COMP3161/9164

Concepts of Programming Languages
FINAL EXAM
Term 3, 2022

- Total Number of **Parts**: 4
- Total Number of **marks**: 100
- All parts **are** of equal value.
- Answer **all** questions.
- Excessively verbose answers may lose marks
- Failure to make the declaration or making a false declaration results in a 100% mark penalty.
- Ensure you are the person listed on the declaration.
- All questions must be attempted **individually** without assistance from anyone else.
- You must **save** your exam paper using the button below **before** time runs out.
- **Late submissions will not be accepted**.
- You may save multiple times before the deadline. Only your final submission will be considered.

Declaration

☐ I, Mr Wang, William (William Wang) (5286124), declare that these answers are **entirely my own**, and that I did not complete this exam with assistance from anyone else.

Part A (25 marks)

Consider the following inductive definition of a toy language **L**, where programs can perform arithmetic computations, and send/receive numbers over the network. We assume a language **Arith** of arithmetic expressions, the details of which are unimportant here. Arithmetic expressions are *open*, in the sense that they may contain variables; for example, we would expect x + 1 **Arith** to hold.

$$\frac{x\ \mathbf{String} \quad e\ \mathbf{Arith} \quad p\ \mathbf{L}}{\mathtt{let}\ x := e\ \mathtt{in}\ p\ \mathbf{L}}\,(a) \qquad \frac{e\ \mathbf{Arith} \quad p\ \mathbf{L}}{\mathtt{send}(e).\,p\ \mathbf{L}}\,(b)$$

$$\frac{x\ \mathbf{String} \quad p\ \mathbf{L}}{\mathtt{receive}(x).\,p\ \mathbf{L}}\,(c) \qquad \frac{e\ \mathbf{Arith}}{\mathtt{return}\ e\ \mathbf{L}}\,(d)$$

Here is its small-step semantics. We assume a standard big-step semantics ⇓ for arithmetic expressions, which means that we treat arithmetic expressions as if their evaluation takes a single step.

$$\frac{e \Downarrow v}{\mathtt{let}\ x := e\ \mathtt{in}\ p \mapsto p[x := v]}\,(1) \qquad \frac{e \Downarrow v}{\mathtt{send}(e).\,p \mapsto p}\,(2)$$

$$\frac{v \in \mathbb{Z}}{\mathtt{receive}(x).\,p \mapsto p[x := v]}\,(3) \qquad \frac{e \Downarrow v \quad e \neq v}{\mathtt{return}\ e \mapsto \mathtt{return}\ v}\,(4)$$

The final states of the small-step semantics are the expressions the form `return` v where $v \in \mathbb{Z}$.

1. (3 marks) Show the step-by-step evaluation of $\mathtt{let}\ x := 2 + 2\ \mathtt{in}\ \mathtt{send}(x).\,\mathtt{return}(x + x)$

⊗ × ÷ → ↦ ⇓ ⊢ ≤ ∧ ∨ ⊤ ⊥ > < ∅ ∈ ⊆ ▻ ◦ ▫ ∀ ∃ ℕ α β Γ ε η λ μ Σ σ τ

Time Remaining
2h 9m 43s

Save

2. Give an intuition for why this language should not be confluent.

3. (4 marks) Rules (1) and (3) of the above semantics uses substitution, but we never defined substitution. Give a recursive definition of substitution. The x in `let x := e in p` binds into p but not into e, and the x in `receive(x).e` binds into p. It suffices to define substitution of a single name for a natural number. You can assume that a suitable definition of substitution for arithmetic expressions already exists.

4. (2 Marks) Do we need the side-condition $e \neq v$ in rule (4)? What happens if we drop it?

5. (6 marks) The semantics above is a substitution semantics. Give an equivalent small-step environment semantics for this language. To write an inference rule with premises, you can write the rule $\frac{A \quad B}{C}$ as `A, B ⊢ C`. You may assume the existence of a big-step environment semantics for arithmetic expressions, without giving its definition. .

6. (3 Marks) Does the above semantics have any stuck states? And does your answer depend on the semantics of **Arith** in any way? Motivate.

7. (3 Marks) The final states of the above semantics gives the return value of an execution. We may also be interested in which messages were sent out during execution. How would you modify the above semantics so that the final state contains that information? You do not need to actually construct such a semantics --- explaining your idea suffices.

