Tema 2. Fonaments dels Llenguatges de Programació

Llenguatges, Tecnologies i Paradigmes de Programació (LTP)

DSIC, ETSInf

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

etsinf
Escuela Técnica Superior de Ingeniería Informática

Descripció formal d'un LP

- **Sintaxi**: quina seqüència de caràcters constitueixen un programa "legal"
  - elements sintàctics del llenguatge
- **Semàntica**: què significa (què calcula) un programa legal donat. Importància:
  1. Ajuda al programador a "raonar" sobre el programa
  2. És necessària per a implementar correctament el llenguatge (models d'execució)
  3. Permet desenvolupar tècniques i eines de:
     - Anàlisi i Optimització
     - Depuració
     - Verificació
     - Transformació

Sintaxi

Us de gramàtiques BNF

Notació BNF:

- Amb <w> es nomena un grup d'expressions definit per alguna regla de construcció de expressions
- el símbol | 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>
```

- els claudàtors [i] se situen al voltant dels items opcionals
- les claus {} (o l'asterisc *) serveixen per a indicar una seqüencia de 0 o més items
- el símbol + serveix per a indicar una seqüencia d'1 o més items

$$\langle realNumber \rangle ::= [+|-] \langle digit \rangle^{+} . [E|e] [+|-] \{\langle digit \rangle\}$$

Sintaxi

