Part 1

Question 1

a) {f: [T1→T2], g: [T1→T2], a:T1} ⊢ (f(g a)) :T2 => false

“a” is of type T1 and “g” is of type [T1→T2], thus “(g a)” is of type T2, and because “f” is of type [T1→T2] it cannot accept parameters (operands) of type T2 (we cannot infer that T1 = T2), thus the statement is false.

b) {x:T1, y:T2, f: [T2→T1]} ⊢ (f y) :T1 => true

“f” is of type [T2→T1] and “y” is of type T2, thus the type of the application “(f y)” is T1.

c) {f: [T1→T2]} ⊢ (lambda(x) (f x)) : [T1→T2] => false

There is no assumption that “x” is of type T1 (and we cannot infer that), but under the assumption “ x: T1” this statement is true.

d) {f: [T1∗T2→T3]} ⊢ (lambda(x) (f x 100)) : [T1→T3] => false

There is no assumption that “x” is of type T1 (and we cannot infer that), in addition, “100” is of type number, and there is no assumption that T2 is number, but under the assumption “ x: T1 and T2 = number” this statement is true.

Question 2

a) `((lambda (x1) (+ x1 1)) 4)`

Stage 1: renaming bound variables

`((lambda (x1) (+ x1 1)) 4) => ((lambda (x) (+ x 1)) 4)`

Stage 2: assign type variables for every sub expression

Expression	Type Variable
`((lambda (x) (+ x 1)) 4)`	T_0
`(lambda (x) (+ x 1))`	T_1
`(+ x 1)`	T_2
`+`	T_+
`x`	T_x
`1`	T_{Num1}
`4`	T_{Num4}

Stage 3: construct type equations

Expression	Equation
`((lambda (x) (+ x 1)) 4)`	$T_1 = [T_{Num4} \text{->} T_0]$
`(lambda (x) (+ x 1))`	$T_1 = [T_x \text{->} T_2]$
`(+ x 1)`	$T_+ = [T_x * T_{Num1} \text{->} T_2]$
`+`	$T_+ = [Number * Number \text{->} Number]$
`1`	$T_{Num1} = Number$
`4`	$T_{Num4} = Number$

Stage 4: solve the equations

Equation	Substitution
1. $T_1 = [T_{Num4} \text{->} T_0]$	{}
2. $T_1 = [T_x \text{->} T_2]$	
3. $T_+ = [T_x * T_{Num1} \text{->} T_2]$	
4. $T_+ = [Number * Number \text{->} Number]$	
5. $T_{Num1} = Number$	
6. $T_{Num4} = Number$	

Step 1: ($T_1 = [T_{Num4} \text{->} T_0]$) ○ Substitution = ($T_1 = [T_{Num4} \text{->} T_0]$)

Substitution = Substitution ○ ($T_1 = [T_{Num4} \text{->} T_0]$)

Equation	Substitution
2. $T_1 = [T_x \text{->} T_2]$	{
3. $T_+ = [T_x * T_{Num1} \text{->} T_2]$	$T_1 := [T_{Num4} \text{->} T_0]$
4. $T_+ = [Number * Number \text{->} Number]$	}
5. $T_{Num1} = Number$	
6. $T_{Num4} = Number$	

Step 2: ($T_1 = [T_x \text{->} T_2]$) ○ Substitution = ($[T_{Num4} \text{->} T_0] = [T_x \text{->} T_2]$)

split to 2 more equations: $T_{Num4} = T_x$ and $T_0 = T_2$

Equation	Substitution
3. $T_+ = [T_x * T_{Num1} \text{->} T_2]$	{
4. $T_+ = [Number * Number \text{->} Number]$	$T_1 := [T_{Num4} \text{->} T_0]$
5. $T_{Num1} = Number$	}
6. $T_{Num4} = Number$	
7. $T_{Num4} = T_x$	
8. $T_0 = T_2$	

Step 3: ($T_+ = [T_x * T_{Num1} \text{ -> } T_2]$) ○ Substitution = ($T_+ = [T_x * T_{Num1} \text{ -> } T_2]$)

Substitution = Substitution ○ ($T_+ = [T_x * T_{Num1} \text{ -> } T_2]$)

Equation	Substitution
4. T_+ = [Number * Number -> Number]	{ $T_1 := [T_{Num4} \text{ -> } T_0]$, $T_+ := [T_x * T_{Num1} \text{ -> } T_2]$ }
5. T_{Num1} = Number	
6. T_{Num4} = Number	
7. $T_{Num4} = T_x$	
8. $T_0 = T_2$	

Step 4: (T_+ = [Number * Number -> Number]) ○ Substitution = ($[T_x * T_{Num1} \text{ -> } T_2]$ = [Number * Number -> Number])

split to 3 more equations: T_x = Number, T_{Num1} = Number and T2 = Number

Equation	Substitution
5. T_{Num1} = Number	{ $T_1 := [T_{Num4} \text{ -> } T_0]$, $T_+ := [T_x * T_{Num1} \text{ -> } T_2]$ }
6. T_{Num4} = Number	
7. $T_{Num4} = T_x$	
8. $T_0 = T_2$	
9. T_x = Number	
10. T_{Num1} = Number	
11. T_2 = Number	

