Tema 2. Fundamentos de los Lenguajes de Programación

Lenguajes, Tecnologías y Paradigmas de Programación (LTP)

DSIC, ETSInf

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

etsinf
Escuela Técnica Superior de Ingeniería Informática

Descripción formal de un LP

- **Sintaxis**: qué secuencia de caracteres constituyen un programa "legal"
  - elementos sintácticos del lenguaje
- **Semántica**: qué significa (qué calcula) un programa legal dado. Importancia:
  1. Ayuda al programador a "razonar" sobre el programa
  2. Es necesaria para implementar correctamente el lenguaje (modelos de ejecución)
  3. Permite desarrollar técnicas y herramientas de:
     - Análisis y Optimización
     - Depuración
     - Verificación
     - Transformación

Sintaxis

Uso de gramáticas BNF

Notación BNF:

- Con <w> se nombra un grupo de expresiones definido por alguna regla de construcción de expresiones
- el símbolo | significa "or"

```
<letter>   ::=   a | b | c | d | A | B | C | D
 <digit>   ::=   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
    <id>   ::=   <letter> | <id><letter> | <id><digit>
```

- los corchetes [y] se sitúan alrededor de los items opcionales
- las llaves {} (o el asterisco *) sirven para indicar una secuencia de 0 o más items
- el símbolo + sirve para indicar una secuencia de 1 o más items

```
<IdNum>   ::=   <letter>+ {<digit>}
```

