Problem 1 (15 points)

Implement the OCaml function `choose` that takes a list `xs` and a *non-negative* integer `n`, and returns a list of all possible ways to choose `n` elements from `xs`. The order of the elements in the returned list does not matter. For example, `choose [1;2;3] 2` should return `[ [1;2]; [1;3]; [2;3] ]` (although your solution can reorder the sub-lists or the integer elements within). Take `choose xs n` to be `[]` if `n` is strictly greater than the length of `xs`. You may assume that `n` is non-negative, and that the input list `xs` has no duplicate elements.

Complete the following code skeleton by filling each labelled box with an OCaml expression from the candidate pool. Each candidate can be used zero, one, or multiple times. You may not use anything other than the candidates provided.

```
let rec choose (n: int) (xs: [1] ) : [2] =
  if n = 0 then [3]
  else
    match xs with
    | [] -> [4]
    | y::ys ->
      let r1 = choose [5] [6] in
      let r2 = choose [7] [8] in
      let r3 = map (fun (zs: [9] ) -> [10] [11] [12] ) r2 in
      r1 [13] r3
```

Candidate pool:

1: `n` 2: `n-1` 3: `n-2`
4: `::` 5: `@` 6: `[]` 7: `[[]]`
8: `xs` 9: `y` 10: `ys` 11: `zs`
12: `[xs]` 13: `[y]` 14: `[ys]` 15: `[zs]`
16: `'a list` 17: `'a list list`

Fill in the table on the right. As an example, we fill [1] with candidate 16, which is `'a list`, since `choose` takes as input a list of items of any type.

Box #	Candidate #
1	16
2	(JR) 17
3	(JR) 7
4	(JR) 6
5	(JR) 1
6	(JR) 10
7	(JR) 2
8	(JR) 10
9	(JR) 16
10	(JR) 9
11	(JR) 4
12	(JR) 11
13	(JR) 5

(JR) Answer:

```
let rec choose (n: int) (xs: 'a list) : 'a list list =
  if n = 0 then [[]]
  else
    (* n > 0 *)
    match xs with
    | [] -> []
    | y::ys ->
      let r1 = choose n ys in
      let r2 = choose (n-1) ys in
      let r3 = map (fun (zs: 'a list) -> y :: zs) r2 in
      r1 @ r3
```

Problem 2 (15 points)

For each the following λ-calculus terms, compute its free variable set.

Expression	**Free variables**
Example: $(x\ y)$	$\{x, y\}$
$\lambda x.\ (\lambda y.\ (f\ x))$	(JR) $\{f\}$
$(x\ (\lambda z.\ \lambda x.\ y))\ ((\lambda y.\ \lambda x.\ z\ y)\ x)$	(JR) $\{x, y, z\}$
$(\lambda z.\ ((\lambda x.\ \lambda y.\ y\ x)\ y))\ z$	(JR) $\{y, z\}$

Problem 3 (15 points)

The following is the definition of the substitution function $c[v \mapsto e]$ you have seen in class:

$$
\begin{aligned}
x[x \mapsto e] &= e \\
y[x \mapsto e] &= y \quad \text{if } y \neq x \\
(\lambda x.c)[x \mapsto e] &= \lambda x.c \\
(\lambda y.c)[x \mapsto e] &= \lambda y.(c[x \mapsto e]) \quad \text{if } y \neq x \wedge y \notin \mathsf{FV}(e) \\
(c_1\, c_2)[x \mapsto e] &= (c_1[x \mapsto e])\,(c_2[x \mapsto e])
\end{aligned}
$$

Recall that in the second last case, if $y \neq x$ but $y \in \mathsf{FV}(e)$, then we need to perform alpha-renaming to avoid variable capture.

For each of the following substitutions $c[x \mapsto y]$, determine whether alpha-renaming is necessary.

- If not, put a "No" in the "Alpha-Renaming Necessary?" column. Then simply write down the result of the substitution $c[v \mapsto e]$ in the "Output" column without performing any alpha-renaming.
- If so, put a "Yes" in the "Alpha-Renaming Necessary?" column. Then alpha-rename c into another c' of your choice (you don't need to exhibit c'). Finally, write down the result of the substitution $c'[v \mapsto e]$ in the "Output" column.

