15-312 Foundations of Programming Languages

Final Examination

December 17, 2002

Name: _____

Andrew User ID: _____

- This is an open book, open notes exam.
- Write your answer legibly in the space provided.
- There are 16 pages in this exam, including 4 worksheets.
- It consists of 4 problems worth a total of 250 points and one extra credit question worth 25 points.
- The extra credit is recorded separately, so only attempt this after you have completed all other questions.
- You have 3 hours for this exam.

Problem 1	Problem 2	Problem 3	Problem 4	Total	EC
70	**50**	**60**	**70**	**250**	**25**

1. Subtyping and Effects (70 pts)

In this problem we explore adding vectors and arrays to MinML and the possibilities for extending the subtyping relation in a sound way. Vectors are *immutable*, while arrays are *mutable*. We show here the typing rules for accessing vector and arrays, and updating array elements. We do not show the language expressions needed to create vectors or arrays (which are unimportant for this problem), nor do we show the new sets of values.

$$\frac{\Gamma \vdash e_1 : \tau \text{ vector} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash \text{sub}_v(e_1, e_2) : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \text{ array} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash \text{sub}_a(e_1, e_2) : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau \text{ array} \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{update}_a(e_1, e_2, e_3) : \tau}$$

For the first set of question we adopt the subset interpretation of subtyping, that is, we assume vectors and arrays have identical representations.

For each of the following proposed rules, state whether it satisfies the *Fundamental Principle of Subtyping* (FPS) as discussed in lecture.

Note that MinML is designed as a statically typed language, and the language extension should preserve this property.

1.1 (5 pts) Vector/vector subtyping.

$$\frac{\tau \leq \sigma}{\tau \text{ vector} \leq \sigma \text{ vector}}$$

Satisfies FPS? (circle one) Yes No

1.2 (5 pts) Array/array subtyping.

$$\frac{\tau \leq \sigma}{\tau \text{ array} \leq \sigma \text{ array}}$$

Satisfies FPS? (circle one) Yes No

1.3 (5 pts) Vector/array subtyping.

$$\frac{\tau \leq \sigma}{\tau \text{ vector} \leq \sigma \text{ array}}$$

Satisfies FPS? (circle one) Yes No

1.4 (5 pts) Array/vector subtyping.

$$\frac{\tau \leq \sigma}{\tau \text{ array} \leq \sigma \text{ vector}}$$

Satisfies FPS? (circle one) Yes No

In the second half of this problem, we adopt the coercion interpretation of subtyping and explore whether it is possible to exploit subtyping to eliminate those pesky "!" operators from MinML with mutable store. We propose the following rule

$$\frac{\tau \leq \sigma}{\tau \text{ ref} \leq \sigma}$$

1.5 (5 pts) Give the type of $\lambda x.x := x + 1$, assuming assignment returns the value of its second argument.

1.6 (10 pts) Show the typing derivation, assuming the context is empty and $+$ is a primitive operator on integers returning an integer.

1.7 (10 pts) Complete the coercion interpretation of the subtyping rule.

$$\frac{f : \tau \leq \sigma}{: \tau \ \mathsf{ref} \leq \sigma}$$

1.8 (10 pts) Does the proposed rule satisfy the *Fundamental Principle of Subtyping*? Briefly explain or show a counterexample.

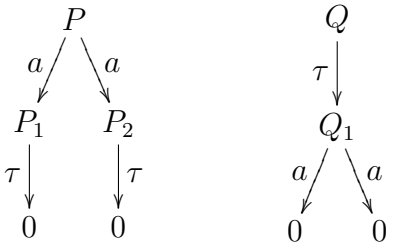
1.9 (15 pts) Is the resulting system *coherent*? Briefly explain or show a counterexample.

2. Bisimulation (50 pts)

For each of the following, indicate whether the two transition systems are strongly bisimilar, weakly bisimilar, or not bisimilar. Of course, strongly bisimilar states are also weakly bisimilar, so circle only the strongest property that is satisfied. If a strong or weak bisimulation exists, exhibit the bi-simulation by writing out which named nodes in the two diagrams it relates. You do not need to write that 0 is bisimilar to itself, which is always satisfied.

