Instructions

- This is a closed-book exam: you may not make use of any books or notes.
- You have 80 minutes to answer all of the questions. The entire exam is worth 80 points.
- Questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.
- Partial credit will be given. All correct answers are short. The back side of each page may be used as a scratch pad.
- Good luck!

Please mark your preferences for the time and date of the final exam.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>can’t do it</th>
<th>rather not</th>
<th>ok with me</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/16</td>
<td>8:30-10:30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/20</td>
<td>1:30-3:30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/21</td>
<td>11-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/21</td>
<td>1:30-3:30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Semantics of simple programming languages

The following four questions concern the following simple programming language:

\[ t ::= \]
\[ \text{true} \]
\[ \text{false} \]
\[ \text{maybe} \]
\[ \text{perhaps } t_1 \text{ then } t_2 \text{ else } t_3 \]
\[ \text{definitely } t_1 \text{ then } t_2 \text{ else } t_3 \]

\[ v ::= \]
\[ \text{true} \]
\[ \text{false} \]
\[ \text{maybe} \]

and its \textit{small-step} operational semantics.

\[ \text{perhaps } \text{true} \text{ then } t_2 \text{ else } t_3 \rightarrow t_2 \] (E-PT)
\[ \text{perhaps } \text{maybe} \text{ then } t_2 \text{ else } t_3 \rightarrow t_2 \] (E-PM)
\[ \text{perhaps } \text{false} \text{ then } t_2 \text{ else } t_3 \rightarrow t_3 \] (E-PF)
\[ \text{definitely } \text{true} \text{ then } t_2 \text{ else } t_3 \rightarrow t_2 \] (E-DT)
\[ \text{definitely } \text{maybe} \text{ then } t_2 \text{ else } t_3 \rightarrow t_3 \] (E-DM)
\[ \text{definitely } \text{false} \text{ then } t_2 \text{ else } t_3 \rightarrow t_3 \] (E-DF)

\[ \frac{t_1 \rightarrow t_1'}{\text{perhaps } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{perhaps } t_1' \text{ then } t_2 \text{ else } t_3} \] (E-P)

\[ \frac{t_1 \rightarrow t_1'}{\text{definitely } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{definitely } t_1' \text{ then } t_2 \text{ else } t_3} \] (E-M)

1. (15 points)

(a) State the structural induction principle for the syntax of this language.

\text{Answer: For all terms } t, P(t) \text{ is true if and only if}

• \(P(\text{true}), P(\text{false}), \text{and } P(\text{maybe}) \text{ are true}
• P(\text{perhaps } t_1 \text{ then } t_2 \text{ else } t_3) \text{ is true given } P(t_1), P(t_2) \text{ and } P(t_3).
• P(\text{definitely } t_1 \text{ then } t_2 \text{ else } t_3) \text{ is true given } P(t_1), P(t_2) \text{ and } P(t_3).
(b) Prove by structural induction, the following statement: For all $t$, either $t$ is a value or $t \rightarrow t'$. Note: If two cases of this proof are extremely similar, you may say that the second case is analogous to the first, instead of writing the case out in full.

Answer: The property that we would like to show is $P(t) = \text{either } t \text{ is a value or } t \rightarrow t'$. 

- Showing that the property is true when $t$ is true, false or maybe is trivial, because $t$ is a value in each of these cases.
- Suppose $t$ is perhaps $t_1$ then $t_2$ else $t_3$. By induction, we know that either $t_1$ is a value, or $t_1 \rightarrow t'_1$.
  - If $t_1$ is true then $t \rightarrow t_2$ by E-PT.
  - If $t_1$ is false then $t \rightarrow t_3$ by E-PF.
  - If $t_1$ is maybe then $t \rightarrow t_2$ by E-PM.
  - If $t_1 \rightarrow t'_1$ then $t \rightarrow \text{perhaps } t'_1$ then $t_2$ else $t_3$ by E-P.
- Suppose $t$ is definitely $t_1$ then $t_2$ else $t_3$. This case is analogous to the previous.
2. (4 points) We can define a new term form \( \text{and} \ t_1 \ t_2 \) with the following operational semantics rules:
\[
\begin{align*}
\text{and true } v & \rightarrow v \\
\text{and maybe true } & \rightarrow \text{maybe} \\
\text{and maybe maybe } & \rightarrow \text{maybe} \\
\text{and maybe false } & \rightarrow \text{false} \\
\text{and false } v & \rightarrow \text{false} \\
\qquad t_1 & \rightarrow t_1' \\
\qquad \text{and } t_1 \ t_2 & \rightarrow \text{and } t_1' \ t_2' \\
\qquad t_2 & \rightarrow t_2' \\
\qquad \text{and } v \ t_2 & \rightarrow \text{and } v \ t_2'
\end{align*}
\]
However, this term is \textit{definable} using the existing constructs of the language. What is its definition?
\( \text{and } t_1 \ t_2 = \)

