LTP Exercises

Theme 2: Foundations of Programming Languages

Part I: Questions

1. According to the following BNF rules:

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

   which of the following sentences is *legal*?

   - `IF X>0 THEN X:=X+1;`
     WRONG. The `ELSE` part is missing.
   - `IF X<0 THEN X:=X+1 ELSE X:=X-1;`
     LEGAL.
   - `IF X>0 THEN X:=X+1 ELSE X<0;`
     WRONG. After `ELSE` there should be an expresion rather than a condition.
   - `IF X>0 THEN X:=X-1 ELSE X:=X+1`
     WRONG. A semicolon is missing at the end of the sentence.

2. According to the following BNF rules:

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

   which of the following claims is *true*?

   - `1+1` is an `<arit>` expression.
     TRUE.
   - `1+2-3` is an `<arit>` expression.
     FALSE.
   - `1+2=X` is an `<expr>` expression.
     TRUE.
   - `Z=2+3;Y=1-4` is an `<expr>` expression.
     TRUE.

3. Check whether the following C program is legal with respect to the grammar which describes the syntax of C-minus, a subset of C (see the appendix below). Provide and answer (yes/no) and a detailed explanation.

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

  while (c<=n) {
    result = result*c;
    c++;
    }

  return result;
}
```

Answer (yes/no): No

Justification: According to the C-minus BNF grammar, variable declarations have the following syntax:

var-declaration $\rightarrow$ *type-specifier* **ID ;** | *type-specifier* **ID [NUM];**

Thus, variable initilization is not allowed in C-minus. More errors:

- Type `long` is not allowed.
- Postincrement operator is not allowed.
- A semicolon must follow every program instruction.

4. Write the semantic rules that define the boolean disjunction.

$$\frac{\langle b_0, s\rangle \Rightarrow \mathit{true}}{\langle b_0 \vee b_1, s\rangle \Rightarrow \mathit{true}} \qquad \frac{\langle b_0, s\rangle \Rightarrow \mathit{false} \qquad \langle b_1, s\rangle \Rightarrow \beta}{\langle b_0 \vee b_1, s\rangle \Rightarrow \beta}$$

5. Consider the following code P:

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

   (a) Write the execution trace that corresponds to the small-step semantics starting from the initial state $s_I = \{\}$.

The trace is as follows:

(1) $\langle(\texttt{X} := 5; \texttt{Y} := \texttt{X}), \{\}\rangle \rightarrow$
$\quad\langle \texttt{X} := 5, \{\}\rangle \rightarrow$
$\quad\quad\langle 5, \{\}\rangle \Rightarrow 5$
$\quad\langle \texttt{skip}, \{X \mapsto 5\}\rangle$
(2) $\langle(\texttt{skip}; \texttt{Y} := \texttt{X}), \{X \mapsto 5\}\rangle \rightarrow$
(3) $\langle \texttt{Y} := \texttt{X}, \{X \mapsto 5\}\rangle \rightarrow$
$\quad\langle \texttt{X}, \{X \mapsto 5\}\rangle \Rightarrow 5$
(4) $\langle \texttt{skip}, \{X \mapsto 5, Y \mapsto 5\}\rangle$

where the steps which are part of the trace are numbered from **(1)** to **(4)**. The intermediate (indented) computations correspond to proofs of the conditions in the premise part of the main small-step rule which is applied to perform each of the main steps. For instance, the third line (i.e., $\langle 5, \{\}\rangle \Rightarrow 5$) which is required for transition **(1)** corresponds to the premise $\langle a, s\rangle \Rightarrow n$ that performs the evaluation of the arithmetic expression a in the application of the assignment rule

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

in the second and fourth lines.

(b) Compute the *big-step* semantics for the initial state $s_I = \{\}$. Show the intermediate computations as well.

$\langle \texttt{X} := 5; \texttt{Y} := \texttt{X}, \{\}\rangle$
$\quad\langle \texttt{X} := 5, \{\}\rangle$
$\quad\quad\langle 5, \{\}\rangle \Rightarrow 5$
$\quad\Downarrow \{X \mapsto 5\}$
$\quad\langle \texttt{Y} := \texttt{X}, \{X \mapsto 5\}\rangle$
$\quad\quad\langle \texttt{X}, \{X \mapsto 5\}\rangle \Rightarrow 5$
$\quad\Downarrow \{X \mapsto 5, Y \mapsto 5\}$
$\Downarrow \{X \mapsto 5, Y \mapsto 5\}$

6. For the following configuration, develop the evaluation of the arithmetic expression by using the appropriate rules:

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

$\langle \texttt{X} + 3, \{X \mapsto 2\}\rangle \Rightarrow$
$\quad\langle \texttt{X}, \{X \mapsto 2\}\rangle \Rightarrow 2$
$\quad\langle 3, \{X \mapsto 2\}\rangle \Rightarrow 3$
5

7. For the following configuration, develop the evaluation of the boolean expression by using the appropriate rules:

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

$\langle \texttt{X} + 3 \leq \texttt{Y}, \{X \mapsto 2, Y \mapsto 0\}\rangle \Rightarrow$
$\quad \langle \texttt{X} + 3, \{X \mapsto 2, Y \mapsto 0\}\rangle \Rightarrow$
$\quad\quad \langle \texttt{X}, \{X \mapsto 2, Y \mapsto 0\}\rangle \Rightarrow 2$
$\quad\quad \langle 3, \{X \mapsto 2, Y \mapsto 0\}\rangle \Rightarrow 3$
$\quad 5$
$\quad \langle \texttt{Y}, \{X \mapsto 2, Y \mapsto 0\}\rangle \Rightarrow 0$
