**COL226: Programming Languages**

II semester 2015-16

Fri 06 May 2016 Major 120 minutes Max Marks 70

Answer only in the space provided on the question paper.

---

1. [**8 marks**] Recall the following definitions of `map` and `foldr` in ML-like languages.

```
fun map f [] = []
  | map f (h::t) = (f h)::(map f t)
fun foldr f e [] = e
  | foldr f e (h::t) = f (h, foldr f e t)
```

Consider the function

```
fun mapr g L = foldr (fn (x,y) => (g x) :: y) nil L
```

Prove that for all lists `L: 'a list` and functions `g: 'a -> 'a`

```
mapr g L = map g L
```

**Solution.**

*Proof:* Let `f` = `fn (x, y) => (g x)::y`

**Basis.** `mapr g []` = `foldr f nil []` = `nil` = `map g []`

**Induction Hypothesis** ($IH$)**.**

| *For all lists* `L` *of length less than* $n > 0$, `mapr g L = map g L` |
|---|

**Induction Step.** Let `L = z::t` be a list of length $n > 0$. Then

$$
\begin{array}{rcl}
 & = & \texttt{mapr g (z :: t)} \\
 & = & \texttt{foldr f nil (z :: t)} \\
 & = & \texttt{foldr (fn(x, y) => (g x) :: y) nil (z :: t)} \\
 & = & \texttt{(fn(x, y) => (g x) :: y) (z, foldr f nil t)} \\
\{\textit{Ind.hyp.}\} & = & \texttt{(fn(x, y) => (g x) :: y) (z, map g t)} \\
 & = & \texttt{(g z) :: (map g t)} \\
 & = & \texttt{map g (z :: t)}
\end{array}
$$

QED

2. [**5 + 6 = 11 marks**] Each comment in the C language

   - begins with the characters "//" and ends with the newline character, or
   - begins with the characters "/*" and ends with "*/" and may run across several lines.

   If the character sequences "//", "/*" and "*/" are allowed to appear in 'quoted' form as "'//'", "'/*'" and "'*/'" respectively within a C comment, then give

   (a) a regular expression for the modified C comments and

   (b) a corresponding DFA for modified C comments

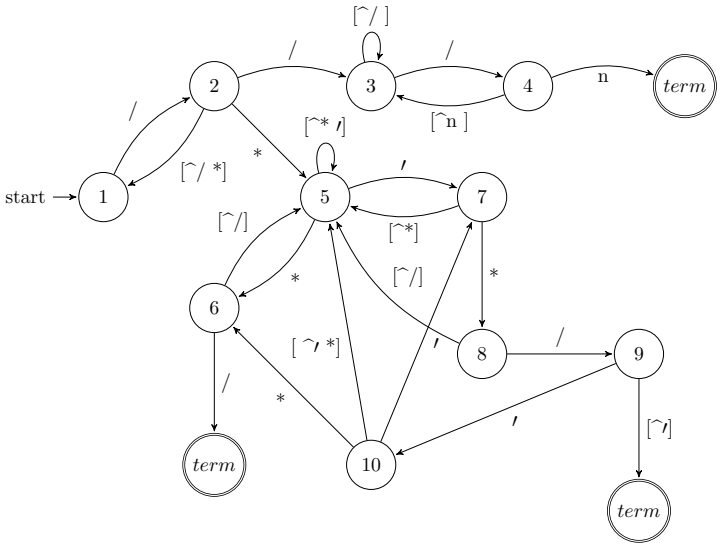
**Solution.** Here is a classic case of what reality some times looks like!