\textit{Answer: definitely } \( t_1 \) \textit{then } \( t_2 \) \textit{else (perhaps } \( t_2 \) \textit{then } \( t_1 \) \textit{else false).}

3. (8 points) We can also encode this language into the untyped lambda calculus. Here is part of the encoding, fill in the missing pieces in the simplest way possible.

\( \text{comp(true} = \lambda x. \lambda y. \lambda z. x (\lambda w. w) \)

\( \text{comp(false} = \)

\textit{Answer: } \( \lambda x. \lambda y. \lambda z. z (\lambda w. w) \)

\( \text{comp(maybe} = \)

\textit{Answer: } \( \lambda x. \lambda y. \lambda z. y (\lambda w. w) \)

\( \text{comp(perhaps } t_1 \ \text{then } t_2 \ \text{else } t_3) = \text{comp}(t_1) (\lambda w. \text{comp}(t_2)) (\lambda w. \text{comp}(t_2)) (\lambda w. \text{comp}(t_3)) \)

\( \text{comp(definitely } t_1 \ \text{then } t_2 \ \text{else } t_3) = \)

\textit{Answer: } \( \text{comp}(t_1) (\lambda w. \text{comp}(t_2)) (\lambda w. \text{comp}(t_3)) (\lambda w. \text{comp}(t_3)) \)
4. (8 points) The following O’Caml definitions implement the small-step evaluation relation almost correctly, but there are several mistakes or omissions. Change the code below to repair these mistakes.

```ocaml
type term = TmTrue | TmFalse | TmMaybe
  | TmPerhaps of term * term * term
  | TmDefinitely of term * term * term

let rec ss t = match t with

  TmPerhaps(t1,t2,t3)
  | (match t1 with
      TmTrue        → ss t2
    | TmMaybe       → ss t2
    | TmFalse       → ss t3
    | _            → TmPerhaps(ss t11, ss t2, t3))
  | TmDefinitely(TmTrue,t2,t3) → ss t2
  | TmDefinitely(TmFalse,t2,t3) → ss t3
  | TmDefinitely(t1,t2,t3) → TmPerhaps(ss t1, t2, t3)
  | TmTrue → ss TmTrue
  | TmFalse → TmFalse
```

Answer:

- **Cases for perhaps true, perhaps false, perhaps maybe and definitely true shouldn’t call ss recursively. Also, the last case for perhaps shouldn’t call ss recursively for t2.**
- **Last case for perhaps should say t1 instead of t11.**
- **Missing case for definitely maybe.**
- **Congruence rule for definitely goes to perhaps.**
- **Because this is a single step semantics, there shouldn’t be cases for true and false, these are values. For these terms, an exception should be raised.**
Untyped lambda-calculus

For each of the following pairs of untyped lambda-terms, answer the following three questions:

(a) What are their normal forms? If a term does not have a normal form, write none.

(b) If they have normal forms, are these normal forms alpha-equivalent? If they are alpha-equivalent write yes, if they are not write no, if at least one term does not have a normal form write not applicable.

(c) Are these terms behaviorally equivalent? Write yes or no.

Recall the following definitions of observational and behavioral equivalence from lecture notes:

- Two terms $s$ and $t$ are observationally equivalent iff either both are normalizable (i.e., they reach a normal form after a finite number of evaluation steps) or both are divergent.