$\quad 5 \leq 0$ is false
false

8. Consider the following program P:

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

(a) Write the *small-step* execution trace together with the intermediate computations for the initial state $\{X \mapsto 2\}$.

(1) $\langle(\texttt{X} := 5; \texttt{if X} > 3 \texttt{ then X} := \texttt{X} - 1 \texttt{ else Y} := \texttt{X}), \{X \mapsto 2\}\rangle \rightarrow$
$\quad \langle \texttt{X} := 5, \{X \mapsto 2\}\rangle \rightarrow$
$\quad\quad \langle 5, \{X \mapsto 2\}\rangle \Rightarrow 5$
$\quad \langle \texttt{skip}, \{X \mapsto 5\}\rangle$
(2) $\langle(\texttt{skip}; \texttt{if X} > 3 \texttt{ then X} := \texttt{X} - 1 \texttt{ else Y} := \texttt{X}), \{X \mapsto 5\}\rangle \rightarrow$
(3) $\langle \texttt{if X} > 3 \texttt{ then X} := \texttt{X} - 1 \texttt{ else Y} := \texttt{X}, \{X \mapsto 5\}\rangle \rightarrow$
$\quad \langle X > 3, \{X \mapsto 5\}\rangle \Rightarrow$
$\quad\quad \langle X, \{X \mapsto 5\}\rangle \Rightarrow 5$
$\quad\quad \langle 3, \{X \mapsto 5\}\rangle \Rightarrow 3$
$\quad\quad 5 > 3$ is true
$\quad$ *true*
(4) $\langle \texttt{X} := \texttt{X} - 1, \{X \mapsto 5\}\rangle \rightarrow$
$\quad \langle \texttt{X} - 1, \{X \mapsto 5\}\rangle \Rightarrow$
$\quad\quad \langle X, \{X \mapsto 5\}\rangle \Rightarrow 5$
$\quad\quad \langle 1, \{X \mapsto 5\}\rangle \Rightarrow 1$
$\quad\quad 5 - 1 = 4$ is computed
$\quad 4$
(5) $\langle \texttt{skip}, \{X \mapsto 4\}\rangle$

(b) Compute now the *big-step* semantics for the same initial state $\{X \mapsto 2\}$ (again, show the intermediate computations)

$\langle(\texttt{X} := 5; \texttt{if X} > 3 \texttt{ then X} := \texttt{X} - 1 \texttt{ else Y} := \texttt{X}), \{X \mapsto 2\}\rangle$
$\quad \langle \texttt{X} := 5, \{X \mapsto 2\}\rangle$
$\quad\quad \langle 5, \{X \mapsto 2\}\rangle \Rightarrow 5$
$\quad \Downarrow \{X \mapsto 5\}$
$\quad \langle \texttt{if X} > 3 \texttt{ then X} := \texttt{X} - 1 \texttt{ else Y} := \texttt{X}, \{X \mapsto 5\}\rangle$
$\quad\quad \langle \texttt{X} > 3, \{X \mapsto 5\}\rangle \Rightarrow$
$\quad\quad\quad \langle \texttt{X}, \{X \mapsto 5\}\rangle \Rightarrow 5$
$\quad\quad\quad \langle 3, \{X \mapsto 5\}\rangle \Rightarrow 3$
$\quad\quad\quad 5 > 3$ is true
$\quad\quad \mathit{true}$
$\quad\quad \langle \texttt{X} := \texttt{X} - 1, \{X \mapsto 5\}\rangle$
$\quad\quad\quad \langle \texttt{X} - 1, \{X \mapsto 5\}\rangle \Rightarrow$
$\quad\quad\quad\quad \langle \texttt{X}, \{X \mapsto 5\}\rangle \Rightarrow 5$
$\quad\quad\quad\quad \langle 1, \{X \mapsto 5\}\rangle \Rightarrow 1$
$\quad\quad\quad\quad 5 - 1 = 4$ is computed
$\quad\quad\quad 4$
$\quad\quad \Downarrow \{X \mapsto 4\}$
$\quad \Downarrow \{X \mapsto 4\}$
$\Downarrow \{X \mapsto 4\}$

9. We want to extend language SIMP with a new instruction *repeat* with the following syntax:

   `repeat i until b`

   This instruction works as follows: the (possibly compound) instruction `i` is executed until condition `b` is satisfied; then the execution of the instruction terminates.

   (a) Provide the rules for the *small-step* semantic description of *repeat*.

   Solution A: Translation into a `while` loop:

$$\frac{}{\langle \texttt{repeat i until b}, s\rangle \rightarrow \langle \texttt{i}; \texttt{while } \neg\texttt{b do i}, s\rangle}$$

   Solution B: Direct definition:

$$\frac{\langle \texttt{b}, s\rangle \Rightarrow \mathit{true}}{\langle \texttt{repeat skip until b}, s\rangle \rightarrow \langle \texttt{skip}, s\rangle}$$

$$\frac{\langle \texttt{b}, s\rangle \Rightarrow \mathit{false}}{\langle \texttt{repeat skip until b}, s\rangle \rightarrow \langle \texttt{repeat skip until b}, s\rangle}$$

$$\frac{\langle \texttt{i}, s\rangle \rightarrow \langle \texttt{i}', s'\rangle}{\langle \texttt{repeat i until b}, s\rangle \rightarrow \langle \texttt{i}'; \texttt{if b then skip else (repeat i until b)}, s'\rangle} \quad \text{if } \texttt{i} \neq \texttt{skip}$$

   (b) Provide the rules for the *big-step* semantic description of *repeat*

$$\frac{\langle i,\ s\rangle \ \Downarrow\ s' \qquad \langle b,\ s'\rangle \Rightarrow \mathit{true}}{\langle \texttt{repeat } i \texttt{ until } b,\ e\rangle \ \ \Downarrow\ \ e'}$$

$$\frac{\langle i,\ s\rangle \ \Downarrow\ s' \qquad \langle b,\ s'\rangle \Rightarrow \mathit{false} \qquad \langle \texttt{repeat } i \texttt{ until } b,\ s'\rangle \ \Downarrow\ \ s''}{\langle \texttt{repeat } i \texttt{ until } b,\ s\rangle \ \ \Downarrow\ \ s''}$$

(1) $\langle(\texttt{X} := \texttt{4}; \texttt{while X} > \texttt{3 do X} := \texttt{X} - \texttt{1}), \{\}\rangle \rightarrow$
$\langle \texttt{X} := \texttt{4}, \{\}\rangle \rightarrow$
$\langle \texttt{4}, \{\}\rangle \Rightarrow 4$
$\langle \texttt{skip}, \{X \mapsto 4\}\rangle$
(2) $\langle(\texttt{skip}; \texttt{while X} > \texttt{3 do X} := \texttt{X} - \texttt{1}), \{X \mapsto 4\}\rangle \rightarrow$
