This sample final exam is LONGER than a real final exam (to give you more practice problems) and has a medium difficulty level. You may take two, two-sided sheets 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 1:45pm.

1 Problem Set

(1) (15 pts) A metric is a function \( m : \alpha \rightarrow \alpha \rightarrow \text{int} \) that computes some notion of distance between two values. The path-length of a list is the sum of the distances between each consecutive pair of elements. For example, if the data has type \( \alpha = \text{int} \) and the metric is absolute difference, then the path-length of list \([7; 10; 6]\) is \(|10 - 7| + |6 - 10| = 7|.

Using only \texttt{List.fold_left} for recursion, implement a function \((\text{pathlen } \mathit{m} \ \ell)\) that computes the path-length of \(\ell\) using metric function \(\mathit{m}\). If \(\ell\) has less than 2 elements, the path-length is 0. Do not use any other \texttt{List} library functions in your implementation.

(2) (15pts) Using Prolog, implement a type-checker for the following small subset of System F:

\[
e \ ::= (\, | \ \lambda v : \tau . e \ | \ \Lambda \alpha . e \ | \ v \\
\tau \ ::= \text{unit} \ | \ \tau_1 \rightarrow \tau_2 \ | \ \forall \alpha . \tau \ | \ \alpha
\]

In particular, implement a predicate \(\text{typ}(E, T)\) that succeeds if and only if expression \(E\) has type \(T\), where expressions and types are represented as the following Prolog structures:

\[
V ::= \text{arbitrary Prolog atoms} \\
E ::= \text{uval} \mid \text{lam}(V, T, E) \mid \text{poly}(V, E) \mid \text{var}(V) \\
T ::= \text{unit} \mid \text{arr}(T_1, T_2) \mid \text{forall}(V, T) \mid \text{tvar}(V)
\]

Your \texttt{typ/2} predicate should fail (reject) if \(E\) has free variables. If you use any helper Prolog predicates that we defined in class, give their exact definitions (since the exact definition you use might affect the correctness of your implementation).

(3) Consider the following recursive definition of function \(f\):

\[
f(x) = (x = 100 \rightarrow 0 \mid x > 100 \rightarrow f(x - 1) + 1 \mid x < 100 \rightarrow f(x + 1) + 1)
\]

(a) (1 pt) Define a non-recursive functional \(F\) whose least fixed point is \(f\).

(b) (4 pts) Give a closed-form function definition \(h\) such that \(h = f\).

(c) (15 pts) Prove by fixed point induction that \(h = \text{fix}(F)\).

(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, just write “uninhabited.”
(a) (3 pts) \( \forall \alpha. \forall \beta. \forall \eta. ((\alpha \times \beta) \rightarrow \eta) \rightarrow (\alpha \rightarrow \beta \rightarrow \eta) \)

(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 \textit{isEven} function in the untyped \( \lambda \)-calculus so that \((\text{isEven} \ n_N)\) evaluates to \text{true} whenever \( n \) is even and to \text{false} whenever \( n \) is odd.

(6) (15 pts) Consider the untyped \( \lambda \)-calculus expression \( \text{foo} \) defined as follows:

\[
\text{foo} = Y (\lambda f. \lambda x. ((\text{iszero} \ x) \ ? \ (f (\text{pred}_N \ x))) )
\]

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

\[
P(g) \equiv \forall (x_N, y_N) \in g . \ y_N = 0_N
\]

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, it is not necessary to write out all the small-step derivations.

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

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

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

\[
\{ x = \bar{n} \} \text{while } x <= -1 \text{ do } x := x + 1 \{ x = \text{max}(\bar{n}, 0) \}
\]

2 Solutions

(1) let pathlen m = function \[\] -> 0 | h::t ->

\[
\text{fst} \ (\text{List.fold_left} \ (\text{fun} \ (s,p) \ x \rightarrow \ (s+(m \ p \ x),x)) \ (0,h) \ t);
\]

(2) \% the pick/2 predicate defined in class...

pick([X|[_]],X).
pick([X|T],Y) :- X \neq Y, pick(T,Y).

\% succeed if type T in context C has No Free Type Variables

nftv(_,unit).
nftv(C,\text{arr}(T1,T2)) :- nftv(C,T1), nftv(C,T2).
nftv(C,forall(A,T)) :- nftv([A|C],T).
nftv(C,tvar(A)) :- pick(C,A).

