Exam for Programming Language Principals, Design and Implementation (Extended)

ID 1803086

After inserting your student ID and the module title in the preamble, write your answers below.

Question 1

(a)

$$M = \lambda f : \mathbb{B} \to \mathbb{B}.\lambda g : \mathbb{B} \to \mathbb{B}.\lambda x : \mathbb{B}.\lambda y : \mathbb{B}.\texttt{if } x \texttt{ then } fy \texttt{ else } gy$$

(i) Prove M is well typed

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \mathbb{B}}\,\texttt{VAR} \qquad \dfrac{\dfrac{}{\Gamma \vdash f : \mathbb{B} \to \mathbb{B}}\,\texttt{VAR} \quad \dfrac{}{\Gamma \vdash y : \mathbb{B}}\,\texttt{VAR}}{\Gamma \vdash fy : \mathbb{B}}\,\texttt{APP} \qquad \Pi_1}{\Gamma, y : \mathbb{B} \vdash \texttt{if } x \texttt{ then } fy \texttt{ else } gy : \mathbb{B}}\,\texttt{ITE}}{\Gamma, x : \mathbb{B} \vdash \lambda y : \mathbb{B}.\texttt{if } x \texttt{ then } fy \texttt{ else } gy : \mathbb{B} \to \mathbb{B}}\,\texttt{ABS}}{\Gamma, g : \mathbb{B} \to \mathbb{B} \vdash \lambda x : \mathbb{B}.\lambda y : \mathbb{B}.\texttt{if } x \texttt{ then } fy \texttt{ else } gy : \mathbb{B} \to \mathbb{B} \to \mathbb{B}}\,\texttt{ABS}}{f : \mathbb{B} \to \mathbb{B} \vdash M_1 : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B} \to \mathbb{B}}\,\texttt{ABS}}{\{\} \vdash M : (\mathbb{B} \to \mathbb{B}) \to (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B} \to \mathbb{B}}\,\texttt{ABS}$$

Where:

- $M_1 = \lambda g : \mathbb{B} \to \mathbb{B}.\lambda x : \mathbb{B}.\lambda y : \mathbb{B}.\texttt{if } x \texttt{ then } fy \texttt{ else } gy$
- $\Pi_1 =$

$$\dfrac{\dfrac{}{\Gamma \vdash g : \mathbb{B} \to \mathbb{B}}\,\texttt{VAR} \qquad \dfrac{}{\Gamma \vdash y : \mathbb{B}}\,\texttt{VAR}}{\Gamma \vdash gy : \mathbb{B}}\,\texttt{APP}$$

(ii) To produce the exclusive function from M we can define the first order parameters F and G as follows:

$$F = \lambda y : \mathbb{B}.\texttt{if } y \texttt{ then false else true} : \mathbb{B} \to \mathbb{B}$$

$$G = \lambda y : \mathbb{B}.\texttt{if } y \texttt{ then true else false} : \mathbb{B} \to \mathbb{B}$$

Alternatively, G can simply be defined as the boolean identity function $\lambda y : \mathbb{B}.y : \mathbb{B} \to \mathbb{B}$. This is the definition I will use in latter parts of the question.

(iii) As the expression (MFG`false true`) has type $\mathbb{B}$, we can be sure when using it that if we provide it to a function of type $\mathbb{B} \rightarrow T$ the evaluation will terminate and we will be returned a value of type T

(iv)

$$\dfrac{\dfrac{}{\lambda f:\mathbb{B}\rightarrow\mathbb{B}.M_1F \rightarrow_v \lambda g:\mathbb{B}\rightarrow\mathbb{B}\lambda x:\mathbb{B}\texttt{if } x \texttt{ then } Fy \texttt{ else } gy}\beta}{\begin{array}{c}(MFG)\texttt{false true} \rightarrow_v \\ ((\lambda g:\mathbb{B}\rightarrow\mathbb{B}\lambda x:\mathbb{B}\texttt{if } x \texttt{ then } Fy \texttt{ else } gy)G)\texttt{false true}\end{array}}\texttt{CTX}_{(\bullet G)\texttt{false true}} \tag{1}$$