(3) $\langle \texttt{while X} > \texttt{3 do X} := \texttt{X} - \texttt{1}, \{X \mapsto 4\}\rangle \rightarrow$
$\langle \texttt{X} > \texttt{3}, \{X \mapsto 4\}\rangle \Rightarrow$
$\langle \texttt{X}, \{X \mapsto 4\}\rangle \Rightarrow 4$
$\langle \texttt{3}, \{X \mapsto 4\}\rangle \Rightarrow 3$
$4 > 3$ holds
true
(4) $\langle(\texttt{X} := \texttt{X} - \texttt{1}; \texttt{while X} > \texttt{3 do X} := \texttt{X} - \texttt{1}), \{X \mapsto 4\}\rangle \rightarrow$
$\langle \texttt{X} := \texttt{X} - \texttt{1}, \{X \mapsto 4\}\rangle \rightarrow$
$\langle \texttt{X} - \texttt{1}, \{X \mapsto 4\}\rangle \Rightarrow$
$\langle \texttt{X}, \{X \mapsto 4\}\rangle \Rightarrow 4$
$\langle \texttt{1}, \{X \mapsto 4\}\rangle \Rightarrow 1$
$4 - 1 = 3$ is obtained
3
$\langle \texttt{skip}, \{X \mapsto 3\}\rangle$
(5) $\langle(\texttt{skip}; \texttt{while X} > \texttt{3 do X} := \texttt{X} - \texttt{1}), \{X \mapsto 3\}\rangle \rightarrow$
(6) $\langle \texttt{while X} > \texttt{3 do X} := \texttt{X} - \texttt{1}, \{X \mapsto 3\}\rangle \rightarrow$
$\langle \texttt{X} > \texttt{3}, \{X \mapsto 3\}\rangle \Rightarrow$
$\langle \texttt{X}, \{X \mapsto 3\}\rangle \Rightarrow 3$
$\langle \texttt{3}, \{X \mapsto 3\}\rangle \Rightarrow 3$
$3 > 3$ is false
false
(7) $\langle \texttt{skip}, \{X \mapsto 3\}\rangle$

Figure 1: Solution to Exercise 10a

10. Consider the following program P:

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

(a) Write the *small-step* execution trace (with the intermediate computations) from the empty state $s = \{\}$.

(b) Write the *big-step* execution tree (with the intermediate computations) from the empty state $s = \{\}$.

$\langle$(X := 4; while X > 3 do X := X − 1), {}$\rangle$
  $\langle$X := 4, {}$\rangle$
    $\langle$4, {}$\rangle \Rightarrow 4$
  $\Downarrow \{X \mapsto 4\}$
  $\langle$while X > 3 do X := X − 1, $\{X \mapsto 4\}\rangle$
    $\langle$X > 3, $\{X \mapsto 4\}\rangle \Rightarrow$
      $\langle$X, $\{X \mapsto 4\}\rangle \Rightarrow 4$
      $\langle$3, $\{X \mapsto 4\}\rangle \Rightarrow 3$
      $4 > 3$ es true
    true
    $\langle$(X := X − 1; while X > 3 do X := X − 1), $\{X \mapsto 4\}\rangle$
      $\langle$X := X − 1, $\{X \mapsto 4\}\rangle$
        $\langle$X − 1, $\{X \mapsto 4\}\rangle \Rightarrow$
          $\langle$X, $\{X \mapsto 4\}\rangle \Rightarrow 4$
          $\langle$1, $\{X \mapsto 4\}\rangle \Rightarrow 1$
          $4 - 1 = 3$ es computado
        3
      $\Downarrow \{X \mapsto 3\}$
    $\langle$while X > 3 do X := X − 1, $\{X \mapsto 3\}\rangle$
      $\langle$X > 3, $\{X \mapsto 3\}\rangle \Rightarrow$
        $\langle$X, $\{X \mapsto 3\}\rangle \Rightarrow 3$
        $\langle$3, $\{X \mapsto 3\}\rangle \Rightarrow 3$
        $3 > 3$ es false
      false
    $\Downarrow \{X \mapsto 3\}$
  $\Downarrow \{X \mapsto 3\}$
$\Downarrow \{X \mapsto 3\}$

11. Consider a new *for* loop for SIMP as follows

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

    where counter `V` takes increasing values from a0 to a1 (both included), as instruction `i` is executed.

    (a) Provide the *small-step* semantic description corresponding to instruction *for*.

Solution A: by transformation into a *while* loop:

$$\frac{}{\langle \texttt{for V := a0 to a1 do i}, s\rangle \rightarrow \langle (\texttt{V := a0; while V <= a1 do (i; V := V + 1)}), s\rangle}$$

Solución B: direct definition using two rules:

$$\frac{\langle \texttt{a0} > \texttt{a1}, s\rangle \Rightarrow \mathit{true}}{\langle \texttt{for V := a0 to a1 do i}, s\rangle \rightarrow \langle \texttt{skip}, s\rangle}$$

$$\frac{\langle \texttt{a0} > \texttt{a1}, s\rangle \Rightarrow \mathit{false}}{\langle \texttt{for V := a0 to a1 do i}, s\rangle \rightarrow \langle (\texttt{V := a0; i; for V := V + 1 to a1 do i}), s\rangle}$$

In solution B, the assignment of `a0` to the control variable `V` before executing `i` is necessary as instruction `i` could use it during its execution. Also, the 'initialization' of `V` to `V+1` in the second call to the loop implements the increase of the counter whilst also takes into account possible changes of `V` during the execution of `i`.

(b) Provide the *big-step* semantic description corresponding to instruction *for*.

$$\frac{\langle \texttt{a0} > \texttt{a1}, s\rangle \Rightarrow \mathit{true}}{\langle \texttt{for V := a0 to a1 do i}, s\rangle \Downarrow s}$$

$$\frac{\langle \texttt{a0} > \texttt{a1}, s\rangle \Rightarrow \mathit{false} \quad \langle \texttt{V := a0}, s\rangle \Downarrow s' \quad \langle \texttt{i}, s'\rangle \Downarrow s'' \quad \langle \texttt{for V := V + 1 to a1 do i}, s''\rangle \Downarrow s'''}{\langle \texttt{for V := a0 to a1 do i}, s\rangle \Downarrow s'''}$$

12. We want to extend SIMP with a new instruction *times* with the following syntax:

