CSCI 3155: Review (Draft)

DRAFT: December 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 **??** questions for a total of **??** 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 **??** pages, including 1 additional scratch page.

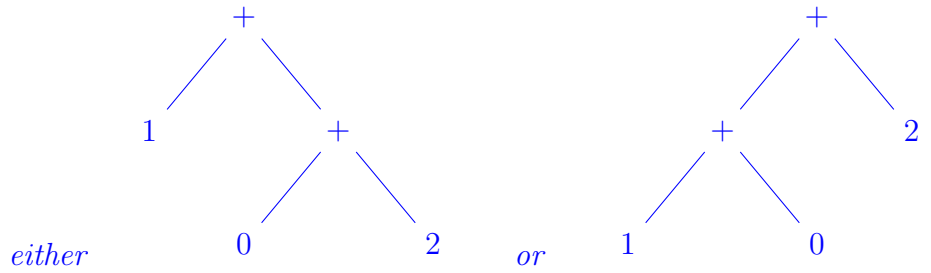
Name: _____

Run LaTeX again to produce the table

Natural numbers	n	::=	`Zero` | `Succ(`n`)`	
Expressions	e	::=	`(`x`)=>{`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.
   i. [2 points] Using the syntax of non-terminal n, encode the numeral constant "0". `Zero`
   ii. [2 points] Using the syntax of non-terminal n, encode the numeral constant "3". `Succ(Succ(Succ(Zero)))`
   iii. [4 points] Using the syntax of non-terminal e, encode "$1 + 0 + 2$" as a parse tree.



   iv. [5 points] Give two distinct parse trees for "$1+0+2+0$". *(solution for this problem is missing/pending)*
   v. [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.
   vi. [3 points] How many distinct parse trees exist for "f `(`g `(`x`))`"? One.
   vii. [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.
   (a) Consider the expression `(`x`)=>{(`y`)=>{`$x + y$`}}`
      i. [3 points] What variables are *defined* in this expression? x, y
      ii. [3 points] What variables are *used* in this expression? x, y
      iii. [3 points] What variables are *used and not defined* in this expression? None.
   (b) Consider the expression f `(`g `(`x`))`
      i. [3 points] What variables are *defined* in this expression? None
      ii. [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)=>{(x)=>{g (f (x))}} ((y)=>{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 plus n_2 is n_3 and understood informally as "the natural numbers n_1 and n_2 sum to n_3". For example, the conclusions `Zero` plus `Zero` is `Zero`, and `Succ(Zero)` plus `Succ(Zero)` is `Succ(Succ(Zero))` are each derivable in this judgement.

i. [10 points] Give two inference rules to define the judgement n_1 plus n_2 is n_3, as described above. Label one rule "zero" and the other "succ", for the base case and inductive case, respectively.

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

ii. [2 points] Give the derivation for `Zero` plus `Zero` is `Zero`. *(solution for this problem is missing/pending)*

iii. [4 points] Give the derivation for `Succ(Zero)` plus `Succ(Zero)` is `Succ(Succ(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 $e = (x)\texttt{=>}\{1\}\ (2)$, then $e \Rightarrow_+ 3$, where 1 plus 2 is 3.

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

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

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

$$\frac{e_1 \Rightarrow_+ n_1}{(x)\texttt{=>}\{e_1\} \Rightarrow_+ n_1}\text{FUN}$$

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

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

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 ::= $\texttt{Nil} \mid \texttt{Cons}(n, l)$

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 $\mathsf{Append}(l_1, l_2) \Downarrow l_3$ 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 $\mathsf{Append}(l_1, l_2) \Downarrow l_3$, as described above. Label one rule "nil" and the other "cons", for the base case and inductive case, respectively.

