Lenguajes de Programación

Sintaxis

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2019

Especificación de LP: Introducción

La especificación precisa de un lenguaje de programación es esencial para describir el comportamiento computacional del mismo. Sin una notación clara del efecto de los **constructores** del lenguaje es imposible determinar dicho comportamiento.

En general la especificación formal de los lenguajes es útil a varios propósitos:

- Ayuda a la comprensión del lenguaje.
- Soporte para la estandarización de lenguajes.
- Guía en el diseño de lenguajes.
- Ayuda al sistema de escritura del compilador y lenguaje.
- Soporta la verificación de correctitud de los programas.
- Modela la especificación de software.

2

Especificación de LP: Introducción

Recordemos que un lenguaje formal es un conjunto de cadenas de caracteres elegidas de una conjunto finito y fijo de símbolos (alfabeto). Las cadenas que pertenecen al lenguaje, son llamadas *frases* o *constructores*.

La descripción de cualquier lenguaje de programación puede clasificarse en:

- **Sintaxis**: que manipula la formación de las *frases*.
- **Semántica**: que manipula el significado de las *frases*.
- **Pragmatismo**: que manipula el uso práctico de las *frases*.

3

Especificación de LP: Introducción

La **sintaxis** es similar a la gramática de un lenguaje natural. Determina la forma de las expresiones, sentencias y unidades.

La **semántica** es mucho más compleja y difícil de expresar porque involucra definir que está ocurriendo en ejecución de una *frase* o programa. Determinan el significado de las expresiones, sentencias y unidades.

El traductor necesita proveer opciones al usuario, para debugear, para interactuar con el sistema operativo y tal vez para interactuar con un entorno de desarrollo de software. Estas opciones conforman el **pragmatismo** del traductor (en general, estas facilidades forman parte del a definición del lenguaje)

4

Especificación de LP

Reconocedores: Un reconocedor lee una cadena de entrada del lenguajes y decide si la cadena de entrada pertenece o no al lenguaje.

Generadores: El generador, cuenta con un dispositivo que genera sentencias del lenguaje. Uno puede determinar si la sintaxis de una sentencia particular es correcta comparándola con la estructura del generador.

5

Especificación de LP: Introducción

Tres mecanismos describen el diseño e implementación de los lenguajes de programación:

1. Expresiones regulares
2. Gramáticas formales
3. Gramáticas de atributos

Las expresiones regulares nos permitirán definir los *lexemas* o *tokens* (trabajo del analizador léxico)

Las gramáticas formales definirán la sintaxis y las gramáticas de atributos, definirán la *semántica estática*.

6

Métodos formales para describir la sintaxis

Ellos son:

1. Gramáticas y reconocedores
2. Gramáticas libres de contexto – BNF – Diagramas sintácticos
3. BNF extendida (mejora la facilidad de lectura y escritura de las BNF tradicionales)

EXAMPLE 3.1 — A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> – <var>
             | <var>
```

7

Métodos formales para describir la sintaxis

EXAMPLE 3.5 — BNF and EBNF Versions of an Expression Grammar

BNF:

```
<expr> → <expr> + <term>
       | <expr> - <term>
       | <term>
<term> → <term> * <factor>
       | <term> / <factor>
       | <factor>
<factor> → <exp> ** <factor>
         | <exp>
<exp> → ( <expr> )
      | id
```

EBNF:

```
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
<factor> → <exp> { ** <exp>}
<exp> → ( <expr> )
      | id
```

8

Métodos formales para describir la sintaxis

Derivación:

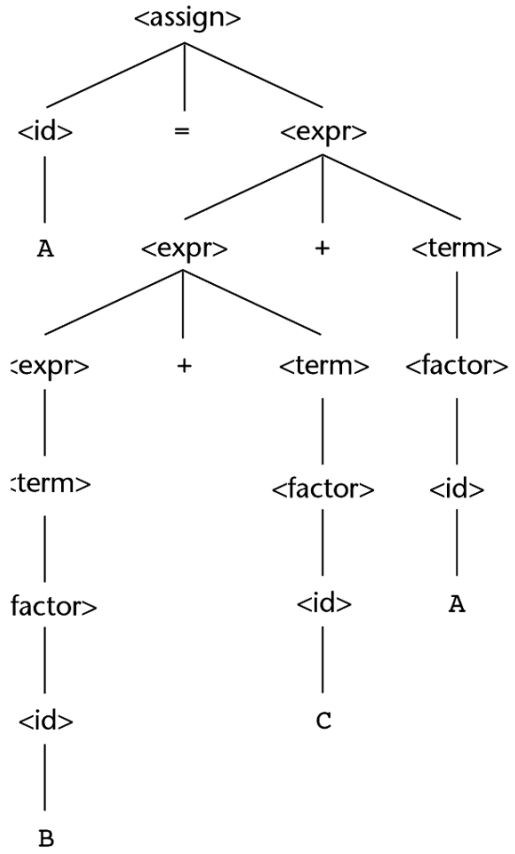
1. Cada cadena de símbolos en la derivación es un forma sentencial.
2. Una sentencia es una forma sentencial que sólo tiene símbolos terminales.
3. Una derivación izquierda es aquella en la cual se expande el noterminal más a la izquierda de cada forma sentencial.

9

Métodos formales para describir la sintaxis

Figure 3.4

A parse tree for `A = B + C + A` illustrating the associativity of addition



Gramáticas de atributos

Una *gramática de atributos* es una mecanismo utilizado para describir más de lo que se puede describir con la gramática libre de contexto.

Es una extensión de las gramáticas libres de contexto, permite la descripción de ciertas reglas del lenguaje como la compatibilidad de tipos.

La idea es agregarle a la gramática tradicional, información semántica a lo largo del árbol de *parsing*.

Semántica estática: considera los aspectos que no pueden ser capturados por la sintaxis (debido a dificultad o imposibilidad de la gramática) pero que pueden ser evaluados en compilación.

11

Gramáticas de atributos: conceptos básicos

Una gramática de atributos es una gramática libre de contexto a la cual se le han agregado atributos, funciones para la evaluación de los atributos y funciones predicado.

Atributo: están asociados a los símbolos de la gramática, son similares a las variables (en el sentido de que pueden tener valores asociados)

Funciones para la evaluación de atributos: también llamadas funciones semánticas, están asociadas a las reglas de la gramática. Son utilizadas para indicar cómo los atributos especificados son computados.

Funciones predicado: determinan la regla semántica estática del lenguaje, están asociadas a reglas de la gramática.

12

Gramáticas de atributos: definición