# Solution 2

May 13, 2016

a) r = (// [^(\n)] \n) | (/*([^*]* *+ [^/]*)|[^*]*/+([^*]*| *+ ([^/]*|(/')+))/)

b)DFA






3. **[10 marks]** Recall that the Church numerals are defined as follows:

$$\underline{0} \stackrel{df}{=} \lambda f\ x[x], \quad \underline{1} \stackrel{df}{=} \lambda f\ x[(f\ x)], \quad \underline{2} \stackrel{df}{=} \lambda f\ x[(f\ (f\ x))], \cdots, \underline{n+1} \stackrel{df}{=} \lambda f\ x[(f^{n+1}\ x)], \cdots$$

and the addition operation on them is defined as

$$\mathsf{add} \stackrel{df}{=} \lambda m\ n\ f\ x[((m\ f)\ (n\ f\ x))]$$

Prove that $\mathsf{add}\ u\ v =_\beta \mathsf{add}\ v\ u$ for all Church numerals $u$ and $v$.

**Solution.**

Let $u \equiv \underline{i} \equiv_\alpha \lambda g\ y[(g^i\ y)]$ and $v \equiv \underline{j} \equiv_\alpha \lambda h\ z[(h^j\ z)]$ be any Church numerals representing the natural numbers $i, j \geq 0$ respectively. We then have

$$\begin{aligned}
& (\mathsf{add}\ u\ v) \\
\longrightarrow_\beta^* \quad & \lambda f\ x[((u\ f)\ (v\ f\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(\lambda y[(f^i\ y)]\ (\lambda z[(f^j z)]\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(\lambda y[(f^i\ y)]\ (f^j\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(f^i\ (f^j\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(f^{i+j}\ x)] \qquad (1)
\end{aligned}$$

Similarly we have

$$\begin{aligned}
& (\mathsf{add}\ v\ u) \\
\longrightarrow_\beta^* \quad & \lambda f\ x[((v\ f)\ (u\ f\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(\lambda z[(f^j\ z)]\ (\lambda y[(f^i y)]\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(\lambda z[(f^j\ z)]\ (f^i\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(f^j\ (f^i\ x))] \\
\longrightarrow_\beta^* \quad & \lambda f\ x[(f^{j+i}\ x)] \qquad (2)
\end{aligned}$$

From (**??**) and (**??**) it follows that $\mathsf{add}\ u\ v =_\beta \mathsf{add}\ v\ u$.

4. **[6 + 6 = 12 marks]** Consider the term algebra defined by the following grammar

$$t, u ::= \varepsilon \mid a \mid b \mid t.u$$

where "." is an associative binary operator

$$\begin{aligned}
a.b \quad & \longrightarrow_\delta \quad \varepsilon \qquad (3) \\
\varepsilon.t \quad & \longrightarrow_\delta \quad t \qquad (4) \\
t.\varepsilon \quad & \longrightarrow_\delta \quad t \qquad (5)
\end{aligned}$$

(a) Give rules for the compatible closure of the rewrite rules.

(b) Prove that $t \longrightarrow_\delta^* \varepsilon$ if and only if the following conditions are satisfied

i. $\#a(t) = \#b(t)$ and

ii. $\#a(s) \geq \#b(s)$ for every prefix $s \preceq t$

where $\#a(t)$ denotes the number of occurrences of $a$ in $t$.

**Solution.**

(a) The compatible closure defines a 1-step $\delta$-reduction

$$\delta_1\ \frac{t \longrightarrow_\delta u}{t \longrightarrow_{\delta_1} u} \quad \delta_1\mathbf{L}\ \frac{t \longrightarrow_{\delta_1} t'}{t.u \longrightarrow_{\delta_1} t'.u} \quad \delta_1\mathbf{R}\ \frac{u \longrightarrow_{\delta_1} u'}{t.u \longrightarrow_{\delta_1} t.u'}$$

(b) The terms of the algebra may be regarded as non-empty strings. It is useful to use a separate symbol $\epsilon$ to denote the empty string since $\varepsilon$ already denotes a constant symbol in the algebra. Let $\#t = \#a(t) + \#b(t) + \#\varepsilon(t)$. The following claim then follows from the $\delta$-rules and the compatibility rules.

