Major Quiz

Graded

Student

Abhinav Shripad

Total Points

25.5 / 40 pts

Question 1

Elaboration in Prolog

10 / 10 pts

+ 0 pts Not answered

✔ **+ 2 pts** Simple Defn Correct

✔ **+ 2 pts** Sequential Correct

✔ **+ 3 pts** Parallel Correct

✔ **+ 3 pts** Local Correct

− 3 pts Cumulative table instead of extension

+ 0 pts Incorrect

+ 0 pts Model Answer
The question clearly asks for an *incremental* table as the result of elaboration.
You were also expected to use the concept as discussed in class, not from some previous course,
Or stuff off the web. As many similar answers have cropped up.

```
elaborate(G, def(X,E), [(X,A)]) :- calculate(G, E, A).
elaborate(G, seq(D1,D2), Gres) :-  elaborate(G, D1, G1), append(G,G1,G2), elaborate(G2, D2, G3), append(G1, (
```

```
elaborate(G, par(D1,D2), Gres) :-  elaborate(G, D1, G1), elaborate(G, D2, G2), append(G1, G2, Gres) .
```
% (appe
correctly acts as union here)

```
elaborate(G, local(D1,D2), Gres) :-  elaborate(G, D1, G1), append(G,G1,G2), elaborate(G2, D2, Gres).
```

Question 2

Combinatory Logic

4 / 4 pts

✔ **+ 2 pts** LHS = S K K P = K P (K P) = P

✔ **+ 1 pt** RHS = I P = P

✔ **+ 1 pt** for arbitrary P, LHS = RHS

+ 0 pts Missing/Incorrect

Question 3

Type Inference

3 / 12 pts

+ 0 pts Missing/Incorrect

+ 4 pts First part is fully correct. Of the following format:
('a * 'b -> 'c) -> ('a list)->('b list)->(c' list)
no partial to be awarded.

+ 4 pts Type of l1 and l2 and return type found from first match case or last match case

+ 4 pts Type of f found from last match case

+ 0 pts Incorrect

+ 3 pts Partial marks awarded. How do you know that l1 is a list?

Question 4

Operational Semantics of Commands

6 / 8 pts

+ 0 pts Incorrect/Not attempted

✔ **+ 4 pts** Iterative case:
g-[c]->g1, g1 | -e :false, g1-[repeat c until e]->g'
g-[repeat c until e]->g'

✔ **+ 4 pts** Termination case:
g-[c]->g', g' | -e :true
g-[repeat c until e]->g'

✔ **− 2 pts** Terminated on e = false

− 3 pts Did not execute c first

+ 0 pts Incorrect

Question 5

Hoare Logic

In Review **2.5 / 6 pts**

+ 0 pts Incorrect/Not attempted/Incomplete

+ 3 pts Correct proof for {p} skip;c {q} => {p} c {q} OR
{p} c;skip {q} => {p} c {q}
Explicit mention of hoare logic rules used in the proof

✔ **+ 2 pts** Correct proof for {p} c {q} => {p} skip;c {q} OR
{p} c {q} => {p} c;skip {q}
Explicit mention of hoare logic rules used in the proof

✔ **+ 1 pt** {Either clear description of how the proofs for {p} c;skip {q} would map to the proof already done for {p} skip;c {q}
Or explicit proof.

+ 0 pts Model Answer

(a-I) |- {p}c{q} => |- {p} skip; c {q}
Assume {p}c {q}. (1)
By HSKIP |- {p} c {p} (2)
By HSEQ , we can deduce |- {p} skip; c {q} (3)

(a-II) |- {p} skip; c {q} => |- {p}c{q}
Assume |- {p} skip; c {q} (3)
By HSEQ (upwards), for some r: |- {p} skip {r} (4) and |- {r} c {q} (5)
By HSKIP |- {p} skip {p} (2),
So if |- {p} skip {r} (4), this is only if |- p -> r (6) and using the HCONSEQUENCE rule on (2) and (6)
But from (2) and (6) using the HCONSEQUENCE rule we can deduce: |- {p} c {q} (1)

(b-I) |- {p}c{q} => |- {p} c; skip {q}
Similar to (a-I) except we will use
HSKIP {q} skip {q} (7)
and HSEQ on (1) and (7) to deduce
|- {p} c; skip {q} (8)

(b-II) |- {p} c; skip{q} => |- {p}c{q}
Similar to (a-II) except we will use |- r-> q (9) from
analysis on |- {r} skip {q}
and the HCONSEQUENCE rule on |- {p}c{r} and (9) to prove |- {p} c {q} (1)

+ 0 pts Incorrect

– 0.5 pts ??? (1) |-hl {p} skip; c {r} -- Why r???
Similarly part is not correct.

Regrade Request **Submitted on: May 13**

Sir in the second last line, there is a typing mistake from my side of using r in place of q. Other than that the 2 proofs are correct, I have even mentioned the rules.

In first part I have shown {p} skip {p} , and {p} c {q} as given in problem, then I have also mentioned that using HSeq, we get {p} skip; c {q},

In second part I have written {p} c {q} as given in problem and {q} skip {q} using HSkip, now combining these 2 using HSeq, we get {p} skip; c {q}. The only mistake I have made is mistyping r in place of q in the last line.

Edit Request

Q1 Elaboration in Prolog

10 Points

Recall that definitions:

$$d \in Defs \ ::= \ \mathbf{def}\ x = e \ \mid \ d_1; d_2 \ \mid \ d_1 || d_2 \ \mid \ \mathbf{local}\ d_1\ \mathbf{in}\ d_2\ \mathbf{ni}$$

are *elaborated* in the context of a given table γ to yield an incremental table γ'. Elaboration $\gamma \vdash d \ggg \gamma'$ is specified inductively according to the rules given in class.

