APUNTES PARA LENGUAJES Y COMPILADORES

Cuando se define un lenguaje de programación, se determina su sintaxis y su semántica. La sintaxis se refiere a las notaciones necesarias para escribir programas, y a las estructuras que pueden tener los mismos para ser sintácticamente correctos. La semántica se refiere al significado que tienen los programas sintácticamente correctos. Sin su semántica, un lenguaje de programación sería solamente notación y estructura sintáctica. Todo lo que ocurre cuando un programa se ejecuta está determinado por su significado, por su semántica.

Semántica de lenguajes de programación

Hay diferentes formas de describir el significado de los programas (o del lenguaje):

- informal, intuitiva: explica el funcionamiento intuitivo de los programas (ejemplo: manuales, documentación tipo javadoc)
- axiomática: establece cómo razonar sobre programas (por ejemplo, cuando escribimos $\{P/v \leftarrow e\}v := e\{P\}$)
- operacional: describe cómo se ejecuta un programa (ejemplo: intérprete)
- denotacional: mapea programas a su significado (ejemplo: compilador)

Se destaca la denotacional, sobre todo a partir del desarrollo de la teoría de dominios, que le dió una forma matemática perfecta. Se la utiliza como "la definición" del lenguaje, y luego, si se proponen otras semánticas (operacional, axiomática), se las demuestra correctas con respecto a dicha definición.

Sintaxis abstracta. Para ejemplificar tomamos el siguiente lenguaje, a pesar de que no es un lenguaje de programación, es un mini-lenguaje de expresiones enteras: constantes no negativas, menos unario y más binario.

<intexp>::= 0 | 1 | 2 | ...
| -<intexp>| <intexp>+ <intexp>

Antes de ilustrar las diferentes maneras de dar la semántica, una "discusión sobre sintaxis": ¿en qué sentido esta gramática describe la sintaxis del lenguaje?

Observemos ejemplos de frases generadas por esta gramática:

142
-15
-15+3
2+3+4
3+-2

La gramátíca es ambigua: algunas frases admiten diferentes maneras de generarse. Por ejemplo, 2+3+4 se puede generar por

<intexp>→ <intexp>+ <intexp>→ 2 + <intexp>
→ 2 + <intexp>+ <intexp>→ . . . → 2 + 3 + 4

y también puede generarse por

<intexp>→ <intexp>+ <intexp>→ <intexp>+ 4
→ <intexp>+ <intexp>+ 4 → . . . → 2 + 3 + 4

La primera manera de generarse dicha frase, se corresponde intuitivamente con asociar a derecha (es decir, con 2 + (3 + 4)) y la segunda con asociar a izquierda (es decir, con (2 + 3) + 4). Pero ninguna de estas dos frases puede generarse porque los paréntesis no están entre los símbolos terminales de la gramática.

Pregunta. ¿Con cuáles de las otras frases mencionadas más arriba ocurre lo mismo?

El problema podría resolverse agregando paréntesis y desambiguando la gramática. Por ejemplo, cambiando la gramática por la siguiente:

<intexp>::= <intexp>+ <termexp>>| <termexp>
<termexp>::= <groundexp>| -<termexp>
<groundexp>::= 0 | 1 | 2 | ... | (<intexp>)

donde queda claro que el + asocia a izquierda y que el menos tiene mayor precedencia. Esta gramática dice cómo se escriben concretamente las frases del lenguaje. Podríamos llamarla **gramática concreta**, y a las frases que genera, **frases concretas**. Podríamos decir que define la **sintaxis concreta** del lenguaje.

La gramática concreta resulta más complicada que la que dimos anteriormente. Y peor aún, oculta algo que en la gramática anterior era evidente: que el mini-lenguaje tiene constantes no negativas, un operador unario (-) y un operador binario (+). Por esta razón, se prefiere la gramática que se dió en primer lugar, a la que llamaremos **gramática abstracta**, en parte porque no expresa detalles de cómo se escriben las expresiones (asociatividades, precedencias, paréntesis) sino que expresa qué construcciones tiene el lenguaje, cuál es la estructura de las frases que hay, cuáles son las subfrases.