```
do n times i
```

The instruction works as follows: instruction `i` is executed `n` times if `n` is a positive number; if $\texttt{n} \leq 0$ then `i` is not executed.

(a) Provide the *small-step* semantic description for instruction *times*.

Solution A:

$$\frac{\langle \texttt{n} > 0, s\rangle \Rightarrow \mathit{false}}{\langle \texttt{do n times i}, s\rangle \rightarrow \langle \texttt{skip}, s\rangle}$$

$$\frac{\langle \texttt{n} > 0, s\rangle \Rightarrow \mathit{true}}{\langle \texttt{do n times i}, s\rangle \rightarrow \langle (\texttt{i; do (n} - 1\texttt{) times i}), s\rangle}$$

Solution B:

$$\frac{}{\langle \texttt{do n times i}, s\rangle \rightarrow \langle \texttt{while n} > 0 \texttt{ do (i; n} := \texttt{n} - 1), s\rangle}$$

(b) Provide the *big-step* semantic description for instruction *times*.

$$\frac{\langle \texttt{n} > 0, s\rangle \Rightarrow \mathit{false}}{\langle \texttt{do n times i}, s\rangle \Downarrow s}$$

$$\frac{\langle \texttt{n} > 0, s\rangle \Rightarrow \mathit{true} \qquad \langle \texttt{i}, s\rangle \Downarrow s' \qquad \langle \texttt{do (n} - 1\texttt{) times i}, s'\rangle \Downarrow s''}{\langle \texttt{do n times i}, s\rangle \Downarrow s''}$$

13. Consider the following code S that computes the maximum of two numbers:

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

(a) Obtain the *small-step* computational trace (with the intermediate computation) for the initial state $\{X \mapsto 3, Y \mapsto 5\}$.

(1) $\langle \texttt{if X} > \texttt{Y then max} := \texttt{X else max} := \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \rightarrow$
$\langle \texttt{X} > \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow$
$\langle \texttt{X}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow 3$
$\langle \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow 5$
$3 > 5$ is false
false
(2) $\langle \texttt{max} := \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \rightarrow$
$\langle \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow 5$
(3) $\langle \texttt{skip}, \{X \mapsto 3, Y \mapsto 5, \mathit{max} \mapsto 5\}\rangle$

(b) Obtain the *big-step* computational tree (with the intermediate computation) for the initial state $\{X \mapsto 3, Y \mapsto 5\}$.

$\langle \texttt{if X} > \texttt{Y then max} := \texttt{X else max} := \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle$
$\langle \texttt{X} > \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow$
$\langle \texttt{X}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow 3$
$\langle \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow 5$
$3 > 5$ es false
false
$\langle \texttt{max} := \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle$
$\langle \texttt{Y}, \{X \mapsto 3, Y \mapsto 5\}\rangle \Rightarrow 5$
$\Downarrow \{X \mapsto 3, Y \mapsto 5, \mathit{max} \mapsto 5\}$
$\Downarrow \{X \mapsto 3, Y \mapsto 5, \mathit{max} \mapsto 5\}$

14. We want to extend SIMP with a new multiple assignment operator with syntax

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

where

- variables $\texttt{x}_i$ are distinct
- the $\texttt{a}_i$ are expressions

and the assignment works as follows (see [Gries81], page 121):

- expressions $\mathtt{a}_1, \mathtt{a}_2, \ldots \mathtt{a}_n$ are first evaluated (the order does not matter) to obtain values $\mathtt{v}_1, \mathtt{v}_2, \ldots, \mathtt{v}_n$, respectively;
- for each i, variable $\mathtt{x}_i$ is given value $\mathtt{v}_i$.

(a) Provide the *small-step* semantic description for the *multiple assignment* operator

$$\frac{\langle \mathtt{a}_1, s\rangle \Rightarrow v_1 \qquad \langle \mathtt{a}_2, s\rangle \Rightarrow v_2 \qquad \cdots \qquad \langle \mathtt{a}_n, s\rangle \Rightarrow v_n}{\langle \mathtt{x}_1, \mathtt{x}_2, \ldots \mathtt{x}_n := \mathtt{a}_1, \mathtt{a}_2, \ldots \mathtt{a}_n, s\rangle \rightarrow \langle \mathtt{skip}, s[x_1 \mapsto v_1, x_2 \mapsto v_2, \ldots x_n \mapsto v_n]\rangle}$$

(b) Provide the *big-step* semantic description for the *multiple assignment* operator