Substitution $c[v \mapsto e]$	**Alpha-Renaming Necessary?**	**Output**
$(x\ (\lambda x.\ y))[y \mapsto x]$	(JR) Yes	(JR) First rename to $x\ (\lambda a.y)$. Then output: $x\ (\lambda a.x)$
$((\lambda y.\ x)\ (\lambda z.\ y))[y \mapsto x]$	(JR) No	(JR) $((\lambda y.\ x)\ (\lambda z.\ x))$
$(\lambda z.\ ((\lambda y.\ y)\ y))[y \mapsto \lambda x.(y\ z)]$	(JR) Yes	(JR) First rename to $\lambda a.\ ((\lambda b.\ b)\ y)$. Then output: $\lambda a.((\lambda b.\ b)\ (\lambda x.(y\ z)))$

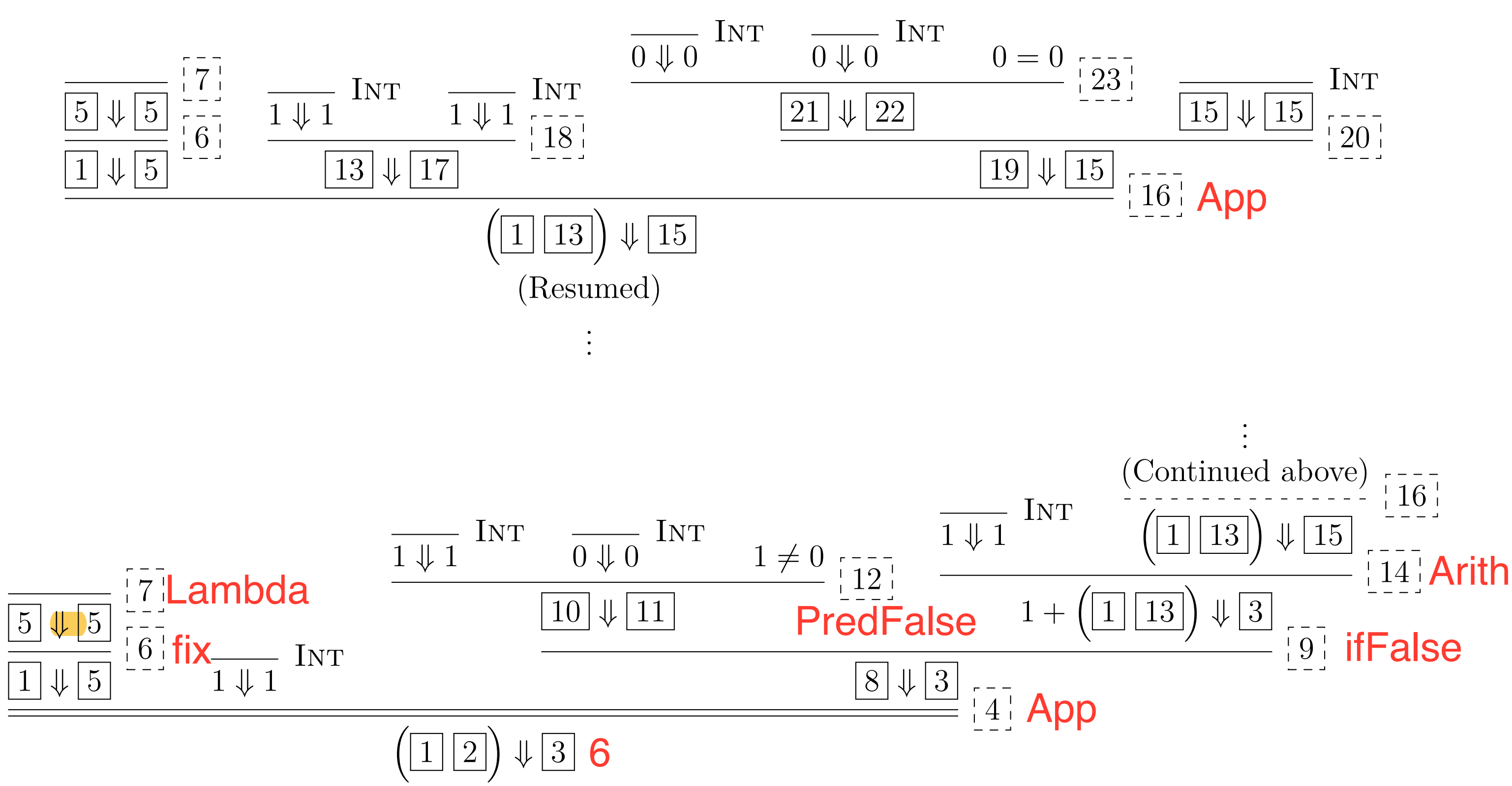
Problem 4 (20 points)

Consider the following (untyped) λ^+ expression:

```
(fix r is lambda n.
    if n = 0 then 5 else n + r (n - 1)) 1
```

Denote the expression above as e. Below is the skeleton of the derivation tree that shows $e \Downarrow v$ for some v. The derivation starts from the bottom of the page with the double bar, and proceeds upwards.

