

CSCI 3155: Review (Draft)

DRAFT: November 7, 2017

- Please begin by acknowledging the CU Honor Code Pledge by signing in the space below the pledge.

On my honor, as a University of Colorado at Boulder student, I will neither give nor receive unauthorized assistance on this work.

Signed: _____

- Please then print your name on this page in the space provided below. Please also write your name at the top of each page in case the pages become separated.
- You have 75 minutes for this exam.
- This exam has 4 questions for a total of 150 points.
- Answer the questions in the spaces provided on the front side of the question sheets. You may use the back side for scratch work. The back side should not contain answers unless clearly marked and referenced.
- This exam has 10 pages, including 1 additional scratch page.

Name: _____

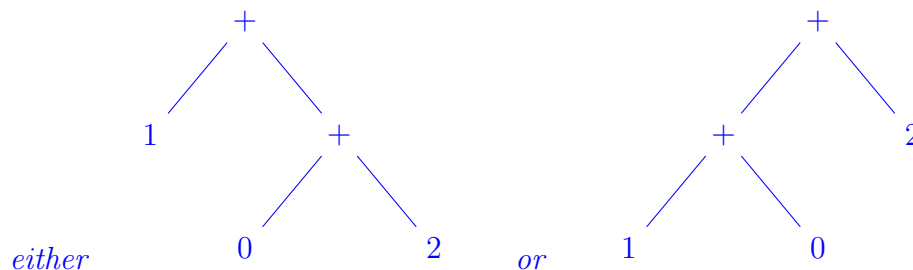
Question:	1	2	3	4	Total
Points:	24	30	46	50	150
Score:					

Natural numbers	$n ::= \text{Zero} \mid \text{Succ}(n)$	
Expressions	$e ::= (x) \Rightarrow \{e_1\}$	Function value
	$ e_1 (e_2)$	Function call
	$ x$	Variable use
	$ e_1 + e_2$	Addition
	$ n$	Number

Figure 1: Syntax definitions for natural numbers and expressions.

1. **Abstract syntax.** Consider the syntax definitions for natural numbers n and expressions e in Figure 1.

- 2 points Using the syntax of non-terminal n , encode the numeral constant “0”.
Zero
- 2 points Using the syntax of non-terminal n , encode the numeral constant “3”.
Succ(Succ(Succ(Zero)))
- 4 points Using the syntax of non-terminal e , encode “1 + 0 + 2” as a parse tree.



- 5 points Give two distinct parse trees for “1 + 0 + 2 + 0”. *(solution for this problem is missing/pending)*
- 5 points How many distinct parse trees exist for “1 + 0 + 2 + 0”, in total? (You do *not* need to draw them all, just count them). *Five.*
- 3 points How many distinct parse trees exist for “ $f (g (x))$ ”? *One.*
- 3 points How many distinct parse trees exist for “ $f (g) (x)$ ”? *One.*

2. **Variable definitions and uses.** Each answer is a (possibly empty) list of variables.

- Consider the expression $(x) \Rightarrow \{(y) \Rightarrow \{x + y\}\}$
 - 3 points What variables are *defined* in this expression? *x, y*
 - 3 points What variables are *used* in this expression? *x, y*
 - 3 points What variables are *used and not defined* in this expression? *None.*
- Consider the expression $f (g (x))$
 - 3 points What variables are *defined* in this expression? *None*
 - 3 points What variables are *used* in this expression? *f, g, x*

- iii. 3 points What variables are *used and not defined* in this expression? f, g, x
- (c) Consider the expression $(f) \Rightarrow \{(x) \Rightarrow \{g(f(x))\}\} ((y) \Rightarrow \{y\})$
- i. 4 points What variables are *defined* in this expression? f, x, y
- ii. 4 points What variables are *used* in this expression? g, f, x, y
- iii. 4 points What variables are *used and not defined* in this expression? g

3. Judgements.

- (a) Consider the syntax definitions for natural numbers
- n
- in Figure 1.

Suppose we want to define a judgement, written formally as $n_1 \text{ plus } n_2 \text{ is } n_3$ and understood informally as “the natural numbers n_1 and n_2 sum to n_3 ”. For example, the conclusions $\text{Zero plus Zero is Zero}$, and $\text{Succ}(\text{Zero}) \text{ plus Succ}(\text{Zero}) \text{ is Succ}(\text{Succ}(\text{Zero}))$ are each derivable in this judgement.

