Exercicis LTP

Tema 2: Fonaments dels Llenguatges de Programació

Part I: qüestions

1. D'acord a les següents regles BNF:

```
<conditional> ::= IF <cond> THEN <exp> ELSE <exp>;
<cond>        ::= X>0 | X<0
<exp>         ::= X:=X+1 | X:=X-1
```

   Indica si les següents sentències són correctes (pugues haver-hi més d'una correcta):

   - `IF X>0 THEN X:=X+1;`
   - `IF X<0 THEN X:=X+1 ELSE X:=X-1;`
   - `IF X>0 THEN X:=X+1 ELSE X<0;`
   - `IF X>0 THEN X:=X-1 ELSE X:=X+1`

2. D'acord a les següents regles BNF:

```
<arit> ::= <num> + <num> | <num> - <num>
<expr> ::= <var> = <arit> | <arit> = <var> | <expr> ; <expr>
<num>  ::= 1 | 2 | 3 | 4 | 5
<var>  ::= X | Y | Z
```

   Indica si les següents afirmacions són correctes (pugues haver-hi més d'una correcta):

   - `1+1` es una expresió `<arit>`.
   - `1+2-3` es una expresió `<arit>`.
   - `1+2=X` es una expresió `<expr>`.
   - `Z=2+3;Y=1-4` es una expresió `<expr>`.

3. Analitza si el següent programa escrit en C-Minus és vàlid pel que fa a la gramàtica inclosa en l'Apèndix, que descriu la sintaxi d'un xicotet subconjunt de C (anomenat C-Minus) i justifica la teua resposta:

```
long factorial(int n)
{
 int c = 2;
 long result = 1;

 while (c <= n)
```

```
 { result = result * c;
   c++
 }
 return result;
}
```

4. Dóna les regles semàntiques per a l'avaluació de l'operació booleana de disjunció.

5. Donat el següent codi P:

```
X:=5;
Y:=X
```

   (a) Desenvolupa la traça d'execució (mostrant els còmputs intermedis) seguint la semàntica *small-step* amb l'estat inicial $s_I = \{\}$.

   (b) Calcula la semàntica *big-step*, mostrant els còmputs intermedis, amb l'estat inicial buit.

6. Donada la següent configuració, desenvolupa l'avaluació de l'expressió aritmètica aplicant les regles corresponents:

$$\langle \texttt{X} + \texttt{3}, \{X \mapsto 2\}\rangle \Rightarrow \ldots$$

7. Donada la següent configuració, desenvolupa l'avaluació de l'expressió booleana aplicant les regles corresponents:

$$\langle \texttt{X} + \texttt{3} \leq \texttt{Y}, \{X \mapsto 2, Y \mapsto 0\}\rangle \Rightarrow \ldots$$

8. Donat el següent codi P:

```
X:=5;
if X>3 then X:= X-1 else Y:=X
```

   (a) Desenvolupa la traça d'execució (mostrant els còmputs intermedis) seguint la semàntica *small-step* amb l'estat inicial $\{X \mapsto 2\}$.

   (b) Calcula la semàntica *big-step*, mostrant els còmputs intermedis, amb l'estat inicial $\{X \mapsto 2\}$.

9. Volem estendre el llenguatge SIMP vist en les transparències amb una nova instrucció *repeat* la sintaxi de la qual és

```
repeat i until b
```

   El comportament d'aquesta instrucció és el següent: ha d'executar-se la instrucció (possiblement composta) `i` fins que se satisfaça la condició `b`, en el moment de la qual es detindrà l'execució del *repeat*.

(a) Dóna la(s) regla(s) semàntica(s) seguint l'estil *small-step* parell la instrucció *repeat*

(b) Dóna la(s) regla(s) semàntica(s) seguint l'estil *big-step* parell la instrucció *repeat*

10. Donat el següent codi P:

```
X:=4;
while X>3 do X:= X-1
```

(a) Desenvolupa la traça d'execució (mostrant els còmputs intermedis) seguint la semàntica *small-step* amb l'estat inicial $s = \{\}$.