$$\frac{\langle \mathtt{a}_1, s\rangle \Rightarrow v_1 \qquad \langle \mathtt{a}_2, s\rangle \Rightarrow v_2 \qquad \cdots \qquad \langle \mathtt{a}_n, s\rangle \Rightarrow v_n}{\langle \mathtt{x}_1, \mathtt{x}_2, \ldots \mathtt{x}_n := \mathtt{a}_1, \mathtt{a}_2, \ldots \mathtt{a}_n, s\rangle \Downarrow s[x_1 \mapsto v_1, x_2 \mapsto v_2, \ldots x_n \mapsto v_n]}$$

15. Consider the following program S:

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

Write the *small-step* execution trace for the initial state $\{x \mapsto 2, y \mapsto 5\}$.

(1) $\langle(\mathtt{t} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{t}), \{x \mapsto 2, y \mapsto 5\}\rangle \rightarrow$
$\quad \langle \mathtt{t} := \mathtt{x}, \{x \mapsto 2, y \mapsto 5\}\rangle \rightarrow$
$\quad\quad \langle \mathtt{x}, \{x \mapsto 2, y \mapsto 5\}\rangle \Rightarrow 2$
$\quad \langle \mathtt{skip}, \{x \mapsto 2, y \mapsto 5, t \mapsto 2\}\rangle$
(2) $\langle(\mathtt{skip}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{t}), \{x \mapsto 2, y \mapsto 5, t \mapsto 2\}\rangle \rightarrow$
(3) $\langle(\mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{t}), \{x \mapsto 2, y \mapsto 5, t \mapsto 2\}\rangle \rightarrow$
$\quad \langle \mathtt{x} := \mathtt{y}, \{x \mapsto 2, y \mapsto 5, t \mapsto 2\}\rangle \rightarrow$
$\quad\quad \langle \mathtt{y}, \{x \mapsto 2, y \mapsto 5, t \mapsto 2\}\rangle \Rightarrow 5$
$\quad \langle \mathtt{skip}, \{x \mapsto 5, y \mapsto 5, t \mapsto 2\}\rangle$
(4) $\langle(\mathtt{skip}; \mathtt{y} := \mathtt{t}), \{x \mapsto 5, y \mapsto 5, t \mapsto 2\}\rangle \rightarrow$
(5) $\langle \mathtt{y} := \mathtt{t}, \{x \mapsto 5, y \mapsto 5, t \mapsto 2\}\rangle \rightarrow$
$\quad \langle \mathtt{t}, \{x \mapsto 5, y \mapsto 5, t \mapsto 2\}\rangle \Rightarrow 2$
(6) $\langle \mathtt{skip}, \{x \mapsto 5, y \mapsto 2, t \mapsto 2\}\rangle$

16. Consider the following program P:

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

Write the *small-step* execution trace for the initial state $\{x \mapsto 3, y \mapsto 7\}$.

(1) $\langle \texttt{x} := \texttt{x} + 1; \texttt{y} := \texttt{y} + \texttt{x}; \texttt{x} := \texttt{x} + 1, \{x \mapsto 3, y \mapsto 7\}\rangle \to$
$\langle \texttt{x} := \texttt{x} + 1, \{x \mapsto 3, y \mapsto 7\}\rangle \to$
$\langle \texttt{x} + 1, \{x \mapsto 3, y \mapsto 7\}\rangle \Rightarrow$
$\langle \texttt{x}, \{x \mapsto 3, y \mapsto 7\}\rangle \Rightarrow 3$
$\langle \texttt{1}, \{x \mapsto 3, y \mapsto 7\}\rangle \Rightarrow 1$
$3 + 1 = 4$ es computado
4
$\langle \texttt{skip}, \{x \mapsto 4, y \mapsto 7\}\rangle$
(2) $\langle \texttt{skip}; \texttt{y} := \texttt{y} + \texttt{x}; \texttt{x} := \texttt{x} + 1, \{x \mapsto 4, y \mapsto 7\}\rangle \to$
(3) $\langle \texttt{y} := \texttt{y} + \texttt{x}; \texttt{x} := \texttt{x} + 1, \{x \mapsto 4, y \mapsto 7\}\rangle$
$\langle \texttt{y} := \texttt{y} + \texttt{x}, \{x \mapsto 4, y \mapsto 7\}\rangle \to$
$\langle \texttt{y} + \texttt{x}, \{x \mapsto 4, y \mapsto 7\}\rangle \Rightarrow$
$\langle \texttt{y}, \{x \mapsto 4, y \mapsto 7\}\rangle \Rightarrow 7$
$\langle \texttt{x}, \{x \mapsto 4, y \mapsto 7\}\rangle \Rightarrow 4$
$7 + 4 = 11$ es computado
11
$\langle \texttt{skip}, \{x \mapsto 4, y \mapsto 11\}\rangle$
(4) $\langle \texttt{skip}; \texttt{x} := \texttt{x} + 1, \{x \mapsto 4, y \mapsto 11\}\rangle \to$
(5) $\langle \texttt{x} := \texttt{x} + 1, \{x \mapsto 4, y \mapsto 11\}\rangle \to$
$\langle \texttt{x} + 1, \{x \mapsto 4, y \mapsto 11\}\rangle \Rightarrow$
$\langle \texttt{x}, \{x \mapsto 4, y \mapsto 11\}\rangle \Rightarrow 4$
$\langle \texttt{1}, \{x \mapsto 4, y \mapsto 11\}\rangle \Rightarrow 1$
$4 + 1 = 5$ es computado
5
(5) $\langle \texttt{skip}, \{x \mapsto 5, y \mapsto 11\}\rangle$

17. Consider the following C code to compute the maximum of two numbers:

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

(a) Write the code of `maximum` using SIMP syntax (where subprograms are not allowed).

Traducimos el cuerpo a la sintaxis SIMP

