Errata

Transitions and trees
Structural operational semantics of programming languages

Hans Hüttel
Aalborg University, Denmark

Updated 9th December 2010
Acknowledgements

The following readers have alerted me to the typos that this errata sheet is intended to correct: Sabrine Mouritsen, Anders Franz Terkelsen, Bo Andersen, Dior Christensen, and Laurence Day.

List of errata

p. 19 The text reads:

Two sets $A$ and $B$ are equal if they contain the same elements, that is, $x \in A$ if and only if $x \in B$. Consequently, $A = B$ if and only if $A \subseteq B$ and $A \subseteq B$.

This should be

Two sets $A$ and $B$ are equal if they contain the same elements, that is, $x \in A$ if and only if $x \in B$. Consequently, $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.

p. 30 In the third item in Section 3.1.2,

In statements we assume that the semicolon operator is left-associative. So $S_1; S_2; S_3$ is to be read as $(S_1; (S_2; S_3))$, the statement whose immediate constituents are $S_1$ and $S_2; S_3$.

should read

In statements we assume that the semicolon operator is right-associative. So $S_1; S_2; S_3$ is to be read as $(S_1; (S_2; S_3))$, the statement whose immediate constituents are $S_1$ and $S_2; S_3$.

p. 61 In Problem 4.18, two arrows are missing. The problem should read

**Problem 4.18** Prove, using a suitable proof technique, that the big-step semantics of statements is deterministic, that is, that for any statement $S$ and state $s$ we have that if $\langle S, s \rangle \rightarrow s'$ and $\langle S, s \rangle \rightarrow s''$ then $s' = s''$. (You may assume that the big-step semantics of arithmetic and Boolean expressions are deterministic.)

p. 88 A semicolon is missing after the second variable declaration in the statement in Figure 6.2, which should be
begin
  var x := 0;
  var y := 42;

  proc p is x := x + 3;
  proc q is call p;

  begin
    var x := 9;
    proc p is x := x + 1;
    call q;
    y := x
  end

end

p. 98 In Figure 7.1, another semicolon is missing. The statement should be
begin
    var y := 0;
    var x := 1;

    proc f(var x) is
    begin
        var z := x - 1;
        y := y * x;
        if x > 1 then
            call f(z)
        else
            skip
        end
    end
    y := 4;
    call f(y);
    z := y
end

p. 99 In Table 7.3, a dash is missing in the last side condition. It should read as follows:

\[
\text{[CALL-R-RECS]} \quad \begin{array}{l}
\quad \text{env}_{V} \cdot [x \mapsto l][\text{next} \mapsto l'], \text{env}_{P} \vdash (S, sto) \rightarrow sto' \\
\quad \text{env}_{V}, \text{env}_{P} \vdash (\text{call } p(y), sto) \rightarrow sto'
\end{array}
\]

where \( \text{env}_{P} p = (S, x, env'_{V}, env'_{P}), env_{V} y = l \) and \( l' = env_{V} \text{next} \) and \( env'_{P} = env'_{P}[p \mapsto (S, x, env'_{V}, env'_{P})] \)

p. 107 Here, the first example has a superfluous semicolon and should read
begin

var y := 3;
var z := 2;

proc q (name x) is
begin
    y := x + 2
end;

z := (z+x)*y;
call q(z)
end

p. 108 The statement in Figure 7.3 has some incorrect semicolons and should read

begin

var y:=2;
proc p(name x) is

begin
    var y := 3;
    var z := 2;

    proc q(name x) is begin y := x+2 end;

    z := (z+x)*y;
call q(z)

end;
call p(y+4)
end

p. 113, line 2 ‘Concurrnet’ should read ‘Concurrent’.
p. 249 In the last line of the page,

\[ DP \in DecP \]

should read

\[ DP \in DecP \]