Nociones Básicas de Sintaxis

Análisis de Lenguajes de Programación

Mauro Jaskelioff

15/08/2017

Sintaxis

- La sintaxis es la forma de un lenguaje.
- Sintaxis concreta:
  Las secuencias exactas de caracteres que son programas sintácticamente válidos.

$$1+2+3 \quad \neq \quad (1+2)+3 \quad \neq \quad (1+2)\\+3$$

- Sintaxis abstracta: La estructura esencial de los programas sintácticamente válidos.

$$1+2+3 \quad = \quad (1+2)+3 \quad = \quad (1+2)\\+3$$

Lenguajes Formales

- Un **lenguaje** es un conjunto de palabras.
- Una **palabra** (cadena) es una secuencia **finita** de símbolos.
- Con ϵ denotamos la **palabra vacía**, o sea la secuencia de cero símbolos.
- Un símbolo es un elemento de un conjunto **finito** denominado alfabeto Σ.
- Ejemplo:

alfabeto	$\Sigma = \{ a, b \}$
palabras	$\varepsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
lenguajes	$\emptyset, \{ \varepsilon \}, \{ a \}, \{ b \}, \{ a, aa \},$
	$\{ \varepsilon, a, aa, aaa \},$
	$\{ a^n \mid n \geqslant 0 \},$
	$\{ a^n \, b^n \mid n \geqslant 0, n \; par \}$

Palabras de un Alfabeto

- Dado un alfabeto Σ, se define el conjunto Σ^* como el conjunto de secuencias sobre Σ. Inductivamente:
  - $\epsilon \in \Sigma^*$
  - Para símbolo $x \in \Sigma$ y palabra $w \in \Sigma^*$, tenemos que $xw \in \Sigma^*$.
- Por ejemplo, para $\Sigma = \{0, 1\}$

$$\begin{aligned}\Sigma^* = \{&\epsilon,\\ &0, 1,\\ &00, 10, 01, 11,\\ &000, 100, 010, 110, 001, 101, 011, 111,\\ &\ldots\}\end{aligned}$$

- Nota: Hay infinitas palabras en Σ^*, pero cada palabra es de longitud **finita**.