- i. 10 points Give two inference rules to define the judgement $n_1 \text{ plus } n_2 \text{ is } n_3$, as described above. Label one rule “zero” and the other “succ”, for the base case and inductive case, respectively.

$$\frac{}{\text{Zero plus } n \text{ is } n} \text{ZERO} \qquad \frac{n_1 \text{ plus } n_2 \text{ is } n_3}{\text{Succ}(n_1) \text{ plus } n_2 \text{ is Succ}(n_3)} \text{SUCC}$$

- ii. 2 points Give the derivation for $\text{Zero plus Zero is Zero}$. *(solution for this problem is missing/pending)*
- iii. 4 points Give the derivation for $\text{Succ}(\text{Zero}) \text{ plus Succ}(\text{Zero}) \text{ is Succ}(\text{Succ}(\text{Zero}))$ *(solution for this problem is missing/pending)*

- (b) Suppose we wish to define a judgement with the general form $e \Rightarrow_+ n$, which means “expression e contains natural numbers whose total sum is n ”.

Example 1. Suppose that $e = 1 \ (2)$, then $1 \ (2) \Rightarrow_+ 3$ is derivable. Note that this derivation is *not* evaluating e , but merely adding the numbers contained in the abstract syntax of e .

Example 2. Suppose that $e = (x) \Rightarrow \{1\} \ (2)$, then $e \Rightarrow_+ 3$, where $1 \text{ plus } 2 \text{ is } 3$.

- i. 2 points Give a rule for $e \Rightarrow_+ n$ in the case where $e = e_1 + e_2$

$$\frac{e_1 \Rightarrow_+ n_1 \quad e_2 \Rightarrow_+ n_2 \quad n_1 \text{ plus } n_2 \text{ is } n_3}{e_1 + e_2 \Rightarrow_+ n_3} \text{PLUS}$$

- ii. 2 points Give a rule for $e \Rightarrow_+ n$ in the case where $e = (x) \Rightarrow \{e_1\}$

$$\frac{e_1 \Rightarrow_+ n_1}{(x) \Rightarrow \{e_1\} \Rightarrow_+ n_1} \text{FUN}$$

- iii. 2 points Give a rule for $e \Rightarrow_+ n$ in the case where $e = e_1 \ (e_2)$

$$\frac{e_1 \Rightarrow_+ n_1 \quad e_2 \Rightarrow_+ n_2 \quad n_1 \text{ plus } n_2 \text{ is } n_3}{e_1 \ (e_2) \Rightarrow_+ n_3} \text{APP}$$

- iv. 2 points Give a rule for $e \Rightarrow_+ n$ in the case where $e = x$

$$\frac{}{x \Rightarrow_+ 0} \text{VAR}$$

Natural number lists $l ::= \text{Nil} \mid \text{Cons}(n, l)$

Figure 2: Syntax definitions for natural number lists.

- v. 2 points Give a rule for $e \Rightarrow_+ n$ in the case where $e = n'$

$$\frac{}{n' \Rightarrow_+ n'} \text{NUM}$$

- (c) Consider the syntax definitions for natural numbers n in Figure 1, and for natural number lists l in Figure 2.

Suppose we want to define a judgement, written formally as $\text{Append}(l_1, l_2) \Downarrow l_3$ and understood informally as “the list l_3 consists of the numbers of list l_1 followed by numbers of list l_2 ”.

- i. 10 points Give two inference rules to define the judgement $\text{Append}(l_1, l_2) \Downarrow l_3$, as described above. Label one rule “nil” and the other “cons”, for the base case and inductive case, respectively.

$$\frac{}{\text{Append}(\text{Nil}, l) \Downarrow l} \text{NIL} \qquad \frac{\text{Append}(l_1, l_2) \Downarrow l_3}{\text{Append}(\text{Cons}(n, l_1), l_2) \Downarrow \text{Cons}(n, l_3)} \text{CONS}$$

- (d) Suppose we wish to define a judgement with the general form $e \Rightarrow_L l$, which means “expression e contains natural numbers l , from left to right”.

Example. In particular, if $e = (x) \Rightarrow \{1\} (2)$, then $e \Rightarrow_L \text{Cons}(1, \text{Cons}(2, \text{Nil}))$. Note that we are *not* evaluating e , but merely listing the numbers contained in the abstract syntax of e .