Step 5: (T_{Num1} = Number) ○ Substitution = (T_{Num1} = Number)

Substitution = Substitution ○ (T_{Num1} = Number)

Equation	Substitution
6. T_{Num4} = Number	{ $T_1 := [T_{Num4} \text{ -> } T_0]$, $T_+ := [T_x *$ **Number** $\text{ -> } T_2]$, T_{Num1} := Number }
7. $T_{Num4} = T_x$	
8. $T_0 = T_2$	
9. T_x = Number	
10. T_{Num1} = Number	
11. T_2 = Number	

Step 5: (T_{Num4} = Number) ○ Substitution = (T_{Num4} = Number)

Substitution = Substitution ○ (T_{Num4} = Number)

<table>
<tr><th>Equation</th><th>Substitution</th></tr>
<tr><td>7. T_{Num4} = T_x</td><td rowspan="7">{
T_1 := [Number -> T_0],
T_+ := [T_x * Number -> T_2] ,
T_{Num1} := Number,
T_{Num4} := Number
}</td></tr>
<tr><td>8. T_0 = T_2</td></tr>
<tr><td>9. T_x = Number</td></tr>
<tr><td>10. T_{Num1} = Number</td></tr>
<tr><td>11. T_2 = Number</td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
</table>

Step 6: (T_{Num4} = T_x) ○ Substitution = (Number = T_x)

Substitution = Substitution ○ (Number = T_x)

<table>
<tr><th>Equation</th><th>Substitution</th></tr>
<tr><td>8. T_0 = T_2</td><td rowspan="7">{
T_1 := [Number -> T_0],
T_+ := [Number * Number -> T_2] ,
T_{Num1} := Number,
T_{Num4} := Number,
T_x := Number
}</td></tr>
<tr><td>9. T_x = Number</td></tr>
<tr><td>10. T_{Num1} = Number</td></tr>
<tr><td>11. T_2 = Number</td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
</table>

Step 7: (T_0 = T_2) ○ Substitution = (T_0 = T_2)

Substitution = Substitution ○ (T_0 = T_2)

<table>
<tr><th>Equation</th><th>Substitution</th></tr>
<tr><td>9. T_x = Number</td><td rowspan="7">{
T_1 := [Number -> T_2],
T_+ := [Number * Number -> T_2] ,
T_{Num1} := Number,
T_{Num4} := Number,
T_x := Number,
T_0 := T_2
}</td></tr>
<tr><td>10. T_{Num1} = Number</td></tr>
<tr><td>11. T_2 = Number</td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
</table>

Step 9: (T_x = Number) ○ Substitution = (Number = Number)

Both sides are atomic, and are equal, do nothing.

Equation	Substitution
10. T_{Num1} = Number	{ T_1 := [Number -> T_2], T_+ := [Number * Number -> T_2] , T_{Num1} := Number, T_{Num4} := Number, T_x := Number, T_0 := T_2 }
11. T2 = Number	

Step 10: (T_{Num1} = Number) ○ Substitution = (Number = Number)

Both sides are atomic, and are equal, do nothing.

Equation	Substitution
11. T2 = Number	{ T_1 := [Number -> T_2], T_+ := [Number * Number -> T_2] , T_{Num1} := Number, T_{Num4} := Number, T_x := Number, T_0 := T_2 }

Step 11: (T2 = Number) ○ Substitution = (T2 = Number)

Substitution = Substitution ○ (T2 = Number)

Equation	Substitution
	{ T_1 := [Number -> **Number**], T_+ := [Number * Number -> **Number**] , T_{Num1} := Number, T_{Num4} := Number, T_x := Number, T_0 := Number }

The type inference succeeds since we have a type for T0, meaning that the expression is well typed. Because there are no free variables, the inferred type of T0 is: Number.

b) `((lambda (f1 x1) (f1 x1 1)) 4 +)`

Stage 1: renaming bound variables

`((lambda (f1 x1) (f1 x1 1)) 4 +) => ((lambda (f x) (f x 1)) 4 +)`

Stage 2: assign type variables for every sub expression

Expression	Type Variable
`((lambda (f x) (f x 1)) 4 +)`	T_0
`(lambda (f x) (f x 1))`	T_1
`(f x 1)`	T_2
`f`	T_f
`x`	T_x
`1`	T_{Num1}
`4`	T_{Num4}
`+`	T_+

Stage 3: construct type equations

Expression	Equation
`((lambda (f x) (f x 1)) 4 +)`	$T_1 = [T_{Num4} * T_+ \text{ -> } T_0]$
`(lambda (f x) (f x 1))`	$T_1 = [T_f * T_x \text{ -> } T_2]$
`(f x 1)`	$T_f = [T_x * T_{Num1} \text{ -> } T_2]$
`1`	T_{Num1} = Number
`4`	T_{Num4} = Number
`+`	T_+ = [Number * Number -> Number]