Part B (25 marks)

⊗ × ÷ → ↦ ⇓ ⊢ ≤ ∧ ∨ ⊤ ⊥ > < ∅ ∈ ⊆ ▻ ∘ ▫ ∀ ∃ ℕ α β Γ ε η λ μ Σ σ τ

Time Remaining
2h 9m 43s

Save

2. (6 marks) Provide the most general type for each of the following MinHS programs, if it exists. If it does not exist, briefly explain why not.
   1. (InL True, False)
   2. recfun f x = x f
   3. recfun f x = case x of (InL y) → y; (InR z) → z
3. (7 marks) Use the Curry-Howard isomorphism to provide a MinHS or Haskell program (if it exists) that constitutes a proof for each of the following logical propositions. If no such program exists, briefly explain why not.
   1. $(A \vee A) \to A$
   2. $(A \to B) \to A \to B$
   3. $((A \to \texttt{False}) \to \texttt{False}) \to A$
4. (5 marks) Using rec, Int, ×, + and 1, write a type that encodes the following rose tree datatype:

$$\begin{array}{rcl} \textbf{datatype}\ \texttt{Tree} & = & \texttt{Leaf Int} \\ & \mid & \texttt{Node [Tree]} \end{array}$$

   Recall that [a] is Haskell notation for the type of lists with element type a. Your type should have a similar structure to the original Tree type. (Hence answering Int is not accepted, even though Int is technically isomorphic to Tree).
5. (4 marks) Write a MinHS expression that has the following type:

$$\exists S.\ S \times (S \to \mathsf{Int}) \times ((S \times S) \to S)$$

   Be explicit about any packing and unpacking of existential types.

Part C (25 marks)

1. (2 marks) Is every unityped language type safe? Explain your reasoning.
2. (4 marks) Alice maintains a programming language. Her users got sick of explicitly converting ints to floats all day, so she added subtyping with the ordering Int ≤ Float. But her users, ever a quarrelsome bunch, are now

⊗ × ÷ → ↦ ⇓ ⊢ ≤ ∧ ∨ ⊤ ⊥ > < ∅ ∈ ⊆ ▻ ◦ ▫ ∀ ∃ ℕ α β Γ ε η λ μ Σ σ τ

Time Remaining
2h 9m 43s

Save

3. (10 marks) Consider the type class Comonad, a cousin of Monad defined as follows:

```
class Comonad m where
  extract :: m a → a
  extend :: (m a → b) → m a → m b
```

An example instance of this type class is infinite lists (aka Streams), defined as follows:

```
datatype Stream a = SCons a (Stream a)
instance Comonad Stream where
  extract (SCons x xs) = x
  extend f (SCons x xs) = SCons (f (SCons x xs)) (extend f xs)
```

1. What is a *type class dictionary*? What would the dictionary for this type class look like?

2. There is no instance of Comonad for the Maybe type constructor, if we require that all methods are total.

```
data Maybe a = Nothing | Just a
```

Why not? *Hint:* how would you define extract?

If we remove the Nothing constructor and only keep Just, we can define an instance. Show how.

4. (6 marks) Give an example of a *covariant*, *contravariant* and *invariant* type constructor.

5. (3 marks) Assume that A is a subtype of B. What, if any, is the subtype relationship between A → A → A and A → B → B ? If there is a subtyping relationship, provide *either* a subtyping derivation for it or a conversion function in terms of this assumed coercion function:

$$\text{coerce} : \mathtt{A} \rightarrow \mathtt{B}$$

Part D (25 marks)

1. (6 marks) For each of the following properties, determine whether it is a *safety* or *liveness* property, or some combination of the two. Explain your answer.

⊗ × ÷ → ↦ ⇓ ⊢ ≤ ∧ ∨ ⊤ ⊥ > < ∅ ∈ ⊆ ▻ ∘ □ ∀ ∃ ℕ α β Γ ε η λ μ Σ σ τ

Time Remaining
2h 9m 43s

Save

3. I won't drink whisky before dinner.

2. (6 marks) In the following program, the shared variables y and z are initially 0. x is a local variable of process P.

Process P	**Process Q**
var x;	y := 1;
x := y + z;	z := 2;

   1. Assume each instruction is executed atomically. What are the possible final values of x?
   2. In reality, the program above may yield a final state where x = 2. Why?
   3. What is the *limited critical reference* restriction, and how does it apply to the above discrepancy?

3. (6 marks) In addition to take and release, most lock implementations will have a tryLock function. The invocation tryLock(l) will attempt to claim the lock l; if successful, return `true`. If we failed to claim the lock, return `false`. Needless to say, an attempt to claim the lock will not succeed if another process currently holds the lock.

```
while(true) {
    (non-critical section)
    while(!tryLock(l));
    (critical section)
    release(l);
}
```

   Assume multiple process are running the above code concurrently, using the same shared lock l. Is this a solution to the critical section problem? Motivate your answer, and explain any further assumptions about the behaviour of tryLock that your answer depends on.

4. (7 marks) In the lecture, we discussed the Software Transactional Memory (STM) approach to critical sections.
   1. In contrast to STM, locks are said to be a *pessimistic* concurrency control model. Why?
   2. Does STM satisfy the *starvation-freedom* liveness property? Why or why not?
   3. How can the type system be used to ensure that STM transactions are only performed inside an atomic block?

END OF EXAM

(don't forget to save!)