Lenguajes

- Dado un alfabeto Σ, un lenguaje L es

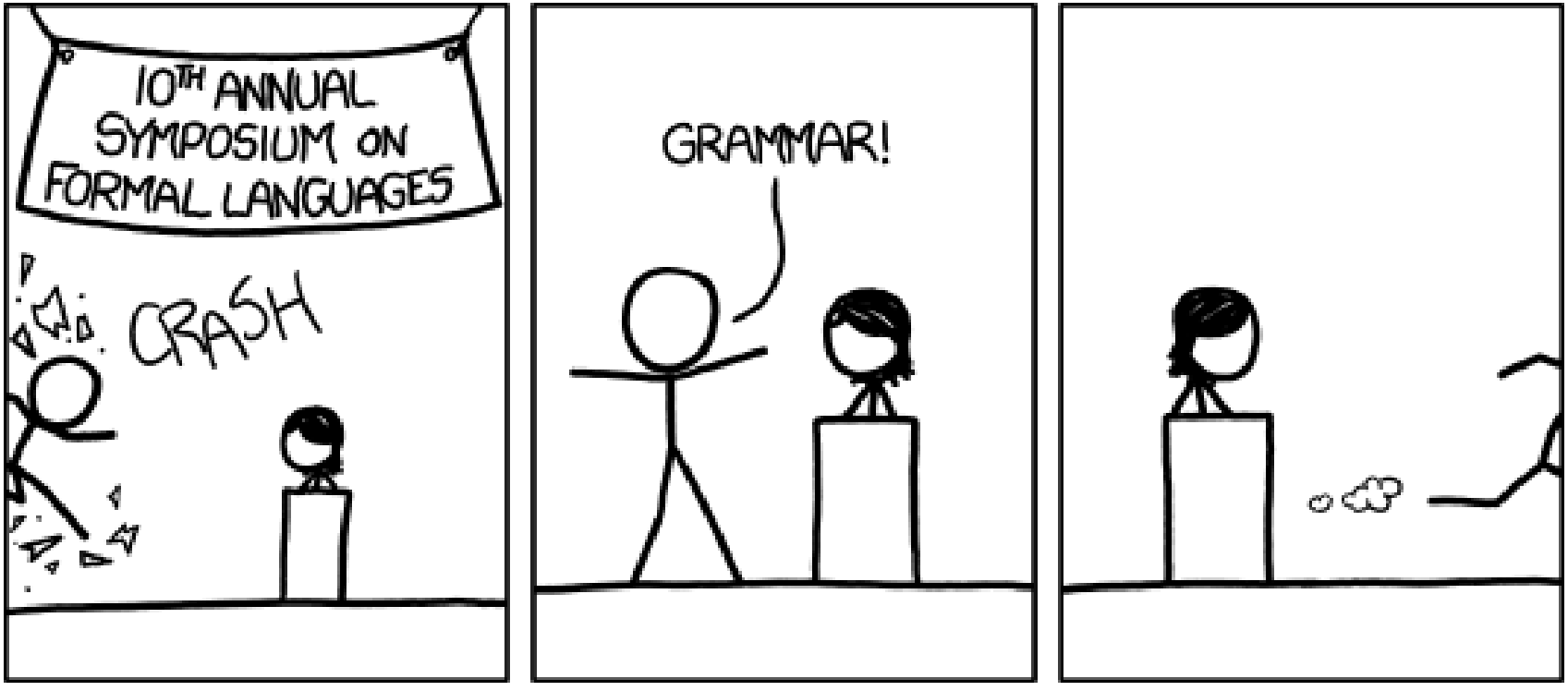
$$L \subseteq \Sigma^*$$

o equivalentemente

$$L \in \mathcal{P}(\Sigma^*)$$

- Ejemplos
  - El conjunto de programas válidos en C es un lenguaje sobre el conjunto de caracteres ASCII.
  - El conjunto de programas válidos en Haskell es un lenguaje sobre el conjunto de caracteres Unicode.

Gramáticas Libres de Contexto



Gramáticas Libres de Contexto

- Las **Gramáticas libres de contexto** (CFG) son una manera de describir lenguajes libres de contexto.
- Capturan nociones frecuentes en lenguajes de programación:
  - estructura anidada,
  - paréntesis balanceados,
  - palabras clave emparejadas como `begin` y `end`.
- La mayoría de los lenguajes razonables pueden ser reconocidos en forma bastante simple: basta un autómata de pila determinístico.

Gramáticas Libres de Contexto

Una gramática libre de contexto es una tupla (N, T, P, S)

- N es un conjunto finito de **no terminales**
- T es un conjunto finito de **terminales** (el alfabeto del lenguaje que se describe)
- $N \cap T = \emptyset$ (N y T son disjuntos)
- S es el **símbolo inicial**, un elemento de N.
- P es un conjunto de **producciones** de la forma $A \to \alpha$, donde $A \in N$ y $\alpha \in (N \cup T)^*$.

Ejemplo de CFG

- $G = (\{S, A\}, \{a, b\}, P, S)$ con P compuesto por las siguientes producciones:

$$S \to \varepsilon$$
$$S \to aA$$
$$A \to bS$$

- Las producciones con el mismo lado izq, se pueden agrupar (Backus-Naur Form o BNF):

$$S \to \varepsilon \mid aA$$

- A veces las producciones se escriben $A ::= \alpha$.