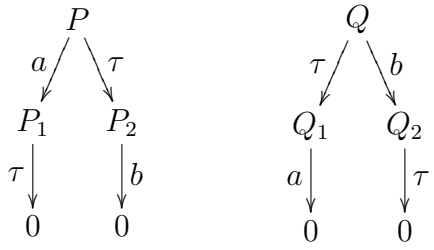
2.1 (15 pts) Processes P and Q are (circle one): (i) strongly (ii) weakly (iii) not bisimilar

Bisimulation (if it exists)



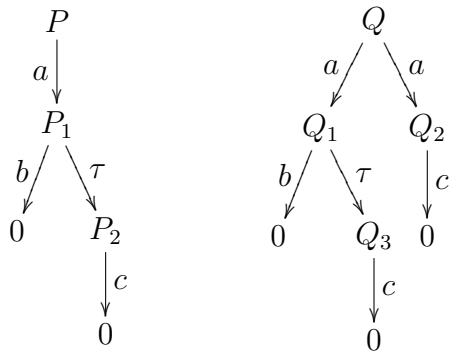
2.2 (15 pts) Processes P and Q are (circle one): (i) strongly (ii) weakly (iii) not bisimilar

Bisimulation (if it exists)



2.3 (20 pts) Processes P and Q are (circle one): (i) strongly (ii) weakly (iii) not bisimilar

Bisimulation (if it exists)



3. Futures and Types (60 pts)

Futures the way we discussed them in class are safe in a statically typed language. However, the type system does not track them in the sense that we cannot distinguish if a given value is an actual value or a future. An alternative is to introduce a type τ future with constructs future(e) and touch(e) with the typing rules

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{future}(e) : \tau\ \mathsf{future}} \qquad \frac{\Gamma \vdash e : \tau\ \mathsf{future}}{\Gamma \vdash \mathsf{touch}(e) : \tau}$$

The intended operational semantics of future(e) is to start computation of e and immediately return a *promise* l (this is sometimes called a future, but we find this confusing, so we use the terminology of Assignment 8). touch(e) evaluates e to a promise l and then blocks until the computation denoted by l has finished computing a value which is then returned.

Assume a machine state $\langle H, e\rangle$ in the small-step operational semantics (also called *structured operational semantics*) as given in lecture. Here H is a collection of process states $l_i{=}e_i$, where the labels l_i also stand for promises, and e is a process singled out to make a step. Note that this is more abstract than the C-machine used in Assignment 8.

3.1 (20 pts) Show all rules concerning future, touch, and promises for the judgment

$$\langle H, e\rangle \mapsto \langle H', e'\rangle.$$

Give an implementation of futures in CML, against the following signature. Note that the type of `future` is slightly different since in ML we are implementing future as a function under call-by-value.

```
signature FUTURE =
sig
  type 'a future
  val future : (unit -> 'a) -> 'a future
  val touch : 'a future -> 'a
end;
```

You should only use pure CML constructs as discussed in lecture and employed in sample code. In particular, you should not use mutable references, exceptions, or letcc. You do not need to account for exceptions raised in the argument to `future`. Be careful to obtain the intended concurrency. It might make it easier for us to assign partial credit if you also briefly explain your functions.

3.2 (5 pts) Define the type `'a future`

3.3 (20 pts) Define the function `future`

3.4 (15 pts) Define the function `touch`

3.5 (25 pts Extra Credit) Define a CML function

```
val future2 : (unit -> 'a) * (unit -> 'a) -> 'a future
```

which takes two functions f and g and immediately starts executing them both, returning only a single promise. When the promise is touched, we block until at least one of the values is available. Does your implementation always return the same value if a promise is touched multiple times?