```
if (x>y)
 then m := x
 else m := y
```

(b) Write the *small-step* and *big-step* execution of `maximum(3,5)` (i.e., for the initial state $\{x \mapsto 3, y \mapsto 5\}$).

Small-step

(1) $\langle \texttt{if (x > y) then m := x else m := y}, \{x \mapsto 3, y \mapsto 5\}\rangle \rightarrow$
$\quad \langle \texttt{x > y}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow$
$\quad\quad \langle \texttt{x}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow 3$
$\quad\quad \langle \texttt{y}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow 5$
$\quad\quad 3 > 5$ es false
$\quad$ *false*

(2) $\langle \texttt{m := y}, \{x \mapsto 3, y \mapsto 5\}\rangle \rightarrow$
$\quad \langle \texttt{y}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow 5$

(3) $\langle \texttt{skip}, \{x \mapsto 3, y \mapsto 5, m \mapsto 5\}\rangle \Rightarrow 5$

Big-step

$\langle \texttt{if (x > y) then m := x else m := y}, \{x \mapsto 3, y \mapsto 5\}\rangle$
$\quad \langle \texttt{x > y}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow$
$\quad\quad \langle \texttt{x}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow 3$
$\quad\quad \langle \texttt{y}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow 5$
$\quad\quad 3 > 5$ es false
$\quad$ *false*
$\quad \langle \texttt{m := y}, \{x \mapsto 3, y \mapsto 5\}\rangle \rightarrow$
$\quad\quad \langle \texttt{y}, \{x \mapsto 3, y \mapsto 5\}\rangle \Rightarrow 5$
$\quad \Downarrow \{x \mapsto 3, y \mapsto 5, m \mapsto 5\}$
$\Downarrow \{x \mapsto 3, y \mapsto 5, m \mapsto 5\}$

18. Compute the weakest precondition $wp(S, Q)$ for program S in question 15 and postcondition Q given by $\texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y}$.

$wp((\texttt{t := x; x := y; y := t}), \texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y}) =$
$wp(\texttt{t := x}, wp(\texttt{x := y}, wp(\texttt{y := t}, \texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y}))) =$ [step 1]
$wp(\texttt{t := x}, wp(\texttt{x := y}, \texttt{x} = \texttt{X} \wedge \texttt{t} = \texttt{Y})) =$ [step 2]
$wp(\texttt{t := x}, \texttt{y} = \texttt{X} \wedge \texttt{t} = \texttt{Y}) =$ [step 3]
$\texttt{y} = \texttt{X} \wedge \texttt{x} = \texttt{Y}$

[step 1] has been performed by using the *assignment* rule to compute the innermost weakest precondition; in particular

$$wp(\texttt{y := t}, \texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y}) = (\texttt{x} = \texttt{X} \wedge \texttt{y} = \texttt{Y})[\texttt{y} \mapsto \texttt{t}] = (\texttt{x} = \texttt{X} \wedge \texttt{t} = \texttt{Y})$$

Similarly, for [step 2] we have:

$$wp(\texttt{x := y}, \texttt{x} = \texttt{X} \wedge \texttt{t} = \texttt{Y}) = (\texttt{x} = \texttt{X} \wedge \texttt{t} = \texttt{Y})[\texttt{x} \mapsto \texttt{y}] = (\texttt{y} = \texttt{X} \wedge \texttt{t} = \texttt{Y})$$

Finally, for [step 3], by using again the assignment rule of the definition of wp, we obtain:

$$wp(\texttt{t := x}, \texttt{y} = \texttt{X} \wedge \texttt{t} = \texttt{Y}) = (\texttt{y} = \texttt{X} \wedge \texttt{t} = \texttt{Y})[\texttt{t} \mapsto \texttt{x}] = (\texttt{y} = \texttt{X} \wedge \texttt{x} = \texttt{Y})$$

According to these results:

(a) What is the functionality implemented by this program? Is there any difference

between program S and the following program S' that uses the multiple assignment introduced in Exercice 14?

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

> Program S' exchanges the values of variables `x` and `y`.
> There are some differences between S and S':