Stage 4: solve the equations

Equation	Substitution
1. $T_1 = [T_{Num4} * T_+ \text{ -> } T_0]$	{}
2. $T_1 = [T_f * T_x \text{ -> } T_2]$	
3. $T_f = [T_x * T_{Num1} \text{ -> } T_2]$	
4. T_{Num1} = Number	
5. T_{Num4} = Number	
6. T_+ = [Number * Number -> Number]	

Step 1: ($T_1 = [T_{Num4} * T_+ \text{->} T_0]$) ○ Substitution = ($T_1 = [T_{Num4} * T_+ \text{->} T_0]$)

Substitution = Substitution ○ ($T_1 = [T_{Num4} * T_+ \text{->} T_0]$)

Equation	Substitution
2. $T_1 = [T_f * T_x \text{->} T_2]$	{ $T_1 := [T_{Num4} * T_+ \text{->} T_0]$ }
3. $T_f = [T_x * T_{Num1} \text{->} T_2]$	
4. T_{Num1} = Number	
5. T_{Num4} = Number	
6. T_+ = [Number * Number -> Number]	

Step 2: ($T_1 = [T_f * T_x \text{->} T_2]$) ○ Substitution = ($[T_{Num4} * T_+ \text{->} T_0] = [T_f * T_x \text{->} T_2]$)

split to 2 more equations: $T_{Num4} = T_f$, $T_+ = T_x$ and $T_0 = T_2$

Equation	Substitution
3. $T_f = [T_x * T_{Num1} \text{->} T_2]$	{ $T_1 = [T_f * T_x \text{->} T_0]$ }
4. T_{Num1} = Number	
5. T_{Num4} = Number	
6. T_+ = [Number * Number -> Number]	
7. $T_{Num4} = T_f$	
8. $T_+ = T_x$	
9. $T_0 = T_2$	

Step 3: ($T_f = [T_x * T_{Num1} \text{->} T_2]$) ○ Substitution = ($T_f = [T_x * T_{Num1} \text{->} T_2]$)

Substitution = Substitution ○ ($T_f = [T_x * T_{Num1} \text{->} T_2]$)

Equation	Substitution
4. T_{Num1} = Number	{ $T_1 = [[T_x * T_{Num1} \text{->} T_2] * T_x \text{->} T_0]$, $T_f = [T_x * T_{Num1} \text{->} T_2]$ }
5. T_{Num4} = Number	
6. T_+ = [Number * Number -> Number]	
7. $T_{Num4} = T_f$	
8. $T_+ = T_x$	
9. $T_0 = T_2$	

Step 4: (T_{Num1} = Number) ○ Substitution = (T_{Num1} = Number)

Substitution = Substitution ○ (T_{Num1} = Number)

Equation	Substitution
5. T_{Num4} = Number	{ T_1 = [[T_x * **Number** -> T_2] * T_x -> T_0], T_f = [T_x * **Number** -> T_2], T_{Num1} = Number }
6. T_+ = [Number * Number -> Number]	
7. T_{Num4} = T_f	
8. T_+ = T_x	
9. T_0 = T_2	

Step 5: (T_{Num4} = Number) ○ Substitution = (T_{Num4} = Number)

Substitution = Substitution ○ (T_{Num4} = Number)

Equation	Substitution
6. T_+ = [Number * Number -> Number]	{ T_1 = [[T_x * **Number** -> T_2] * T_x -> T_0], T_f = [T_x * **Number** -> T_2], T_{Num1} = Number, T_{Num4} = Number }
7. T_{Num4} = T_f	
8. T_+ = T_x	
9. T_0 = T_2	

Step 6: (T_+ = [Number * Number -> Number]) ○ Substitution = (T_+ = [Number * Number -> Number])

Substitution = Substitution ○ (T_+ = [Number * Number -> Number])

Equation	Substitution
7. T_{Num4} = T_f	{ T_1 = [[T_x * **Number** -> T_2] * T_x -> T_0], T_f = [T_x * **Number** -> T_2], T_{Num1} = Number, T_{Num4} = Number, T_+ = [Number * Number -> Number] }
8. T_+ = T_x	
9. T_0 = T_2	

Step 7: (T_{Num4} = T_f) ○ Substitution = (Number = [T_x * Number -> T_2])

We get the conflicting equation:

(Number = [Tx * Number -> T2]) and we can say that the expression is not well typed.

Part 2

Question 2.2 (b)

Since we are returning an asynced function (using the keyword "async"), the return type of this function is always a promise (no matter what the return type of the returned value is).

Part 3

Question 3.1

Typing rule define:

For every: type environment _Tenv,

variable _x1

expressions _e1 and

type expressions _S1, _U1:

If $\_Tenv \circ \{\_x1: \_S1\} \vdash \_e1: \_U1$

Then $\_Tenv \vdash$ (define _x1 _e1) : Void

Typing rule set!:

For every: type environment _Tenv,

variable _x1

expressions _e1 and

type expressions _S1:

If $\_Tenv \vdash \_x1: \_S1$

$\_Tenv \vdash \_e1: \_S1$

Then $\_Tenv \vdash$ (set! _x1 _e1) : Void