Us de gramàtiques BNF

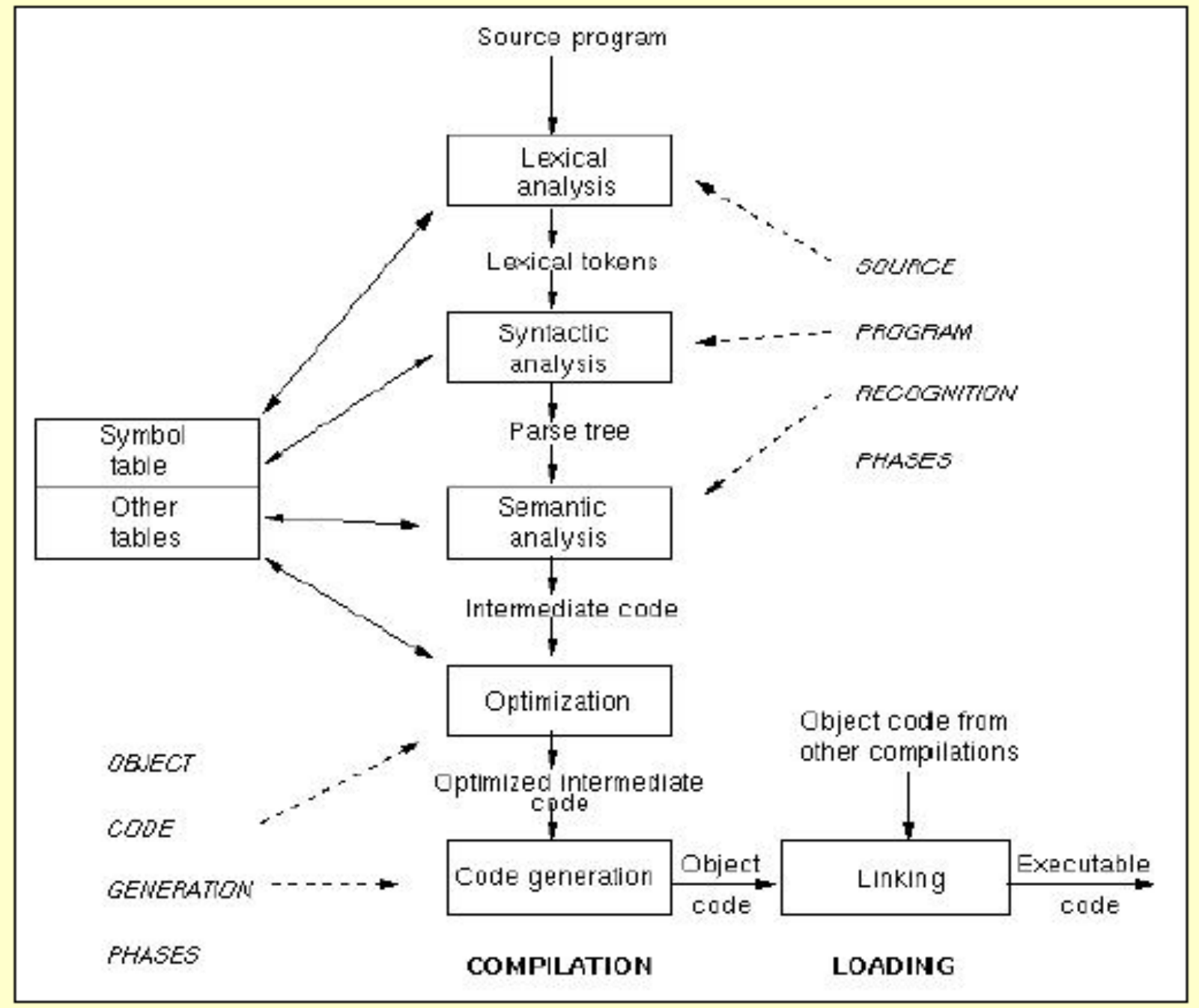
Exemple: sintaxi del bucle while

Java

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

Modula-2

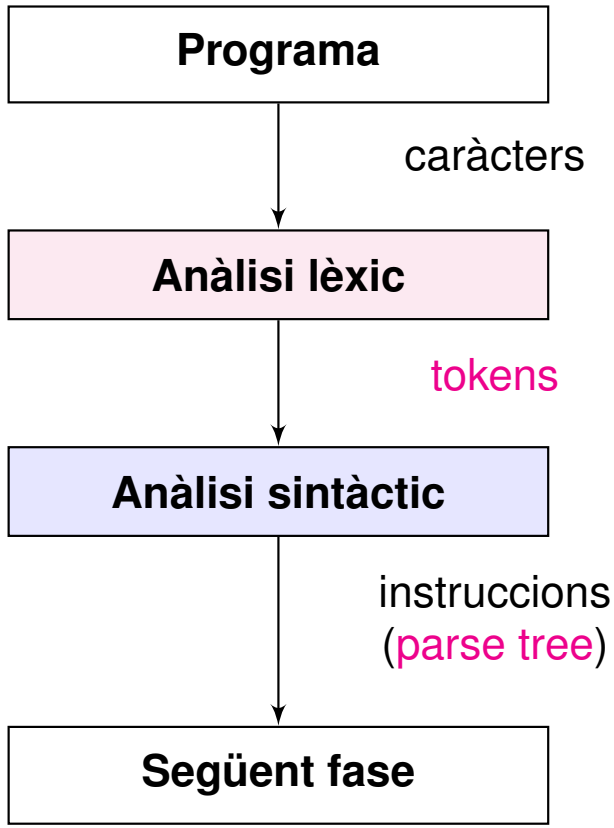
< while_statement> ::= `WHILE` <expression> `DO`
<statement> { `;` <statement> }
`END`



Processament d'un programa font

Anàlisi lèxic y sintàctic

- El analitzador lèxic (scanner) divideix una seqüència de caràcters (el programa) en una seqüència de components sintàctics primitius o paraules (tokens) que actuen com a identificadors, nombres, paraules reservades, etc.
- El analitzador sintàctic (parser) reconeix una seqüència de tokens i obté una seqüència de instruccions en forma de arbre sintàctic estructura



Example

1. Seqüència de caràcters

```
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
```

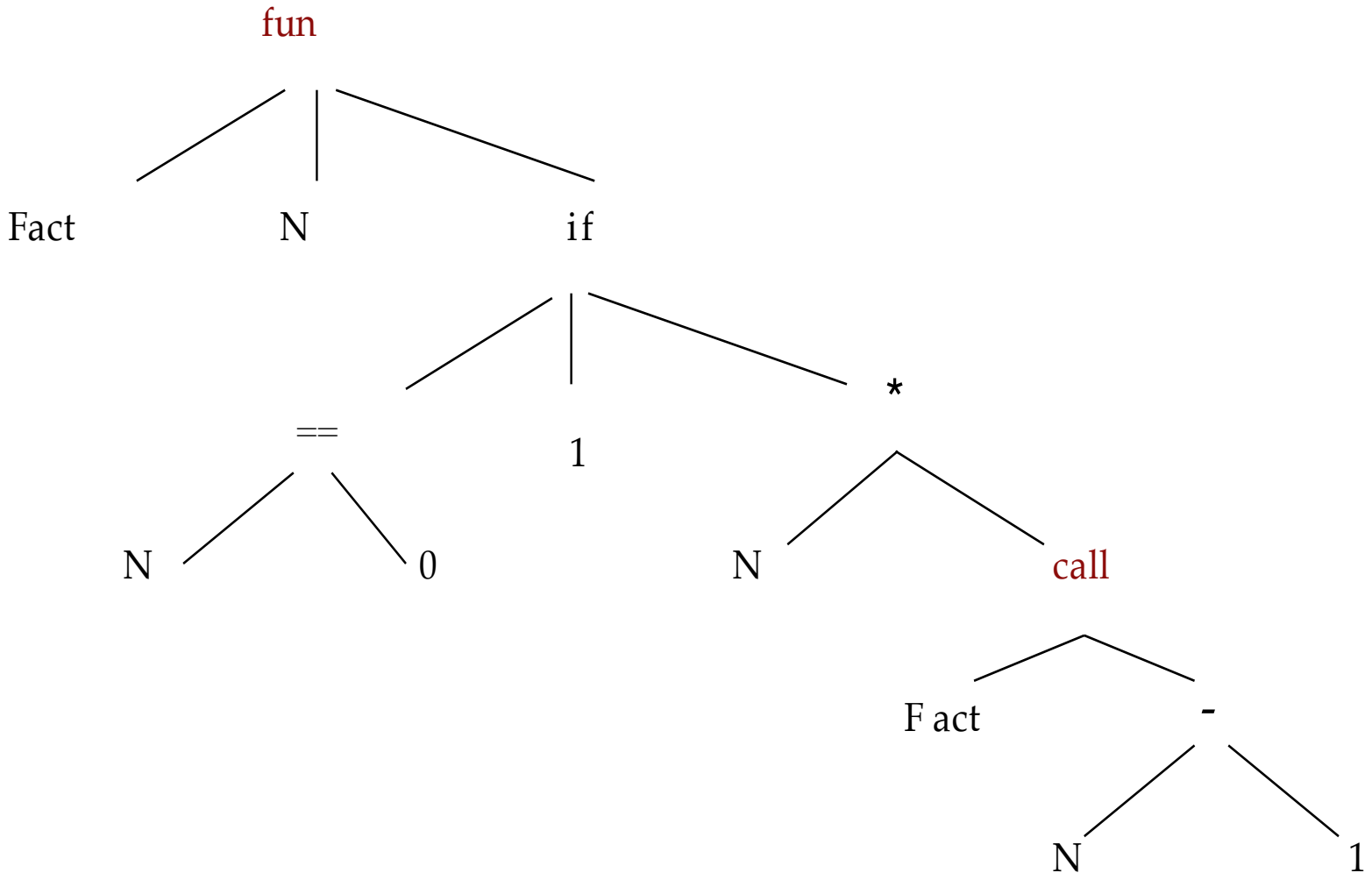
2. Seqüència de paraules (tokens)

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

3. Instrucció

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

Example



Arbre sintàctic (parse tree)

Anàlisi semàntic

descripció Semàntica: concepto i necessitat

Semàntica estática: restriccions de la sintaxi que no poden expressar-se mitjançant la notació BNF però que sí poden comprovar-se en temps de compilació

Exemple