Sintaxis

Uso de gramáticas BNF

Ejemplo: sintaxis del bucle while

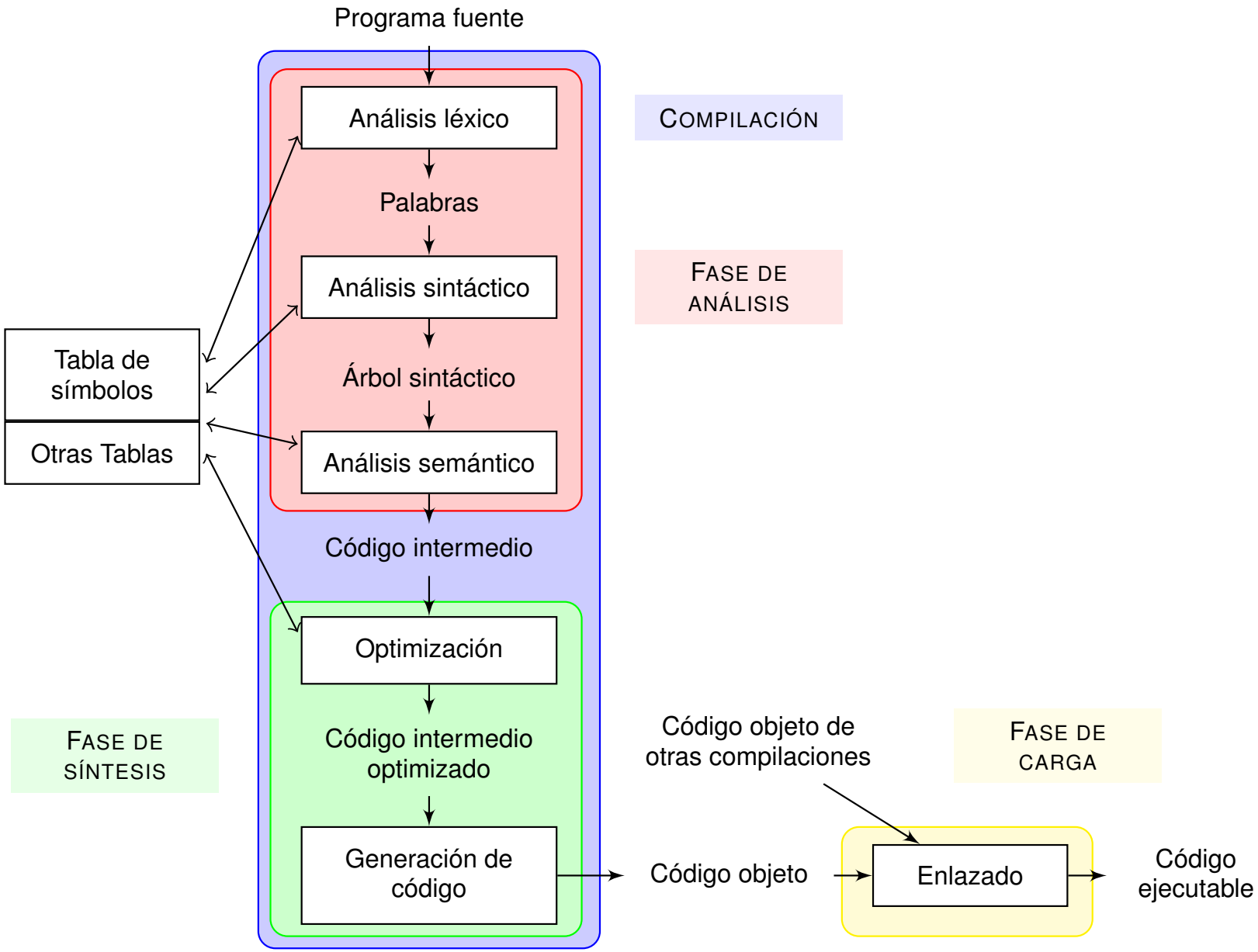
Java

<while_statement> ::= `while` (<expression>) <statement>

Modula-2

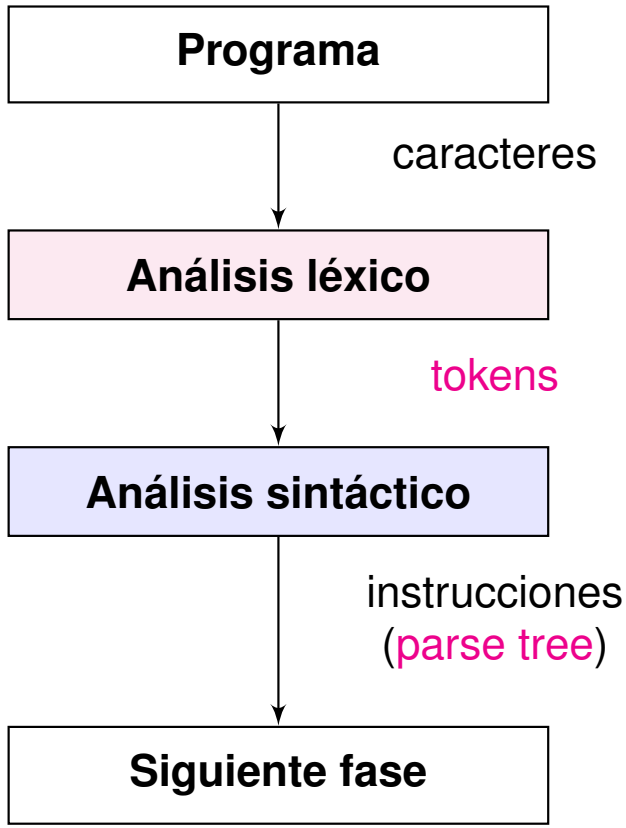
<while_statement> ::= `WHILE` <expression> `DO`
<statement> { `;` <statement> }
`END`

Procesamiento de un programa fuente:



Análisis léxico y sintáctico

- El analizador léxico (scanner) divide una secuencia de caracteres (el programa) en una secuencia de componentes sintácticos primitivos o palabras (tokens) que actúan como identificadores, números, palabras reservadas, etc.
- El analizador sintáctico (parser) reconoce una secuencia de tokens y obtiene una secuencia de instrucciones en forma de árbol sintáctico estructura



Ejemplo

1. Secuencia de caracteres

```
f,u,n,{,F,a,c,t,’’,N,},\n,’’,i,f,’’,N, =,=,0, ’’, t,h,e,n,
’’,1,\n,’’,e,l,s,e,’’,N,*,{,F,a, c,t,’’,N, -,1,},’
’,e,n,d,i,f,\n,e,n,d
```

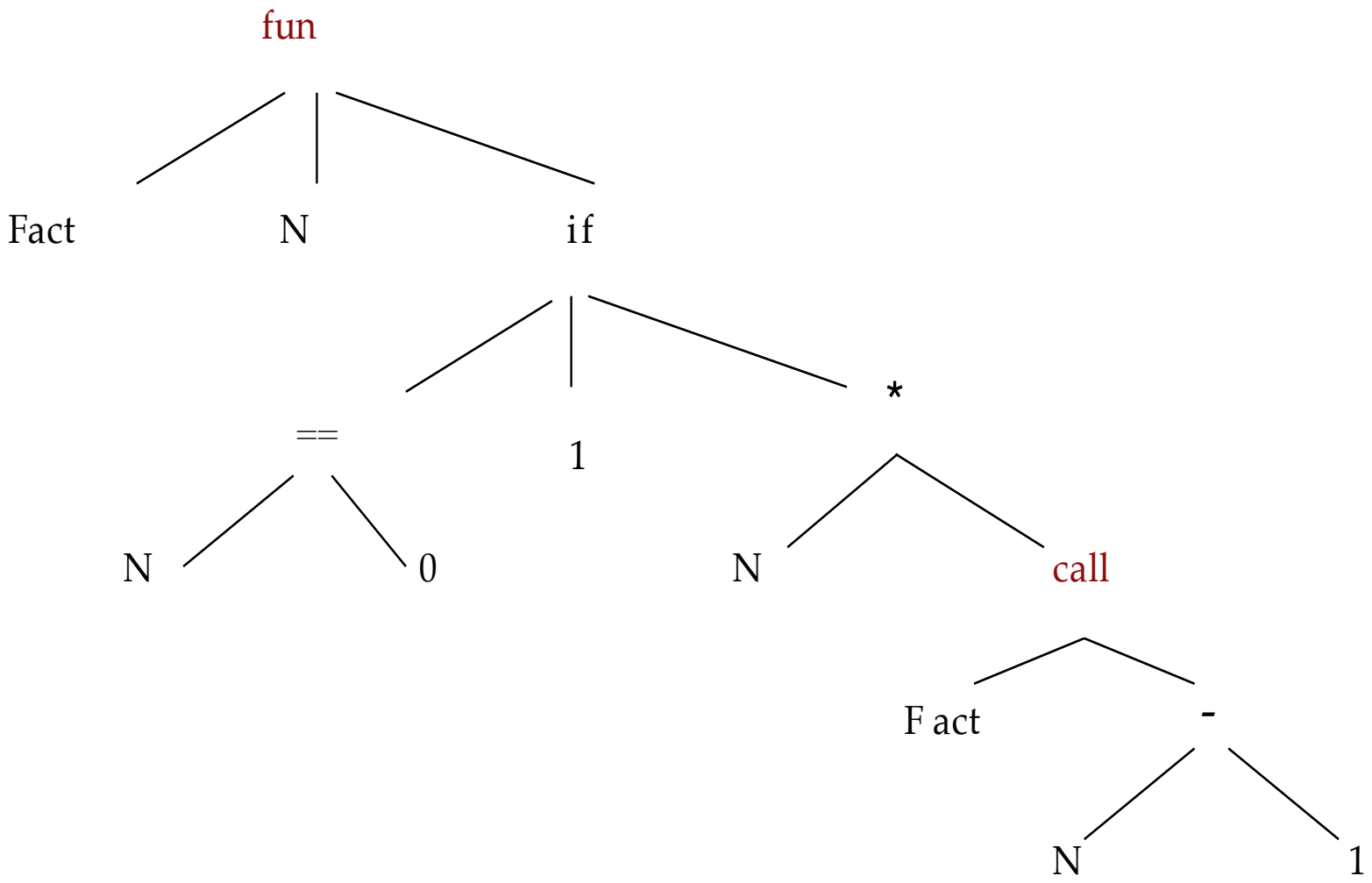
2. Secuencia de palabras (tokens)

```
fun,{,Fact,N,},if,N,==,0,then,1,else,N,*,{,Fact,N,
-,1,},endif,end
```