(b) Calcula la semàntica *big-step*, mostrant els còmputs intermedis, amb l'estat inicial $s = \{\}$.

11. Volem estendre el llenguatge SIMP vist en les transparències amb una nova instrucció *for* la sintaxi de la qual és

```
for V:=a0 to a1 do i
```

El comportament d'aquesta instrucció és el següent: la variable `V` (el "comptador") va prenent valors, en ordre creixent, des de `a0` fins a `a1` (tots dos inclusivament) i, després de cada assignació, s'executa la instrucció `i`.

(a) Dóna la(s) regla(s) semàntica(s) seguint l'estil *small-step* parell la instrucció *for*

(b) Dóna la(s) regla(s) semàntica(s) seguint l'estil *big-step* parell la instrucció *for*

12. Volem estendre el llenguatge SIMP vist en les transparències amb una nova instrucció *times* la sintaxi de la qual és

```
do n times i
```

El comportament d'aquesta instrucció és el següent: s'executa la instrucció `i` tantes vegades com indique el nombre natural `n` (si `n=0` la instrucció `i` no s'executa).

(a) Dóna la(s) regla(s) semàntica(s) seguint l'estil *small-step* parell la instrucció *times*

(b) Dóna la(s) regla(s) semàntica(s) seguint l'estil *big-step* parell la instrucció *times*

13. Donat el següent codi S on es calcula el màxim de dos nombres:

```
if X>Y then max:=X else max:=Y
```

(a) Desenvolupa la traça d'execució (mostrant els còmputs intermedis) seguint la semàntica *small-step* amb l'estat inicial $\{X \mapsto 3, Y \mapsto 5\}$.

(b) Calcula la semàntica *big-step*, mostrant els còmputs intermedis, amb l'estat inicial $\{X \mapsto 3, Y \mapsto 5\}$.

14. Volem estendre el llenguatge SIMP vist en les transparències amb un nou operador d'assignació múltiple la sintaxi de la qual és

$\mathtt{x}_1\mathtt{,x}_2\mathtt{,...x}_n$:= $\mathtt{a}_1\mathtt{,a}_2\mathtt{,...a}_n$

on

- les $\mathtt{x}_i$ són diferents entre si,
- les $\mathtt{a}_i$ són expressions

i el seu comportament ha de ser el següent (veure [Gries81], pàgina 121):

- primer s'avaluen les expressions $\mathtt{a}_1\mathtt{,a}_2\mathtt{,...a}_n$ (en qualsevol ordre), obtenint-se els valors $\mathtt{v}_1\mathtt{,v}_2\mathtt{,...v}_n$ respectivament;
- per a cada i, s'assigna a $\mathtt{x}_i$ el valor $\mathtt{v}_i$.

(a) Dóna una o més regles semàntiques seguint l'estil *small-step* per a la instrucció *times*

(b) Dóna una o més regles semàntiques seguint l'estil *big-step* per a la instrucció *times*

15. Donat el següent programa S:

```
t:=x;
x:=i;
i:=t;
```

Desenvolupa la traça de l'execució a partir de l'estat $\{x \mapsto 2, y \mapsto 5\}$ usant la semàntica operacional de pas xicotet.

16. Donat el següent fragment de codi P:

```
x:=x+1;
y:=y+x;
x:=x+1;
```

usant la semàntica operacional de pas xicotet, construeix la traça d'execució a partir de l'estat inicial $\{x \mapsto 3, y \mapsto 7\}$.

17. Considera el següent codi en C que retorna el màxim de dos nombres:

```
int maximo (int x, int y)
{
if (x>y)
return x ;
else
return y ;
} ;
```

(a) Escriu el cos de la funció `maximo` en la sintaxi del llenguatge SIMP (en SIMP no hi ha subprogrames).