\% type-checker

typ(_,_,uval,unit).
typ(_,G,var(V),T) :- pick(G,\(V,T\)).
typ(C,G,\text{lam}(V,VT,E),\text{arr}(VT,T)) :- nftv(C,VT), typ(C,[(V,VT)|G],E,T).
typ(C,G,\text{poly}(A,E),forall(A,T)) :- typ([A\|C],G,E,T).
typ(E,T) :- typ([],[],E,T).
(3) (a) \( F(g) = \lambda x. (x=100 \rightarrow 0 \mid x>100 \rightarrow g(x-1) + 1 \mid x<100 \rightarrow g(x+1) + 1) \)
(b) \( h(x) = |x - 100| \)
(c) Proof. Define proposition \( P(g) = \forall x \in g^{-}. g(x) = |x - 100| \). We wish to prove that \( P(\text{fix}(F)) \) holds.

**Base Case:** \( P(\bot) \) holds vacuously.

**Inductive Case:** As the inductive hypothesis, assume that \( P(g) \) holds. We must prove that \( P(F(g)) \) holds. Let \( x \in (g)^{-} \) be given.

Case 1: Suppose \( x = 100 \). Then by definition of \( F \), \( F(g)(x) = 0 = |x - 100| \).

Case 2: Suppose \( x > 100 \). Then by definition of \( F \), \( F(g)(x) = g(x - 1) + 1 \). By inductive hypothesis, \( g(x-1) = |x - 100| \). Since \( x > 100 \), \( |x - 100| + 1 = x - 100 + 1 = x - 100 = |x - 100| \).

Case 3: Suppose \( x < 100 \). Then by definition of \( F \), \( F(g)(x) = g(x + 1) + 1 \). Since \( x < 100 \), \( |x + 1 - 100| + 1 = -(x + 1 - 100) + 1 = -(x - 100) = |x - 100| \). □

(4) (a) \( \Lambda \alpha. \Lambda \beta. \Lambda \eta. \lambda f:((\alpha \times \beta) \rightarrow \eta). \lambda x: \alpha. \lambda y: \beta. f(x, y) \)
(b) \( \Lambda \alpha. \text{in}_{2}^{\alpha+\text{unit}}() \)
(c) uninhabited
(d) The type is inhabited. The following is a term that inhabits it:

\[
\lambda \alpha. \lambda x: \alpha + \beta . \text{case } x \text{ of } \text{in}_{1}(y) \rightarrow \text{in}_{2}^{(\alpha \rightarrow \beta) + (\beta \rightarrow \alpha)}(\lambda z: \beta . y) \\
\mid \text{in}_{2}(y) \rightarrow \text{in}_{1}^{(\alpha \rightarrow \beta) + (\beta \rightarrow \alpha)}(\lambda z: \alpha . y)
\]

(5) The `iseven` function can be encoded this way:

\[
\text{iseven} = Y(\lambda f. \lambda n. ((\text{iszero}_{\mathbb{N}} n) ? \text{true} : ((\text{iszero}_{\mathbb{N}} (\text{pred}_{\mathbb{N}} n)) \ ? \text{false} : (f (\text{pred}_{\mathbb{N}} (\text{pred}_{\mathbb{N}} n))))))
\]

(6) Proof. Define functional \( \Gamma \) by

\[
\Gamma(f) = \lambda x. ((\text{iszero}_{\mathbb{N}} x) ? x : (f (\text{pred}_{\mathbb{N}} x)))
\]

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

**Base Case:** \( P(\bot) \) 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 \( \Gamma \), \( \Gamma(x_{\mathbb{N}}) = x_{\mathbb{N}} = 0_{\mathbb{N}} \), so \( y_{\mathbb{N}} = 0_{\mathbb{N}} \).

Case 2: Suppose \( x_{\mathbb{N}} \neq 0_{\mathbb{N}} \). Then by definition of \( \Gamma \), \( y_{\mathbb{N}} = (g (\text{pred}_{\mathbb{N}} x_{\mathbb{N}})) = (g (x - 1)_{\mathbb{N}}) \).