3. Instrucción

```
fun {Fact N}
   if N == 0 then 1 else N*{Fact N-1}
   endif
end
```

Ejemplo



Arbol sintáctico (parse tree)

Análisis semántico

descripción semántica: concepto y necesidad

Semántica estática: restricciones de la sintaxis que no pueden expresarse mediante la notación BNF pero que sí pueden comprobarse en tiempo de compilación

Ejemplo

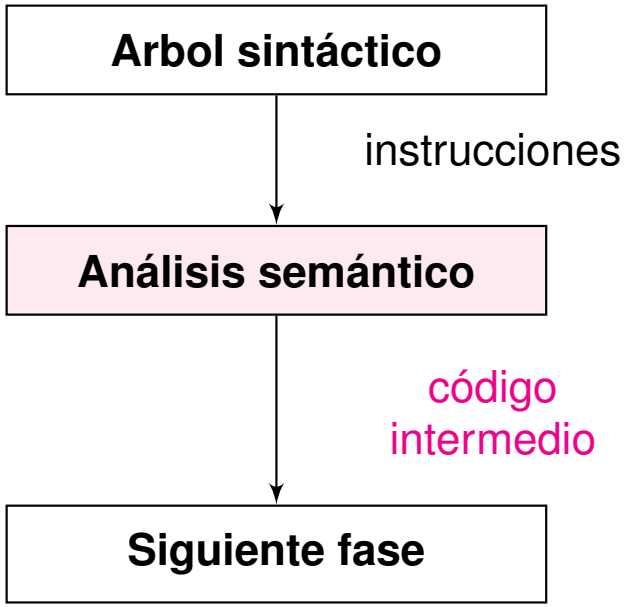
A := B + C podría no ser legal si A, B o C no han sido declaradas previamente

Semántica dinámica: restricciones que solo se pueden comprobar durante la ejecución del programa (e.g. comprobación de índices dentro del rango del vector)

Análisis semántico

Semántica estática

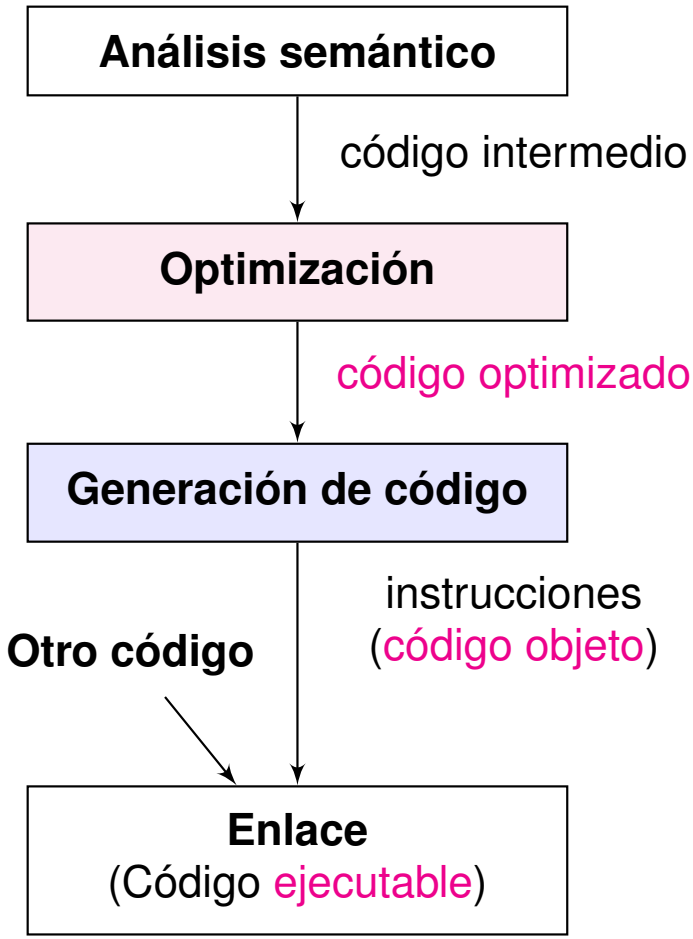
- Comprobaciones en el analizador semántico:
  1. Declaración de variables previa a su uso
  2. Compatibilidad y conversión de tipos (coercion)
  3. Signatura de las funciones: los parámetros reales coinciden en número y tipo con los formales
  4. ...
- Produce un código intermedio que es la base para el proceso de compilación posterior



