

Write all of your answers and scratch work in your exam booklet(s), not on this exam paper; only material written in your booklet(s) will be graded. Remember to write your name on the front of your exam booklet. You may take a single, two-sided sheet of notes with you into the exam. All other books or notes must remain closed throughout the exam. You will have 2 hours and 45 minutes to complete the exam; all papers must be turned in by 4:45pm. When turning in your answers, place your exam paper inside your exam booklet and turn them both in together.

This sample exam has been made intentionally longer than the real final exam in order to give you a comprehensive collection of practice problems.

1 Problem Set

- (1) (8 pts) Implement a **tail-recursive** OCaml function named `split` that takes as input a list l and returns a pair of lists such that one member of the pair consists of the even-indexed elements of l and the other consists of the odd-indexed elements of l . For example, (`split [2;37;13]`) might return (`[13;2], [37]`) or (`[37], [2;13]`). The order of elements in the returned lists does not matter. The lists should be polymorphic, so your code should start with something like:

```
let rec split (l:'a list) : ('a list * 'a list) = ...
```

Do not use any of the OCaml List library functions in your implementation, except that you may use `List.fold_left` if you wish. Remember that your code must be tail-recursive to receive full credit!

- (2) (5 pts) Implement a Prolog predicate `disjoint(X,Y)` that succeeds if and only if lists X and Y contain no common elements.
- (3) Consider the OCaml program p defined by `(fun x y -> (fun z -> z) y)`.
- (a) (1 pt) Write an α -equivalent OCaml program.
 - (b) (2 pts) Write a β -equivalent OCaml program.
 - (c) (3 pts) Write an η -equivalent OCaml program.
- (4) For each of the following System F types, say whether the type is inhabited or not. If the type is inhabited, give an example of a System F term that inhabits it. (Do not prove that your term inhabits the type, just state it.) If the type is not inhabited, change the type into a logical predicate and show that the predicate is not tautological.

- (a) (2 pts) $\forall \alpha. \alpha$
 - (b) (3 pts) $\forall \alpha. (\alpha + \text{unit})$
 - (c) (4 pts) $\forall \alpha. \forall \beta. (\alpha + \beta) \rightarrow (\alpha \times \beta)$
 - (d) (7 pts) $\forall \alpha. \forall \beta. ((\alpha + \beta) \rightarrow ((\alpha \rightarrow \beta) + (\beta \rightarrow \alpha)))$
- (5) (5 pts) Encode an *even?* function in the untyped λ -calculus so that (*even?* $n_{\mathbb{N}}$) evaluates to *true* whenever n is even and to *false* whenever n is odd.
- (6) (15 pts) Consider the untyped λ -calculus expression *foo* defined as follows:

$$\text{foo} = Y(\lambda f. \lambda x. \text{if } (\text{natzero? } x) \text{ then } x \text{ else } (f (\text{natpred } x)))$$

Prove by fixed-point induction that $P(\text{foo})$ holds, where P is the property defined by

$$P(g) \equiv \forall (x_{\mathbb{N}}, y_{\mathbb{N}}) \in g. y_{\mathbb{N}} = 0_{\mathbb{N}}$$

In your proof when you claim that an expression e_1 evaluates to another expression e_2 , you may do so without a formal proof of $e_1 \rightarrow^* e_2$. That is, you need not formally expand all abbreviations and then write out a small-step derivation.

- (7) (5 pts) Derive the following typing judgment using the typing rules for the simply-typed λ -calculus:

$$\{\} \vdash (\lambda x : \text{int}. x) 3 : \text{int}$$

- (8) (20 pts) Derive the following partial correctness assertion using Hoare Logic:

$$\{x = \bar{n}\} \text{while } x \leq -1 \text{ do } x := x + 1 \{x = \max(\bar{n}, 0)\}$$

2 Solutions

- (1)

```
let split =
  let rec helper (l1,l2) l3 = (match l3 with [] -> (l1,l2)
                               | h::t -> helper (l2,h::l1) t)
  in helper ([], []);;
```
- (2)

```
nonmember(_, []).
nonmember(X, [Y|T]) :- X \= Y, nonmember(X, T).
disjoint([], _).
disjoint([X|T], L) :- nonmember(X, L), disjoint(T, L).
```
- (3) (a)

```
(fun a b -> (fun c -> c) b)
```