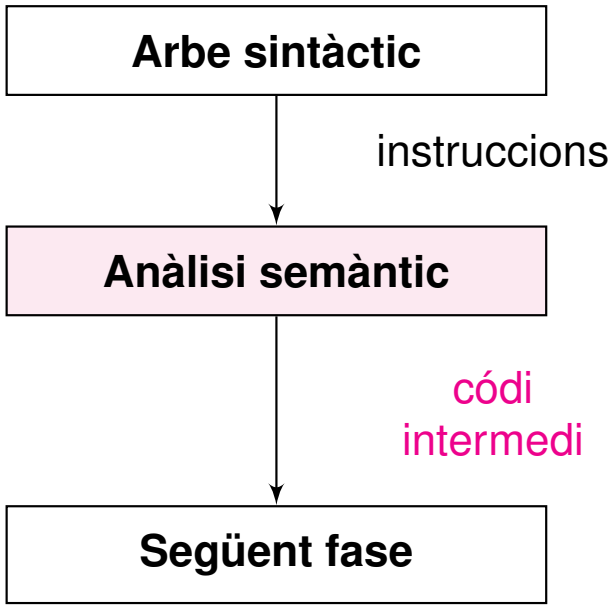
A := B + C podria no ser legal si A, B o C no han sigut declarades prèviament

Semàntica dinàmica: restriccions que només es poden comprovar durant l'execució del programa (e.g. comprovació d'índexs dins del rang del vector)

Anàlisi semàntic

Semàntica estática

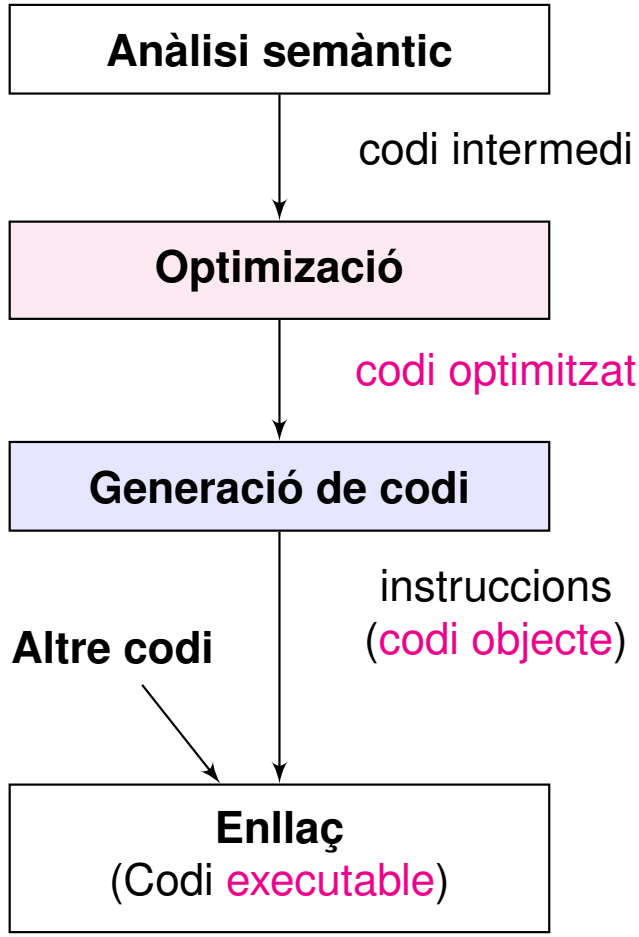
- Comprovacions en el analitzador semàntic:
  1. Declaració de variables prèvia al seu ús
  2. Compatibilitat i conversió de tipus (coerció)
  3. Signatura de les funcions: els paràmetres reals coincideixen en nombre i tipus amb els formals
  4. ...
- Produeix un còdi intermedi que és la base per al procés de compilació posterior



Compilació i Enllaç

Generació de codi executable

- Primer es optimitza el codi intermedi rebut de la fase anterior
- La fase de generació de codi produeix el codi objecte del programa
- El codi objecte es enllaça amb codi procedent d'altres programes o llibreries per a obtenir el codi executable.



Example

Evolució de la representació interna d'un programa durant les diferents etapes del procés de compilació.

Considerarem el següent programa:[1]

```
posicio = inicial + velocitat * 60
```

on les variables posicio, inicial i velocitat són de tipus real.