**Claim 0.1** *i.* $t \longrightarrow_\delta^1 t'$ *implies* $\#t = \#t' + 1$

**Lemma 0.2** *$t \longrightarrow_\delta^* \varepsilon$ iff one (or more) of the following holds.*
*Case (0) $t \equiv \varepsilon$* *Case (1) $t \equiv u.\varepsilon.v$* *Case (2) $t \equiv u.a.v.b.w$*
*where $u \neq \epsilon \Rightarrow u \longrightarrow_\delta^* \varepsilon$, $v \neq \epsilon \Rightarrow v \longrightarrow_\delta^* \varepsilon$ and $w \neq \epsilon \Rightarrow w \longrightarrow_\delta^* \varepsilon$.*

*Proof:* We proceed by induction on $\#t$. Notice that for any actual term $t$, $\#t > 0$.

**Basis.** $\#t = 1$. Then $t \longrightarrow_\delta^* \varepsilon$ iff $t \equiv \varepsilon$ which is merely case (1) with $u = \epsilon = v$.
**Induction Hypothesis ($IH$).**

> *For any term $s$ with $1 \leq \#s < n$, $s \longrightarrow_\delta^* \varepsilon$ iff one (or more) of the cases (0-2) holds.*

**Induction Step.**

Let $\#t = n > 1$. Then $t \longrightarrow_\delta^* \varepsilon$ iff for some term $t_1$, $t \longrightarrow_\delta^1 t_1 \longrightarrow_\delta^* \varepsilon$ where $t_1$ satisfies the induction hypothesis. If $t \longrightarrow_\delta^1 t_1$ because of an application of either rule (**??**) or (**??**), then *Case (1)* applies. If $t \longrightarrow_\delta^1 t_1$ because of an application of rule (**??**) then *Case (2)* applies. QED

**Corollary 0.3** *$t \longrightarrow_\delta^* \varepsilon$ implies $\#a(t) = \#b(t) \wedge \forall s \preceq t[\#a(s) \geq \#b(s)]$*

**Lemma 0.4** *$\#a(t) = \#b(t) \wedge \forall s \preceq t[\#a(s) \geq \#b(s)]$ implies $t \longrightarrow_\delta^* \varepsilon$ for each term $t$.*

*Proof:* We proceed by induction on $\#a(t)$ assuming that $\#a(t) = \#b(t) \wedge \forall s \preceq t[\#a(s) \geq \#b(s)]$.

**Basis.** $\#a(t) = \#b(t) = 0$. then clearly $t \equiv \varepsilon^k$, for some $k \geq 1$. By applying rules (**??**) or (**??**), $k - 1$ times the required result follows.
**Induction Hypothesis ($IH$).**

> *For every term $u$, $0 \leq \#a(u) = \#b(u) < n \wedge \forall s \preceq u[\#a(s) \geq \#b(s)]$ implies $u \longrightarrow_\delta^* \varepsilon$*

**Induction Step.** Assume $\#a(t) = \#b(t) = n > 0$. Then there exists a left-most occurrence of $b$ in the string $t$, $t \equiv x.b.y$ for some terms $x$ and $y$ ($y$ could be empty, but $x$ is not). Since $x.b \preceq t$ we have $\#a(x) = \#a(x.b) \geq \#b(x.b) = 1$ and $\#b(x) = 0$. Consider the right-most occurrence of $a$ in $x$. We then have for some terms $u$, $v$ (either or both of which could be empty) $x = u.a.v$ such that $\#a(v) = 0$. Since $\#b(x) = 0$ we also have $\#b(v) = 0$ which implies $v = \varepsilon^i$ for some $i \geq 0$. Hence $t \equiv u.a.\varepsilon^i.b.y \longrightarrow_\delta^i u.a.b.y \longrightarrow_\delta^1 u.y \equiv t'$, where $\#a(t') = \#a(t) - 1 = \#b(t) - 1 = \#b(t')$. By the induction hypothesis $t' \longrightarrow_\delta^* \varepsilon$. Hence $t \longrightarrow_\delta^* \varepsilon$.