(b) Construeix les traces de l'execució de la trucada `maximo(3,5)` (és a dir, partint de l'estat inicial $\{x \mapsto 3, y \mapsto 5\}$), usant les semàntiques de pas xicotet i gran.

18. Calcula la precondició més feble $wp(S, Q)$ per al programa S de la pregunta **??** i la postcondició Q donada per $\texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y}$. D'acord als resultats obtinguts:

    (a) Què fa aquest programa? Quina és la diferència (si la hi ha) entre el programa S i el següent programa S' que usa l'assignació múltiple introduïda en la pregunta **??**?

    ```
    x, y := y, x
    ```

    (b) Considerant la semàntica axiomàtica, hi ha alguna diferència entre S i S' pel que fa a la postcondició Q?

    (c) Són equivalents els programes S i S' des del punt de vista operacional o es podrien distingir usant la semàntica?

19. Donat el següent programa S:

    ```
    t:=x;
    x:=y;
    y:=t;
    ```

    i donada la precondició $P = (\texttt{x=a} \wedge \texttt{y=b} \wedge \texttt{z=c})$ i la postcondició $Q = (\texttt{x=b} \wedge \texttt{y=a})$, és correcte S pel que fa a P i Q ? Usa la semàntica axiomàtica (precondició més feble) per a la demostració i mostra els càlculs realitzats per a comprovar la correcció o no correcció.

20. Donat el següent programa S:

    ```
    X:=X-1
    ```

    i donada la precondició $P = (\texttt{X=1})$ i la postcondició $Q = (\texttt{X} \geq \texttt{0})$, és correcte S pel que fa a P i Q ? Usa la semàntica axiomàtica (precondició més feble) per a la demostració i mostra els càlculs realitzats per a comprovar la correcció o no correcció.

21. Calcula la precondició més feble dels següents programes tenint en compte la postcondició indicada en cada cas:

    (a) `x:=1 {x=1}`

    (b) `x:=y {x=0}`

    (c) `x:=x-1 {x=0}`

    (d) `x:=x-1 {y>0}`

    (e) `if (x>0) then x:=y else y:=x {x>=y, y>0}`

    (f) `if (x=0) then x:=1 else x:=x+1 {x>y, y<=0}`

    (g) `x:=y; y:=5 {x>0}`

    (h) `x:=x+1; if (x>0) then x:=y else y:=x {x>0}`

Part II: Test

22. D'acord amb l'esquema de compilació vist en classe, la següent sentència Java:

```
int 3 = x;
```

quin tipus d'error produciria?

- A Error lèxic.
- B Error sintàctic.
- C Error semàntic.
- D No conté cap error.

23. Indica quina de les següents afirmacions sobre la semàntica estàtica d'un llenguatge és **CERTA**

- A Consisteix en les restriccions de sintaxis que no es poden expressar en BNF però sí comprovar en temps de compilació.
- B En la compilació, la comprovació de les restriccions de la semàntica estàtica es realitza des de l'anàlisi lèxic fins a l'anàlisi semàntic.
- C Consisteix en les restriccions que només es poden comprovar en temps d'execució.
- D Consisteix en les restriccions de sintaxis que no es poden expressar en BNF però sí comprovar en temps d'execució.

24. El resultat de l'anàlisi lèxic és:

- A Una seqüència de caràcters.
- B Una seqüència de paraules.
- C Una seqüència d'instruccions.
- D Un arbre sintàctic.

25. Indica quina de les següents afirmacions és **FALSA**:

- A La semàntica dinàmica es calcula en temps de compilació.
- B La semàntica dinàmica ens permet treballar amb propietats i/o errors que es manifesten en temps d'execució.
- C La semàntica operacional és un estil de definició de la semàntica dinàmica.
- D La semàntica estàtica no és suficient per a detectar, per exemple, si va a ocórrer una divisió per zero durant l'execució d'un programa.