[1] En les pàgines 12 i 13 de *Aho, Sethi and Ullman. Compiladores: Principios, técnicas y herramientas. Addisón-Wesley Iberoamericana, 1990.*

Semàntica dinàmica

Per què la semàntica no és sempre "estàtica"?

El compilador no pot detectar tots els errors possibles:

1. Alguns errors només es manifesten durant l'execució:
   - Z=X/Y produeix un error si s'executa amb Y = 0
   - Z=V[Y] produeix un error si Y té un valor que cau fora del rang del vector V
2. Moltes propietats interessants d'un programa no són decidibles.
   - la acabació (però és 'semidecidible': hi ha prou en executar el programa per a "semi-decidir-ho")
   - si dos programes qualssevol computen la mateixa funció
   - si dues descripcions BNF generen el mateix llenguatge

Semàntica dinàmica

Estils de definició Semàntica

- **Operacional**
- **Axiomàtica**
- Declarativa:
  - Denotacional
  - Algebraica
  - Teoria de models
  - Punt fix

Semàntica Operacional

Consisteix a definir una màquina (abstracta) M i expressar el significat de cada construcció del llenguatge en termes de les accions a realitzar per la màquina per a executar aquesta instrucció.

Semàntica Operacional

- Representem l'estat de la màquina (abstracta) que executa el programa com una funció $s : \mathcal{X} \to D$ que assigna valors en un domini D a les variables $x, y, \ldots \in \mathcal{X}$ del programa.