La relación de derivación directa

- Para obtener el lenguaje generado por una gramática $G = (N, T, P, S)$ definimos la relación binaria $\Rightarrow$ para cadenas de $N \cup T$ como la menor relación tal que

$$\alpha A \gamma \Rightarrow \alpha \beta \gamma$$

  cuando $A \to \beta$ es una producción de G.
- Una producción puede ser aplicada sin importar el contexto (de ahí, libre de contexto).
- Ejemplo: para la gramática

$$S \to \varepsilon \mid aA$$
$$A \to bS$$

  tenemos que

$$S \Rightarrow \varepsilon \qquad aA \Rightarrow abS$$
$$S \Rightarrow aA \qquad SaAaa \Rightarrow SabSaa$$

La relación de derivación

- La relación de derivación $\Rightarrow^*$ es la clausura reflexiva transitiva de $\Rightarrow$.
- O sea, es la menor relación sobre cadenas en $N \cup T$ tal que:
  - $\alpha \Rightarrow^* \beta$ si $\alpha \Rightarrow \beta$
  - $\alpha \Rightarrow^* \alpha$
  - $\alpha \Rightarrow^* \beta$ si $\alpha \Rightarrow^* \gamma \quad \wedge \quad \gamma \Rightarrow^* \beta$
- Ejemplo: para la gramática

$$S \to \varepsilon \mid aA$$
$$A \to bS$$

tenemos que

$$S \Rightarrow^* \varepsilon \qquad S \Rightarrow^* abS$$
$$S \Rightarrow^* aA \qquad S \Rightarrow^* ababS$$
$$aA \Rightarrow^* abS \qquad S \Rightarrow^* abab$$

Lenguaje generado por una gramática

- El lenguaje generado por una gramática $G = (N, T, P, S)$, se denota $L(G)$, y se define como:

$$L(G) = \{w \mid w \in T^* \wedge S \Rightarrow^* w\}$$

- Un lenguaje L es libre de contexto (CFL) sii $L = L(G)$ para algún CFG G.
- Una cadena $\alpha \in (N \cup T)^*$ es una forma sentencial sii $S \Rightarrow^* \alpha$.
- Ejemplo: para la gramática G

$$S \to \varepsilon \mid aA$$
$$A \to bS$$

$$L(G) = \{(ab)^i \mid i \geqslant 0\} = \{\epsilon, ab, abab, ababab, \dots\}$$

Árbol de Parseo

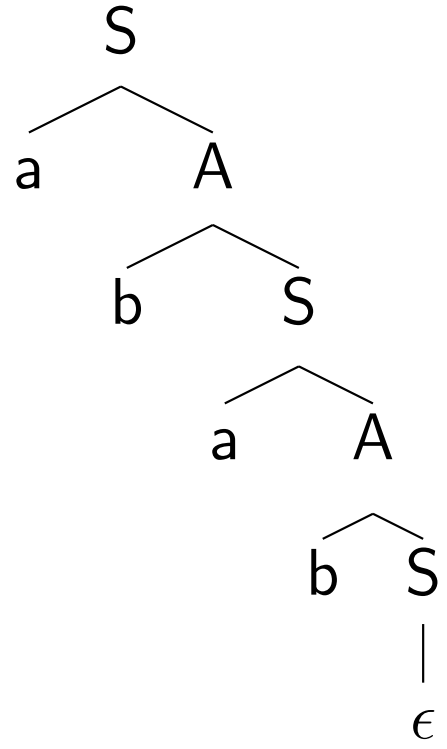
Un árbol es una derivación o **árbol de parseo** de un CFG $G = (N, T, P, S)$ si:

- cada nodo tiene una etiqueta en $N \cup T \cup \{\epsilon\}$,
- la etiqueta de la raíz es S,
- la etiquetas de los nodos interiores están en N,
- si el nodo n tiene etiqueta A e hijos $n_1, n_2, \ldots, n_k$ (de izq. a der.) con etiquetas $X_1, X_2, \ldots, X_k$, entonces $A \to X_1 X_2 \ldots X_k$ es una producción en P,
- si un nodo n tiene etiqueta ϵ, entonces n es una hoja y es hijo único.

Árbol de parseo: Ejemplo

- Dada la gramática G

$$S \to \varepsilon \mid aA$$
$$A \to bS$$



La cadena $abab \in L(G)$ tiene el árbol de parseo

- La cadena de etiquetas en las hojas de izq. a der. es el **resultado** del árbol.
- El resultado es una forma sentencial de G.
- Propiedad: α es el resultado de un árbol sii $S \Rightarrow^* \alpha$.