4. Pattern matching (70 pts)

In this question we explore adding pattern matching to MinML. We extend the language as follows

$$\begin{array}{rrcl} \text{Expressions} & e & ::= & \ldots \mid \mathsf{case}\ e\ \mathsf{of}\ ms\ \mathsf{end} \\ \text{Matches} & ms & ::= & \cdot \mid (p \Rightarrow e \mid ms) \\ \text{Patterns} & p & ::= & x \mid \langle p_1, p_2 \rangle \mid \langle\,\rangle \end{array}$$

We have the following new typing judgments

$\Gamma \vdash ms : \sigma \Rightarrow \tau$	in context Γ, all branches of ms map a subject of type σ to a result of type τ
$\Gamma \vdash p : \sigma$	pattern p matches σ with context Γ.

We only show the rules for the new constructs.

$$\frac{\Gamma \vdash e : \sigma \quad \Gamma \vdash ms : \sigma \Rightarrow \tau}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ ms\ \mathsf{end} : \tau}$$

$$\frac{}{\Gamma \vdash \cdot : \sigma \Rightarrow \tau} \qquad \frac{\Gamma' \vdash p : \sigma \quad \Gamma, \Gamma' \vdash e : \tau \quad \Gamma \vdash ms : \sigma \Rightarrow \tau}{\Gamma \vdash (p \Rightarrow e \mid ms) : \sigma \Rightarrow \tau}$$

$$\frac{}{x{:}\sigma \vdash x : \sigma} \qquad \frac{\Gamma_1 \vdash p_1 : \sigma_1 \quad \Gamma_2 \vdash p_2 : \sigma_2}{\Gamma_1, \Gamma_2 \vdash \langle p_1, p_2 \rangle : \sigma_1 \times \sigma_2}\ (*) \qquad \frac{}{\cdot \vdash \langle\,\rangle : 1}$$

(*) The rule for pair patterns $\langle p_1, p_2 \rangle$ has a side condition stating that the variables declared in Γ_1 and Γ_2 must be disjoint. In other words, patterns may not contain repeated variables.

4.1 (10 pts) Write the function fst : $\forall t.\forall s.t \times s \to t$ using pattern matching.

4.2 (15 pts) Extend the language of patterns to handle disjoint sums $\sigma_1 + \sigma_2$ and give the new typing rules.

In the second part of this question we write out a big-step operational semantics (also called an *evaluation semantics*) employing environments. The new judgments are:

$\eta \vdash v_1 > ms \Downarrow v$	in environment η, evaluating a branch in ms matching v_1 yields value v
$v_1/p \Downarrow \eta$	matching v_1 against pattern p yields environment η.

We only show the rules concerned with the new constructs.

$$\frac{\eta \vdash e_1 \Downarrow v_1 \quad \eta \vdash v_1 > ms \Downarrow v}{\eta \vdash \mathsf{case}\ e_1\ \mathsf{of}\ ms\ \mathsf{end} \Downarrow v}$$

$$\frac{v_1/p \Downarrow \eta_1 \quad \eta, \eta_1 \vdash e \Downarrow v}{\eta \vdash v_1 > (p \Rightarrow e \mid ms) \Downarrow v} \qquad \frac{\eta \vdash v_1 > ms \Downarrow v}{\eta \vdash v_1 > (p \Rightarrow e \mid ms) \Downarrow v}$$

$$\frac{}{v_1/x \Downarrow x{=}v_1} \qquad \frac{v_1/p_1 \Downarrow \eta_1 \quad v_2/p_2 \Downarrow \eta_2}{\langle v_1, v_2\rangle/\langle p_1, p_2\rangle \Downarrow \eta_1, \eta_2} \qquad \frac{}{\langle\,\rangle/\langle\,\rangle \Downarrow \cdot}$$

4.3 (15 pts) Show the new evaluation rules for disjoint sums.

4.4 (20 Pts) The resulting language should satisfy type preservation. State the invariants that relate the new typing and the new evaluation judgments that are necessary for an inductive proof. You may freely refer to the judgment $\eta : \Gamma$ introduced in class. You do not need to show the proof.

4.5 (10 pts) Does the resulting language satisfy determinacy? If yes, give a brief explanation; if not give a counterexample.

Worksheet

Worksheet

Worksheet

Worksheet