 (b)

```
(fun x y -> y)
```


 (c)

```
(fun x -> (fun z -> z))
```

- (4) (a) The type is not inhabited. Changing it into a logical predicate yields $\forall p.p$, which is not a tautology because there exists a p that falsifies it—namely $p = F$.
- (b) The type is inhabited. The following is a term that inhabits it: $\Lambda\alpha.\text{in}_2^{\alpha+\text{unit}}()$.
- (c) The type is not inhabited. Changing it into a logical predicate yields $\forall p.\forall q.(p \vee q) \Rightarrow (p \wedge q)$. This is not a tautology because there exists a p and a q that falsify it. In particular, if $p = T$ and $q = F$ then $(T \vee F) \Rightarrow (T \wedge F) \equiv T \Rightarrow F \equiv F$.
- (d) The type is inhabited. The following is a term that inhabits it:

$$\Lambda\alpha.\Lambda\beta.\lambda x:\alpha+\beta.\text{case } x \text{ of } \text{in}_1(y) \rightarrow \text{in}_2^{(\alpha\rightarrow\beta)+(\beta\rightarrow\alpha)}(\lambda z:\beta.y) \\ \mid \text{in}_2(y) \rightarrow \text{in}_1^{(\alpha\rightarrow\beta)+(\beta\rightarrow\alpha)}(\lambda z:\alpha.y)$$

- (5) The *even?* function can be encoded this way:

$$\text{even?} = Y(\lambda f.\lambda n.(if (\text{natzero? } n) \text{ then true} \\ \text{else if } (\text{natzero? } (\text{natpred } n)) \text{ then false} \\ \text{else } (f (\text{natpred } (\text{natpred } n))))))$$

- (6) *Proof.* Define functional Γ by

$$\Gamma(f) = \lambda x.if (\text{natzero? } x) \text{ then } x \text{ else } (f (\text{natpred } x))$$

Since $\text{foo} = Y\Gamma = \text{fix}(\Gamma)$, we can prove the theorem by fixed-point induction on Γ .

Base Case: $P(\perp)$ holds vacuously.

Inductive Case: We must prove that $P(g)$ implies $P(\Gamma(g))$. Therefore, assume $P(g)$ holds and let $(x_{\mathbb{N}}, y_{\mathbb{N}}) \in \Gamma(g)$ be given. We wish to prove that $y_{\mathbb{N}} = 0_{\mathbb{N}}$.

Case 1: Suppose $x_{\mathbb{N}} = 0_{\mathbb{N}}$. By the definition of Γ , $\Gamma(x_{\mathbb{N}}) = x_{\mathbb{N}} = 0_{\mathbb{N}}$, so $y_{\mathbb{N}} = 0_{\mathbb{N}}$.

Case 2: Suppose $x_{\mathbb{N}} \neq 0_{\mathbb{N}}$. Then by definition of Γ , $y_{\mathbb{N}} = (g (\text{natpred } x_{\mathbb{N}})) = (g (x - 1)_{\mathbb{N}})$. This is the same as saying $((x - 1)_{\mathbb{N}}, y_{\mathbb{N}}) \in g$. Since we assumed $P(g)$ holds, it follows that $y_{\mathbb{N}} = 0_{\mathbb{N}}$. \square

- (7) The following typing derivation proves the typing judgment:

$$\frac{\frac{\frac{\overline{\{(x, \text{int})\} \vdash x : \text{int}}^{(9)}}{\{\} \vdash (\lambda x:\text{int}.x) : \text{int} \rightarrow \text{int}}^{(11)}}{\{\} \vdash (\lambda x:\text{int}.x)3 : \text{int}}^{(12)}}{\{\} \vdash 3 : \text{int}}^{(10)}}{\{\} \vdash (\lambda x:\text{int}.x)3 : \text{int}}^{(12)}$$