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

(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 $e = \texttt{(}x\texttt{)=>\{1\} (2)}$, then $e \Rightarrow_L \texttt{Cons(1,Cons(2,Nil))}$. 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 $e = e_1 + e_2$

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

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

$$\frac{e_1 \Rightarrow_L l_1}{\texttt{(}x\texttt{)=>\{}e_1\texttt{\}} \Rightarrow_L l_1}\text{FUN}$$

iii. [2 points] Give a rule for $e \Rightarrow_L l$ in the case where $e = e_1\ \texttt{(}e_2\texttt{)}$

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

Environments	E	::=	$\cdot$	Empty environment
		|	$E, x \mapsto v$	Extended environment
Values	v	::=	`(x)=>{`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 $e = x$

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

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

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

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 "`Succ(Zero)`", and "2" in place of "`Succ(Succ(Zero))`", etc. You may do the same in your answsers 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:

$$\frac{}{E \vdash x \Downarrow E(x)}\text{Eval-Var} \qquad \frac{}{E \vdash v \Downarrow v}\text{Eval-Val}$$

$$\frac{E \vdash e_1 \Downarrow n_1 \qquad E \vdash e_2 \Downarrow n_2 \qquad n_1 \,\mathsf{plus}\, n_2 \,\mathsf{is}\, n_3}{E \vdash e_1 + e_2 \Downarrow n_3}\text{Eval-Plus}$$

$$\frac{E \vdash e_1 \Downarrow \texttt{(x)=>\{}e_1'\texttt{\}} \qquad E \vdash e_2 \Downarrow v_2 \qquad E, x \mapsto v_2 \vdash e_1' \Downarrow v_3}{E \vdash e_1\ \texttt{(}e_2\texttt{)} \Downarrow v_3}\text{Eval-Call}$$

i. [5 points] Consider the statement $E \vdash$ `(x)=>{Zero} (Zero)` $\Downarrow v$.
Is it derivable? If yes, give the value v. If not, say what "goes wrong". Yes, $v = \texttt{Zero}$

ii. [10 points] Consider the statement $E \vdash$ `(x)=>{(y)=>{`$x + y$`}} (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).

$\diamond$

(b) **Stepping judgement**.

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

$$\textsc{Step-Call1}\ \frac{e_1 \longrightarrow e_1'}{e_1\ (e_2) \longrightarrow e_1'\ (e_2)} \qquad \textsc{Step-Call2}\ \frac{e_2 \longrightarrow e_2'}{e_1\ (e_2) \longrightarrow e_1\ (e_2')} \qquad \textsc{Step-Call3}\ \frac{}{(x)\texttt{=>}\{e\}\ (v) \longrightarrow e[v/x]}$$

$$\textsc{Step-Plus1}\ \frac{e_1 \longrightarrow e_1'}{e_1 + e_2 \longrightarrow e_1' + e_2} \qquad \textsc{Step-Plus2}\ \frac{e_2 \longrightarrow e_2'}{e_1 + e_2 \longrightarrow e_1 + e_2'} \qquad \textsc{Step-Plus3}\ \frac{n_1\ \mathsf{plus}\ n_2\ \mathsf{is}\ n_3}{n_1 + n_2 \longrightarrow n_3}$$

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 $(x)\texttt{=>}\{\texttt{Zero}\}\ (\texttt{Zero}) \longrightarrow^* v$.
Is it derivable? If yes, give the value v. If not, say what "goes wrong". Yes, $v = \texttt{Zero}$

ii. [10 points] Consider the statement $(x)\texttt{=>}\{(y)\texttt{=>}\{x + y\}\}\ (1)\ (2) \longrightarrow^* v$.
Is it derivable? If yes, give the value v. If not, say what goes "wrong". Yes, $v =$ "3" $= \texttt{Succ(Succ(Succ(Zero)))}$

(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. $(x)\texttt{=>}\{(x)\texttt{=>}\{(y)\texttt{=>}\{y + x\}\}\ (1)\ (2)\}\ (0)$

$\diamond$

Types	τ	::=	`num`	Numbers. Inhabitant form n.
		|	$\tau_1 \to \tau_2$	Functions from τ_1 to τ_2, inhabitant form `(`x`)=>{`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 *allocated pointers* ℓ to types τ
Values	v	::=	n | `(`x`)=>{`e`}` | ℓ	Values consist of numbers, functions and pointers
Memory	M	::=	$\cdot$ | $(M, \ell \mapsto v)$	Memory is a finite map from pointers to values
Expressions	e	::=	`(`x`)=>{`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

$$\textsc{Step-Call1}\ \frac{\langle M_1, e_1\rangle \longrightarrow \langle M_2, e_1'\rangle}{\langle M_1, e_1\ \texttt{(}e_2\texttt{)}\rangle \longrightarrow \langle M_2, e_1'\ \texttt{(}e_2\texttt{)}\rangle}$$