This is the same as saying \( ((x - 1)_{\mathbb{N}}, y_{\mathbb{N}}) \in g \). Since we assumed \( P(g) \) holds, it follows that \( y_{\mathbb{N}} = 0_{\mathbb{N}} \). □
(7) The following typing derivation proves the typing judgment:

\[
\begin{align*}
\{x, \text{int}\} & \vdash \ x : \text{int} \quad (10) \\
\{\} & \vdash (\lambda x : \text{int}. x) : \text{int} \rightarrow \text{int} \quad (11) \\
\{\} & \vdash 3 : \text{int} \quad (9) \\
\{\} & \vdash (\lambda x : \text{int}. x)3 : \text{int} \quad (12)
\end{align*}
\]

(8) Choose loop invariant \( I = ((x \leq 0) \lor (x = \bar{n})) \land (x \geq \bar{n}) \) and derive the following:

\[
\begin{align*}
| I \land b & \Rightarrow C \quad \{C\} x := x + 1 \{I\} \quad (4) \quad | I \Rightarrow I \quad (6) \\
| A & \Rightarrow I \quad \{I\} p \{-b \land I\} \quad (5) \quad | -b \land I \Rightarrow B \quad (6)
\end{align*}
\]

where

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

3 Reference

In addition to the material in this reference section, you will also be provided any relevant material from the reference section of the sample midterm exam.

3.1 Syntax of SIMPL

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

\[
b ::= \text{true} \mid \text{false} \mid a_1 \leq a_2 \mid b_1 \land b_2 \mid b_1 \lor b_2 \mid \neg b
\]

arithmetic expressions

\[
a ::= n \mid v \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2
\]

variable names

\[
v
\]

integer constants

\[
n
\]
3.2 Axiomatic Semantics of SIMPL

\[
\begin{align*}
\{A\}\text{skip}\{A\} & \quad (1) \\
\{A\}c_1\{C\} \quad \{C\}c\{B\} & \quad (2) \\
\{A\}c_1; c_2\{B\} & \\
\{A \land b\}c_1\{B\} \quad \{A \land \neg b\}c_2\{B\} & \quad (3) \\
\{A\}\text{if } b \text{ then } c_1 \text{ else } c_2\{B\} & \\
\{B[a/v]\}v := a\{B\} & \quad (4) \\
\{I \land b\}c\{I\} & \\
\{I\}\text{while } b \text{ do } c\{\neg b \land I\} & \\
\models A \Rightarrow A' \quad \{A'\}c\{B'\} & \quad B' \Rightarrow B \\
\{A\}c\{B\} & \quad (6)
\end{align*}
\]

3.3 Untyped Lambda Calculus

3.3.1 Syntax and Semantics of Untyped \(\lambda\)-calculus

\[
e ::= v \mid \lambda v.e \mid e_1e_2
\]

\[
e_1 \rightarrow_1 e_1' \\
e_1e_2 \rightarrow_1 e_1'e_2
\]

\[
(\lambda v.e_1)e_2 \rightarrow_1 e_1[e_2/v]
\]

3.3.2 Abbreviations in Untyped \(\lambda\)-calculus

\[
\begin{align*}
\text{true} &= (\lambda x.\lambda y.x) \\
\text{false} &= (\lambda x.\lambda y.y) & \text{pair} &= (\lambda x.\lambda y.\lambda b.(b?e_1:e_2)) \\
e_1?e_2:e_3 &= (e_1e_2e_3) & \pi_1 &= (\lambda x . x \text{ true}) \\
\text{not} &= (\lambda b.(b?\text{false}:\text{true})) & \pi_2 &= (\lambda x . x \text{ false}) \\
\text{and} &= (\lambda a.\lambda b.(a?b:\text{false})) & 0_N &= (\lambda x.x) \\
\text{or} &= (\lambda a.\lambda b.(a?\text{true}:b)) & \text{succ}_N &= (\text{pair false}) \\
Y &= (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))) & \text{pred}_N &= \pi_2 \\
& & \text{iszero}_N &= \pi_1 \\
\text{add}_N &= (Y(\lambda f.\lambda m.\lambda n.(\text{iszero}_N m)?n:(f(\text{pred}_N m)(\text{succ}_N n)))) \\
\text{sub}_N &= (Y(\lambda f.\lambda m.\lambda n.(\text{iszero}_N n)?m:(f(\text{pred}_N m)(\text{pred}_N n)))) \\
\text{mult}_N &= (Y(\lambda f.\lambda m.\lambda n.(\text{iszero}_N m)?0_N:(\text{add}_N (f (\text{pred}_N m) n) n))))
\end{align*}
\]
3.4 Simply-typed Lambda Calculus