Complete the derivation tree by filling in the table on the next page. A $\boxed{\text{solid box}}$ should be filled with an λ^+ expression, while a dashed box should be filled with the name of an *operational semantics* rule.



Always write the full, concrete expression or rule name in a cell. Do *not* use abbreviations, and do *not* use names ("e", "v", etc.) or numbers to refer to other expressions or rules.

#	Expression/Rule Name
1	`(fix r is lambda n. if n = 0 then 5 else n + r (n - 1))`
2	`1`
3	(JR) 6
4	App
5	`(lambda n. if n = 0 then 5 else n + (fix r is lambda n. if n = 0 then 5 else n + r (n - 1))(n - 1))`
6	(JR) Fix
7	(JR) Lambda
8	`if 1 = 0 then 5 else 1 + (fix r is lambda n. if n = 0 then 5 else n + r (n - 1))(1 - 1))`
9	(JR) IfFalse
10	(JR) 1=0
11	(JR) false

#	Expression/Rule Name
12	(JR) PredFalse
13	(JR) 1-1
14	(JR) Arith
15	(JR) 5
16	(JR) App
17	(JR) 0
18	(JR) Arith
19	`if 0 = 0 then 5 else 0 + (fix r is lambda n. if n = 0 then 5 else n + r (n - 1))(0 - 1))`
20	(JR) IfTrue
21	(JR) 0=0
22	(JR) true
23	(JR) PredTrue