- i. 2 points Give a rule for $e \Rightarrow_L l$ in the case where $e = e_1 + e_2$

$$\frac{e_1 \Rightarrow_L l_1 \quad e_2 \Rightarrow_L l_2 \quad \text{Append}(l_1, l_2) \Downarrow l_3}{e_1 + e_2 \Rightarrow_L l_3} \text{PLUS}$$

- ii. 2 points Give a rule for $e \Rightarrow_L l$ in the case where $e = (x) \Rightarrow \{e_1\}$

$$\frac{e_1 \Rightarrow_L l_1}{(x) \Rightarrow \{e_1\} \Rightarrow_L l_1} \text{FUN}$$

- iii. 2 points Give a rule for $e \Rightarrow_L l$ in the case where $e = e_1 (e_2)$

$$\frac{e_1 \Rightarrow_+ l_1 \quad e_2 \Rightarrow_+ l_2 \quad \text{Append}(l_1, l_2) \Downarrow l_3}{e_1 (e_2) \Rightarrow_+ l_3} \text{APP}$$

Environments	$E ::= \cdot$	Empty environment
	$ E, x \mapsto v$	Extended environment
Values	$v ::= (x) \Rightarrow \{e_1\}$	Function value
	$ n$	Number value

Figure 3: Syntax definitions for values v and environments E .

- iv. 2 points Give a rule for $e \Rightarrow_L l$ in the case where $e = x$

$$\frac{}{x \Rightarrow_+ \text{Nil}} \text{VAR}$$

- v. 2 points Give a rule for $e \Rightarrow_L l$ in the case where $e = n$

$$\frac{}{n \Rightarrow_+ \text{Cons}(n, \text{Nil})} \text{NUM}$$

4. Judgements for operational semantics

Consider the syntax for expressions e in Figure 1, and the corresponding syntax for values v , in Figure 3.

Notation for naturals: Below, we use “1” in place of “ $\text{Succ}(\text{Zero})$ ”, and “2” in place of “ $\text{Succ}(\text{Succ}(\text{Zero}))$ ”, etc. You may do the same in your answers to the questions that follow this note.

(a) **Evaluation judgement.**

Consider the evaluation judgement $E \vdash e \Downarrow v$ defined by the following four rules:

$$\begin{array}{c}
 \text{EVAL-VAR} \qquad \text{EVAL-VAL} \qquad \text{EVAL-PLUS} \\
 \frac{}{E \vdash x \Downarrow E(x)} \qquad \frac{}{E \vdash v \Downarrow v} \qquad \frac{E \vdash e_1 \Downarrow n_1 \quad E \vdash e_2 \Downarrow n_2 \quad n_1 \text{ plus } n_2 \text{ is } n_3}{E \vdash e_1 + e_2 \Downarrow n_3} \\
 \\
 \text{EVAL-CALL} \\
 \frac{E \vdash e_1 \Downarrow (x) \Rightarrow \{e'_1\} \quad E \vdash e_2 \Downarrow v_2 \quad E, x \mapsto v_2 \vdash e'_1 \Downarrow v_3}{E \vdash e_1 (e_2) \Downarrow v_3}
 \end{array}$$

- i. 5 points Consider the statement $E \vdash (x) \Rightarrow \{\text{Zero}\} (\text{Zero}) \Downarrow v$. Is it derivable? If yes, give the value v . If not, say what “goes wrong”. Yes, $v = \text{Zero}$
- ii. 10 points Consider the statement $E \vdash (x) \Rightarrow \{(y) \Rightarrow \{x + y\}\} (1) (2) \Downarrow v$. Is it derivable? If yes, give the value v . If not, say what “goes wrong”. No, when evaluating subexpression $x + y$, variable x is *undefined* (not in the environment).

(b) **Stepping judgement.**

Consider the stepping judgement $e_1 \longrightarrow e_2$ defined by the following six rules:

$\frac{\text{STEP-CALL1} \quad e_1 \longrightarrow e'_1}{e_1 (e_2) \longrightarrow e'_1 (e_2)}$	$\frac{\text{STEP-CALL2} \quad e_2 \longrightarrow e'_2}{e_1 (e_2) \longrightarrow e_1 (e'_2)}$	$\frac{\text{STEP-CALL3}}{(x) \Rightarrow \{e\} (v) \longrightarrow e[v/x]}$
$\frac{\text{STEP-PLUS1} \quad e_1 \longrightarrow e'_1}{e_1 + e_2 \longrightarrow e'_1 + e_2}$	$\frac{\text{STEP-PLUS2} \quad e_2 \longrightarrow e'_2}{e_1 + e_2 \longrightarrow e_1 + e'_2}$	$\frac{\text{STEP-PLUS3} \quad n_1 \text{ plus } n_2 \text{ is } n_3}{n_1 + n_2 \longrightarrow n_3}$

Notation for multiple steps: Recall that we write $e_1 \longrightarrow^* e_2$ to mean “ e_1 steps to e_2 in zero or more single steps”. That is, $e_1 \longrightarrow^* e_2$ means that either $e_1 = e_2$, or there exists e'_1 such that $e_1 \longrightarrow e'_1$ and $e'_1 \longrightarrow^* e_2$.