- Terms $s$ and $t$ are behaviorally equivalent iff, for every finite sequence of values $v_1, \ldots, v_n$, the applications

$$s v_1 \ldots v_n$$

and

$$t v_1 \ldots v_n$$

are observationally equivalent.

5. (6 points) $(\lambda x. x x) (\lambda x. x x)$ and $(\lambda x. x x x) (\lambda x. x x x)$.

(a) Answer: none and none

(b) Answer: not applicable

(c) Answer: yes

6. (6 points) $(\lambda x. \lambda y. x)$ and $(\lambda x. \lambda y. (\lambda w. w) x)$.

(a) Answer: $(\lambda x. \lambda y. x)$ and $(\lambda x. \lambda y. (\lambda w. w) x)$

(b) Answer: no

(c) Answer: yes

7. (6 points) $(\lambda x. \lambda y. x) (\lambda z. y)$ and $(\lambda x. \lambda y. x) (\lambda x. w)$.

(a) Answer: $\lambda w. \lambda z. y$ and $\lambda y. \lambda x. w$

(b) Answer: no

(c) Answer: yes
8. (9 points) The following is a slightly different encoding of natural numbers in the untyped lambda calculus.

\[
\begin{align*}
    s_0 &= \lambda s.\lambda z.z \\
    s_1 &= \lambda s.\lambda z.s\ s_0\ z \\
    s_2 &= \lambda s.\lambda z.s\ s_1\ (s\ s_0\ z) \\
    s_3 &= \lambda s.\lambda z.s\ s_2\ (s\ s_1\ (s\ s_0\ z)) \\
    \text{scc} &= \lambda n.\lambda s.\lambda z.s\ n\ (n\ s\ z)
\end{align*}
\]

(a) Define the predecessor function \( \text{prd} \) for this encoding, using the simplest term you can.

\[\text{Answer: } \text{prd} = \lambda n.\ n\ (\lambda m.\ \lambda x.\ m)\ s_0\]

(b) Define the addition function \( \text{plus} \) for this encoding, using the simplest term you can.

\[\text{Answer: } \text{plus} = \lambda n.\ \lambda m.\ n\ (\lambda x.\ \text{scc})\ m\]

or the same definition for \( \text{plus} \) as for Church numerals:

\[\text{plus} = \lambda n.\ \lambda m.\ \lambda s.\ \lambda z.\ n\ s\ (m\ s\ z)\]

(c) Define the function \( \text{sumupto} \) that, given the encoding of a number \( m \), calculates the sum of all the numbers less than or equal to \( m \). Use the simplest term you can, and do not use \( \text{fix} \).

\[\text{Answer: } \text{sumupto} = \lambda m.\ m\ \text{plus}\ m\ is\ the\ simplest\ answer.}\]

Several people gave a function that sums all of the numbers less than \( m \), such as \( \lambda m.\ m\ \text{plus}\ s_0 \). Partial credit was awarded for this function.
Typed arithmetic expressions

The full definition of the language of typed arithmetic and boolean expressions is reproduced, for your reference, on page 10.

9. (6 points) Suppose we add the following two new rules to the evaluation relation:

\[
\text{pred true} \rightarrow \text{pred false} \\
\text{pred false} \rightarrow \text{pred true}
\]

Which of the following properties will remain true in the presence of this rule? For each one, circle either “remains true” or else “becomes false.” If a property becomes false, also write down a counterexample to the property.

(a) Termination of evaluation (for every term \( t \) there is some normal form \( t' \) such that \( t \rightarrow^* t' \))

remains true becomes false, because ....

Answer: Becomes false.

(b) Progress (if \( t \) is well typed, then either \( t \) is a value or else \( t \rightarrow t' \) for some \( t' \))

remains true becomes false, because ....

Answer: Remains true

(c) Preservation (if \( t \) has type \( T \) and \( t \rightarrow t' \), then \( t' \) also has type \( T \))

remains true becomes false, because ....