>
> - S uses three variables, whereas S' only requires two variables.
> - When considering the *small-step* operational semantic description of S and S', we notice that the trace for S is *longer* than the trace for S', which actually consists of a single step.

(b) When considering an axiomatic semantics, is there any difference between S and S' as above with regard to postcondition Q?

> If a rule for the computation of *wp* of the multiple assignment operation is provided (similar to the usual assignment), we would obtain $wp(\texttt{x,y := y,x}, Q) = (\texttt{y=X} \wedge \texttt{x=Y})$, meaning that there is no difference in the functionality of S and S'. They can be viewed as equivalent.

(c) Are S and S' equivalent from the operational point of view, or there is a way to distinguish them?

> As discussed above, we can use the small-step semantics to distinguish them.

19. Consider the following program S:

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

Given the precondition $P = (\texttt{x=a} \wedge \texttt{y=b} \wedge \texttt{z=c})$ and the postcondition $Q = (\texttt{x=b} \wedge \texttt{y=a})$, is S correct with respect to P and Q (i.e., correctness of $\{P\}\ S\ \{Q\}$)? Use the weakest precondition calculus to prove it and show the steps of the correctness proof.

> In order to prove $\{P\}\ S\ \{Q\}$ correct, we have to compute $wp(S, Q)$ and then prove $P \Rightarrow wp(S, Q)$.
>
> (a) Computation of $wp(S, Q)$:
>
> $wp(\texttt{(t:=x;x:=y;y:=t)}, \texttt{x=b} \wedge \texttt{y=a}) =$
> $wp(\texttt{t:=x}, wp(\texttt{(x:=y;y:=t)}, \texttt{x=b} \wedge \texttt{y=a})) =$
> $wp(\texttt{t:=x}, wp(\texttt{x:=y}, wp(\texttt{y:=t}, \texttt{x=b} \wedge \texttt{y=a}))) =$
> $wp(\texttt{t:=x}, wp(\texttt{x:=y}, \texttt{x=b} \wedge \texttt{t=a})) =$
> $wp(\texttt{t:=x}, \texttt{y=b} \wedge \texttt{t=a}) =$
> $\texttt{y=b} \wedge \texttt{x=a}$
>
> (b) We easily see that $(\texttt{x=a} \wedge \texttt{y=b} \wedge \texttt{z=c}) \Rightarrow (\texttt{y=b} \wedge \texttt{x=a})$ holds (the equalities in the consequent are present in the antecedent). Thus, S is correct with respect to P and Q.

20. Consider the following program S:

`X:=X-1`

Given the precondition $P = (\texttt{X=1})$ and the postcondition $Q = (\texttt{X} \geq 0)$, is S correct with respect to P and Q?. Use the weakest precondition calculus to prove it and show the steps of the correctness proof.

First we compute $wp(S, Q)$:

$wp(\texttt{X:=X-1}, \texttt{X} \geq 0) = (\texttt{X} - 1 \geq 0) \Leftrightarrow (\texttt{X} \geq 1)$

Now, since $\texttt{X} = 1 \Rightarrow \texttt{X} \geq 1$ clearly holds, correctness of S with respect to P and Q is proved.

21. Compute the weakest precondition of the following programs S for the given postcondition Q:

(a) $S = (\texttt{x:=1})$, $Q = (\texttt{x} = 1)$.

$wp(\texttt{x:=1}, \texttt{x=1}) = (\texttt{1=1}) = \mathit{true}$

(b) $S = (\texttt{x:=y})$, $Q = (\texttt{x} = 0)$.

$wp(\texttt{x:=y}, \texttt{x=0}) = (\texttt{y=0})$

(c) $S = (\texttt{x:=x-1})$, $Q = (\texttt{x} = 0)$.

$wp(\texttt{x:=x-1}, \texttt{x=0}) = (\texttt{x-1=0}) = (\texttt{x=1})$

(d) $S = (\texttt{x:=x-1})$, $Q = (\texttt{y} > 0)$.

$wp(\texttt{x:=x-1}, \texttt{y>0}) = (\texttt{y>0})$

(e) $S = (\texttt{if (x>0) then x:=y else y:=x})$, $Q = (\texttt{x} \geq \texttt{y} \wedge \texttt{y} > 0)$.

$wp(\texttt{if } (\texttt{x} > 0) \texttt{ then x} := \texttt{y else y} := \texttt{x}, (\texttt{x} \geq \texttt{y} \wedge \texttt{y} > 0)) =$