- i. 5 points Consider the statement $(x) \Rightarrow \{\text{Zero}\} (\text{Zero}) \longrightarrow^* v$.
Is it derivable? If yes, give the value v . If not, say what “goes wrong”. Yes, $v = \text{Zero}$
- ii. 10 points Consider the statement $(x) \Rightarrow \{(y) \Rightarrow \{x + y\}\} (1) (2) \longrightarrow^* v$.
Is it derivable? If yes, give the value v . If not, say what goes “wrong”. Yes, $v = \text{“3”} = \text{Succ}(\text{Succ}(\text{Succ}(\text{Zero})))$

(c) **Determinism of operational semantics.**

- i. 5 points **Determinism of evaluation.** Give a short program e such that e evaluates to v_1 (that is, $\cdot \vdash e \Downarrow v_1$), and e evaluates to v_2 (that is, $\cdot \vdash e \Downarrow v_2$) and such that $v_1 \neq v_2$. If no such program exists, say “no such program exists”, and briefly explain why (just a few words will suffice). No such program exists; the evaluation judgement is deterministic
- ii. 5 points **Determinism of single steps.** Give a short program e such that e steps to expression e_1 (that is, $e \longrightarrow e_1$), and e steps to e_2 (that is, $e \longrightarrow e_2$) and such that $e_1 \neq e_2$. If no such program exists, say “no such program exists”, and briefly explain why (just a few words will suffice). (1 + 2) + (3 + 4)
- iii. 5 points **Determinism of completed steppings (stepping to values).** Give a short program e such that e steps to value v_1 (that is, $e \longrightarrow^* v_1$), and e steps to v_2 (that is, $e \longrightarrow^* v_2$) and such that $v_1 \neq v_2$. If no such program exists, say “no such program exists”, and briefly explain why (just a few words will suffice). No such program exists; completed steppings are deterministic
- iv. 5 points **Evaluation vs stepping.** Give a short program e such that it evaluates to v_1 (that is, $\cdot \vdash e \Downarrow v_1$), it steps to v_2 (that is, $e \longrightarrow^* v_2$) and such that $v_1 \neq v_2$. If no such program exists, say “no such program exists”, and briefly explain why (just a few words will suffice).

Hint: Notice that we are asking you to compare evaluation and stepping here, that these are different definitions, and that one uses an environment and the other uses substitution. (x) => {(x) => {(y) => {y + x}} (1) (2)} (0)

Types	τ	::=	num	Numbers. Inhabitant form n .
			$\tau_1 \rightarrow \tau_2$	Functions from τ_1 to τ_2 , inhabitant form $(x) \Rightarrow \{e\}$
			ptr (τ)	Pointer type, inhabitant form ℓ
Typing contexts	Γ	::=	\cdot	Empty typing contexts
			$\Gamma, x : \tau$	Typing contexts map variables x to types τ
			$\Gamma, \ell : \tau$	Typing contexts also map <i>allocated pointers</i> ℓ to types τ
Values	v	::=	$n \mid (x) \Rightarrow \{e\} \mid \ell$	Values consist of numbers, functions and pointers
Memory	M	::=	$\cdot \mid (M, \ell \mapsto v)$	Memory is a finite map from pointers to values
Expressions	e	::=	$(x) \Rightarrow \{e_1\}$	Function value
			$e_1 (e_2)$	Function call
			x	Variable use
			$e_1 + e_2$	Addition
			n	Number
			new (e)	Allocate a pointer, holding value of e
			get (e)	Get the current value of pointer e
			$e_1 := e_2$	Assign the pointer e_1 the value of e_2
			ℓ	Pointer, names a “current value” in memory

Figure 4: This figure extends Figure 1 with values v , and expression forms for pointers