Notació

ja que en un programa utilitzem un conjunt finit de variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, podem representar l'estat com un conjunt de parells variable-valor:

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

- La configuració de la màquina és un parell

$$\langle i, s \rangle$$

que registra el estat actual (s) al costat de la instrucció a avaluar (i), bé siga simple o composta (un programa es considera una instrucció composta).

Semàntica Operacional

- Per a formalitzar la ejecució del programa en la màquina utilitzem una relació de transició '$\rightarrow$' entre configuracions.
- La relació es defineix mitjançant regles de transició:

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

que descriuen la configuració $\langle i', s'\rangle$ obtinguda a partir la configuració de partida $\langle i, s\rangle$ quan es satisfà la premissa o condició sobre la configuració $\langle i, s\rangle$.

- També utilitzem altres relacions per a descriure
  - la avaluació de expressions aritmètiques ($\langle exp, s\rangle \Rightarrow n$).
  - la obtenció directa d'un estat final ($\langle i, s\rangle \Downarrow s'$).

  i les definim mitjançant regles similars a (1).

El llenguatge SIMP

Sintaxi

La gramàtica en estil BNF del minillenguatge imperatiu SIMP que utilitzarem en aquest tema es defineix així:

- Expressions aritmètiques:

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

on C i V denoten les constants numèriques $(0, 1, 2, \ldots)$ i les variables $(x, y, \ldots)$ respectivament

- Expressions booleanes:

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

- Instruccions:

$$i ::= \mathit{skip} \mid V := a_1 . \mid i ; i_1 \mid \texttt{if}\ b\ \texttt{then}\ i_1\ \texttt{else}\ i_2 \mid \texttt{while}\ b\ \texttt{do}\ i$$

on *skip* denota la instrucció buida.

El llenguatge SIMP

Avaluació d'expressions

- Escrivim $\langle exp, s \rangle \Rightarrow n$ per a indicar que l'expressió *exp* s'avalua a *n* en l'estat *s*.
- Usem aquesta relació de avaluació per a avaluar les expressions aritmètiques i booleanes.

El llenguatge SIMP

Avaluació de expressions aritméticas

- Constants numèriques:

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

- Variables:

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

Recordem que l'estat s és una funció de variables en valors. $s(x)$ no és més que el valor de la variable x en l'estat de la màquina s.

- Addició:

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

  si n és la suma d'i n_1 n_2.
- Resta i producte: similar.

El llenguatge SIMP

Avaluació de expressions booleanas

- Valors booleans:

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

- Igualtat:

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 = a_2, s\rangle \Rightarrow \mathit{true}} \quad \text{si } n_1 \text{ i } n_2 \text{ són iguals}$$

$$\frac{\langle a_1,s\rangle \Rightarrow n_1 \qquad \langle a_2,s\rangle \Rightarrow n_2}{\langle a_1 = a_2, s\rangle \Rightarrow \mathit{false}} \quad \text{si } n_1 \text{ i } n_2 \text{ són diferents}$$

- Menor o igual:

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

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

- Negació: $\dfrac{\langle b,s\rangle \Rightarrow \mathit{true}}{\langle \neg b,s\rangle \Rightarrow \mathit{false}}$ $\dfrac{\langle b,s\rangle \Rightarrow \mathit{false}}{\langle \neg b,s\rangle \Rightarrow \mathit{true}}$
- Disjunció: EXERCICI

Semàntica Operacional

Pas xicotet (*small-step*)

- En la descripció semàntica operacional de Pas xicotet l'execució d'un programa es pot seguir instrucció a instrucció.
- En executar un programa P a partir del estat inicial s_I (on cap variable està assignada a cap valor, és a dir: $s_I = \{\}$), s'obté una seqüència de configuracions (denominada traça):

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

Distingim dues situacions:

1. P_n és la instrucció buida (*skip*) per a algun $n \geq 1$. Llavors l'execució del programa acaba amb un estat final $s_F = s_n$.
2. P_n mai arriba a ser la instrucció buida per a cap n: l'execució del programa no acaba.

El llenguatge SIMP

Semàntica de Pas xicotet (I)

- Seqüència:

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

- Assignació:

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

on el nou estat $s[x \mapsto n]$ es defineix eliminant del s possible vincle que existisca per a x i afegint en qualsevol cas el nou vincle $x \mapsto n$:

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

El llenguatge SIMP

Semàntica de Pas xicotet (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*:

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

Exercici

Definir la semàntica del bucle *while* amb una única regla utilitzant la instrucció condicional.