$(\texttt{x} > 0 \wedge wp(\texttt{x} := \texttt{y}, (\texttt{x} \geq \texttt{y} \wedge \texttt{y} > 0))) \vee (\texttt{x} \leq 0 \wedge wp(\texttt{y} := \texttt{x}, (\texttt{x} \geq \texttt{y} \wedge \texttt{y} > 0))) =$
$(\texttt{x} > 0 \wedge \texttt{y} \geq \texttt{y} \wedge \texttt{y} > 0) \vee (\texttt{x} \leq 0 \wedge \texttt{x} \geq \texttt{x} \wedge \texttt{x} > 0) =$
$(\texttt{x} > 0 \wedge \texttt{y} > 0) \vee (\texttt{x} \leq 0 \wedge \texttt{x} > 0) =$
$(\texttt{x} > 0 \wedge \texttt{y} > 0)$

(f) $S = (\texttt{if (x=0) then x:=1 else x:=x+1})$, $Q = (\texttt{x} > \texttt{y} \wedge \texttt{y} \leq 0)$.

$wp(\texttt{if } (\texttt{x} = 0) \texttt{ then x} := 1 \texttt{ else x} := \texttt{x} + 1, (\texttt{x} > \texttt{y} \wedge \texttt{y} \leq 0)) =$
$(\texttt{x} = 0 \wedge wp(\texttt{x} := 1, (\texttt{x} > \texttt{y} \wedge \texttt{y} \leq 0))) \vee (\texttt{x} \neq 0 \wedge wp(\texttt{x} := \texttt{x} + 1, (\texttt{x} > \texttt{y} \wedge \texttt{y} \leq 0))) =$
$(\texttt{x} = 0 \wedge 1 > \texttt{y} \wedge \texttt{y} \leq 0) \vee (\texttt{x} \neq 0 \wedge \texttt{x} + 1 > \texttt{y} \wedge \texttt{y} \leq 0) =$
$(\texttt{x} = 0 \wedge \texttt{y} \leq 0) \vee (\texttt{x} \neq 0 \wedge \texttt{x} > \texttt{y} - 1 \wedge \texttt{y} \leq 0) =$
$\texttt{y} \leq 0 \wedge (\texttt{x} = 0 \vee (\texttt{x} \neq 0 \wedge \texttt{x} > \texttt{y} - 1)) =$
$\texttt{y} \leq 0 \wedge (\texttt{x} = 0 \vee (\texttt{x} \neq 0 \wedge \texttt{x} \geq \texttt{y}))$

(g) $S = (\texttt{x:=y; y:=5})$, $Q = (\texttt{x} > 0)$.

$wp(\texttt{x} := \texttt{y}; \texttt{y} := 5, \texttt{x} > 0) =$
$wp(\texttt{x} := \texttt{y}, wp(\texttt{y} := 5, \texttt{x} > 0)) =$
$wp(\texttt{x} := \texttt{y}, \texttt{x} > 0) = (\texttt{y} > 0)$

(h) $S = (\texttt{x:=x+1; if (x>0) then x:=y else y:=x})$, $Q = (\texttt{x} > \texttt{z})$.

$wp((\texttt{x} := \texttt{x} + 1; \texttt{if } (\texttt{x} > 0) \texttt{ then x} := \texttt{y else y} := \texttt{x}), \texttt{x} > \texttt{z}) =$
$wp(\texttt{x} := \texttt{x} + 1, wp(\texttt{if } (\texttt{x} > 0) \texttt{ then x} := \texttt{y else y} := \texttt{x}, \texttt{x} > \texttt{z})) =$
$wp(\texttt{x} := \texttt{x} + 1, ((\texttt{x} > 0 \wedge wp(\texttt{x} := \texttt{y}, \texttt{x} > \texttt{z})) \vee (\texttt{x} \leq 0 \wedge wp(\texttt{y} := \texttt{x}, \texttt{x} > \texttt{z}))) =$
$wp(\texttt{x} := \texttt{x} + 1, ((\texttt{x} > 0 \wedge \texttt{y} > \texttt{z}) \vee (\texttt{x} \leq 0 \wedge \texttt{x} > \texttt{z})) =$
$(\texttt{x} + 1 > 0 \wedge \texttt{y} > \texttt{z}) \vee (\texttt{x} + 1 \leq 0 \wedge \texttt{x} + 1 > \texttt{z}) =$
$(\texttt{x} \geq 0 \wedge \texttt{y} > \texttt{z}) \vee (\texttt{x} < 0 \wedge \texttt{x} \geq \texttt{z})$

22. B
23. A
24. B
25. A
26. C
27. A
28. D
29. C
30. A
31. A
32. A
33. B
34. C
35. C
36. D
37. B
38. D
39. A

A C-Minus BNF grammar

```
<ID>                 ::= <letter><letter>*
<NUM>                ::= <digit><digit>*
<letter>             ::= a | ...| z | A | ...| Z
<digit>              ::= 0 | ...| 9
<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> ) <compound-stmt>
<params>             ::= <param-list> | void
<param-list>         ::= <param-list> , <param> | <param>
<param>              ::= <type-specifier> <ID> | <type-specifier> <ID> [ ]
<compound-stmt>      ::= { <local-declarations> <statement-list> }
<local-declarations> ::= <local-declarations> <var-declaration> | empty
<statement-list>     ::= <statement-list> <statement> | empty
<statement>          ::= <expression-stmt> | <compound-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>
```