26. Donada la següent execució amb la semàntica operacional de pas xicotet (small-step):

$$\langle \textbf{while } X > 0 \textbf{ do } X := X + 1,\ \{X \mapsto 0\}\rangle \ \rightarrow\ \boxed{?}$$

completa adecuadamente lo que falta (indicat amb $\boxed{?}$):

A $\{X \mapsto 0\}$.

B $\{X \mapsto 1\}$.

C $\langle skip, \{X \mapsto 0\}\rangle$.

D $\langle skip, \{X \mapsto 1\}\rangle$.

27. Suposa que s'amplia la sintaxi del llenguatge Imperatiu Simple vist en classe amb la instrucció **do** i **loop** b en la qual s'executa la instrucció i fins que es compleix la condició b per a eixir del bucle (nota que i s'executa sempre almenys una vegada). Assumint que la seua semàntica operacional *big-step* és:

$$\frac{\langle i,\ e\rangle \ \Downarrow\ e' \qquad \langle b,\ e'\rangle \Rightarrow true}{\langle \textbf{do } i \textbf{ loop } b,\ e\rangle \ \Downarrow\ e'}$$

$$\frac{\langle i,\ e\rangle \ \Downarrow\ e' \qquad \langle b,\ e'\rangle \Rightarrow false \qquad \langle \textbf{do } i \textbf{ loop } b,\ e'\rangle \ \Downarrow\ e''}{\langle \textbf{do } i \textbf{ loop } b,\ e\rangle \ \Downarrow\ e''}$$

indica quin és el valor d'e en el següent pas:

$$\langle \textbf{do } X := X + 1 \textbf{ loop } X = Y,\ \{X \mapsto 1, Y \mapsto 3\}\rangle \ \Downarrow\ e$$

A $\{X \mapsto 3, Y \mapsto 3\}$

B $\{X \mapsto 2, Y \mapsto 3\}$

C $\{X \mapsto 1, Y \mapsto 3\}$

D $\{X \mapsto 0, Y \mapsto 3\}$

28. En la semàntica operacional per al llenguatge Imperatiu Simple vist en classe, com completaries el buit en la següent regla que defineix l'avaluació de les restes $a_0 - a_1$?

$$\frac{\langle a_0, e\rangle \Rightarrow n_0 \qquad \langle a_1, e\rangle \Rightarrow n_1}{\langle a_0 - a_1, e\rangle \boxed{\ ?\ }} \quad n \text{ es la diferencia de } n_0 \text{ y } n_1$$

A $\rightarrow \langle skip, e[X \mapsto n]\rangle$

B $\Rightarrow \langle skip, \{e \mapsto n\}\rangle$

C $\Downarrow n$

D $\Rightarrow n$

29. Considerem la terna de Hoare $\{P\}$ $\mathtt{x} := \mathtt{x} + \mathtt{y}$ $\{Q\}$, sent Q el asert $\mathtt{y} = 0$.Quin dels següents asertos P fa que la terna siga *correcta*?

   A $\mathtt{x} = \mathtt{x}$.

   B $\mathtt{x} + \mathtt{y} = 0$.

   C $\mathtt{y} = 0$.

   D $\mathtt{x} = \mathtt{x} + 0$.

30. Indica quina és la precondició més feble del següent programa `S`

```
x:= x+1;
y:= x+y
```

   pel que fa a la postcondició `{Q}={y>5}`

   A `pmd(S,Q)={x+y>4}`.

   B `pmd(S,Q)={x+y>=4}`.

   C `pmd(S,Q)={x>5}`.

   D `pmd(S,Q)={x+y>5}`.

31. Indica quina de les següents afirmacions és FALSA :

   A La semàntica dinàmica es calcula en temps de compilació.

   B La semàntica dinàmica ens permet treballar amb propietats i/o errors que es manifesten en temps d'execució.