Decimos que la gramática abstracta describe la **sintaxis abstracta** y determina las **frases abstractas**. La palabra "abstracta" refiere siempre a las estructuras que se describen independientemente de la notación concreta a utilizar. Trabajar a este nivel de abstracción es muy conveniente ya que nos permite desentendernos de detalles que cuando se trata de dar significado a las frases resultan irrelevantes.

El libro de Reynolds (a partir de ahora, "el libro") hace un tratamiento detallado del significado preciso de gramática abstracta. Es muy interesante y lectura recomendada para entender esto con precisión.

Para el caso que nos ocupa, la gramática abstracta especifica que hay un conjunto $\mathcal{D}$ y ciertos "constructores" $c_0, c_1, \ldots, c_-, c_+$ tales que

$$\begin{array}{rcll} c_i & \in & \{\cdot\} \to \mathcal{D} & \forall i \in \mathbb{N} \\ c_- & \in & \mathcal{D} \to \mathcal{D} & \\ c_+ & \in & \mathcal{D} \times \mathcal{D} \to \mathcal{D} & \end{array}$$

donde $\{\cdot\}$ es un conjunto unitario. Que sean constructores significa que los codominios de cualquier par de estas funciones son disjuntos, que cada uno de ellos es inyectivo, y que cubren todo $\mathcal{D}$.

En realidad, el conjunto $\mathcal{D}$ puede generarse a partir de los constructores:

$$\begin{array}{rcl}
\mathcal{D}_0 & = & \{c_0(\cdot), c_1(\cdot), \ldots\} \\
\mathcal{D}_1 & = & \{c_0(\cdot), c_1(\cdot), \ldots\} \cup \{c_-(x) | x \in \mathcal{D}_0\} \cup \{c_+(x,y) | x, y \in \mathcal{D}_0\} \\
 & \vdots & \\
\mathcal{D}_{i+1} & = & \{c_0(\cdot), c_1(\cdot), \ldots\} \cup \{c_-(x) | x \in \mathcal{D}_i\} \cup \{c_+(x,y) | x, y \in \mathcal{D}_i\} \\
\mathcal{D} & = & \bigcup_{i=0}^{\infty} D_i
\end{array}$$

La gramática es abstracta en el sentido de que no especifica concretamente quién es $\mathcal{D}$ y cuáles son los constructores. Cualquier $\mathcal{D}$ con constructores como los mencionados es igualmente aceptable.

De todas formas, la gramática abstracta permite generar un $\mathcal{D}$ y constructores concretos de manera mecánica. Hay varias formas de hacerlo. Por ejemplo, tomamos $\mathcal{U}$ (del que $\mathcal{D}$ será subconjunto) como el conjunto de todas las cadenas formadas por números naturales, símbolos -, + y paréntesis. Definimos $c_i(\cdot) = i$, la cadena con un sólo elemento, el número natural i. Definimos $c_-(u) =$ "(−" ++u ++")". Definimos $c_+(u,v) =$ "(" ++u ++" + " ++v ++")". Es obvio que

$$\begin{array}{rcll}
c_i & \in & \{\cdot\} \to \mathcal{U} & \forall i \in \mathbb{N} \\
c_- & \in & \mathcal{U} \to \mathcal{U} & \\
c_+ & \in & \mathcal{U} \times \mathcal{U} \to \mathcal{U} &
\end{array}$$

y que estos satisfacen las propiedades de los constructores, excepto la de cubrir todo $\mathcal{U}$. Pero $\mathcal{D}$ puede generarse a través de la secuencia $\mathcal{D}_0, \mathcal{D}_1, \ldots$ ya descripta, obteniendo

$$\begin{array}{rcll}
c_i & \in & \{\cdot\} \to \mathcal{D} & \forall i \in \mathbb{N} \\
c_- & \in & \mathcal{D} \to \mathcal{D} & \\
c_+ & \in & \mathcal{D} \times \mathcal{D} \to \mathcal{D} &
\end{array}$$

y ahora sí se satisfacen todas las propiedades de los constructores.

