Semántica formal de un lenguaje de programación

1 – Lenguaje mínimo

Maximiliano Cristiá
Universidad Nacional de Rosario
Argentina

2019

Este documento describe el modelo matemático usado para formalizar la semántica del lenguaje de programación presentado en el vídeo `https://youtu.be/3ybrhTCU6BQ`.

La gramática del lenguaje de programación

En esta sección presentamos la gramática del lenguaje de programación en BNF.

Sea $\mathbb{Z}$ el conjunto de los números enteros y Var el conjunto de los nombres de variables que se pueden utilizar en el lenguaje de programación.

$$Expr ::= \mathbb{Z} \mid Var \mid Expr + Expr$$

$$Sentencia ::= Var = Expr \mid Estructura$$

$$Estructura ::= \\ \quad \mathsf{if}\ Expr\ \mathsf{then}\ Programa\ \mathsf{fi} \\ \quad \mid \mathsf{while}\ Expr\ \mathsf{do}\ Programa\ \mathsf{done}$$

$$Programa ::= Sentencia \mid Programa\ ;\ Sentencia$$

La semántica formal del lenguaje de programación

Daremos la *semántica operativa estructural de paso grande* (*big-step structural semantics*) para nuestro lenguaje de programación.

Definimos una memoria como una función que determina el valor de cada variable del programa:

$$Var \to \mathbb{Z}$$

Definimos la función *eval* para evaluar expresiones ($Expr$) de nuestro lenguaje de programación en una memoria dada. Es decir, *eval* toma una memoria y una

Expr y retorna el valor de esa *Expr* de acuerdo a los valores que tienen las variables en esa memoria. El resultado de *eval* es un número entero ($\mathbb{Z}$).

$$eval : (Var \to \mathbb{Z}) \times Expr \to \mathbb{Z}$$

pero para simplificar y acortar las fórmulas que vamos a escribir vamos a definir un sinónimo para *eval* de la siguiente forma:

$$eval(m, e) = m[\![e]\!]$$

Donde la idea es que $m[\![e]\!]$ se lea como "dada m evaluar e". Entonces tenemos que la definición de $m[\![e]\!]$ es la siguiente:

$$m[\![e]\!] = \begin{cases} e & \text{si } e \in \mathbb{Z} \\ m(e) & \text{si } e \in Var \\ m[\![e_1]\!] + m[\![e_2]\!] & \text{si } e \text{ es de la forma } e_1 + e_2 \end{cases}$$

A continuación definimos la función *exec* que es la que dará la semántica formal de nuestro lenguaje de programación. *exec* toma una memoria y un *Programa* y devuelve una memoria:

$$exec : (Var \to \mathbb{Z}) \times Programa \to (Var \to \mathbb{Z})$$

La interpretación de *exec* es la siguiente: $exec(m, P)$ es la memoria que se obtiene al ejecutar el programa P partiendo de la memoria m. Es decir, la semántica de un programa está dada por cuál es el valor de cada una de sus variables luego de ejecutar todas las sentencias que lo componen, partiendo de una cierta memoria.

De la misma forma que hicimos con *eval* vamos a definir un sinónimo para *exec* de forma de acortar las fórmulas que vamos a escribir:

$$exec(m, P) = m[\![P]\!]$$

O sea que usamos el mismo sinónimo para expresiones *eval* y *exec* aunque no debería haber confusión entre ambos sinónimos porque el primero se usa para *Expr* mientras que el segundo se usa para *Programa*. En este caso la idea es que $m[\![P]\!]$ se lea como "dada m ejecutar P".

La definición de *exec* se da para cada una de las construcciones de la gramática del lenguaje de programación:

- Para la asignación $x = expr$; es decir x es un elemento de *Var* y *expr* se construye como indica *Expr*.

  $$m[\![x = expr]\!] = m \oplus (x, m[\![expr]\!])$$

  donde si m es una memoria la expresión $m \oplus (a, b)$ es una memoria igual a m excepto que el valor de la variable a es b.

- Para la estructura if. En nuestro lenguaje de programación las sentencias internas a una estructura if se ejecutan solo si la expresión evalúa a número positivo.

$$m[\![\textsf{if } e \textsf{ then } P \textsf{ fi}]\!] = \begin{cases} m[\![P]\!] & \text{si } m[\![e]\!] > 0 \\ m & \text{si } m[\![e]\!] \leq 0 \end{cases}$$

- Para la estructura while:

$$m[\![\textsf{while } e \textsf{ do } P \textsf{ done}]\!] = \begin{cases} m[\![P \ ; \ \textsf{while } e \textsf{ do } P \textsf{ done}]\!] & \text{si } m[\![e]\!] > 0 \\ m & \text{si } m[\![e]\!] \leq 0 \end{cases}$$

- Para la concatenación de sentencias:

$$m[\![P \ ; \ Q]\!] = m[\![P]\!][\![Q]\!]$$

La notación se podría simplificar más si omitimos la memoria cuando está clara en el contexto.

Ejercicios

1. Extienda la gramática y la semántica con la instrucción skip.
2. Extienda la gramática y la semántica con la instrucción if-then-else.
3. Extienda la gramática y la semántica con la instrucción repeat-until.
4. Extienda la gramática y la semántica con la instrucción for.
5. Reescriba las reglas semánticas omitiendo la memoria siempre que esté clara en el contexto.