$$\dfrac{\dfrac{}{\begin{array}{c}(\lambda g:\mathbb{B}\rightarrow\mathbb{B}.\lambda x:\mathbb{B}.\texttt{if } x \texttt{ then } Fy \texttt{ else } gy)G \rightarrow_v \\ \lambda x:\mathbb{B}.\lambda y:\mathbb{B}.\texttt{if } x \texttt{ then } Fy \texttt{ else } Gy\end{array}}\beta}{\begin{array}{c}((\lambda g:\mathbb{B}\rightarrow\mathbb{B}.\lambda x:\mathbb{B}.\texttt{if } x \texttt{ then } Fy \texttt{ else } gy)G)\texttt{false true} \rightarrow_v \\ (\lambda x:\mathbb{B}.\lambda y:\mathbb{B}.\texttt{if } x \texttt{ then } Fy \texttt{ else } Gy)\texttt{false true}\end{array}}\texttt{CTX}_{(\bullet)\texttt{false true}} \tag{2}$$

$$\dfrac{\dfrac{}{\begin{array}{c}(\lambda x:\mathbb{B}.\lambda y:\mathbb{B}.\texttt{if } x \texttt{ then } Fy \texttt{ else } Gy)\texttt{false} \rightarrow_v \\ \lambda y:\mathbb{B}.\texttt{if false then } Fy \texttt{ else } Gy\end{array}}\beta}{\begin{array}{c}(\lambda x:\mathbb{B}.\lambda y:\mathbb{B}.\texttt{if } x \texttt{ then } Fy \texttt{ else } Gy)\texttt{false true} \rightarrow_v \\ (\lambda y:\mathbb{B}.\texttt{if false then } Fy \texttt{ else } Gy)\texttt{true}\end{array}}\texttt{CTX}_{\bullet\texttt{true}} \tag{3}$$

$$\dfrac{}{\begin{array}{c}(\lambda y:\mathbb{B}.\texttt{if false then } Fy \texttt{ else } Gy)\texttt{true} \rightarrow_v \\ \texttt{if false then } F\texttt{true} \texttt{ else } G\texttt{true}\end{array}}\beta \tag{4}$$

$$\dfrac{}{\texttt{if false then } F\texttt{true} \texttt{ else } G\texttt{true} \rightarrow_v G\texttt{true}}\texttt{IteF} \tag{5}$$

$$\dfrac{}{(\lambda y:\mathbb{B}.y)\texttt{true} \rightarrow_v \texttt{true}}\beta \tag{6}$$

Where:

- M_1 is as defined the same as above.

`true` $\in V$, $\therefore$ (MFG)`false true` computes to a value.

(b)

$$\texttt{Stack} = \forall\alpha.(\mathbb{N}\rightarrow\alpha\rightarrow\alpha)\rightarrow\alpha\rightarrow\alpha$$

(i)

$$\lambda\alpha.\lambda f:\mathbb{N}\rightarrow\alpha\rightarrow\alpha.\lambda x:\alpha.f0(f0(f1x))$$

(ii)

$$\texttt{peek} = \lambda d:\mathbb{N}.\lambda s:\texttt{Stack}.s\{\mathbb{N}\rightarrow\mathbb{N}\}GId$$

Where:

- $G = \lambda n:\mathbb{N}.\lambda g:\mathbb{N}\rightarrow\mathbb{N}.\lambda x:\mathbb{N}.gn$
- $I = \lambda x:\mathbb{N}.x$

$$
\dfrac{
\dfrac{\dfrac{\Pi_1 \qquad \Pi_2}{\Gamma \vdash s\{\mathbb{N} \to \mathbb{N}\}G : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})}\,\texttt{APP}
\qquad
\dfrac{\dfrac{}{\Gamma, x : \mathbb{N} \vdash x : \mathbb{N}}\,\texttt{VAR}}{\Gamma \vdash I : \mathbb{N} \to \mathbb{N}}\,\texttt{ABS}}
{\Gamma \vdash ((s\{\mathbb{N} \to \mathbb{N}\}G)I) : \mathbb{N} \to \mathbb{N}}\,\texttt{APP}
\qquad
\dfrac{}{\Gamma \vdash d : \mathbb{N}}\,\texttt{VAR}
}{
\dfrac{\Gamma, s : \texttt{Stack} \vdash ((s\{\mathbb{N} \to \mathbb{N}\}G)I)d : \mathbb{N} \to \texttt{Stack} \to \mathbb{N} : \mathbb{N}}{
\dfrac{d : \mathbb{N} \vdash \lambda s : \texttt{Stack}.s\{\mathbb{N} \to \mathbb{N}\}GId : \mathbb{N} \to \texttt{Stack} \to \mathbb{N} : \texttt{Stack} \to \mathbb{N}}{\{\} \vdash \lambda d : \mathbb{N}.\lambda s : \texttt{Stack}.s\{\mathbb{N} \to \mathbb{N}\}GId : \mathbb{N} \to \texttt{Stack} \to \mathbb{N}}\,\texttt{ABS}}\,\texttt{ABS}
}\,\texttt{APP}
$$