QED

5. **[5+5+5=15 marks]** The Python language allows a "multiple-assignment" command. For example the following two programs (which compute the largest fibonacci number under 100) are equivalent in Python.

```
#!/usr/bin/python          |        #!/usr/bin/python
a,b = 0,1                  |        a,b = 0,1
while b < 100:             |        while b < 100:
    a,b = b, a+b           |            b,a = a+b,b
```

Assume that in addition to the simple assignment command, the pure WHILE language also has the following multiple-assignment command.

```
x1, x2 := e1, e2
```

(a) Define an operational rule of inference for the above command.

(b) From your operational rule prove that the following two programs are equivalent whenever `x1` and `x2` are distinct.

```
x1, x2 := e1, e2          |        x2, x1 := e2, e1
```

(c) Give an example in the WHILE language to show that the following three programs all yield different final states, even if they all begin execution from the same initial state, even if `x1` and `x2` are distinct.

```
x1, x2 := e1, e2  | x1:= e1; x2 := e2  | x2:= e2; x1 := e1
```

**Solution.**

(a) Assume in the following that `m1` and `m2` are normal forms of values of the appropriate type compatible with the types of variables `x1` and `x2`.

Then **Assgn2** is the required rule of inference.

$$\textbf{Assgn2}\ \frac{\begin{array}{c}\langle\sigma, \texttt{e1}\rangle \longrightarrow_e^* \langle\sigma, \texttt{m1}\rangle \\ \langle\sigma, \texttt{e2}\rangle \longrightarrow_e^* \langle\sigma, \texttt{m2}\rangle\end{array}}{\gamma \vdash \langle\sigma, \texttt{x1}, \texttt{x2} := \texttt{e1}, \texttt{e2}\rangle \longrightarrow_c^* [\gamma(x2) \mapsto \texttt{m2}][\gamma(x1) \mapsto \texttt{m1}]\sigma}$$

**Assgn2** may be derived from the following three rules which outline exactly how the evaluation of the expressions and the subsequent change of store takes place.

$$\textbf{Assgn2.0}\ \frac{}{\gamma \vdash \langle\sigma, \texttt{x1}, \texttt{x2} := \texttt{m1}, \texttt{m2}\rangle \ \longrightarrow_c^1\ [\gamma(x2) \mapsto \texttt{m2}][\gamma(x1) \mapsto \texttt{m1}]\sigma}$$

$$\textbf{Assgn2.1}\ \frac{\gamma \vdash \langle\sigma, \texttt{e2}\rangle \ \longrightarrow_e\ \langle\sigma, \texttt{e2}'\rangle}{\gamma \vdash \langle\sigma, \texttt{x1}, \texttt{x2} := \texttt{m1}, \texttt{e2}\rangle \ \longrightarrow_c^1\ \langle\sigma, \texttt{x1}, \texttt{x2} := \texttt{m1}, \texttt{e2}'\rangle}$$

$$\textbf{Assgn2.2}\ \frac{\gamma \vdash \langle\sigma, \texttt{e1}\rangle \ \longrightarrow_e\ \langle\sigma, \texttt{e1}'\rangle}{\gamma \vdash \langle\sigma, \texttt{x1}, \texttt{x2} := \texttt{e1}, \texttt{e2}\rangle \ \longrightarrow_c^1\ \langle\sigma, \texttt{x1}, \texttt{x2} := \texttt{e1}', \texttt{e2}\rangle}$$

- The two expressions `e1` and `e2` are evaluated in order. But the values stored in the locations of the variables `x1` and `x2` does not change until both expressions `e1` and `e2` have been completely evaluated.
- It is possible to derive the rule **Assgn2** by induction on the sum of the number of steps it takes to evaluate expressions `e1` and `e2` in sequence using the rules **Assgn2.2**, **Assgn2.1** and **Assgn2.0**.