Una gramática de atributos es una gramática libre de contexto G=(S,N,T,P) con los siguientes agregados:

- Para cada símbolo de la gramática *X*, hay un conjunto *A(X)* de atributos.
- Cada regla tiene un conjunto de funciones que definen ciertos atributos de los no-terminales en la regla.
- Cada regla tiene un conjunto (posiblemente vacío) de predicados para chequear la consistencia de los atributos.

13

Gramáticas de atributos: definición

- Para cada símbolo de la gramática *X*, hay un conjunto *A(X)* de atributos.

  El conjunto se divide en dos conjuntos disjuntos S(X) y I(X).

  Que definen a los atributos sintetizados y heredados respectivamente

  Hay atributos intrínsecos en las hojas del árbol de derivación

14

Gramáticas de atributos: definición

EXAMPLE 3.6 — An Attribute Grammar for Simple Assignment Statements

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type = int) and
   (<var>[3].actual_type = int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → `A | B | C`
   Semantic rule: <var>.actual_type ← look-up(<var>.string)

The look-up function looks up a given variable name in the symbol table and returns the variable's type.

15

Gramáticas de atributos: definición

EXAMPLE 3.6 — An Attribute Grammar for Simple Assignment Statements

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type = int) and
   (<var>[3].actual_type = int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → `A | B | C`
   Semantic rule: <var>.actual_type ← look-up(<var>.string)

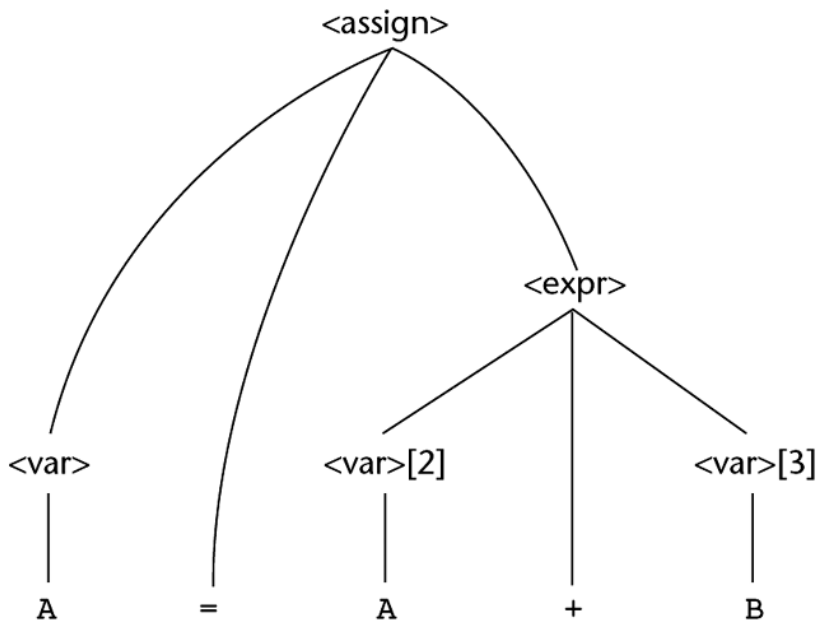
The look-up function looks up a given variable name in the symbol table and returns the variable's type.

16

Gramáticas de atributos: definición

Figure 3.6

A parse tree for
`A = A + B`

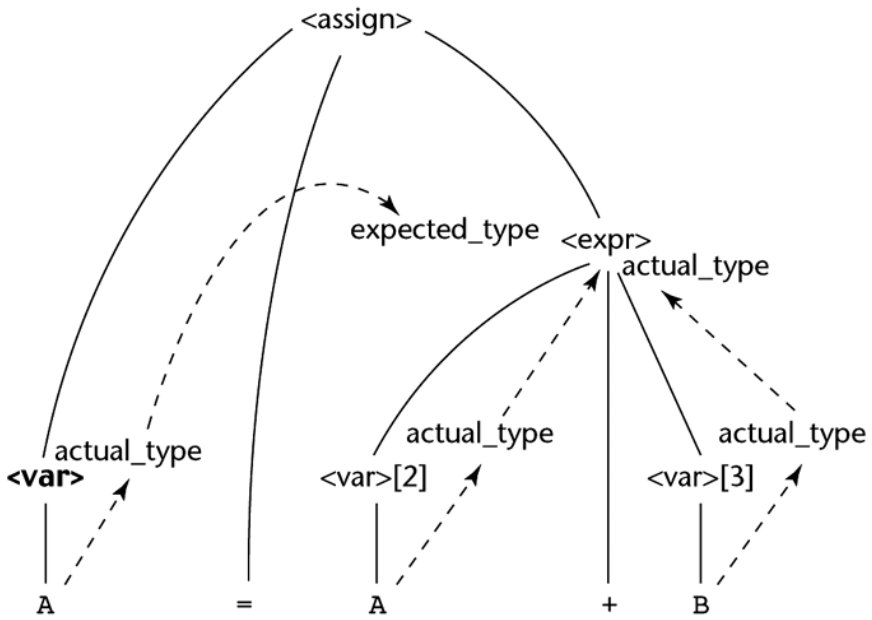


17

Gramáticas de atributos: definición

Figure 3.7

The flow of attributes in the tree



Gramáticas de atributos: definición

Figure 3.8

A fully attributed parse tree