Answer: Remains true

10. (6 points) Suppose, instead, that we add this new rule to the typing relation:

\[
\frac{t_2 : \text{Nat}}{\text{if true then } t_2 \text{ else } t_3 : \text{Nat}}
\]

Which of the following properties remains true? (Answer in the same style as the previous question.)

(a) Termination of evaluation (for every term \( t \) there is some normal form \( t' \) such that \( t \rightarrow^* t' \))

remains true becomes false, because ....

Answer: Remains true

(b) Progress (if \( t \) is well typed, then either \( t \) is a value or else \( t \rightarrow t' \) for some \( t' \))

remains true becomes false, because ....

Answer: Remains true

(c) Preservation (if \( t \) has type \( T \) and \( t \rightarrow t' \), then \( t' \) also has type \( T \))

remains true becomes false, because ....
11. (6 points) Suppose, instead, that we add this new rule to the typing relation:

\[
\begin{array}{c}
t : \text{Bool} \\
\text{succ } t : \text{Bool}
\end{array}
\]

Which of the following properties remains true? (Answer in the same style as the previous question.)

(a) Termination of evaluation (for every term \( t \) there is some normal form \( t' \) such that \( t \rightarrow^* t' \))

remains true becomes false, because ....

\textit{Answer: Remains true}

(b) Progress (if \( t \) is well typed, then either \( t \) is a value or else \( t \rightarrow t' \) for some \( t' \))

remains true becomes false, because ....

\textit{Answer: Becomes false. \texttt{succ true} is well-typed, but stuck.}

(c) Preservation (if \( t \) has type \( T \) and \( t \rightarrow t' \), then \( t' \) also has type \( T \))

remains true becomes false, because ....

\textit{Answer: Remains true}
For reference: Boolean and arithmetic expressions

Syntax

t ::= 
  true
  false
  if t then t else t
  0
  succ t
  pred t
  iszero t

v ::= 
  true
  false
  nv

nv ::= 
  0
  succ nv

T ::= 
  Bool
  Nat

Evaluation

\[
\begin{align*}
\text{if true then } t_2 \text{ else } t_3 & \rightarrow t_2 \\
\text{if false then } t_2 \text{ else } t_3 & \rightarrow t_3
\end{align*}
\]

\[
\frac{t_1 \rightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3}
\]

\[
\frac{t_1 \rightarrow t_1'}{\text{succ } t_1 \rightarrow \text{succ } t_1'}
\]

\[
\frac{\text{pred } 0 \rightarrow 0}{\text{pred } (\text{succ } n v_1) \rightarrow n v_1}
\]

\[
\frac{t_1 \rightarrow t_1'}{\text{pred } t_1 \rightarrow \text{pred } t_1'}
\]

\[
\frac{\text{iszero } 0 \rightarrow \text{true}}{\text{iszero } (\text{succ } n v_1) \rightarrow \text{false}}
\]

\[
\frac{t_1 \rightarrow t_1'}{\text{iszero } t_1 \rightarrow \text{iszero } t_1'}
\]

continued on next page...
Typing

true : Bool
false : Bool

\( t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T \)

if \( t_1 \) then \( t_2 \) else \( t_3 \) : T

0 : Nat

\( t_1 : \text{Nat} \)

suc \( t_1 \) : Nat

\( t_1 : \text{Nat} \)

pred \( t_1 \) : Nat

\( t_1 : \text{Nat} \)

iszero \( t_1 \) : Bool
For reference: Untyped lambda calculus

Syntax

\[ t ::= \]
\[ \lambda x.t \]
\[ t \cdot t \]

\[ v ::= \]
\[ \lambda x: T. t \]

Evaluation

\[
\frac{t_1 \rightarrow t_1'}{t_1 t_2 \rightarrow t_1' t_2} \quad (E-\text{APP1})
\]

\[
\frac{t_2 \rightarrow t_2'}{v_1 t_2 \rightarrow v_1 t_2'} \quad (E-\text{APP2})
\]

\[
(\lambda x. t_{12}) v_2 \rightarrow [x \mapsto v_2] t_{12} \quad (E-\text{APPABS})
\]