Encode this big-step elaboration as a predicate $\texttt{elaborate}(G, D, G1)$ in Prolog, assuming tables are represented as lists of "variable-answer" pairs. You may assume a Prolog predicate $\texttt{calculate}(G, E, A)$ for calculating the result of an expression (wrt a given table).
You may also assume that in $d_1||d_2$, $dv(d_1) \cap dv(d_2) = \{\}$.

```
elaborate(G, def(X, E), G1) :- calculate(G, E, A), update_table(G, X, A, G1). /// def x
= e
elaborate(G, sequence(D1,D2), G2) :- elaborate(G, D1, G1), elaborate(G1, D2,
G2). /// d1;d2
elaborate(G, local(D1) in D2, G2) :-elaborate(G, D1, G1),elaborate(G1, D2, G2). ///
local d1 in d2 , same as sequential
elaborate(G, D1 || D2, G3) :- elaborate(G, D1, G1), elaborate(G, D2,
G2),merge_tables(G1, G2, G3). ///d1||d2

merge_tables([], G, G):-!.
merge_tables([(X, A)|T], G, [(X, A)|T2]) :- \+ member((X, _), G), merge_tables(T, G,
T2).
merge_tables([(X, _)|T], G, T2) :-member((X, _), G), merge_tables(T, G, T2).

update_table([], X, A, [(X, A)]):-!.
update_table([(X, _)|T], X, A, [(X, A)|T]):-!.
update_table([H|T], X, A, [H|T1]) :-update_table(T, X, A, T1).
```

Q2 Combinatory Logic

4 Points

Recall the **S**, **K** and **I** combinators of Combinatory Logic, and their equational rules (for all *x*, *y* and *z*):

I *x* = *x*

K *x y* = *x*

S *x y z* = (*x z*) (*y z*)

Prove that **S K K** = **I**

[Hint: Apply both sides to an arbitrary CL term *P*]

=cl denotes combinatory logic equality.
S K K P
=cl (K P) (K P) /// third relation
=cl (K P (KP)) /// (LEFT ASSOCIATIVE)
=cl P /// second relation
=cl I P // first relation in opposite direction.
Thus S K K P = I P , and we know that =cl is also posited to be a congruence with respect to the constructs of CL (i.e., =cl is preserved both in the operator and argument position), and Since P is arbitrary we can conclude that S K K = I

Q3 Type Inference

12 Points

Consider the following OCaml program:

```
exception UnEqualLength;;
let rec zipwith f l1 l2 = match l1, l2 with
   [ ], [ ] -> [ ]
  | [ ], _ -> raise UnEqualLength
  | _ , [ ] -> raise UnEqualLength
  | x::xs, y::ys -> f(x,y)::(zipwith f xs ys)
;;
```

What is its type?

(t2 -> t3 -> t4) -> [t2] -> [t3] -> [t4]

Show your working, clearly indicating what type assumptions you are making, what type constraints you have for sub-expressions, and what type equational constraints are obtained.

type assumptions are :-
zipwidth : t0
f : t1
l1 : [t2] /// list of elements of type t2
l2 : [t3] /// list of elements of type t3
Let us name the 3 cases in the declaration of the code as 1,2,3.
From 3 we can conclude that f takes in a element of l1 ie t2, and l2 ie t3 and gives an element of type say t4
thus t1 = t2 -> t3 -> t4 ....(1)
and this t4 must have the same type as that of an element in the list returned by zipwith, thus type of (zipwith f xs ys) : t4
Thus combining all these
wipwith takes in f of type t1 = (t2 -> t3 -> t4) , and a list's of element type t2 and t3
and returns t4.
Thus
(t2 -> t3 -> t4) -> [t2] -> [t3] -> [t4] is the type.

Q4 Operational Semantics of Commands

8 Points

In addition to a "while loop", Pascal contains an iterative construct $\textbf{repeat}\ c\ \textbf{until}\ e$ with the following informal specification: execute command c; then test if boolean condition e is true. If yes, then exit the loop (and proceed to the next command if it exists). Otherwise, repeatedly execute the loop until e becomes true.

Write a big-step semantics specification for this construct $\gamma -[\ \textbf{repeat}\ c\ \textbf{until}\ e\ ]\!\rightarrow \gamma'$

```
y -> [c] -> y'        y' -> e => True     y' -> [repeat c until e] -> y''
---------------------------------------------------------------------------------RepeatTrue
y -> [repeat c until e] -> y''


y->[c]->y'          y' -> e => False
------------------------------------------------ RepeatFalse
y -> [repeat c until e] -> y'
```

Q5 Hoare Logic

6 Points

Definw two commands to be HL-equivalent, written $c_1 \approx_{HL} c_2$, if for *all* predicates p and q:
$\vdash_{HL} \{p\} c_1 \{q\}$ *if and only if* $\vdash_{HL} \{p\} c_2 \{q\}$.

Prove that for all commands *c*:
$\mathbf{skip}; c \approx_{HL} c \approx_{HL} c; \mathbf{skip}$
[Hint: Use the Hoare Logic rules. Do **not** use induction on *c*.]

Let p and q be any two arbitrary predicates, such that |-hl {p} c{q} ....(1)
then by HSkip
|-hl {p} skip {p}
and from (1)
|-hl {p} c{q} ,
combining these 2 using Hseq we get |-hl {p} skip; c {r}

Similarly
From (1)
|-hl {p} c {q}
and from Hskip
|-hl {q} skip {q}
combining these 2 using Hseq we get |-hl {p} skip; c {r}

Since p and q are arbitrary, we can conclude that
skip; c = c = c;skip