$$\textsc{Step-Call2}\ \frac{\langle M_1, e_2\rangle \longrightarrow \langle M_2, e_2'\rangle}{\langle M_1, v_1\ \texttt{(}e_2\texttt{)}\rangle \longrightarrow \langle M_2, v_1\ \texttt{(}e_2'\texttt{)}\rangle}$$

$$\textsc{Step-Call3}\ \frac{}{\langle M, \texttt{(}x\texttt{)=>\{}e\texttt{\}\ (}v\texttt{)}\rangle \longrightarrow \langle M, e[v/x]\rangle}$$

$$\textsc{Step-Plus1}\ \frac{\langle M_1, e_1\rangle \longrightarrow \langle M_2, e_1'\rangle}{\langle M_1, e_1 + e_2\rangle \longrightarrow \langle M_2, e_1' + e_2\rangle}$$

$$\textsc{Step-Plus2}\ \frac{\langle M_1, e_2\rangle \longrightarrow \langle M_2, e_2'\rangle}{\langle M_1, v_1 + e_2\rangle \longrightarrow \langle M_2, v_1 + e_2'\rangle}$$

$$\textsc{Step-Plus3}\ \frac{n_1\ \mathsf{plus}\ n_2\ \mathsf{is}\ n_3}{\langle M, n_1 + n_2\rangle \longrightarrow \langle M, n_3\rangle}$$

$$\textsc{Step-PtrNew1}\ \frac{\langle M_1, e_1\rangle \longrightarrow \langle M_2, e_1'\rangle}{\langle M_1, \texttt{new}(e_1)\rangle \longrightarrow \langle M_2, \texttt{new}(e_1')\rangle}$$

$$\textsc{Step-PtrNew2}\ \frac{\ell \notin \mathsf{dom}(M)}{\langle M, \texttt{new}(v_1)\rangle \longrightarrow \langle (M, \ell{\mapsto}v_1), v_1\rangle}$$

$$\textsc{Step-PtrGet1}\ \frac{\langle M_1, e_1\rangle \longrightarrow \langle M_2, e_1'\rangle}{\langle M_1, \texttt{get}(e_1)\rangle \longrightarrow \langle M_2, \texttt{get}(e_1')\rangle}$$

$$\textsc{Step-PtrGet2}\ \frac{M(\ell) = v}{\langle M, \texttt{get}(\ell)\rangle \longrightarrow \langle M, v\rangle}$$

$$\textsc{Step-PtrAssign1}\ \frac{\langle M_1, e_1\rangle \longrightarrow \langle M_2, e_1'\rangle}{\langle M_1, e_1\ \texttt{:=}\ e_2\rangle \longrightarrow \langle M_2, e_1'\ \texttt{:=}\ e_2\rangle}$$

$$\textsc{Step-PtrAssign2}\ \frac{\langle M_1, e_2\rangle \longrightarrow \langle M_2, e_2'\rangle}{\langle M_1, v_1\ \texttt{:=}\ e_2\rangle \longrightarrow \langle M_2, v_1\ \texttt{:=}\ e_2'\rangle}$$

$$\textsc{Step-PtrAssign3}\ \frac{}{\langle M, \ell\ \texttt{:=}\ v\rangle \longrightarrow \langle (M, \ell{\mapsto}v), v\rangle}$$

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 $e = \texttt{(}x\texttt{)=>\{}e_1\texttt{\}}$

5. Give a typing rule for expressions e when $e = e_1\ \texttt{(}e_2\texttt{)}$

6. Give a typing rule for expressions e when $e = e_1 + e_2$

7. Give a typing rule for expressions e when $e = \texttt{new}(e_1)$

8. Give a typing rule for expressions e when $e = \texttt{get}(e_1)$

9. Give a typing rule for expressions e when $e = e_1 \texttt{ := } e_2$

Practice using the typing rules:
Suppose that we want to use the type system that you just defined to type programs.
(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 $e = \texttt{(}x\texttt{)=>\{(}y\texttt{)=>\{}x + y\texttt{\}\}}$

2. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = \texttt{(}x\texttt{)=>\{(}y\texttt{)=>\{get(}x\texttt{)} + 1\texttt{\}\}}$

3. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = \texttt{(}x\texttt{)=>\{(}y\texttt{)=>\{}1 + \texttt{get(}y\texttt{)\}\}}$

4. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = \texttt{(}x\texttt{)=>\{(}y\texttt{)=>\{}x\texttt{ (}y\texttt{)\}\}}$

5. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = \texttt{(}x\texttt{)=>\{(}y\texttt{)=>\{get(}x\texttt{) (}y\texttt{)\}\}}$

6. Give a typing derivation for $\cdot \vdash e : \tau$ when $e = \texttt{(}x\texttt{)=>\{(}y\texttt{)=>\{}x\texttt{ (get(}y\texttt{))\}\}}$

Regexp's	r	::=	n	Atom: Match mingle number constant n.
		|	$r_1 \vee r_2$	Union: Match either r_1 or r_2 (or both).
		|	$r_1; r_2$	Concatenation: Match r_1 followed by r_2.
		|	$(r)^*$	Kleene star: Match r zero or more times.

Figure 5: Syntax definitions for regular expressions.

More practice defining judgements, relations defined *inductively*, with a fixed set of *inference rules*:

1. Define $\mathsf{pred}(n) \Downarrow n'$, read as "the predecessor of n is n'". How many inference rules were required? (Try to do it with one.)

2. Define $\mathsf{eq}(n, n')$, read as "n is equal to n'". How many inference rules were required? (Try to do it with two.)

3. Define $\mathsf{neq}(n, n')$, read as "n is *not* equal to n'". How many inference rules were required? (Try to do it with two.)

4. Define $\mathsf{lte}(n, n')$, read as "n is *less than or equal to* n'". How many inference rules were required? (Try to do it with two.)

5. Define $\mathsf{gt}(n, n')$, read as "n is *greater than or equal to* n'". How many inference rules were required? (Try to do it with two.)

6. Define $\mathsf{minus}(n_1, n_2) \Downarrow n_3$, read as "$n_1$ minus n_2 is n_3". How many inference rules were required? Try to do it with two, and without using any other definitions (don't use plus). Then, try to do it with one, by using plus.

7. Define $\mathsf{count}(l) \Downarrow n$, read as "list l contains n numbers". How many inference rules were required? Try to do it with two.

8. Define $\mathsf{genUp}(n) \Downarrow l$, read as "list l contains numbers $[0, \cdots, n]$". How many inference rules were required? Try to do it with two.

9. Define $\mathsf{genDn}(n) \Downarrow l$, read as "list l contains numbers $[n, \cdots, 0]$". How many inference rules were required? Try to do it with two.

Semantics of regular expression matching:
Consider regular expressions r, whose abstract syntax is given in Figure 5.

1. Write inference rules that define the inductive judgement with the form $\mathsf{match}(r, l_1) \Downarrow l_2$, read as "matching r against the prefix of numbers in list l_1 succeeds, leaving unmatched the list suffix l_2". Or for short, "r matches l_1, leaving l_2".

2. Use the inference rules to derive the following examples.

   (a) $\mathsf{match}(a; (b)^*, [a, c]) \Downarrow [c]$

$\diamond$

(b) $\mathsf{match}(a; (b)^*, [a, b, c]) \Downarrow [c]$

(c) $\mathsf{match}(a; (b)^*, [a, b, b, c]) \Downarrow [c]$

(d) $\mathsf{match}(a \vee b, [a, c]) \Downarrow [c]$

(e) $\mathsf{match}(a \vee b, [b, c]) \Downarrow [c]$

(f) $\mathsf{match}(((b; a) \vee b); b, [b, b, c]) \Downarrow [c]$

3. If we consider r, l_1 as inputs, and list l_2 as an output, is the relation $\mathsf{match}(r, l_1) \Downarrow l_2$ a function?

   That is, does it ever relate one input with several outputs? (If so, recall it's *not* a function). If so, give an example of the input, and the two different outputs. For practice, *construct derivations* of these two different outputs.

4. Add additional syntax forms to r for "options", $r?$ (zero or one matches of some r are permitted), and for "conjunction", $r_1 \wedge r_2$ (*both* of two sub-expressions must match). Add rules for these forms, and write two examples for each.

$\diamond$