3.4.1 Syntax of \( \lambda \rightarrow \)

expressions
\[
e ::= n \mid v \mid \lambda v : \tau. e \mid e_1 e_2 \mid \text{true} \mid \text{false} \mid e_1 \text{aop} e_2 \mid e_1 \text{bop} e_2 \mid e_1 \text{cmp} e_2 \mid (e_1, e_2) \mid \pi_1 e \mid \pi_2 e \mid () \mid \text{in}_{\tau_1+\tau_2} e \mid \text{in}_{\tau_1+\tau_2}^2 e \mid (\text{case } e \text{ of } \text{in}_1(v_1) \rightarrow e_1 \mid \text{in}_2(v_2) \rightarrow e_2)
\]

types
\[
\tau ::= \text{int} \mid \text{bool} \mid \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid \text{unit} \mid \tau_1 + \tau_2 \mid \text{void}
\]

arithmetic ops
\[
aop ::= + \mid - \mid *
\]

boolean ops
\[
bop ::= \land \mid \lor
\]

comparisons
\[
cmp ::= \leq \mid \geq \mid < \mid > \mid =
\]

3.4.2 Static Semantics of \( \lambda \rightarrow \)

\[
\begin{align*}
\Gamma \vdash n : \text{int} & \quad \text{(9)} \\
\Gamma \vdash v : \Gamma(v) & \quad \text{(10)} \\
\Gamma[v \mapsto \tau_1] \vdash e : \tau_2 & \quad \text{(11)} \\
\Gamma \vdash (\lambda v : \tau_1 . e) : \tau_1 \rightarrow \tau_2 & \quad \text{(12)} \\
\Gamma \vdash e_1 : \tau \rightarrow \tau' \quad \Gamma \vdash e_2 : \tau & \quad \text{(13)} \\
\Gamma \vdash e_1 e_2 : \tau' & \quad \text{(14)} \\
\Gamma \vdash \text{false} : \text{bool} & \quad \text{(15)} \\
\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} & \quad \text{(16)} \\
\Gamma \vdash e_1 \text{bop} e_2 : \text{bool} & \quad \text{(17)} \\
\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} & \quad \text{(18)} \\
\Gamma \vdash e_1 \text{cmp} e_2 : \text{bool} & \quad \text{(19)} \\
\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 & \quad \text{(20)} \\
\Gamma \vdash e : \tau_1 \times \tau_2 \quad i \in \{1,2\} & \quad \text{(21)} \\
\Gamma \vdash \pi_i e : \tau_i & \quad \text{(22)} \\
\Gamma \vdash () : \text{unit} & \quad \text{(23)} \\
\Gamma \vdash \text{in}_i : \tau_i \quad i \in \{1,2\} & \quad \text{(24)} \\
\Gamma \vdash \text{in}_{\tau_1+\tau_2} e : \tau_1 + \tau_2 & \quad \text{(25)} \\
\Gamma \vdash \text{case } e \text{ of } \text{in}_1(v_1) \rightarrow e_1 \mid \text{in}_2(v_2) \rightarrow e_2 : \tau & \quad \text{(26)}
\end{align*}
\]

3.5 System F

expressions
\[
e ::= \cdots \mid \Lambda \alpha. e \mid e[\tau]
\]
types
\[
\tau ::= \cdots \mid \alpha \mid \forall \alpha. \tau
\]

\[
(\Lambda \alpha. e)[\tau] \rightarrow_1 e[\tau/\alpha]
\]

\[
\begin{align*}
\Gamma \vdash e : \tau & \quad \text{(23)} \\
\Gamma \vdash \Lambda \alpha. e : \forall \alpha. \tau & \quad \text{(24)} \\
\Gamma \vdash e[\tau] : \tau'[\tau/\alpha] & \quad \text{(25)}
\end{align*}
\]