Semàntica Operacional

Pas gran (*big-step*)

- En la descripció semàntica operacional de Pas gran (big-step) s'especifica l'execució d'un programa P com una transició directa des de la configuració inicial $\langle P, s_I \rangle$ al estat final s_F.
- A diferència de la semàntica de pas xicotet, doncs, la relació de transició de pas gran $\Downarrow$ relaciona configuracions amb estats: $\langle P, s \rangle \Downarrow s'$

El llenguatge SIMP

Semàntica de Pas gran

- Instrucció buida:

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

- Seqüència:

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

- Assignació:

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

- 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 llenguatge SIMP

Semàntica de Pas gran

Exercici

Definir la semàntica del bucle *while* amb una única regla utilitzant la instrucció condicional.

Semàntica d'unprograma

Definim la semàntica $\mathcal{S}(P)$ d'un programa ("terminant") P mitjançant les descripcions operacionals *small-step* i *big-step*:

- $\mathcal{S}^{small}(P)$ és la traça finita (única)

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

  obtinguda a partir del sistema de transició small-step.
- $\mathcal{S}^{big}(P)$ és l'estat final s_F obtingut en utilitzar el sistema de transició big-step per a calcular $\langle P, s_I \rangle \Downarrow s_F$.

Ambdues estan relacionades (mateix s_F). Però $\mathcal{S}^{big}$ té un nivell d'abstracció major que $\mathcal{S}^{small}$ ($\mathcal{S}^{big}$ no guarda els detalls del còmput de s_F)

Exercici

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

Semàntica Axiomàtica

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

- una precondició P (que restringeix els estats de entrada a S) i
- una postcondició Q (que representa els estats de eixida desitjats)

Corrección d'un programa

Sempre que un estat s satisfà P, l'estat final s' resultant d'executar S satisfarà Q

Exemples

$$\{y = 4\} \quad x{:=}y \quad \{x = 4\} \tag{2}$$

$$\{y \leq x\} \quad z{:=}x;\ z{:=}z{+}1 \quad \{y < z\} \tag{3}$$

Semàntica Axiomàtica

Dijkstra va idear un transformador de predicats que associa a cada tipus d'instrucció *i* i postcondició Q una precondició més feble $pmd(i, Q)$

Aquesta *precondició més debil* ha de complir l'estat anterior a l'execució de *i* perquè, després d'aquesta execució, es garantisca Q

La correcció d'una instrucció S simple o composta (programa) respecte a P i Q, és a dir $\{P\}\ S\ \{Q\}$, es comprova com segueix:

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

Semàntica Axiomàtica

El transformador de predicats *pmd*

- Assignació:

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

ací $x \mapsto a$ és una sustitució que reemplaça una variable x en una expressió per una altra expressió a. Així, $Q[x \mapsto a]$ és el resultat d'aplicar aqueixa substitució a l'expressió lògica Q.

- Condicional:

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

- Seqüència:

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

Semàntica Axiomàtica

Example de cálculo amb *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. Càlcul (de baix dalt) de P' (ací 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. Com $P_1 = pmd(S, Q)$, comprovem $P \Rightarrow P_1$, i.e.,

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

que és clarament cert.

Semàntica Axiomàtica

Example

Donada la següent terna de Hoare:

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

podem dir que el programa és correcte?

Solució: Atès que

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

i que

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

concloem que el programa és correcte respecte a P i Q .

Propiedades Semànticas

Equivalencia de programas

La semàntica d'un llenguatge ens permet raonar sobre l'equivalència de programes

Equivalència semàntica

Dos programes P i P' són equivalents respecte a una descripció Semàntica $\mathcal{S}$ (e.g., $\mathcal{S}^{big}$ o $\mathcal{S}^{small}$) si i solament si

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

Denotem açò escrivint $P \equiv_{\mathcal{S}} P'$.

Per exemple, per als programes

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

tenim $P \equiv_{\mathcal{S}^{big}} P'$, però $P \not\equiv_{\mathcal{S}^{small}} P'$ (Per què?).