Where:

- $\Pi_1 =$

$$
\dfrac{\dfrac{}{\Gamma \vdash s : \texttt{State}}\,\texttt{VAR}}{\Gamma \vdash s\{\mathbb{N} \to \mathbb{N}\} : (\mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})) \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})}\,\texttt{TAPP}
$$

- $\Pi_2 =$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \mathbb{N} \to \mathbb{N}}\,\texttt{VAR} \qquad \dfrac{}{\Gamma \vdash n : \mathbb{N}}\,\texttt{VAR}}{\Gamma, x : \mathbb{N} \vdash gn : \mathbb{N}}\,\texttt{APP}}{\Gamma, g : \mathbb{N} \to \mathbb{N} \vdash \lambda x : \mathbb{N}.gn : \mathbb{N} \to \mathbb{N}}\,\texttt{ABS}}{\Gamma, n : \mathbb{N} \vdash \lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}}\,\texttt{ABS}}{\Gamma \vdash G : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}}\,\texttt{ABS}
$$

(iii) Prove:

$$\texttt{peek}\ \ ds_2 \to_v^* m$$

$$
\dfrac{\dfrac{}{(\lambda d : \mathbb{N}.\lambda s : \texttt{Stack}.s\{\mathbb{N} \to \mathbb{N}\}GId)d \to_v \lambda s : \texttt{Stack}.s\{\mathbb{N} \to \mathbb{N}\}GId}\,\beta}{\texttt{peek}\ \ ds_2 \to_v (\lambda s : \texttt{Stack}.s\{\mathbb{N} \to \mathbb{N}\}GId)s_2}\,\texttt{CTX}_{\bullet s_2}
$$

$$
\dfrac{}{(\lambda s : \texttt{Stack}.s\{\mathbb{N} \to \mathbb{N}\}GId)s_2 \to_v s_2\{\mathbb{N} \to \mathbb{N}\}GId}\,\beta
$$

$$
\dfrac{\dfrac{}{\begin{array}{c}\lambda\alpha.\lambda f : \mathbb{N} \to \alpha \to \alpha.\lambda x : \alpha.fn(fmx)\{\mathbb{N} \to \mathbb{N}\} \to_v \\ \lambda f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}).\lambda x : (\mathbb{N} \to \mathbb{N}).fn(fmx)\end{array}}\,T_\beta}{\begin{array}{c}((s_2\{\mathbb{N} \to \mathbb{N}\}G)I)d \to_v \\ (((\lambda f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}).\lambda x : \mathbb{N} \to \mathbb{N}.fn(fmx))G)I)d\end{array}}\,\texttt{CTX}_{((\bullet G)I)d}
$$

$$
\dfrac{\dfrac{}{\begin{array}{c}(\lambda f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}).\lambda x : \mathbb{N} \to \mathbb{N}.fn(fmx))G \to_v \\ \lambda x : \mathbb{N} \to \mathbb{N}.Gn(Gmx)\end{array}}\,\beta}{\begin{array}{c}(((\lambda f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}).\lambda x : \mathbb{N} \to \mathbb{N}.fn(fmx))G)I)d \to_v \\ ((\lambda x : \mathbb{N} \to \mathbb{N}.Gn(Gmx))I)d\end{array}}\,\texttt{CTX}_{((\bullet)I)d}
$$

$$
\dfrac{\dfrac{}{(\lambda x : \mathbb{N} \to \mathbb{N}.Gn(Gmx))I \to_v Gn(GmI)}\,\beta}{((\lambda x : \mathbb{N} \to \mathbb{N}.Gn(Gmx))I)d \to_v (Gn(GmI)d)}\,\texttt{CTX}_{(\bullet)d}
$$

