Ejercicios LTP

Tema 2: Fundamentos de los Lenguajes de Programación

Parte I: Cuestiones

1. De acuerdo a las siguientes reglas BNF:

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

   Indica si las siguientes sentencias son correctas (puede haber más de 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. De acuerdo a las siguientes reglas 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 las siguientes afirmaciones son correctas (puede haber más de una correcta):

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

3. Analiza si el siguiente programa escrito en C-Minus es válido con respecto a la gramática incluida en el Apéndice, que describe la sintaxis de un pequeño subconjunto de C (llamado C-Minus) y justifica tu respuesta:

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

 while (c <= n)
```

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

4. Da las reglas semánticas para la evaluación de la operación booleana de disyunción.

5. Dado el siguiente código P:

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

   (a) Desarrolla la traza de ejecución (mostrando los cómputos intermedios) siguiendo la semántica *small-step* con el estado inicial $s_I = \{\}$.

   (b) Calcula la semántica *big-step*, mostrando los cómputos intermedios, con el estado inicial vacío.

6. Dada la siguiente configuración, desarrolla la evaluación de la expresión aritmética aplicando las reglas correspondientes:

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

7. Dada la siguiente configuración, desarrolla la evaluación de la expresión booleana aplicando las reglas correspondientes:

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

8. Dado el siguiente código P:

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

   (a) Desarrolla la traza de ejecución (mostrando los cómputos intermedios) siguiendo la semántica *small-step* con el estado inicial $\{X \mapsto 2\}$.

   (b) Calcula la semántica *big-step*, mostrando los cómputos intermedios, con el estado inicial $\{X \mapsto 2\}$.

9. Queremos extender el lenguaje SIMP visto en las transparencias con una nueva instrucción *repeat* cuya sintaxis es

```
repeat i until b
```

   El comportamiento de esta instrucción es el siguiente: debe ejecutarse la instrucción (posiblemente compuesta) `i` hasta que se satisfaga la condición `b`, en cuyo momento se detendrá la ejecución del *repeat*.

(a) Da la(s) regla(s) semántica(s) siguiendo el estilo *small-step* par la instrucción *repeat*

(b) Da la(s) regla(s) semántica(s) siguiendo el estilo *big-step* par la instrucción *repeat*

10. Dado el siguiente código P:

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

(a) Desarrolla la traza de ejecución (mostrando los cómputos intermedios) siguiendo la semántica *small-step* con el estado inicial $s = \{\}$.

(b) Calcula la semántica *big-step*, mostrando los cómputos intermedios, con el estado inicial $s = \{\}$.

11. Queremos extender el lenguaje SIMP visto en las transparencias con una nueva instrucción *for* cuya sintaxis es

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

El comportamiento de esta instrucción es el siguiente: la variable `V` (el "contador") va tomando valores, en orden creciente, desde `a0` hasta `a1` (ambos inclusive) y, tras cada asignación, se ejecuta la instrucción `i`.

(a) Da la(s) regla(s) semántica(s) siguiendo el estilo *small-step* par la instrucción *for*

(b) Da la(s) regla(s) semántica(s) siguiendo el estilo *big-step* par la instrucción *for*

12. Queremos extender el lenguaje SIMP visto en las transparencias con una nueva instrucción *times* cuya sintaxis es