Propiedades Semànticas

Example

Assumint que el llenguatge SIMP s'ha enriquit amb el producte i la divisió, tant a nivelll sintàctic com a semàntic (EXERCICI), considerem els programes

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

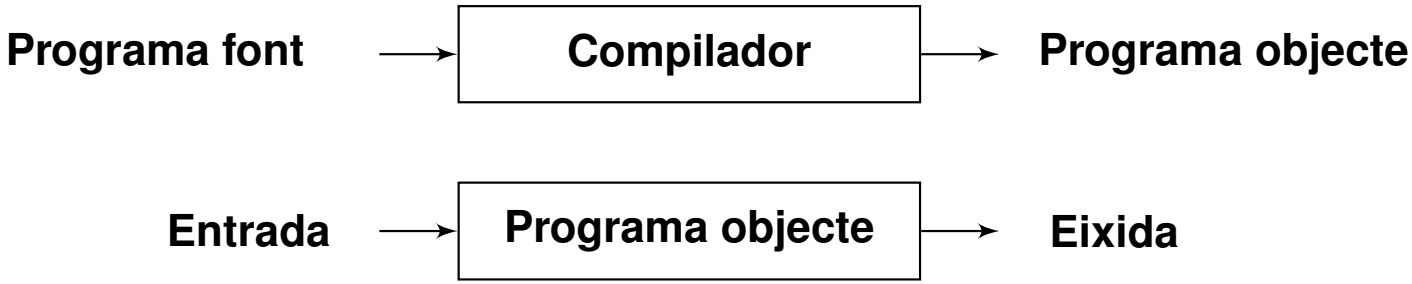
P' :
$sum := 0;$
$i{:=}1;$
`while` $i \leq n$ `do`
$sum{:=}sum{+}1;$
$i{:=}i{+}1;$

per a calcular $1 + 2 + \cdots + n$ per a un enter positiu n donat.

- Des del punt de vista del nombre de passos de còmput, quin dels dos és més eficient?
- Podem capturar açò amb $\mathcal{S}^{small}$ o $\mathcal{S}^{big}$?
- Só equivalents respecte a $\mathcal{S}^{small}$ o $\mathcal{S}^{big}$ (o ambdues) Per què?

Implementació dels llenguatges de programació

- Llenguatges compilats

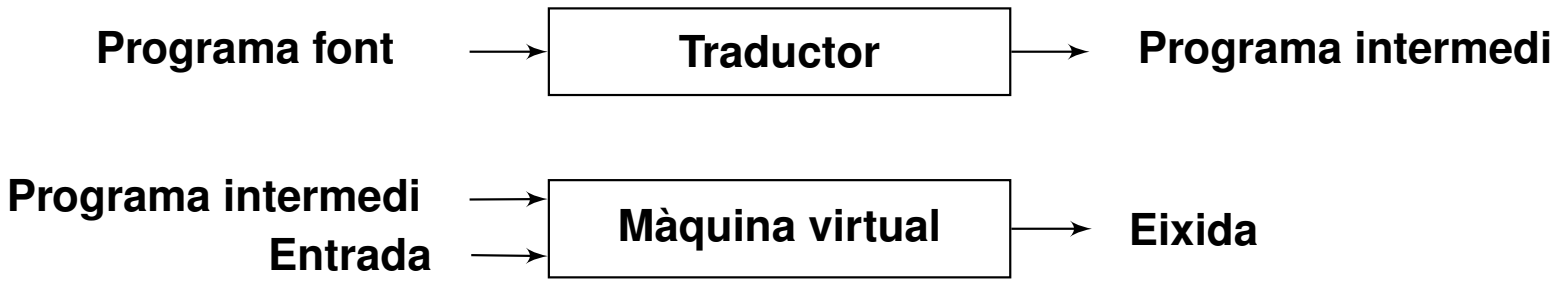


- Llenguatges interpretats

Els bons entorns inclouen tant intèrpret (per a ser usat en la fase de desenvolupament) com a compilador (per a usar-se en explotació).

Traducció vs interpretació (I)

- La traducció i interpretació pura constitueixen dos extrems
- En la pràctica no se sol usar la traducció pura excepte quan els leguajes són de nivelll molt pròxim (com en el cas dels assembladors)
- La interpretació pura tampoc és molt freqüent, excepte en llenguatges de control de S.O. (scripting) o en llenguatges interactius
- És més comú una implementació mixta: el programa es tradueix primer de la forma original a una altra d'execució més fàcil, que s'executa per interpretació.