(b) Since `x1` and `x2` are distinct variables (they are not aliases of each other) and hence in any environment $\gamma$, $\gamma(\texttt{x1}) \neq \gamma(\texttt{x2})$. It follows that

$$[\gamma(\texttt{x2}) \mapsto \texttt{m2}][\gamma(\texttt{x1}) \mapsto \texttt{m1}]\sigma = [\gamma(\texttt{x1}) \mapsto \texttt{m1}][\gamma(\texttt{x2}) \mapsto \texttt{m2}]\sigma \tag{6}$$

Applying rule **Assgn2** to the programs

```
x1, x2 := e1, e2          |          x2, x1 := e2, e1
```

and by the identity (**??**) we see that the two programs are equivalent.

(c) Let $\sigma_0$ be the initial state with $\sigma_0(\gamma(\texttt{x1})) = 1$ and $\sigma_0(\gamma(\texttt{x2})) = 2$. Let $\texttt{e1} \equiv \texttt{x1} + \texttt{x2} + 10$ and $\texttt{e2} \equiv \texttt{x1} + \texttt{x2} + 20$. Then it is easy to see that the final states $\sigma_f$ are given by the following table.

| $\sigma_0$ | | | $\sigma_f$ | |
|---|---|---|---|---|
| $\gamma$(x1) | $\gamma$(x2) | Program | $\gamma$(x1) | $\gamma$(x2) |
| 1 | 2 | x1, x2 := x1+x2+10, x1+x2+20 | 13 | 23 |
| 1 | 2 | x1 := x1+x2+10; x2 := x1+x2+20 | 13 | 35 |
| 1 | 2 | x2 := x1+x2+20; x1 := x1+x2+10 | 34 | 23 |

6. **[14 marks]**

Using the type rules

$$\textbf{Var}\ \frac{}{\Gamma \vdash x : \Gamma(x)\ \ \rhd_{\emptyset}\ \ \emptyset}$$

$$\textbf{Abs}\ \frac{\Gamma, x : \sigma \vdash L : \tau\ \ \rhd_T\ \ C}{\Gamma \vdash \lambda x[L] : \sigma{\rightarrow}\tau\ \ \rhd_T\ \ C}$$

$$\textbf{App}\ \frac{\begin{array}{c}\Gamma \vdash L : \sigma\ \ \rhd_{T_1}\ \ C_1 \\ \Gamma \vdash M : \tau\ \ \rhd_{T_2}\ \ C_2\end{array}}{\Gamma \vdash (L\ M) : {'\mathtt{a}}\ \ \rhd_{T'}\ \ C'}\ \ (\textbf{Conditions 1. and 2.})$$

where

- **Condition 1.** $T_1 \cap T_2 = T_1 \cap TVar(\tau) = T_2 \cap TVar(\sigma) = \emptyset$
  **Condition 2.** ${'\mathtt{a}} \notin T_1 \cup T_2 \cup TVar(\sigma) \cup TVar(\tau) \cup TVar(C_1) \cup TVar(C_2)$.
- $T' = T_1 \cup T_2 \cup \{{'\mathtt{a}}\}$
- $C' = C_1 \cup C_2 \cup \{\sigma = \tau{\rightarrow}{'\mathtt{a}}\}$

give a formal proof of the principal type of the combinator $\mathsf{S} \stackrel{df}{=} \lambda x\ y\ z[((x\ z)\ (y\ z))]$.