Optimización y Enlace

Optimización de código

- Primero se optimiza el código intermedio recibido de la fase anterior
- La fase de generación de código produce el código objeto del programa
- El código objeto se enlaza con código procedente de otros programas o librerías para obtener el código ejecutable.



Ejemplo

Evolución de la representación interna de un programa durante las distintas etapas del proceso de compilación.

Consideraremos el siguiente programa:[1]

```
posicion = inicial + velocidad * 60
```

donde las variables posición, inicial y velocidad son de tipo real.

[1] En las páginas 12 y 13 de *Aho, Sethi and Ullman. Compiladores: Principios, técnicas y herramientas. Addison-Wesley Iberoamericana, 1990.*

Semántica dinámica

¿Por qué la semántica no es siempre "estática"?

El compilador no puede detectar todos los errores posibles:

1. Algunos errores sólo se manifiestan durante la ejecución:
   - Z=X/Y produce un error si se ejecuta con Y = 0
   - Z=V[Y] produce un error si Y tiene un valor que cae fuera del rango del vector V
2. Muchas propiedades interesantes de un programa no son decidibles.
   - la terminación (pero es 'semidecidible': basta ejecutar el programa para "semi-decidirlo")
   - si dos programas cualesquiera computan la misma función
   - si dos descripciones BNF generan el mismo lenguaje

Semántica dinámica

Estilos de definición semántica

- **Operacional**
- **Axiomática**
- Declarativa:
  - Denotacional
  - Algebraica
  - Teoría de modelos
  - Punto fijo

Semántica Operacional

Consiste en definir una máquina (abstracta) M y expresar el significado de cada construcción del lenguaje en términos de las acciones a realizar por la máquina para ejecutar dicha instrucción.

Semántica Operacional

- Representamos el estado de la máquina (abstracta) que ejecuta el programa como una función $s : \mathcal{X} \to D$ que asigna valores en un dominio D a las variables $x, y, \ldots \in \mathcal{X}$ del programa.

Notación

Puesto que en un programa utilizamos un conjunto finito de variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, podemos representar el estado como una conjunto de pares variable-valor:

$$s = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}.$$

- La configuración de la máquina es un par

$$\langle i, s \rangle$$

que registra el estado actual (s) junto a la instrucción a evaluar (i), bien sea simple o compuesta (un programa se considera una instrucción compuesta).

Semántica Operacional

- Para formalizar la ejecución del programa en la máquina utilizamos una relación de transición '$\rightarrow$' entre configuraciones.
- La relación se define mediante reglas de transición:

$$\frac{premisa}{\langle i, s\rangle \rightarrow \langle i', s'\rangle} \quad (1)$$

que describen la configuración $\langle i', s'\rangle$ obtenida a partir la configuración de partida $\langle i, s\rangle$ cuando se satisface la premisa o condición sobre la configuración $\langle i, s\rangle$.

- También utilizamos otras relaciones para describir
  - la evaluación de expresiones aritméticas ($\langle exp, s\rangle \Rightarrow n$).
  - la obtención directa de un estado final ($\langle i, s\rangle \Downarrow s'$).

  y las definimos mediante reglas similares a (1).

El lenguaje SIMP

Sintaxis

La gramática en estilo BNF del minilenguaje imperativo SIMP que utilizaremos en este tema se define así:

- Expresiones aritméticas:

$$a ::= C \mid V \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$$

  donde C y V denotan las constantes numéricas $(0, 1, 2, \ldots)$ y las variables $(x, y, \ldots)$ respectivamente

- Expresiones booleanas:

$$b ::= \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \vee b_2$$

- Instrucciones:

$$i ::= \mathit{skip} \mid V := a \mid i_1 ; i_1 \mid \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2 \mid \texttt{while } b \texttt{ do } i$$

  donde *skip* denota la instrucción vacía.

El lenguaje SIMP

Evaluación de expresiones

- Escribimos $\langle exp, s \rangle \Rightarrow n$ para indicar que la expresión exp se evalúa a n en el estado s.
- Usamos esta relación de evaluación para evaluar las expresiones aritméticas y booleanas.

El lenguaje SIMP

Evaluación de expresiones aritméticas

- Constantes numéricas:

$$\langle \mathtt{n}, s \rangle \Rightarrow n$$

- Variables:

$$\langle x, s \rangle \Rightarrow s(x)$$

Recordemos que el estado s es una función de variables en valores. $s(x)$ no es más que el valor de la variable x en el estado de la máquina s.

- Adición:

$$\frac{\langle a_1, s \rangle \Rightarrow n_1 \qquad \langle a_2, s \rangle \Rightarrow n_2}{\langle a_1 + a_2, s \rangle \Rightarrow n}$$

  si n es la suma de n_1 y n_2.
- Resta y producto: similar.