Semántica del minilenguaje. A continuación ilustramos las distintas maneras de dar semántica a este lenguaje:

- intuitiva: el significado de una expresión e es el valor que resulta de resolver la expresión e, ejemplo: el significado de 3+(5+2)+(-2) es 8.
- axiomática: se establecen axiomas: e + (f + g) = (e + f) + g, -(e+f) = -e + -f, etc. Da significado de una manera indirecta: las frases deben cumplir estas propiedades, entonces ya no son frases arbitrarias, no pueden significar cualquier cosa. Si el sistema de axiomas es completo el significado queda totalmente determinado.
- operacional: se dice cómo obtener el valor, por ejemplo "para calcular el valor de e + f evaluar primero e y luego f y finalmente sumar ambos valores", etc

- denotacional: se define un dominio semántico, $\mathbb{Z}$, y un mapeo $[\![\ ]\!]$ de <intexp> en dicho dominio semántico:

$$\begin{array}{rcl} [\![\ ]\!] & \in & <\text{intexp}> \to \mathbb{Z} \\ [\![0]\!] & = & 0 \\ [\![1]\!] & = & 1 \\ & \vdots & \\ [\![-e]\!] & = & -[\![e]\!] \\ [\![e+f]\!] & = & [\![e]\!] + [\![f]\!] \end{array}$$

Estas ecuaciones dan significado a las frases. Por ejemplo,

$$\begin{array}{rcl} [\![5+2]\!] & = & [\![5]\!] + [\![2]\!] \\ & = & 5+2 \\ & = & 7 \\ [\![-(5+2)]\!] & = & -[\![5+2]\!] \\ & = & -7 \\ [\![-(5+2)+10]\!] & = & [\![-(5+2)]\!] + [\![10]\!] \\ & = & -7+10 \\ & = & 3 \end{array}$$

Unicidad del mapeo $[\![\ ]\!]$. Se puede demostrar que las ecuaciones que definen el mapeo $[\![\ ]\!]$ determinan una única función de <intexp> en $\mathbb{Z}$. Para ello se utiliza la definición que hemos dado de $\mathcal{D}$ como unión de todos los $\mathcal{D}_i$. Esto permite razonar por inducción en i.

No cualquier conjunto de ecuaciones tienen la propiedad de dar significaco único. Cuando el conjunto de ecuaciones sigue algún criterio que garantiza existencia y unicidad decimos que son **ecuaciones semánticas**..

Un conjunto de ecuaciones es **dirigido por sintaxis** cuando se satisfacen las siguientes condiciones:

1. hay 1 ecuación por cada producción de la gramática abstracta
2. cada ecuación que expresa el significado de una frase compuesta, lo hace puramente en función de los significados de sus subfrases inmediatas

Se puede demostrar en general que dirección por sintaxis implica existencia y unicidad del significado.

Se dice que una semántica es **composicional**, cuando el significado de una frase no depende de ninguna propiedad de sus subfrases, salvo de sus significados.

Composicionalidad es muy importante ya que implica que podemos reemplazar una subfrase f de e por otra de igual significado que f sin alterar el significado de la frase e.

Composicionalidad no es lo mismo que dirección por sintaxis: composicionalidad habla de una propiedad de la semántica, mientras que dirección por sintaxis habla de la forma en que se definió dicha semántica.

Una definición dirigida por sintaxis necesariamente determina una función semántica composicional.

Concluyendo, dirección por sintaxis garantiza existencia y unicidad del significado y también garantiza composicionalidad, todas propiedades deseables. Por ello, insistiremos en que nuestras ecuaciones sean dirigidas por sintaxis.

Lenguaje y metalenguaje. Observemos nuevamente las ecuaciones que obtuvimos para este mini-lenguaje de expresiones:

$$\begin{array}{rcl} [\![\ ]\!] & \in & <\text{intexp}> \to \mathbb{Z} \\ [\![0]\!] & = & 0 \\ [\![1]\!] & = & 1 \\ & \vdots & \\ [\![-e]\!] & = & -[\![e]\!] \\ [\![e+f]\!] & = & [\![e]\!] + [\![f]\!] \end{array}$$

La presentación utiliza 2 lenguajes: el mini-lenguaje de expresiones y el lenguaje en que hicimos la definición. Al primero se lo llama simplemente **lenguaje** y al segundo **metalenguaje**:

En $[\![0]\!] = 0$, el primer 0 es del lenguaje (es la frase 0) y el segundo 0 es del metalenguaje (es el número entero 0).