- (8) Choose loop invariant $I = (x \leq 0) \vee (x = \bar{n})$ and derive the following:

$$\frac{\frac{\frac{\frac{\vdash I \wedge b \Rightarrow C}{\vdash I \wedge b} \quad \frac{\overline{\{C\}x:=x+1\{I\}}^{(4)}}{\{I \wedge b\}x:=x+1\{I\}}^{(5)}}{\vdash A \Rightarrow I} \quad \frac{\vdash I \Rightarrow I}{\vdash -b \wedge I \Rightarrow B}^{(6)}}{\vdash A \Rightarrow I} \quad \frac{\vdash -b \wedge I \Rightarrow B}{\{A\}p\{B\}}^{(6)}}{\{A\}p\{B\}}^{(6)}$$

where

$$\begin{aligned}
p &= \text{while } x \leq -1 \text{ do } x := x + 1 \\
b &\equiv (x \leq -1) \\
I &\equiv (x \leq 0) \vee (x = \bar{n}) \\
A &\equiv (x = \bar{n}) \\
B &\equiv (x = \max(\bar{n}, 0)) \\
C &\equiv I[x/x + 1] \equiv (x + 1 \leq 0) \vee (x + 1 = \bar{n})
\end{aligned}$$

3 Reference

3.1 Syntax of IMP

commands $c ::= \text{skip} \mid c_1; c_2 \mid v := a \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c$
boolean expressions $b ::= \text{true} \mid \text{false} \mid a_1 \leq a_2 \mid b_1 \ \&\& \ b_2 \mid b_1 \ || \ b_2 \mid !b$
arithmetic expressions $a ::= i \mid v \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$
variable names v
integer constants i

3.2 Axiomatic Semantics of IMP

$$\{A\} \text{skip} \{A\} \tag{1}$$

$$\frac{\{A\}c_1\{C\} \quad \{C\}c_2\{B\}}{\{A\}c_1; c_2\{B\}} \tag{2}$$

$$\frac{\{A \wedge b\}c_1\{B\} \quad \{A \wedge \neg b\}c_2\{B\}}{\{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\}} \tag{3}$$

$$\{B[v/a]\}v := a\{B\} \tag{4}$$

$$\frac{\{I \wedge b\}c\{I\}}{\{I\} \text{while } b \text{ do } c \{-b \wedge I\}} \tag{5}$$

$$\frac{\models A \Rightarrow A' \quad \{A'\}c\{B'\} \quad \models B' \Rightarrow B}{\{A\}c\{B\}} \tag{6}$$

3.3 Untyped Lambda Calculus

3.3.1 Syntax of Untyped λ -calculus

expressions $e ::= v \mid \lambda v. e \mid e_1 e_2$
variables v

3.4.2 Static Semantics of λ^{\rightarrow}

$$\Gamma \vdash v : \Gamma(v) \quad (9)$$

$$\Gamma \vdash i : int \quad (10)$$

$$\frac{\Gamma[v \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \lambda v : \tau_1. e : \tau_1 \rightarrow \tau_2} \quad (11)$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \quad (12)$$

3.5 System F

expressions	$e ::= v \mid i \mid \mathbf{true} \mid \mathbf{false} \mid \lambda v : \tau. e \mid e_1 e_2 \mid e_1 \mathit{aop} e_2 \mid e_1 \mathit{bop} e_2 \mid$ $\neg e \mid e_1 \mathit{cmp} e_2 \mid (e_1, e_2) \mid \pi_1 e \mid \pi_2 e \mid () \mid \mathbf{in}_1^{\tau_1 + \tau_2} e \mid \mathbf{in}_2^{\tau_1 + \tau_2} \mid$ $(\mathbf{case} e \mathbf{of} \mathbf{in}_1(v_1) \rightarrow e_1 \mid \mathbf{in}_2(v_2) \rightarrow e_2) \mid \Lambda \alpha. e \mid e[\tau]$
arithmetic ops	$\mathit{aop} ::= + \mid - \mid *$
boolean ops	$\mathit{bop} ::= \wedge \mid \vee$
comparisons	$\mathit{cmp} ::= \leq \mid \geq \mid < \mid > \mid =$
types	$\tau ::= int \mid bool \mid \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid unit \mid void \mid \alpha \mid \forall \alpha. \tau$
integers	i
variables	v
type variables	α