El lenguaje SIMP

Evaluación de expresiones booleanas

- Valores booleanos:

$$\langle \texttt{false}, s\rangle \Rightarrow \mathit{false} \qquad \langle \texttt{true}, s\rangle \Rightarrow \mathit{true}$$

- Igualdad:

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 = a_2, s\rangle \Rightarrow \mathit{true}} \quad \text{si } n_1 \text{ y } n_2 \text{ son iguales}$$

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 = a_2, s\rangle \Rightarrow \mathit{false}} \quad \text{si } n_1 \text{ y } n_2 \text{ son distintos}$$

- Menor o igual:

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 \leq a_2, s\rangle \Rightarrow \mathit{true}} \quad \text{si } n_1 \text{ es menor o igual que } n_2$$

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 \leq a_2, s\rangle \Rightarrow \mathit{false}} \quad \text{si } n_1 \text{ es mayor que } n_2$$

- Negación:

$$\frac{\langle b,s\rangle \Rightarrow \mathit{true}}{\langle \neg b, s\rangle \Rightarrow \mathit{false}} \qquad \frac{\langle b,s\rangle \Rightarrow \mathit{false}}{\langle \neg b, s\rangle \Rightarrow \mathit{true}}$$

- Disyunción: EJERCICIO

Semántica Operacional

Paso pequeño (*small-step*)

- En la descripción semántica operacional de paso pequeño la ejecución de un programa se puede seguir instrucción a instrucción.
- Al ejecutar un programa P a partir del estado inicial s_I (donde ninguna variable está asignada a ningún valor, es decir: $s_I = \{\}$), se obtiene una secuencia de configuraciones (denominada traza):

$$\langle P, s_I \rangle = \langle P_1, s_1 \rangle \rightarrow \langle P_2, s_2 \rangle \rightarrow \cdots \rightarrow \langle P_n, s_n \rangle$$

Distinguimos dos situaciones:

1. P_n es la instrucción vacía (*skip*) para algún $n \geq 1$. Entonces la ejecución del programa termina con un estado final $s_F = s_n$.
2. P_n nunca llega a ser la instrucción vacía para ningún n: la ejecución del programa no termina.

El lenguaje SIMP

Semántica de paso pequeño (I)

- Secuencia:

$$\frac{}{\langle skip;i,s\rangle \to \langle i,s\rangle} \qquad \frac{\langle i_1,s\rangle \to \langle i_1',s'\rangle}{\langle i_1;i_2,s\rangle \to \langle i_1';i_2,s'\rangle}$$

- Asignación:

$$\frac{\langle a,s\rangle \Rightarrow n}{\langle x{:=}a,s\rangle \to \langle skip, s[x \mapsto n]\rangle}$$

donde el nuevo estado $s[x \mapsto n]$ se define eliminando de s el posible vínculo que exista para x y añadiendo en cualquier caso el nuevo vínculo $x \mapsto n$:

$$s[x \mapsto n](y) = \begin{cases} s(y) & \text{si } y \neq x \\ n & \text{si } y = x \end{cases}$$

El lenguaje SIMP

Semántica de paso pequeño (II)

- Condicional:

$$\frac{\langle b,s\rangle \Rightarrow true}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \rightarrow \langle i_1, s\rangle} \qquad \frac{\langle b,s\rangle \Rightarrow false}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \rightarrow \langle i_2, s\rangle}$$

- Bucle *while*:

$$\frac{\langle b,s\rangle \Rightarrow false}{\langle \texttt{while } b \texttt{ do } i, s\rangle \rightarrow \langle skip, s\rangle} \qquad \frac{\langle b,s\rangle \Rightarrow true}{\langle \texttt{while } b \texttt{ do } i, s\rangle \rightarrow \langle i\texttt{;while } b \texttt{ do } i, s\rangle}$$

Ejercicio

Definir la semántica del bucle *while* con una única regla utilizando la instrucción condicional.

Semántica Operacional

Paso grande (*big-step*)

- En la descripción semántica operacional de paso grande (big-step) se especifica la ejecución de un programa P como una transición directa desde la configuración inicial $\langle P, s_I \rangle$ al estado final s_F.
- A diferencia de la semántica de paso pequeño, pues, la relación de transición de paso grande $\Downarrow$ relaciona configuraciones con estados: $\langle P, s \rangle \Downarrow s'$

El lenguaje SIMP

Semántica de paso grande

- Instrucción vacía:

$$\frac{}{\langle skip, s\rangle \Downarrow s}$$

- Secuencia:

$$\frac{\langle i_1, s\rangle \Downarrow s_1 \qquad \langle i_2, s_1\rangle \Downarrow s'}{\langle i_1; i_2, s\rangle \Downarrow s'}$$

- Asignación:

$$\frac{\langle a, s\rangle \Rightarrow n}{\langle x{:=}a, s\rangle \Downarrow s[x \mapsto n]}$$

- Condicional:

$$\frac{\langle b,s\rangle \Rightarrow true \qquad \langle i_1,s\rangle \Downarrow s'}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \Downarrow s'} \qquad \frac{\langle b,s\rangle \Rightarrow false \qquad \langle i_2,s\rangle \Downarrow s'}{\langle \texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, s\rangle \Downarrow s'}$$

- Bucle *while*:

$$\frac{\langle b,s\rangle \Rightarrow false}{\langle \texttt{while } b \texttt{ do } i, s\rangle \Downarrow s} \qquad \frac{\langle b,s\rangle \Rightarrow true \quad \langle i,s\rangle \Downarrow s' \quad \langle \texttt{while } b \texttt{ do } i, s'\rangle \Downarrow s''}{\langle \texttt{while } b \texttt{ do } i, s\rangle \Downarrow s''}$$

El lenguaje SIMP

Semántica de paso grande

Ejercicio

Definir la semántica del bucle *while* con una única regla utilizando la instrucción condicional.