Ejercicio

$$G = (\{E, E_p, V, I, D\},$$
$$\{+, -, *, /, (,), x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\},$$
$$P, E)$$

donde P consiste de las producciones

$$\begin{aligned}
E &\rightarrow E + E \mid E - E \mid E * E \mid E \,/\, E \mid E_p \\
E_p &\rightarrow V \mid I \mid (E) \\
V &\rightarrow x \mid y \mid z \\
I &\rightarrow DI \mid D \\
D &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}$$

Dibujar el árbol de parseo de las siguientes expresiones

- $1 + x$
- $42 * x - z$
- $(13 + 7 * x * (y - 123))$
- $(1$

Gramáticas Ambiguas

- Una CFG G es ambigua si alguna palabra en $L(G)$ tiene **más de un árbol de parseo**.
- Un CFL para el que toda CFG es ambigua es **inherentemente ambiguo**.
- La mayoría de los CFLs no son inherentemente ambiguos, por lo que una CFG G ambigua, usualmente puede ser **transformada** a una **equivalente** G' que no es ambigua.
- En general, la ambigüedad de una CFG **no es decidible**.

Eliminando Ambigüedad: el `else` colgado

- La siguiente gramática posee el problema del "`else` colgado" (dangling else).

$$\begin{aligned} \mathit{Stmt} \rightarrow\ & \textbf{if } \mathit{Expr} \textbf{ then } \mathit{Stmt} \\ \mid\ & \textbf{if } \mathit{Expr} \textbf{ then } \mathit{Stmt} \textbf{ else } \mathit{Stmt} \\ \mid\ & \textbf{other} \end{aligned}$$

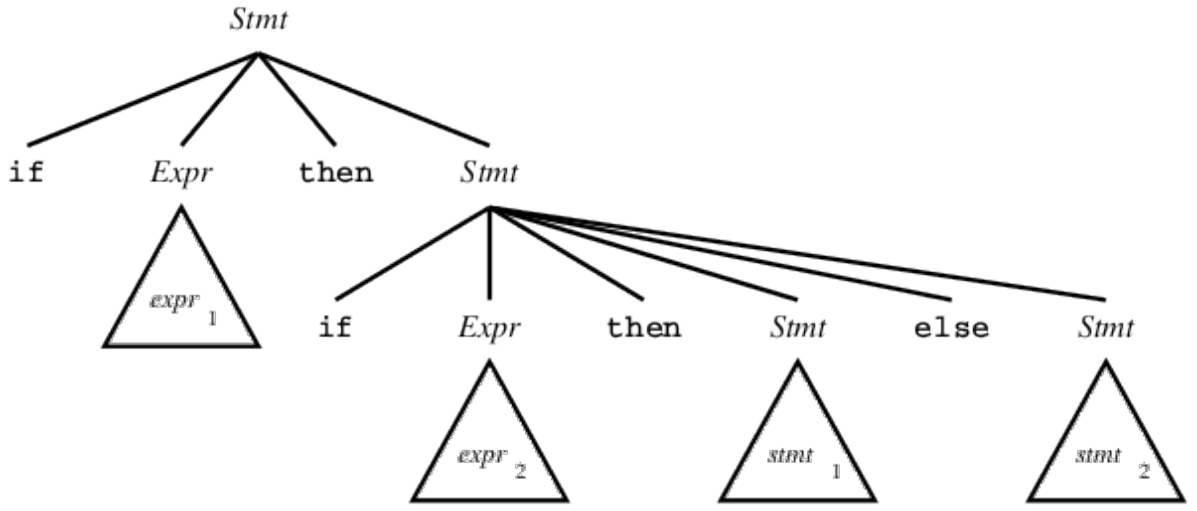
- el programa

$$\textbf{if } expr_1 \textbf{ then if } expr_2 \textbf{ then } stmt_1 \textbf{ else } stmt_2$$

