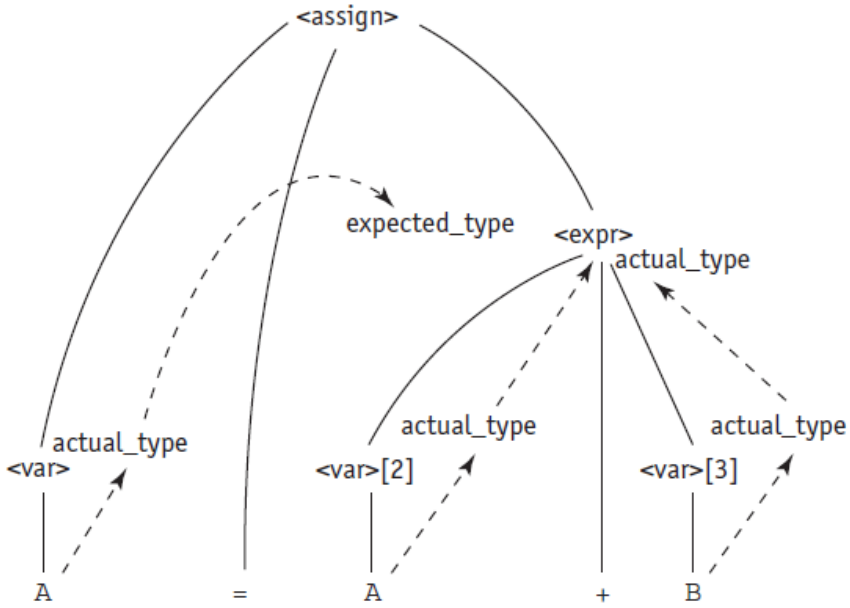
1. <var>.actual_type ← look-up(A) (Rule 4)
2. <expr>.expected_type ← <var>.actual_type (Rule 1)
3. <var>[2].actual_type ← look-up(A) (Rule 4)
   <var>[3].actual_type ← look-up(B) (Rule 4)
4. <expr>.actual_type ← int o real (Rule 2)
5. <expr>.expected_type == <expr>.actual_type
   y TRUE o FALSE (Regla 2)

Gramática de Atributos

1. <var>.actual_type ← look-up(A) (Rule 4)
2. <expr>.expected_type ← <var>.actual_type (Rule 1)
3. <var>[2].actual_type ← look-up(A) (Rule 4)
   <var>[3].actual_type ← look-up(B) (Rule 4)
4. <expr>.actual_type ← int o real (Rule 2)
5. <expr>.expected_type == <expr>.actual_type
   y TRUE o FALSE (Regla 2)



Gramática de Atributos



<assign>

<expr>

expected_type = real_type
actual_type = real_type

actual_type = real_type

<var>

<var>[2]

actual_type = real_type

<var>[3]