Semántica de un programa

Definimos la semántica $\mathcal{S}(P)$ de un programa (terminante) P mediante las descripciones operacionales *small-step* y *big-step*:

- $\mathcal{S}^{small}(P)$ es la traza finita (única)

  $$\langle P, s_I \rangle = \langle P_1, s_1 \rangle \rightarrow \langle P_2, s_2 \rangle \rightarrow \cdots \rightarrow \langle P_n, s_n \rangle = \langle skip, s_F \rangle$$

  obtenida a partir del sistema de transición small-step.
- $\mathcal{S}^{big}(P)$ es el estado final s_F obtenido al utilizar el sistema de transición big-step para calcular $\langle P, s_I \rangle \Downarrow s_F$.

Ambas están relacionadas (mismo s_F). Pero $\mathcal{S}^{big}$ tiene un nivel de abstracción mayor que $\mathcal{S}^{small}$ ($\mathcal{S}^{big}$ no guarda los detalles del cómputo de s_F)

Ejercicio

Calcular la semántica de: P= (x:=4; while x>3 do x:=x-1)

Semántica Axiomática

Una terna de Hoare (Hoare triple) $\{P\}\ S\ \{Q\}$ representa la corrección de un programa S respecto a

- una precondición P (que restringe los estados de entrada a S) y
- una postcondición Q (que representa los estados de salida deseados)

Corrección de un programa

Siempre que un estado s satisface P, el estado final s' resultante de ejecutar S satisfará Q

Ejemplos

$$\begin{array}{ccc} \{y = 4\} & x{:=}y & \{x = 4\} \\ \{y \leq x\} & z{:=}x;\ z{:=}z{+}1 & \{y < z\} \end{array}$$

Semántica Axiomática

Dijkstra ideó un transformador de predicados que asocia a cada tipo de instrucción *i* y postcondición Q una precondición más débil $pmd(i, Q)$

Dicha *precondición más debil* ha de cumplir el estado anterior a la ejecución de *i* para que, después de dicha ejecución, se garantice Q

La corrección de una instrucción S simple o compuesta (programa) respecto a P y Q, es decir $\{P\}\ S\ \{Q\}$, se comprueba como sigue:

1. Calcular $P' = pmd(S, Q)$.
2. Comprobar que $P \Rightarrow P'$.

Semántica Axiomática

El transformador de predicados *pmd*

- Asignación:

$$pmd(x{:=}a, Q) = Q[x \mapsto a]$$

aquí $x \mapsto a$ es una sustitución que reemplaza una variable x en una expresión por otra expresión a. Así, $Q[x \mapsto a]$ es el resultado de aplicar esa sustitución a la expresión lógica Q.

- Condicional:

$$pmd(\texttt{if } b \texttt{ then } i_1 \texttt{ else } i_2, Q) \;=\; (b \wedge pmd(i_1, Q)) \;\vee\; (\neg b \wedge pmd(i_2, Q))$$

- Secuencia:

$$pmd(i_1; i_2, Q) = pmd(i_1, pmd(i_2, Q))$$

Semántica Axiomática

Ejemplo de cálculo con *pmd*

$$
\begin{array}{rcl}
\{P\} & = & \{x = 0 \wedge y = 1 \wedge z = 2\} \\
 & & \{P_1\} \\
 & & \quad t := x \\
 & & \{P_2\} \\
 & & \quad x := y \\
 & & \{P_3\} \\
 & & \quad y := t \\
\{Q\} & = & \{x = 1 \wedge y = 0\}
\end{array}
$$

1. Cálculo (de abajo a arriba) de P' (aquí igual a P_1):
   - $P_3 = pmd(y{:=}t, Q) = Q[y \mapsto t] = (x = 1 \wedge t = 0)$.
   - $P_2 = pmd(x{:=}y, P_3) = P_3[x \mapsto y] = (y = 1 \wedge t = 0)$.
   - $P_1 = pmd(t{:=}x, P_2) = P_2[t \mapsto x] = (y = 1 \wedge x = 0)$.
2. Como $P_1 = pmd(S, Q)$, comprobamos $P \Rightarrow P_1$, i.e.,

$$(x = 0 \wedge y = 1 \wedge z = 2) \Rightarrow (y = 1 \wedge x = 0)$$

que es claramente cierto.

Semántica Axiomática

Ejemplo

Dada la siguiente terna de Hoare:

$$\begin{aligned} \{P\} &= \{x = 1\} \\ & \quad x := x - 1 \\ \{Q\} &= \{x \geq 0\} \end{aligned}$$

¿podemos decir que el programa es correcto?

