Cs358 Exam

Name: ______________________ ID: ______________________

This assignment has **6** questions, for a total of **100** marks.

Question 1: **Formalising capability machines** .... 25 marks

Capability machines extend assembly instructions with explicit capabilities such that reading and writing on memory is only allowed if a capability is provided.

Take ULC (with Nats, pairs and products) and add a heap from natural numbers (i.e., as the one from assignment 6), make allocation deterministic starting from 0, each new location is at the next number. Extend the language with capabilities and formalise their semantics. You choose how to model them, choose wisely according to their behaviour as described below.

Capabilities are unforgeable and unobservable tokens which the program can create. Every time a memory location is created, it is unprotected. The language must provide primitives for protecting a location given a capability, this should only be possible if the location is unprotected. Reading and writing a memory location is always possible if the location is unprotected. However, if the location is protected with a capability, reading and writing that location is only possible if the same capability is provided at reading and writing time.

Question 2: **Thread-based concurrency** .... 20 marks

Take STLC with sums and pairs. Add a new term for spawning a process, call that *fork* t. *fork* t must not reduce t, instead it must return 0 and immediately spawn a thread whose body is t. Threads are scheduled nondeterministically for now, though you may want to look at the next exercise to build a more robust solution.

Formalise the statics and dynamic semantics of such a concurrent language and show all changes to the formalisation of the language.

Question 3: **Round-robin scheduling** . . . . 15 marks

Take the language from the previous exercise. Change the scheduling to be round-robin. A thread executes for 10 steps and then control passes to the next thread (if there is one). If a thread terminates before 10 steps, control passes to the next thread. The next thread to be scheduled must be the next one in order of spawning.

Formalise the scheduling process and show changes to the formalisation of the language.

Make the formalisation elegant.

Question 4: **Free theorems** .... 15 marks

Consider type $\tau = \forall\alpha.\forall\beta.\alpha \to \beta$. Is it inhabited? If yes, show a term that inhabits τ and prove that such a term is in the term relation for τ. If not, prove that there is no term in the term relation for τ.

Question 5: **Subtyping** .... 10 marks

Subtyping lets you use a term of type τ at a super-type τ'. Intuitively, it is the same principle by which you can use an object of a class as if it were of a super class. Concretely, subtyping is achieved by introducing the subsumption rule below, as well as an ordering on types, indicated with $<:$.

$$\frac{\Gamma \vdash t : \tau \qquad \tau <: \tau'}{\Gamma \vdash t : \tau'}\ \textsf{(Subsumption)}$$

Consider the standard typing of the sequencing rule and its related reduction. Suppose we extend our types with *Unit*, so $\tau ::= \cdots \mid \mathit{Unit}$ and our terms with *unit* (a new value), so $t ::= \cdots \mid \mathit{unit}$.

$$\frac{\Gamma \vdash t : \mathit{Unit} \qquad \Gamma \vdash t' : \tau}{\Gamma \vdash t; t' : \tau}\ \textsf{(Type-seq)} \qquad \frac{}{\mathit{unit}; t \rightsquigarrow^{p} t}\ \textsf{(Eval-seq)}$$

Consider STLC with sums and pairs. Can we introduce an ordering on types to allow this kind of reduction: $v; t \rightsquigarrow^{p} t$ for any value v? If yes, show such an ordering. If no, argue why not.

Question 6: **Program equivalence** .... 15 marks

For each of these programs, tell if they are equivalent or not. If they are equivalent, show what they reduce to, no matter the input. If they are not, argue why and if possible, show a context that tells them apart.

1. $z : Ref\ (N \to N)$.
   $t_1 = \lambda x : N. !z\ 0; 2 + x$
   $t_2 = \lambda x : N. if\ x > 0\ then\ x + 2\ else\ !z\ x; x + 2$.

2. $t_1 = let\ x : N = \lambda y : \forall \alpha.\alpha \to \alpha.\, \lambda z : N.\, y\ [N]\ (x + 1)\ in\ x$
   $t_2 = \lambda y : \forall \alpha.\alpha \to \alpha.\, \lambda x : N.\, (y\ [N]\ x) + 1$.

3. $f : (Ref\ N) \to N$.
   $t_1 = let\ x = new\ 0\ in\ f\ x; !x$
   $t_2 = let\ x = new\ 1\ in\ f\ (new\ 0); x := (!x - 1)$.

4. $r : Ref\ N$.
   $t_1 = let\ x = !r\ in\ let\ y = new\ x\ in\ r := !y; !y$
   $t_2 = let\ x = new\ 0\ in\ let\ y = !x; !r\ in\ y$.

5. $t_1 = \lambda x : N. \langle x, 1 \rangle .1$
   $t_2 = let\ x = \Lambda \alpha.\lambda x : \alpha.\, x\ in\ x$.