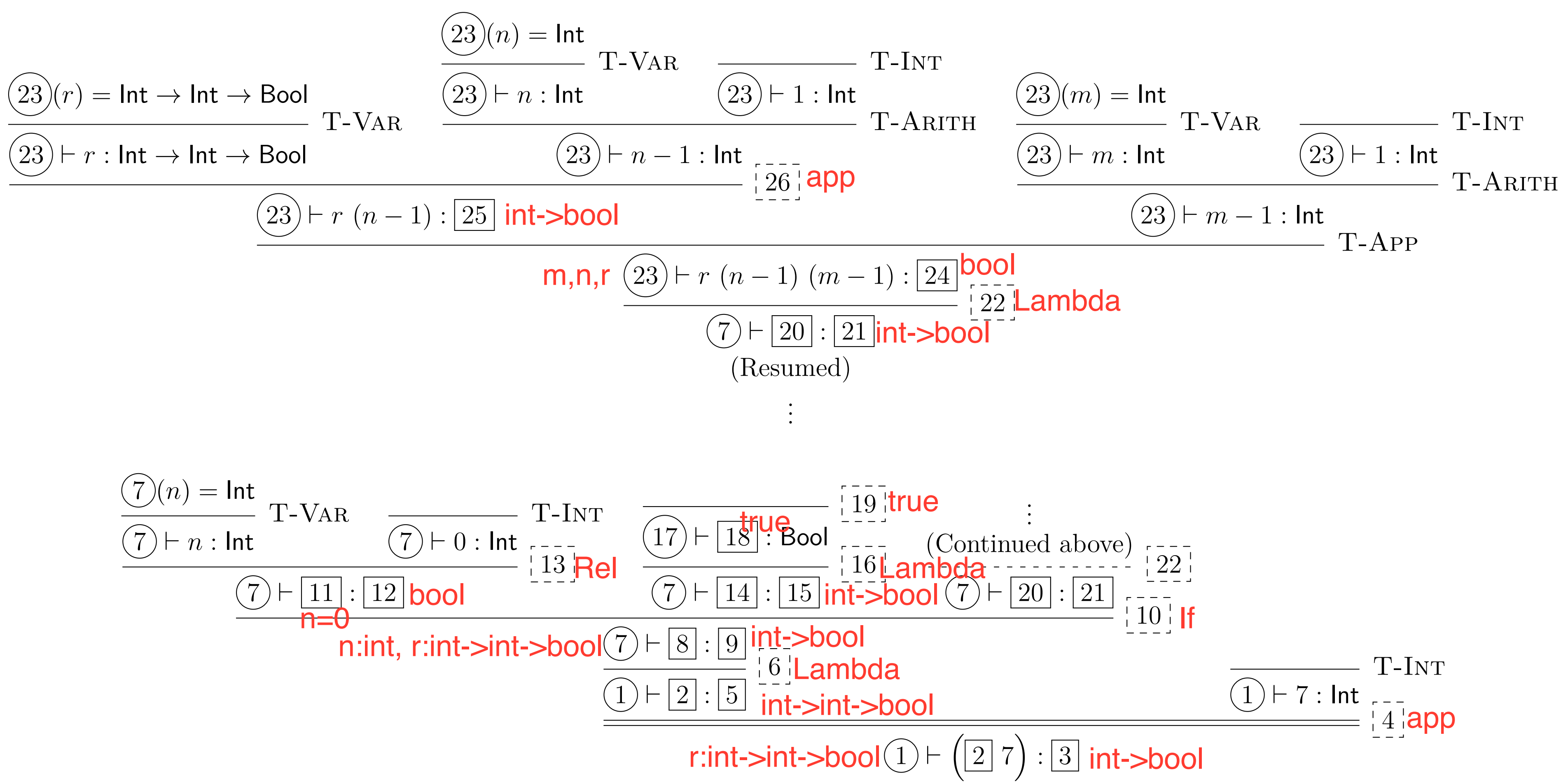
Text

Problem 5 (20 points)

Consider the following λ^+ expression:

```
(lambda n: Int.
    if n = 0 then (lambda k: Int. true) else (lambda m: Int. r (n-1) (m-1))) 7
```

Denote the above expression as e. We claim e is well-typed under the environment ① $= r : \mathsf{Int} \to \mathsf{Int} \to \mathsf{Bool}$. Below is the skeleton of the derivation tree that shows ① $\vdash e : \boxed{3}$ for some type $\boxed{3}$. The derivation starts from the bottom of the page with the double bar, and proceeds upwards. Complete the derivation tree by filling in the table on the next page. A $\boxed{\text{solid box}}$ should be filled with an λ^+ expression or type, a dashed box with the name of a *typing rule*, and a circled number (n) with a typing environment. Note that type equalities of the form $\mathsf{T}_1 = \mathsf{T}_2$ are omitted from the tree, but you should mentally check that they hold.



Always write the full concrete expression, the rule name, or the typing environment in a cell. Do *not* use abbreviations, and do *not* use names ("Γ", "e", "T", etc.) or numbers to refer to other expressions or rules.

#	Your Answer
1	r : Int → Int → Bool
2	`lambda n: Int. if n = 0 then (lambda k: Int. true)else (lambda m: Int. r (n-1)(m-1))`
3	(JR) Int → Bool
4	T-App
5	(JR) Int → Int → Bool
6	(JR) T-Lambda
7	(JR) n : Int, r : Int → Int → Bool
8	`if n = 0 then (lambda k: Int. true)else (lambda m: Int. r (n-1)(m-1))`
9	(JR) Int → Bool
10	(JR) T-If
11	(JR) $n = 0$
12	(JR) Bool
13	(JR) T-Rel

#	Your Answer
14	`lambda k: Int. true`
15	(JR) Int → Bool
16	(JR) T-Lambda
17	(JR) k : Int, n : Int, r : Int → Int → Bool
18	(JR) true
19	(JR) T-True
20	`lambda m: Int. r (n-1)(m-1)`
21	(JR) Int → Bool
22	(JR) T-Lambda
23	(JR) m : Int, n : Int, r : Int → Int → Bool
24	(JR) Bool
25	(JR) Int → Bool
26	(JR) T-App

Abbreviations

$\Gamma_0 = id : T_0 \to T_0$

$\Gamma_1 = n : T_1, \Gamma_0$

Problem 1 Below is a skeleton of the derivation tree for inferring the type of the expression " let $id = \lambda x.x$ in $(id\ (\lambda n.n+1))\ (id\ 0)$ ".