Solución: Dado que

$$pmd(x := x - 1, x \geq 0) = (x - 1 \geq 0) \Leftrightarrow x \geq 1$$

y que

$$x = 1 \Rightarrow x \geq 1,$$

concluimos que el programa es correcto respecto a P y Q.

Propiedades semánticas

Equivalencia de programas

La semántica de un lenguaje nos permite razonar sobre la equivalencia de programas

Equivalencia semántica

Dos programas P y P' son equivalentes respecto a una descripción semántica $\mathcal{S}$ (e.g., $\mathcal{S}^{big}$ o $\mathcal{S}^{small}$) si y solo si

$$\mathcal{S}(P) = \mathcal{S}(P')$$

Denotamos esto escribiendo $P \equiv_{\mathcal{S}} P'$.

Por ejemplo, para los programas

$$P: \quad x{:=}1; \; x{:=}2; \qquad\qquad P': \quad x{:=}2;$$

tenemos $P \equiv_{\mathcal{S}^{big}} P'$, pero $P \not\equiv_{\mathcal{S}^{small}} P'$ (¿Por qué?).

Propiedades semánticas

Ejemplo

Asumiendo que el lenguaje SIMP se ha enriquecido con el producto y la división, tanto a nivel sintáctico como semántico (ejercicio), consideremos los programas

P : $sum{:=}(n * (n+1))/2;$

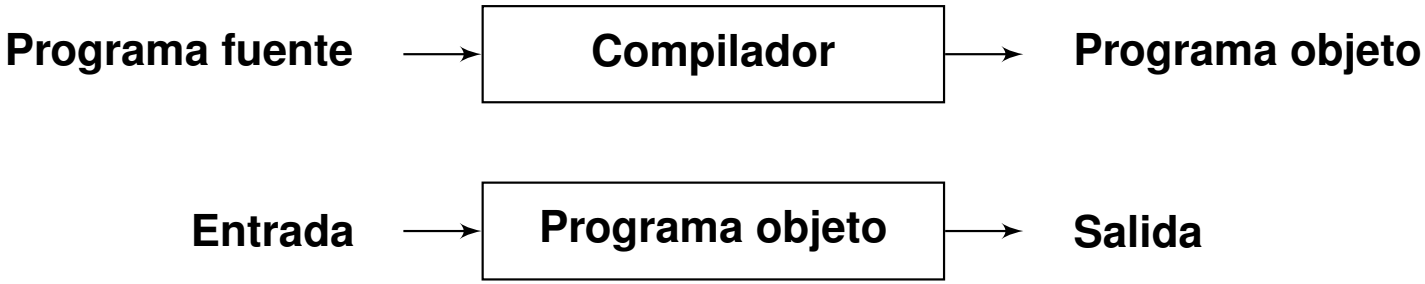
P' :
$sum := 0;$
$i{:=}1;$
`while` $i \leq n$ `do`
$sum{:=}sum{+}1;$
$i{:=}i{+}1;$

para calcular $1 + 2 + \cdots + n$ para un entero positivo n dado.

- Desde el punto de vista del número de pasos de cómputo, ¿cuál de los dos es más eficiente?
- ¿Podemos capturar esto con $\mathcal{S}^{small}$ o $\mathcal{S}^{big}$?
- ¿Son equivalentes respecto a $\mathcal{S}^{small}$ o $\mathcal{S}^{big}$ (o ambas)?¿Por qué?

Implementación de los lenguajes de programación

- Lenguajes compilados

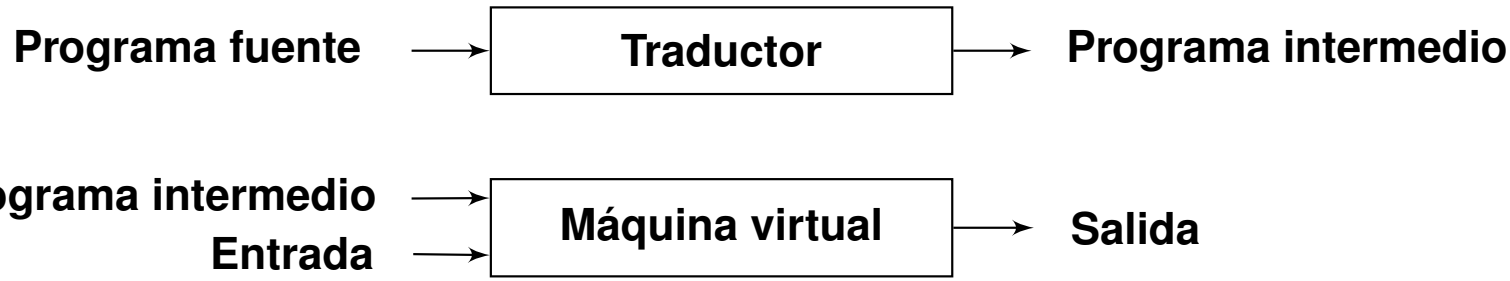


- Lenguajes interpretados

Los buenos entornos incluyen tanto intérprete (para ser usado en la fase de desarrollo) como compilador (para usarse en explotación).