Traducció vs interpretació (II)

- Llenguatges típicament compilats:

C, C++, Fortran, Ada

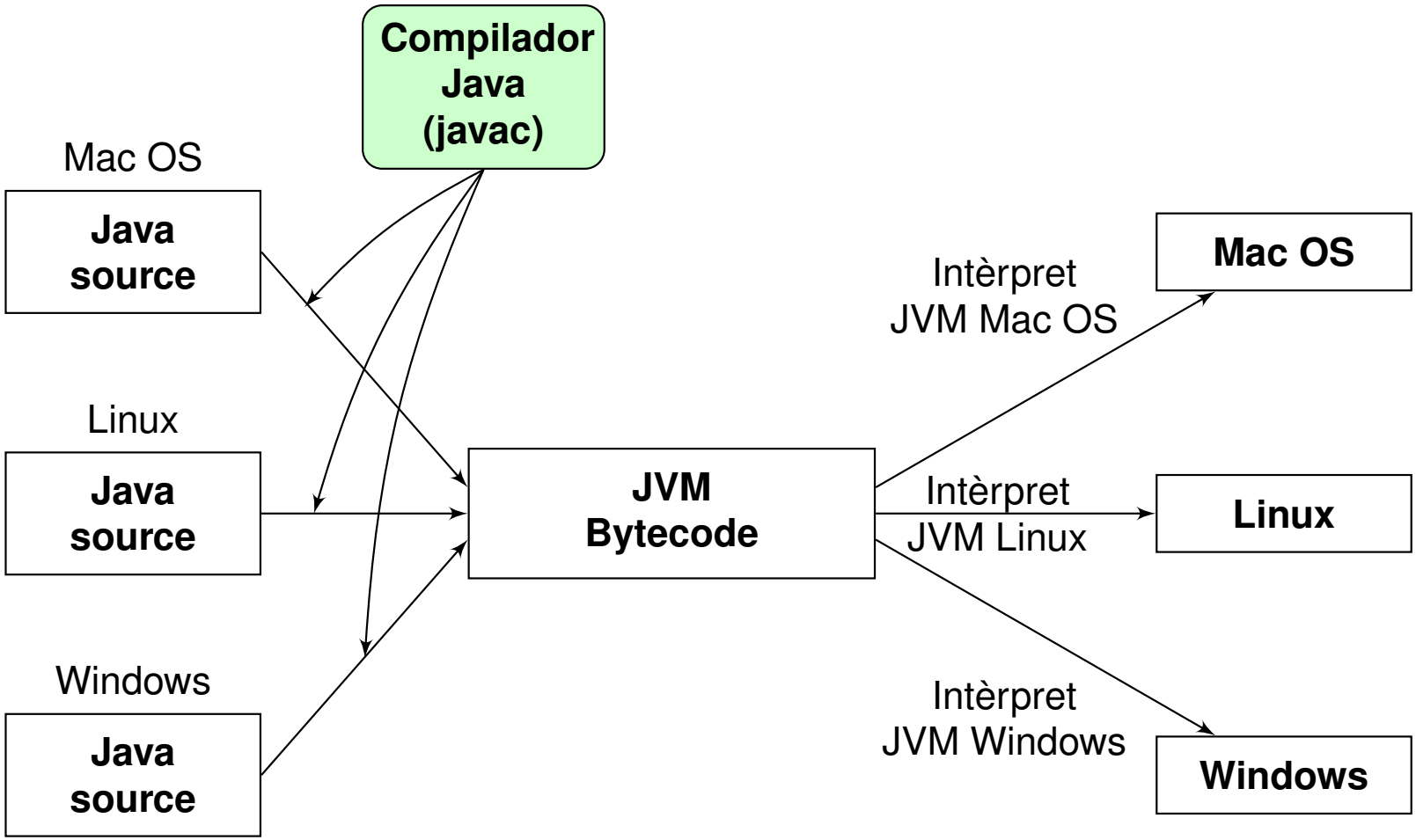
- Llenguatges típicament interpretats:

LISP, ML, Smalltalk, Perl, Postscript

- Llenguatges amb implementació mixta (açò facilita la portabilitat a qualsevol plataforma):

  - Pascal (P-code),
  - Prolog (WAM-code),
  - Java (byte-code, el codi de la JVM, i.e., el format estàndard per a la distribució de codi Java)

Java Virtual Machine (JVM)



Bibliografía

Bàsica:

- 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). Addisón-Wesley, 1995.