actual_type = int_type

A = A + B

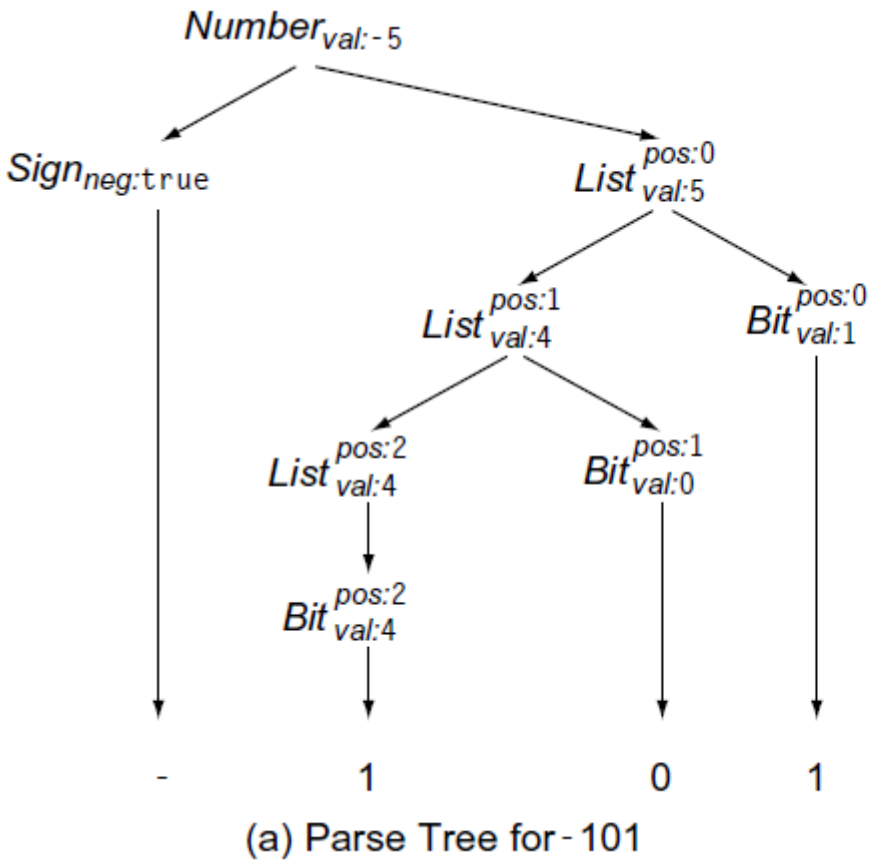
Gramática de Atributos

- ¿Cómo son computados los valores de los atributos?
  - Si todos los atributos fuesen heredados, el árbol puede ser decorado en orden decreciente
  - Si todos los atributos fuesen sintetizados, el árbol puede ser decorado en orden creciente
  - En muchos casos, ambos los tipos de atributos son usados, y una combinación de creciente y decreciente necesita ser usada
- No existe un manera única de determinar el orden de evaluación, cabe al desarrollador del compilador/interpretador definirlo

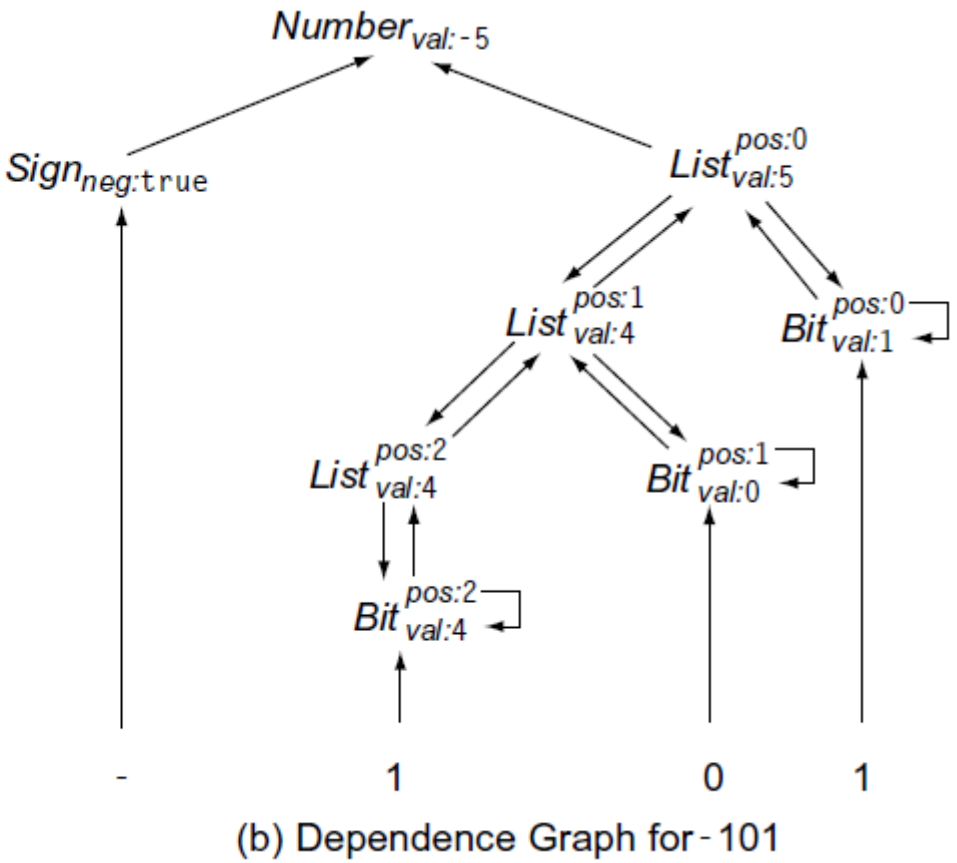
- Ejemplo de cálculo de valor decimal para números binarios

	Production	Attribution Rules
1	$Number \rightarrow Sign\ List$	$List.position \leftarrow 0$ if $Sign.negative$ then $Number.value \leftarrow -List.value$ else $Number.value \leftarrow List.value$
2	$Sign \rightarrow +$	$Sign.negative \leftarrow false$
3	$Sign \rightarrow -$	$Sign.negative \leftarrow true$
4	$List \rightarrow Bit$	$Bit.position \leftarrow List.position$ $List.value \leftarrow Bit.value$
5	$List_0 \rightarrow List_1\ Bit$	$List_1.position \leftarrow List_0.position + 1$ $Bit.position \leftarrow List_0.position$ $List_0.value \leftarrow List_1.value + Bit.value$
6	$Bit \rightarrow 0$	$Bit.value \leftarrow 0$
7	$Bit \rightarrow 1$	$Bit.value \leftarrow 2^{Bit.position}$

- Ejemplo de cálculo de valor decimal para números binarios



(a) Parse Tree for -101



(b) Dependence Graph for -101

Ejemplo simple de atributos heredados

$expr \longrightarrow$ `const` $expr\_tail$

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

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

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

$expr\_tail \longrightarrow \epsilon$

- ▷ expr_tail.val := expr_tail.st

Ejemplo simple de atributos heredados

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