Traducción vs interpretación (I)

- La traducción e interpretación pura constituyen dos extremos
- En la práctica no se suele usar la traducción pura excepto cuando los lenguajes son de nivel muy próximo (como en el caso de los ensambladores)
- La interpretación pura tampoco es muy frecuente, excepto en lenguajes de control de S.O. (scripting) o en lenguajes interactivos
- Es más común una implementación mixta: el programa se traduce primero de la forma original a otra de ejecución más fácil, que se ejecuta por interpretación.



Traducción vs interpretación (II)

- Lenguajes típicamente compilados:

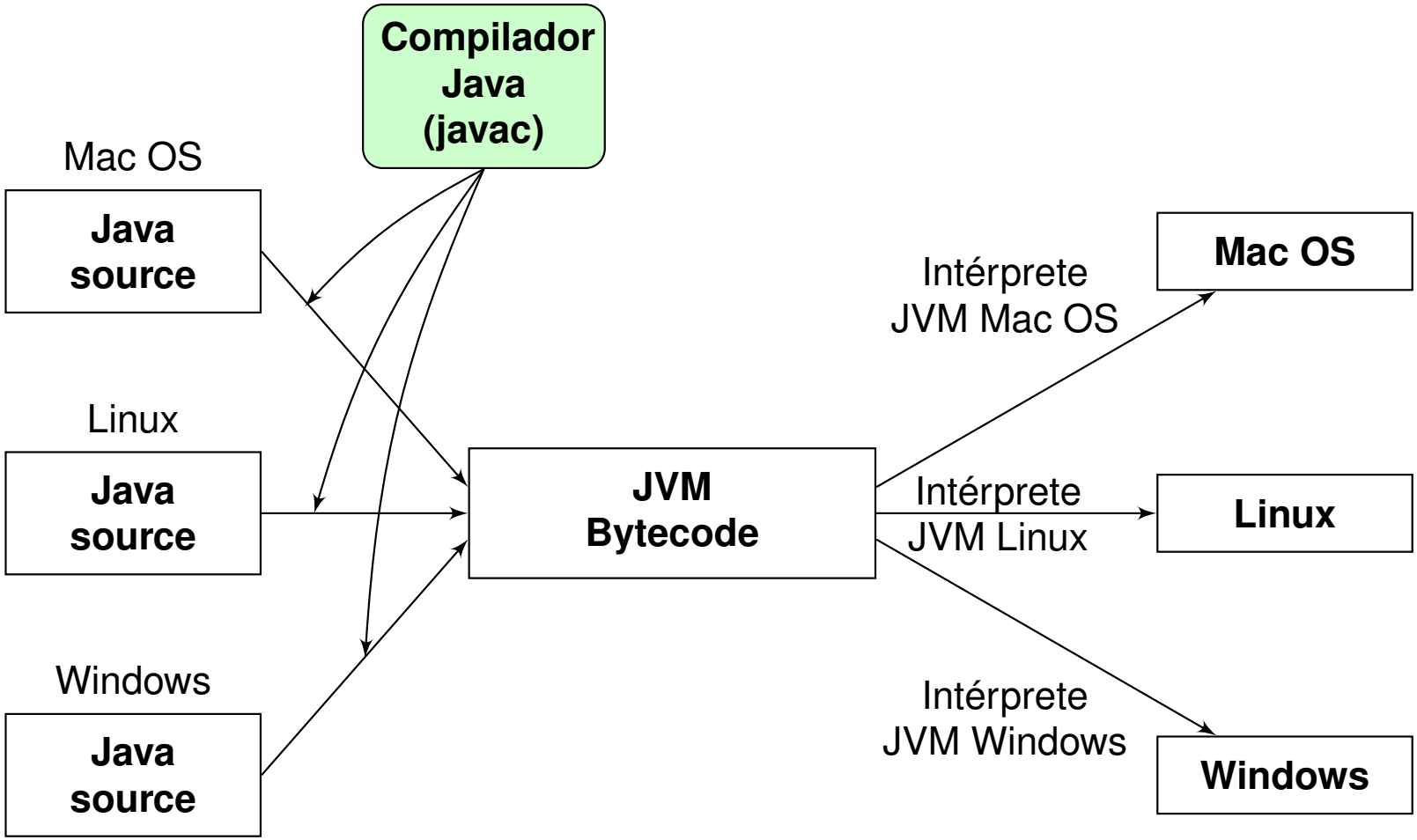
C, C++, Fortran, Ada

- Lenguajes típicamente interpretados:

LISP, ML, Smalltalk, Perl, Postscript

- Lenguajes con implementación mixta (esto facilita la portabilidad a cualquier plataforma):
  - Pascal (P-code),
  - Prolog (WAM-code),
  - Java (byte-code, el código de la JVM, i.e., el formato estándar para la distribución de código Java)

Java Virtual Machine (JVM)



Bibliografía

Básica:

- Winskel, G. The formal Semantics of Programming Languages. An introduction. MIT Press, 1993.
- Pratt, T.W. and Zelkowitz, M.V. Lenguajes de programación: diseño e implementación, Prentice-Hall, 1998.
- Scott, M.L. Programming Language Pragmatics, Morgan Kaufmann Publishers, 2003.

Complementaria

- Stuart, T. Understanding Computation (Capítulo 2). Ed. O'Reilly, 2013.
- Kenneth Slonneger, Barry L. Kurtz. Formal Syntax and Semantics of Programming Languages. A Laboratory Based Approach (Capítulos 1 y 11). Addison-Wesley, 1995.