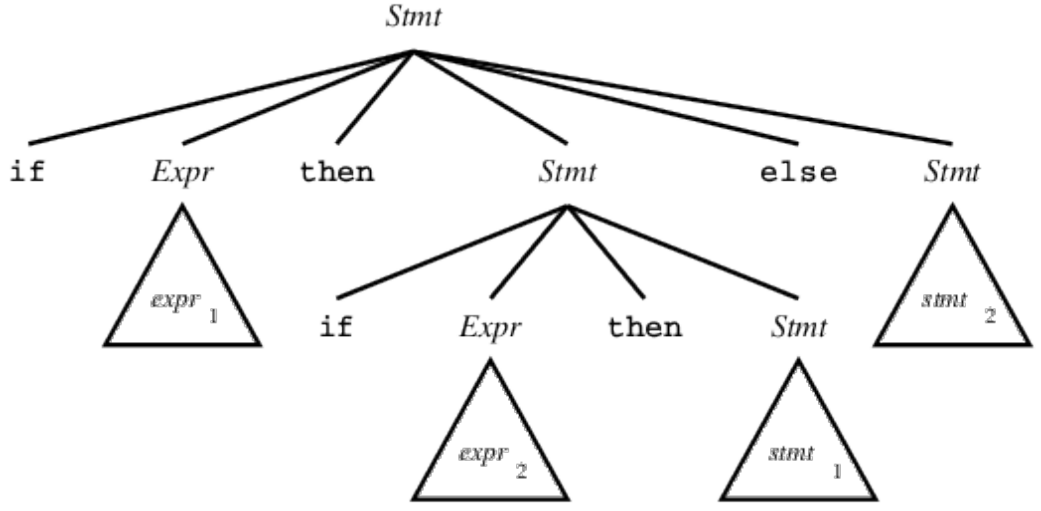
  tiene 2 árboles de parseo posibles.
- Por lo tanto la gramática es ambigua.

Eliminando Ambigüedad: el `else` colgado

Árbol 1:



Árbol 2:



Desambiguando gramáticas

- Resolver la ambigüedad es importante.
  ¡Los dos árboles tienen diferente semántica!
- Por ejemplo, suponer que $expr_1$ evalúa a $True$ y $expr_2$ a $False$. ¿Qué sentencias se ejecutan en cada caso?
- Elegimos una interpretación

  > *"Asociamos cada* **else** *al* **then** *más cercano que no esté asociado a otro* **else**."

  - O sea, preferimos el árbol 1.
- ¿Cómo lo formalizamos?
- Transformando la gramática en una equivalente pero sin ambigüedades.

Desambiguando gramáticas

- Idea: Una sentencia entre un **then** y un **else** debe ser una sentencia "matcheada".

$$
\begin{aligned}
\mathit{Stmt} &\rightarrow \mathit{MatchedStmt} \\
&\mid \mathit{UnmatchedStmt} \\
\mathit{MatchedStmt} &\rightarrow \textbf{if } \mathit{Expr} \textbf{ then } \mathit{MatchedStmt} \\
&\qquad\qquad \textbf{else } \mathit{MatchedStmt} \\
&\mid \textbf{other} \\
\mathit{UnmatchedStmt} &\rightarrow \textbf{if } \mathit{Expr} \textbf{ then } \mathit{Stmt} \\
&\mid \textbf{if } \mathit{Expr} \textbf{ then } \mathit{MatchedStmt} \\
&\qquad\qquad \textbf{else } \mathit{UnmatchedStmt}
\end{aligned}
$$

Ejercicio

Considere la gramática para expresiones aritméticas vista anteriormente.
¿Es ambigua? Justificar.

- Las expresiones aritméticas se desambiguan mediante convenciones como precedencia de operadores y asociatividad.
- Las mismas ideas se pueden usar para desambiguar gramáticas.

Sintaxis Concreta y Abstracta

- Los árboles de parseo que vimos se refieren a la **sintaxis concreta**.
  - Se refieren a cadenas sin estructura.
  - Contienen detalles poco interesantes.
  - Su estructura no es única si la gramática es ambigua.
- Por esto, es conveniente trabajar con árboles de **sintaxis abstracta**.
  - La sintaxis abstracta es un árbol.
    No puede haber problemas de ambigüedad.
  - Las gramáticas son mas simples ya que no es necesario tener en cuenta formato ni paréntesis.