   C La semàntica operacional és un estil de definició de la semàntica dinàmica.

   D La semàntica estàtica no és suficient per a detectar, per exemple, si va a ocórrer una divisió per zero durant l'execució d'un programa.

32. Quina de les següents afirmacions sobre la semàntica operacional de pas xicotet (*small-step*) és **CERTA** ?

   A L'avaluació d'expressions aritmètiques i booleanes es descriu com en la de pas gran (*big-step*).

   B En tot pas de transició sempre es modifica l'estat.

   C L'estat final no pot obtenir-se a partir de l'última configuració de la traça.

   D Les configuracions consten d'un asert i una instrucció de programa.

33. En emprar la semàntica operacional de pas gran (*big-step*), què hem d'escriure en lloc de l'interrogant en la següent expressió?

$$\langle \texttt{while x} > 0 \texttt{ do x} := \texttt{x} - 1, \{\texttt{x} \mapsto 1, \texttt{y} \mapsto 1\}\rangle \Downarrow \boxed{\ ?\ }$$

A $\langle \texttt{x} := \texttt{x} - 1; \texttt{while x} > 0 \texttt{ do x} := \texttt{x} - 1, \{\texttt{x} \mapsto 1, \texttt{y} \mapsto 1\}\rangle$.

B $\{\texttt{x} \mapsto 0, \texttt{y} \mapsto 1\}$.

C $\langle skip, \{\texttt{x} \mapsto 0, \texttt{y} \mapsto 1\}\rangle$.

D $\langle \texttt{while x} > 0 \texttt{ do x} := \texttt{x} - 1, \{\texttt{x} \mapsto 0, \texttt{y} \mapsto 1\}\rangle$.

34. Quin és el resultat de $\mathsf{pmd}(\texttt{x} := \texttt{x} + \texttt{y}, \{\texttt{y} = 0\})$?

A $\{\texttt{x} = \texttt{x}\}$

B $\{\texttt{x} + \texttt{y} = 0\}$

C $\{\texttt{y} = 0\}$

D $\{\texttt{x} = \texttt{x} + 0\}$

35. Suposant la següent definició del transformador de predicats pmd que associa a una instrucció d'assignació múltiple ($::=$) i un predicat Q la seua precondició més feble canviant *simultàniament* les variables $\mathsf{x_i}$ per les seues corresponents expressions $\mathsf{expr_i}$:

$$\mathsf{pmd}(\text{“}\mathsf{x_0}, \mathsf{x_1}, \ldots, \mathsf{x_n} ::= \mathsf{exp_0}, \mathsf{exp_1}, \ldots, \mathsf{exp_n}\text{”}, \mathsf{Q}) = \mathsf{Q}[\mathsf{x_i} \mapsto \mathsf{exp_i}]_{\mathsf{i}=0}^{\mathsf{n}}$$

Quin és el resultat de $\mathsf{pmd}(\texttt{x}, \texttt{y} ::= 1, \texttt{x} + 1, \{\texttt{x} > 0, \texttt{y} \geq 1\})$?

A $\{\texttt{x} > 1, \texttt{y} \geq 2\}$

B $\{\texttt{x} > 0, 2 \geq 2\}$

C $\{1 > 0, \texttt{x} \geq 0\}$ o també $\{1 > 0, \texttt{x} + 1 \geq 1\}$ o $\{\texttt{x} \geq 0\}$

D $\{\texttt{x} > 1, \texttt{y} \geq 2\}$ o també $\{\texttt{x} > 0\}$

36. Donats els següents programes P_1 i P_2,

P_1:

```
x:=0;
if x>0 then y:=10
       else y:=5
```

P_2:

```
x:=0;
x:=x+5;
y:=x
```

podem dir que:

A P_1 i P_2 són equivalents, independentment de la semàntica que se seguisca.

B P_1 i P_2 són equivalents pel que fa a la semàntica big-step.