En $[\![-e]\!] = -[\![e]\!]$, el primer "-" es del lenguaje (es el operador unario "-") y el segundo "-" es del metalenguaje (es la función que devuelve el opuesto de su argumento).

Similarmente en la otra ecuación.

Cada una de las tres últimas ecuaciones representa, en realidad, infinitas ecuaciones. Por ejemplo, $[\![-e]\!] = -[\![e]\!]$ establece una propiedad que vale cualquiera sea la expresión e. Pero e no es una expresión, sólo es un objeto que representa cualquier expresión del lenguaje. Es una variable. Pero no es una variable del lenguaje (revisemos la gramática para comprobar que no hay variables en el lenguaje, sólo constantes y operadores). Es una variable del metalenguaje que utilizamos para escribir una ecuación como $[\![-e]\!] = -[\![e]\!]$ en vez de infinitas ecuaciones, una para cada expresión. A estas variables del metalenguaje se las llama **metavariables**.

La primer ecuación, en cambio, fue escrita sólo para la frase 0, y los puntos suspensivos expresan que hay una ecuación como esa para cada número natural. Podríamos escribir $[\![n]\!] = n$ para cada $n \in \mathbb{N}$. Pero acá estaríamos abusando de la notación, ya que el primer n es una frase y el segundo es un número natural. La metavariable n no representa en ambos lugares exactamente lo mismo. Esto suele resolverse escribiendo $[\![\overline{n}]\!] = n$ para cada $n \in \mathbb{N}$, donde $\overline{n}$ es la manera de escribir el número natural n en el lenguaje.

El metalenguaje que se utilizará a todo lo largo de la materia es la teoría de conjuntos habitual de la matemática. Por el permanente uso de funciones matemáticas, muchas de las definiciones serán fácilmente traducibles a lenguajes de programación funcionales. Una diferencia importante entre el metalenguaje y, por ejemplo, Haskell, es que en el metalenguaje todas las funciones son totales.

Así, en nuestro metalenguaje las siguientes definiciones no son válidas:

```
fact :: ℤ → ℤ
fact n = if n == 0 then 1 else n * fact (n-1)

head :: [a] → a
head (a:as) = a
```

En efecto, fact no está definida para los enteros negativos y head no lo está para las listas vacías. En Haskell ambas definiciones son válidas, a pesar de que la evaluación de fact (-1) no termina y la de head [ ] da error.

Un caso extremo de no terminación lo proporciona la siguiente definición válida de Haskell:

bottom :: a → b
bottom a = bottom a

que es una función que no está definida para ningún argumento.

Metacircularidad. Volvamos a observar las ecuaciones que obtuvimos para este mini-lenguaje de expresiones:

$$\begin{aligned} [\![0]\!] &= 0 \\ [\![1]\!] &= 1 \\ &\vdots \\ [\![-e]\!] &= -[\![e]\!] \\ [\![e+f]\!] &= [\![e]\!] + [\![f]\!] \end{aligned}$$

Las ecuaciones no parecen decir mucho, definen la semántica del operador del lenguaje - (resp +) en función de la función correspondiente del metalenguaje - (resp +). Como ya observamos, no hay circularidad en la definición ya que en un caso se trata de un operador del lenguaje y en otro del metalenguaje. De todas formas, esta aparente circularidad tiene un nombre: **metacircularidad**.

La metacircularidad en algunas ecuaciones es habitual al definir semántica. Hay que estar atentos ya que se corre el riesgo de llevar inconscientemente "vicios" del metalenguaje al lenguaje que se intenta definir.

Observación final. Hemos presentado el mini-lenguaje en 3 etapas:

1. presentación de la gramática abstracta
2. definición de los dominios sintácticos
3. presentación de ecuaciones dirigidas por sintaxis que determinan la semántica denotacional

A lo largo de la materia estudiaremos varios lenguajes siguiendo siempre estas etapas. Para cada uno de los lenguajes así definidos, analizaremos sus propiedades.