Ejemplo de sintaxis concreta

- La *sintaxis concreta* de un lenguaje de expresiones aritméticas (simplificado) es, en BNF:

$$
\begin{array}{lll}
\langle digit \rangle & ::= & \texttt{'0'} \mid \texttt{'1'} \mid \ldots \mid \texttt{'9'} \\
\langle nat \rangle & ::= & \langle digit \rangle \mid \langle digit \rangle \ \langle nat \rangle \\
\langle intexp \rangle & ::= & \langle nat \rangle \\
 & & \mid \langle intexp \rangle \ \texttt{'+'} \ \langle intexp \rangle \\
 & & \mid \langle intexp \rangle \ \texttt{'*'} \ \langle intexp \rangle \\
 & & \mid \texttt{'('} \ \langle intexp \rangle \ \texttt{')'}
\end{array}
$$

- La sintaxis es ambigua
  $10 * 2 + 3 = ?$
- La desambiguamos especificando asociatividad y precedencia de operadores.

Sintaxis Abstracta

- Lo que nos interesa son los naturales y expresiones como entidades abstractas.
- El *árbol de sintaxis abstracta* (AST) en BNF es:

$$\begin{aligned}\langle intexp\rangle ::= & \langle nat\rangle \\ & \mid \langle intexp\rangle + \langle intexp\rangle \\ & \mid \langle intexp\rangle * \langle intexp\rangle\end{aligned}$$

- Los AST no contienen ambigüedades.
- ¿Pero qué es lo que estamos definiendo?

Definiendo Términos

- Los AST me definen un conjunto de términos
- El conjunto de términos del lenguaje es el menor conjunto T tal que
  1. $n \in \mathbb{N}$ entonces $n \in T$
  2. $t, u \in T$ entonces $t + u \ \in T$
  3. $t, u \in T$ entonces $t * u \ \in T$
- También podemos definir el conjunto mediante reglas de inferencia

$$\frac{n \in \mathbb{N}}{n \in T} \qquad \frac{t \in T \quad u \in T}{t + u \ \in T} \qquad \frac{t \in T \quad u \in T}{t * u \ \in T}$$

- Los símbolos $+$ y $*$ son arbitrarios y pueden ser diferentes a la sintaxis concreta.

Implementación en Haskell

- En Haskell podemos usar un tipo de datos algebraico para representar el AST de un lenguaje.

$$\begin{aligned}\mathbf{data}\ IntExp = {} & Num\ Int \\ \mid {} & Sum\ \ IntExp\ IntExp \\ \mid {} & Prod\ \ IntExp\ IntExp\end{aligned}$$

- Los elementos de $IntExp$ son árboles con etiquetas Num, Sum y $Prod$ en sus nodos.
- En la implementación se ve bien claro que tratamos con árboles y que los nombres de los constructores son arbitrarios.

Lenguaje de Expresiones Aritméticas

Consideremos el árbol de sintaxis abstracta (AST) de un lenguaje:

$$
\begin{aligned}
t ::= \ & \texttt{true} \\
\mid \ & \texttt{false} \\
\mid \ & \texttt{if } t \texttt{ then } t \texttt{ else } t \\
\mid \ & 0 \\
\mid \ & \texttt{succ } t \\
\mid \ & \texttt{pred } t \\
\mid \ & \texttt{iszero } t
\end{aligned}
$$

Un programa es simplemente un término del lenguaje:

- El programa $\texttt{if false then } 0 \texttt{ else } 1$ evalúa a 1.
- El programa $\texttt{iszero } (\texttt{pred } (\texttt{succ } 0))$ evalúa $\texttt{true}$.

Detalles de notación

- Las *metavariables* están en el metalenguaje. Se usan para representar entidades del lenguaje objeto.
- La metavariable t, letras cercanas como u, s y r, y variaciones como t_1 o t' representan términos del lenguaje.
- Un **término** es una frase que computa.
- Una **expresión** es más general (por ejemplo podríamos tener una expresión de tipos.)
- Salvo que se aclare siempre nos referimos a la sintaxis abstracta. Los paréntesis se usan exclusivamente para expresar un árbol como una cadena de texto.