```
do n times i
```

El comportamiento de esta instrucción es el siguiente: se ejecuta la instrucción `i` tantas veces como indique el número natural `n` (si `n=0` la instrucción `i` no se ejecuta).

(a) Da la(s) regla(s) semántica(s) siguiendo el estilo *small-step* par la instrucción *times*

(b) Da la(s) regla(s) semántica(s) siguiendo el estilo *big-step* par la instrucción *times*

13. Dado el siguiente código S donde se calcula el máximo de dos números:

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

(a) Desarrolla la traza de ejecución (mostrando los cómputos intermedios) siguiendo la semántica *small-step* con el estado inicial $\{X \mapsto 3, Y \mapsto 5\}$.

(b) Calcula la semántica *big-step*, mostrando los cómputos intermedios, con el estado inicial $\{X \mapsto 3, Y \mapsto 5\}$.

14. Queremos extender el lenguaje SIMP visto en las transparencias con un nuevo operador de asignación múltiple cuya sintaxis es

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

donde

- las $\texttt{x}_i$ son distintas entre sí,
- las $\texttt{a}_i$ son expresiones

y su comportamiento debe ser el siguiente (ver [Gries81], página 121):

- primero se evalúan las expresiones $\texttt{a}_1\texttt{,a}_2\texttt{,...a}_n$ (en cualquier orden), obteniéndose los valores $\texttt{v}_1\texttt{,v}_2\texttt{,...v}_n$ respectivamente;
- para cada i, se asigna a $\texttt{x}_i$ el valor $\texttt{v}_i$.

(a) Da la(s) regla(s) semántica(s) siguiendo el estilo *small-step* par la instrucción *times*

(b) Da la(s) regla(s) semántica(s) siguiendo el estilo *big-step* par la instrucción *times*

15. Dado el siguiente programa S:

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

Desarrolla la traza de la ejecución a partir del estado $\{x \mapsto 2, y \mapsto 5\}$ usando la semántica operacional de paso pequeño.

16. Dado el siguiente fragmento de código P:

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

usando la semántica operacional de paso pequeño, construye la traza de ejecución a partir del estado inicial $\{x \mapsto 3, y \mapsto 7\}$.

17. Considera el siguiente código en C que devuelve el máximo de dos números:

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

(a) Escribe el cuerpo de la función `maximo` en la sintaxis del lenguaje SIMP (en SIMP no hay subprogramas).

(b) Construye las trazas de la ejecución de la llamada `maximo(3,5)` (es decir, partiendo del estado inicial $\{x \mapsto 3, y \mapsto 5\}$), usando las semánticas de paso pequeño y grande.

18. Calcula la precondición más débil $pmd(S, Q)$ para el programa S de la pregunta 15 y la postcondición Q dada por $\texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y}$. De acuerdo a los resultados obtenidos:

    (a) ¿Qué hace este programa? ¿Cuál es la diferencia (si la hay) entre el programa S y el siguiente programa S' que usa la asignación múltiple introducida en la pregunta 14?

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

    (b) Considerando la semántica axiomática, ¿hay alguna diferencia entre S y S' con respecto a la postcondición Q?

    (c) ¿Son equivalentes los programas S y S' desde el punto de vista operacional o se podrían distinguir usando la semántica?

19. Dado el siguiente programa S:

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

    y dada la precondición $P = (\texttt{X=1})$ y la postcondición $Q = (\texttt{X} \geq \texttt{0})$, ¿es correcto S con respecto a P y Q? Usa la semántica axiomática (precondición más débil) para la demostración y muestra los cálculos realizados para comprobar la corrección o no corrección.

20. Calcula la precondición más débil de los siguientes programas teniendo en cuenta la postcondición indicada en cada caso:

    (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}`

Parte II: Test

21. De acuerdo con el esquema de compilación visto en clase, la siguiente sentencia Java:

```
int 3 = x;
```

¿qué tipo de error produciría?

A Error léxico.

B Error sintáctico.

C Error semántico.

D No contiene ningún error.

22. Indica cuál de las siguientes afirmaciones sobre la semántica estática de un lenguaje es **CIERTA**

A Consiste en las restricciones de sintaxis que no se pueden expresar en BNF pero sí comprobar en tiempo de compilación.

B En la compilación, la comprobación de las restricciones de la semántica estática se realiza desde el análisis léxico hasta el análisis semántico.

C Consiste en las restricciones que sólo se pueden comprobar en tiempo de ejecución.

D Consiste en las restricciones de sintaxis que no se pueden expresar en BNF pero sí comprobar en tiempo de ejecución.

23. El resultado del análisis léxico es:

A Una secuencia de caracteres.

B Una secuencia de palabras.

C Una secuencia de instrucciones.

D Un árbol sintáctico.

24. Indica cuál de las siguientes afirmaciones es **FALSA**:

A La semántica dinámica se calcula en tiempo de compilación.

B La semántica dinámica nos permite trabajar con propiedades y/o errores que se manifiestan en tiempo de ejecución.

C La semántica operacional es un estilo de definición de la semántica dinámica.

D La semántica estática no es suficiente para detectar, por ejemplo, si va a ocurrir una división por cero durante la ejecución de un programa.

25. Dada la siguiente ejecución con la semántica operacional de paso pequeño (small-step):

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

completa adecuadamente lo que falta (indicado con $\boxed{?}$):

A $\{X \mapsto 0\}$.
B $\{X \mapsto 1\}$.
C $\langle skip, \{X \mapsto 0\}\rangle$.
D $\langle skip, \{X \mapsto 1\}\rangle$.

26. Supón que se amplía la sintaxis del lenguaje IMPerativo Simple visto en clase con la instrucción **do** i **loop** b en la que se ejecuta la instrucción i hasta que se cumple la condición b para salir del bucle (nota que i se ejecuta siempre por lo menos una vez). Asumiendo que su semántica operacional *big-step* es:

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

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

indica cuál es el valor de e en el siguiente paso:

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

A $\{X \mapsto 3, Y \mapsto 3\}$
B $\{X \mapsto 2, Y \mapsto 3\}$
C $\{X \mapsto 1, Y \mapsto 3\}$
D $\{X \mapsto 0, Y \mapsto 3\}$

27. En la semántica operacional para el lenguaje IMPerativo Simple visto en clase, ¿cómo completarías el hueco en la siguiente regla que define la evaluación de las restas $a_0 - a_1$?

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

A $\rightarrow \langle skip, e[X \mapsto n]\rangle$
B $\Rightarrow \langle skip, \{e \mapsto n\}\rangle$
C $\Downarrow n$
D $\Rightarrow n$

28. Consideremos la terna de Hoare $\{P\}$ $\mathtt{x} := \mathtt{x} + \mathtt{y}$ $\{Q\}$, siendo Q el aserto $\mathtt{y} = 0$. ¿Cuál de los siguientes asertos P hace que la terna sea *correcta*?

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

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

C $\mathtt{y} = 0$.

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

29. Indica cuál es la precondición más débil del siguiente programa `S`

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

con respecto a la postcondición `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).`

30. ¿Cuál de las siguientes afirmaciones acerca de la semántica operacional de paso pequeño (*small-step*) es **CIERTA** ?

A La evaluación de expresiones aritméticas y booleanas se describe como en la de paso grande (*big-step*).

B En todo paso de transición siempre se modifica el estado.

C El estado final no puede obtenerse a partir de la última configuración de la traza.

D Las configuraciones constan de un aserto y una instrucción de programa.

31. Al emplear la semántica operacional de paso grande (*big-step*), ¿qué debemos escribir en lugar del interrogante en la siguiente expresión?

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

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

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

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

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

32. Suponiendo la siguiente definición del transformador de predicados pmd que asocia a una instrucción de asignación múltiple (::=) y un predicado Q su precondición más débil cambiando *simultáneamente* las variables x_i por sus correspondientes expresiones $expr_i$:

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

¿Cuál es el resultado de $pmd(x, y ::= 1, x + 1, (x > 0, y \geq 1))$?

A $(x > 1 \wedge y \geq 2)$

B $(x > 0 \wedge 2 \geq 2)$

C $(1 > 0 \wedge x \geq 0)$ o también $(1 > 0 \wedge (x + 1 \geq 1))$ o $(x \geq 0)$

D $(x > 1 \wedge y \geq 2)$ o también $(x > 0)$

33. Dados los siguientes programas P_1 y P_2,

P_1:

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

P_2:

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

podemos decir que:

A P_1 y P_2 son equivalentes, independientemente de la semántica que se siga.

B P_1 y P_2 son equivalentes con respecto a la semántica big-step.

C P_1 y P_2 son equivalentes con respecto a la semántica small-step.

D P_1 y P_2 no son equivalentes, independientemente de que consideremos la semántica big-step o small-step.

34. ¿Los siguientes programas son equivalentes?

```
x := 1;
x := 2;
x := 3;
```

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

A Según la semántica de paso grande NO lo son.

B Según la semántica de paso pequeño NO lo son.

C Si el estado inicial es $\{x \mapsto 0\}$, según la semántica de paso pequeño SÍ lo son.

D Nunca pueden ser equivalentes.

35. Señale la opción **FALSA**:

A Un compilador recibe un programa fuente y devuelve un programa objeto.

B Un programa objeto recibe datos de entrada y produce datos de salida.

C Un intérprete recibe un programa fuente y datos de entrada y devuelve datos de salida.

D Un programa fuente recibe datos de entrada y devuelve un programa objeto.

36. ¿En qué consiste una implementación mixta de un lenguaje de programación?

A Primero se traduce el código a un lenguaje intermedio, y después el código resultante se interpreta.

B Consiste en que partes del programa se traducen y otras, las más difíciles, se interpretan.

C Primero se interpreta el código y, luego, se traduce a lenguaje máquina.

D En el esquema mixto siempre se traduce el programa a lenguaje máquina, que es luego interpretado en la máquina virtual del procesador.

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> [ ]
<compount-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
```