Fill in the blanks with typing constraints of the form $\boxed{? = ??}$ where ? and ?? are λ^+ types (which may contain type variables).

$$\dfrac{\dfrac{}{\Gamma_1 \vdash n : T_1}\ \text{CT-Var} \qquad \dfrac{}{\Gamma_1 \vdash 1 : Int}\ \text{CT-Int} \qquad \boxed{T_1 = Int}}{\Gamma_1 \vdash n+1 : Int}\ \text{CT-Arith}$$

$$\dfrac{\Gamma_1 \vdash n+1 : Int}{\Gamma_0 \vdash \lambda n.n+1 : T_1 \to Int}\ \text{CT-}\lambda$$

$$\dfrac{\Gamma_0 \vdash id : T_0 \to T_0 \qquad \Gamma_0 \vdash \lambda n.n+1 : T_1 \to Int \qquad \boxed{T_0 \to T_0 = T_2 \to T_3} \qquad \boxed{T_2 = T_1 \to Int}}{\Gamma_0 \vdash id\ (\lambda n.n+1) : T_3}\ \text{CT-App}$$

$$\dfrac{\dfrac{}{\Gamma_0 \vdash id : T_0 \to T_0}\ \text{CT-Var} \qquad \dfrac{}{\Gamma_0 \vdash 0 : Int}\ \text{CT-Int} \qquad \boxed{T_0 \to T_0 = T_4 \to T_5} \qquad \boxed{T_4 = Int}}{\Gamma_0 \vdash id\ \ 0 : T_5}\ \text{CT-App}$$

$$\dfrac{\Gamma_0 \vdash id\ (\lambda n.n+1) : T_3 \qquad \Gamma_0 \vdash id\ \ 0 : T_5 \qquad \boxed{T_3 = T_6 \to T_7} \qquad \boxed{T_6 = T_5}}{\Gamma_0 \vdash (id\ (\lambda n.n+1))\ (id\ 0) : T_7}\ \text{CT-App}$$

$$\dfrac{\dfrac{}{x : T_0 \vdash x : T_0}\ \text{CT-Var}}{\vdash \lambda x.x : T_0 \to T_0}\ \text{CT-}\lambda$$

$$\dfrac{\vdash \lambda x.x : T_0 \to T_0 \qquad \Gamma_0 \vdash (id\ (\lambda n.n+1))\ (id\ 0) : T_7}{\vdash \text{let } id = \lambda x.x \text{ in } (id\ (\lambda n.n+1))\ (id\ 0) : T_7}\ \text{CT-Let}$$

Constraints

$T_1 = Int$
$T_0 \to T_0 = T_2 \to T_3$
$T_2 = T_1 \to Int$
$T_0 \to T_0 = T_4 \to T_5$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\boxed{\mathcal{C}}$

Unification Steps

$\cancel{T_1 = Int}$
$T_0 \to T_0 = T_2 \to T_3$
$T_2 = \cancel{T_1}^{Int} \to Int$
$T_0 \to T_0 = T_4 \to T_5$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$T_0 = T_2$
$T_0 = T_3$
$\cancel{T_0 \to T_0 = T_2 \to T_3}$
$T_2 = Int \to Int$
$T_0 \to T_0 = T_4 \to T_5$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$T_0 = T_2$
$T_2\ \cancel{T_0} = T_3$
$T_2 = Int \to Int$
$\cancel{T_0}^{T_2} \to \cancel{T_0}^{T_2} = T_4 \to T_5$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$\cancel{T_2 = T_3}$
$T_3\ \cancel{T_2} = Int \to Int$
$T_3\ \cancel{T_2} \to \cancel{T_2}^{T_3} = T_4 \to T$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$\dots \Rightarrow$

$\cancel{T_3 = Int \to Int}$
$Int \to Int\ \cancel{T_3} \to \cancel{T_3}^{Int \to Int} = T_4 \to T_5$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$T_4 = Int \to Int$
$T_5 = Int \to Int$
$\cancel{(Int \to Int) \to (Int \to Int) = T_4 \to T_5}$
$T_4 = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$\cancel{T_4 = Int \to Int}$
$T_5 = Int \to Int$
$Int \to Int\ \cancel{T_4} = Int$
$T_3 = T_6 \to T_7$
$T_6 = T_5$

$\Rightarrow$

$\cancel{T_5 = Int \to Int}$
$Int \to Int = Int$
$T_3 = T_6 \to T_7$
$T_6 = \cancel{T_5}\ Int \to Int$

$\Rightarrow$

stuck.

$\boxed{Int \to Int = Int}$
$T_3 = T_6 \to T_7$
$T_6 = Int \to Int$

Problem 2

Simulate the execution of the unification algorithm on the constraint system $\boxed{\mathcal{C}}$.
If the system is solvable, also write down the substitution σ.
Otherwise, indicate where unification gets stuck.