Los Términos dados Inductivamente

- Podemos definir el conjunto de términos inductivamente: Los términos son el menor conjunto $\mathcal{T}$ tal que
  1. $\{\texttt{true}, \texttt{false}, 0\} \subseteq \mathcal{T}$;
  2. si $t_1 \in \mathcal{T}$, entonces $\{\texttt{succ}\ t_1, \texttt{pred}\ t_1, \texttt{iszero}\ t_1\} \subseteq \mathcal{T}$;
  3. si $t_1 \in \mathcal{T}, t_2 \in \mathcal{T}$, y $t_3 \in \mathcal{T}$, entonces $\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \in \mathcal{T}$
- El requerimiento que $\mathcal{T}$ sea el conjunto más chico dice que en $\mathcal{T}$ sólo están los elementos requeridos por las tres cláusulas.
- Recordar que $\mathcal{T}$ es un conjunto de **árboles**.

Los Términos dados por Reglas de Inferencia

$$\texttt{true} \in \mathcal{T} \qquad \texttt{false} \in \mathcal{T} \qquad 0 \in \mathcal{T}$$

$$\frac{t_1 \in \mathcal{T}}{\texttt{succ}\ t_1 \in \mathcal{T}} \qquad \frac{t_1 \in \mathcal{T}}{\texttt{pred}\ t_1 \in \mathcal{T}} \qquad \frac{t_1 \in \mathcal{T}}{\texttt{iszero}\ t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T} \quad t_2 \in \mathcal{T} \quad t_3 \in \mathcal{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \in \mathcal{T}}$$

- En este estilo, usualmente se deja implícito que uno está interesado en el menor conjunto que satisface las reglas.
- A menudo decimos "reglas de inferencia", pero cuando contienen metavariables, son en realidad **esquemas de reglas**.

Los Términos en forma concreta

- Para cada $i \in \mathbb{N}$, definimos un conjunto S_i.

$$\begin{aligned} S_0 &= \emptyset \\ S_{i+1} &= \{\texttt{true}, \texttt{false}, 0\} \\ &\cup \{\texttt{succ}\ t_1, \texttt{pred}\ t_1, \texttt{iszero}\ t_1 \mid t_1 \in S_i\} \\ &\cup \{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \mid t_1, t_2, t_3 \in S_i\} \end{aligned}$$

- Luego, definimos

$$S = \bigcup_{i \in \mathbb{N}} S_i$$

- Ejercicio: ¿Cuántos elementos tiene S_3?

  Solución: $|S_{i+1}| = |S_i|^3 + |S_i| \times 3 + 3$, $|S_0| = 0$.

$$|S_3| = 59439$$

Equivalencia de las definiciones

- La definición inductiva y la dada por reglas caracterizan los términos como un conjunto que satisface cierta propiedad de clausura.
- La definición concreta muestra como construir el conjunto como el límite de una secuencia.

Proposición ($\mathcal{T} = S$)

El conjunto $\mathcal{T}$ fue definido como el menor conjunto que satisface ciertas condiciones. Para probar el teorema basta:

1. probar que S satisface las condiciones;
2. probar que todo conjunto que satisface las condiciones contiene a S.

Resumen

- Los lenguajes de programación son lenguajes formales.
- En general, basta gramáticas libres de contexto para describir su sintaxis concreta.
- La sintaxis concreta se refiere a cadenas, por lo que puede ser ambigua.
- La sintaxis abstracta trata con árboles, por lo que no es ambigua.
- Para procesar un programa, el primer paso es obtener su árbol de sintaxis abstracta (AST).
- Un AST es un conjunto de términos definidos inductivamente.
  - Caracterizaciones. Pruebas por inducción estructural.

Bibliografía

- *Types and Programming Languages*. B.C. Pierce.
- *Foundations of Programming Languages*. J.C. Mitchell.