C P_1 i P_2 són equivalents pel que fa a la semàntica small-step.

D P_1 i P_2 no són equivalents, independentment que considerem la semàntica big-step o small-step.

37. Els següents programes són equivalents?

```
x := 1;                    x := 3;
x := 2;                    while x > 5 do
x := 3;                       x := 1; x := 2;
```

A Segons la semàntica de pas gran NO ho són.

B Segons la semàntica de pas xicotet NO ho són.

C Si l'estat inicial és $\{\mathtt{x} \mapsto 0\}$, segons la semàntica de pas xicotet SÍ ho són.

D Mai poden ser equivalents.

38. Assenyale l'opció **FALSA**:

A Un compilador rep un programa font i retorna un programa objecte.

B Un programa objecte rep dades d'entrada i produeix dades d'eixida.

C Un intèrpret rep un programa font i dades d'entrada i retorna dades d'eixida.

D Un programa font rep dades d'entrada i retorna un programa objecte.

39. ¿En què consisteix una implementació mixta d'un llenguatge de programació?

A Primer es tradueix el codi a un llenguatge intermedi, i després el codi resultant s'interpreta.

B Consisteix que parts del programa es tradueixen i unes altres, les més difícils, s'interpreten.

C Primer s'interpreta el codi i, després, es tradueix a llenguatge màquina.

D En l'esquema mixt sempre es tradueix el programa a llenguatge màquina, que és després interpretat en la màquina virtual del processador.

A BNF Grammar for C-Minus

Keywords: `else if int return void while`

Special symbols: `+ - * / < <= > >= == = ; , ( ) [ ] /* */`!

Comments: `/* ... */`

```
<program>             ::= <declaration-list>
<declaration-list>    ::= <declaration-list> <declaration> | <declaration>
<declaration>         ::= <var-declaration> | <fun-declaration>
<var-declaration>     ::= <type-specifier> <ID> ; | <type-specifier> <ID> [ <NUM> ] ;
<type-specifier>      ::= int | void
<fun-declaration>     ::= <type-specifier> <ID> ( <params> ) <compount-stmt>
<params>              ::= <param-list> |  empty
<param-list>          ::= <param-list> , <param> | <param>
<param>               ::= <type-specifier> <ID> | <type-specifier> <ID> [ ]
<compound-stmt>       ::= { <local-declarations> <statement-list> }
<local-declarations>  ::= <local-declarations> <var-declarations> |  empty
<statement-list>      ::= <statement-list> <statement> | empty
<statement>           ::= <expression-stmt> | <compount-stmt> | <selection-stmt> |
                          <iteration-stmt> | <return-stmt>
<expression-stmt>     ::= <expression> ; | ;
<selection-stmt>      ::= if ( <expression> ) <statement> |
                          if ( <expression> ) <statement> else <statement>
<iteration-stmt>      ::= while ( <expression> ) <statement>
<return-stmt>         ::= return ; | return <expression> ;
<expression>          ::= <var> = <expression> | <simple-expression>
<var>                 ::= <ID> | <ID> [ <expression> ]
<simple-expression>   ::= <additive-expression> <relop> <additive-expression> |
                          <additive-expression>
<relop>               ::= <= | < | > | >= | == | !=
<additive-expression> ::= <additive-expression> <addop> <term> | <term>
<addop>               ::= + | -
<term>                ::= <term> <mulop> <factor> | <factor>
<mulop>               ::= * | /
<factor>              ::= ( <expression> ) | <var> | <call> | <NUM>
<call>                ::= <ID> ( <args> )
<args>                ::= <arg-list> |  empty
<arg-list>            ::= <arg-list> , <expression> | <expression>
<ID>                  ::= <letter>+
<NUM>                 ::= <digit>+
<letter>              ::= a | b | ... | z | A | B | ... | Z
<digit>               ::= 0 | 1 | ... | 9
```