**Solution.**

1. $x : {'\mathtt{a}}, y : {'\mathtt{b}}, z : {'\mathtt{c}} \vdash x : {'\mathtt{a}}\ \ \rhd_{\emptyset}\ \ \emptyset$ (**Var**)
2. $x : {'\mathtt{a}}, y : {'\mathtt{b}}, z : {'\mathtt{c}} \vdash y : {'\mathtt{b}}\ \ \rhd_{\emptyset}\ \ \emptyset$ (**Var**)
3. $x : {'\mathtt{a}}, y : {'\mathtt{b}}, z : {'\mathtt{c}} \vdash z : {'\mathtt{c}}\ \ \rhd_{\emptyset}\ \ \emptyset$ (**Var**)
4. $x : {'\mathtt{a}}, y : {'\mathtt{b}}, z : {'\mathtt{c}} \vdash (y\ z) : {'\mathtt{d}}\ \ \rhd_{\{'\mathtt{d}\}}\ \ \{{'\mathtt{b}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{d}}\}$ (**App**)
5. $x : {'\mathtt{a}}, y : {'\mathtt{b}}, z : {'\mathtt{c}} \vdash (x\ z) : {'\mathtt{e}}\ \ \rhd_{\{'\mathtt{d},'\mathtt{e}\}}\ \ \{{'\mathtt{b}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{d}}, {'\mathtt{a}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{e}}\}$ (**App**)
6. $x : {'\mathtt{a}}, y : {'\mathtt{b}}, z : {'\mathtt{c}} \vdash ((x\ z)\ (y\ z)) : {'\mathtt{f}}\ \ \rhd_{\{'\mathtt{d},'\mathtt{e},'\mathtt{f}\}}\ \ \{{'\mathtt{b}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{d}}, {'\mathtt{a}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{e}}, {'\mathtt{e}} = {'\mathtt{d}}{\rightarrow}{'\mathtt{f}}\}$ (**App**)
7. $x : {'\mathtt{a}}, y : {'\mathtt{b}} \vdash \lambda z[((x\ z)\ (y\ z))] : {'\mathtt{c}}{\rightarrow}{'\mathtt{f}}\ \ \rhd_{\{'\mathtt{d},'\mathtt{e},'\mathtt{f}\}}\ \ \{{'\mathtt{b}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{d}}, {'\mathtt{a}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{e}}, {'\mathtt{e}} = {'\mathtt{d}}{\rightarrow}{'\mathtt{f}}\}$ (**Abs**)
8. $x : {'\mathtt{a}} \vdash \lambda y\ z[((x\ z)\ (y\ z))] : {'\mathtt{b}}{\rightarrow}{'\mathtt{c}}{\rightarrow}{'\mathtt{f}}\ \ \rhd_{\{'\mathtt{d},'\mathtt{e},'\mathtt{f}\}}\ \ \{{'\mathtt{b}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{d}}, {'\mathtt{a}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{e}}, {'\mathtt{e}} = {'\mathtt{d}}{\rightarrow}{'\mathtt{f}}\}$ (**Abs**)
9. $\vdash \lambda x\ y\ z[((x\ z)\ (y\ z))] : {'\mathtt{a}}{\rightarrow}{'\mathtt{b}}{\rightarrow}{'\mathtt{c}}{\rightarrow}{'\mathtt{f}}\ \ \rhd_{\{'\mathtt{d},'\mathtt{e},'\mathtt{f}\}}\ \ \{{'\mathtt{b}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{d}}, {'\mathtt{a}} = {'\mathtt{c}}{\rightarrow}{'\mathtt{e}}, {'\mathtt{e}} = {'\mathtt{d}}{\rightarrow}{'\mathtt{f}}\}$ (**Abs**)

Substituting from the constraints for ${'\mathtt{a}}$, ${'\mathtt{b}}$ and ${'\mathtt{e}}$ we get

$$\mathsf{S} : ({'\mathtt{c}}{\rightarrow}{'\mathtt{d}}{\rightarrow}{'\mathtt{f}}){\rightarrow}({'\mathtt{c}}{\rightarrow}{'\mathtt{d}}){\rightarrow}{'\mathtt{c}}{\rightarrow}{'\mathtt{f}}$$

which is a principal type scheme for $\mathsf{S}$.