1 New Material: Types and State

$$\frac{\text{STEP-CALL1} \quad \langle M_1, e_1 \rangle \longrightarrow \langle M_2, e'_1 \rangle}{\langle M_1, e_1 (e_2) \rangle \longrightarrow \langle M_2, e'_1 (e_2) \rangle}$$

$$\frac{\text{STEP-CALL2} \quad \langle M_1, e_2 \rangle \longrightarrow \langle M_2, e'_2 \rangle}{\langle M_1, v_1 (e_2) \rangle \longrightarrow \langle M_2, v_1 (e'_2) \rangle}$$

$$\frac{\text{STEP-CALL3}}{\langle M, (x) \Rightarrow \{e\} (v) \rangle \longrightarrow \langle M, e[v/x] \rangle}$$

$$\frac{\text{STEP-PLUS1} \quad \langle M_1, e_1 \rangle \longrightarrow \langle M_2, e'_1 \rangle}{\langle M_1, e_1 + e_2 \rangle \longrightarrow \langle M_2, e'_1 + e_2 \rangle}$$

$$\frac{\text{STEP-PLUS2} \quad \langle M_1, e_2 \rangle \longrightarrow \langle M_2, e'_2 \rangle}{\langle M_1, v_1 + e_2 \rangle \longrightarrow \langle M_2, v_1 + e'_2 \rangle}$$

$$\frac{\text{STEP-PLUS3} \quad n_1 \text{ plus } n_2 \text{ is } n_3}{\langle M, n_1 + n_2 \rangle \longrightarrow \langle M, n_3 \rangle}$$

$$\frac{\text{STEP-PTRNEW1} \quad \langle M_1, e_1 \rangle \longrightarrow \langle M_2, e'_1 \rangle}{\langle M_1, \text{new}(e_1) \rangle \longrightarrow \langle M_2, \text{new}(e'_1) \rangle}$$

$$\frac{\text{STEP-PTRNEW2} \quad \ell \notin \text{dom}(M)}{\langle M, \text{new}(v_1) \rangle \longrightarrow \langle (M, \ell \mapsto v_1), v_1 \rangle}$$

$$\frac{\text{STEP-PTRGET1} \quad \langle M_1, e_1 \rangle \longrightarrow \langle M_2, e'_1 \rangle}{\langle M_1, \text{get}(e_1) \rangle \longrightarrow \langle M_2, \text{get}(e'_1) \rangle}$$

$$\frac{\text{STEP-PTRGET2} \quad M(\ell) = v}{\langle M, \text{get}(\ell) \rangle \longrightarrow \langle M, v \rangle}$$

$$\frac{\text{STEP-PTRASSIGN1} \quad \langle M_1, e_1 \rangle \longrightarrow \langle M_2, e'_1 \rangle}{\langle M_1, e_1 := e_2 \rangle \longrightarrow \langle M_2, e'_1 := e_2 \rangle}$$

$$\frac{\text{STEP-PTRASSIGN2} \quad \langle M_1, e_2 \rangle \longrightarrow \langle M_2, e'_2 \rangle}{\langle M_1, v_1 := e_2 \rangle \longrightarrow \langle M_2, v_1 := e'_2 \rangle}$$

$$\frac{\text{STEP-PTRASSIGN3}}{\langle M, \ell := v \rangle \longrightarrow \langle (M, \ell \mapsto v), v \rangle}$$

Practice defining the typing rules:

Suppose that we want to give a *type system* for the imperative language in Figure 4. (You may see something like this on the exam).

1. Give a typing rule for expressions e when $e = x$
2. Give a typing rule for expressions e when $e = n$
3. Give a typing rule for expressions e when $e = \ell$
4. Give a typing rule for expressions e when $e = (x) \Rightarrow \{e_1\}$
5. Give a typing rule for expressions e when $e = e_1 (e_2)$

6. Give a typing rule for expressions e when $e = e_1 + e_2$
7. Give a typing rule for expressions e when $e = \mathbf{new}(e_1)$
8. Give a typing rule for expressions e when $e = \mathbf{get}(e_1)$
9. Give a typing rule for expressions e when $e = e_1 := e_2$

Practice using the typing rules:

Suppose that we want to use the type system that you just defined to type programs.

For each question below, give the derivation in question, **or say “none exists”** if no typing derivation exists for the given program.

(Hint: Doing these exercises will be much more time-consuming than the corresponding final exam question, and will serve as critical practice for that question).

1. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = x$
2. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = x + y$
3. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = (x) \Rightarrow \{(y) \Rightarrow \{x + y\}\}$
4. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = (x) \Rightarrow \{(y) \Rightarrow \{\mathbf{get}(x) + 1\}\}$
5. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = (x) \Rightarrow \{(y) \Rightarrow \{1 + \mathbf{get}(y)\}\}$
6. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = (x) \Rightarrow \{(y) \Rightarrow \{x \ (y)\}\}$
7. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = (x) \Rightarrow \{(y) \Rightarrow \{\mathbf{get}(x) \ (y)\}\}$
8. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = (x) \Rightarrow \{(y) \Rightarrow \{x \ (\mathbf{get}(y))\}\}$