$$
\dfrac{\dfrac{}{(\lambda n : \mathbb{N}.\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)n \to_v \lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn}\,\beta}{(Gn(GmI)d) \to_v ((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)(GmI))d}\,\texttt{CTX}_{(\bullet(GmI))d}
$$

$$\dfrac{\dfrac{}{(\lambda n : \mathbb{N}\lambda g : \mathbb{N} \to \mathbb{N}\lambda x : \mathbb{N}.gn)m \to_v \lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gm}\ \beta}{\begin{array}{c}((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)(GmI))d \to_v \\ ((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gm)I))d\end{array}}\ \mathtt{CTX}_{((\lambda g:\mathbb{N}\to\mathbb{N}.\lambda x:\mathbb{N}.gn)(\bullet I))d}$$

$$\dfrac{\dfrac{}{(\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gm)I \to_v \lambda x : \mathbb{N}.Im}\ \beta}{\begin{array}{c}((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gm)I))d \to_v \\ ((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)(\lambda x : \mathbb{N}.Im))d\end{array}}\ \mathtt{CTX}_{((\lambda g:\mathbb{N}\to\mathbb{N}.\lambda x:\mathbb{N}.gn)(\bullet))d}$$

$$\dfrac{\dfrac{}{(\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)(\lambda x : \mathbb{N}.Im) \to_v \lambda x : \mathbb{N}.(\lambda x : \mathbb{N}Im)n}\ \beta}{\begin{array}{c}((\lambda g : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.gn)(\lambda x : \mathbb{N}.Im))d \to_v \\ (\lambda x : \mathbb{N}.(\lambda x : \mathbb{N}.Im)n)d\end{array}}\ \mathtt{CTX}_{(\bullet)d}$$

$$\dfrac{}{(\lambda x : \mathbb{N}.(\lambda x : \mathbb{N}.Im)n)d \to_v (\lambda x : \mathbb{N}.Im)n}\ \beta$$

$$\dfrac{}{(\lambda x : \mathbb{N}.Im)n \to_v Im}\ \beta$$

$$\dfrac{}{Im \equiv (\lambda x : \mathbb{N}.x)n \to_v m}\ \beta$$

(iv) an abstract *stack* datatype can be defined as follows:

$$\begin{aligned}\mathtt{pack}\ & \langle \mathtt{Stack}, \langle s_0, \langle \mathtt{push}, \langle \mathtt{peek}, \mathtt{pop}\rangle\rangle\rangle\rangle \\ & \ \ \mathtt{as} \\ & \exists \mathtt{stack}.\mathtt{stack} \times (\mathbb{N} \to \mathtt{stack} \to \mathtt{stack}\times \\ & (\mathtt{stack} \to \mathbb{N} \to \mathbb{N} \times \mathtt{stack} \to \mathtt{stack}))\end{aligned}$$

Where:

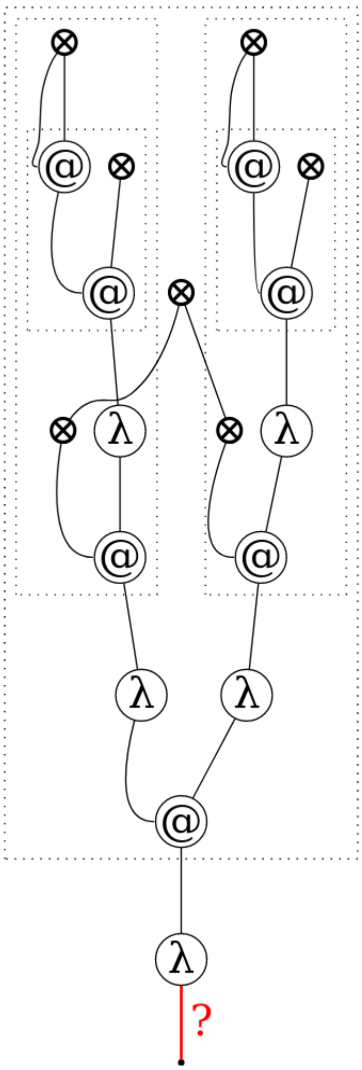
- $s_0 = \lambda\alpha.\lambda f : \mathbb{N} \to \alpha \to \alpha.\lambda x : \alpha.x$
- $\mathtt{Stack} \ = \forall\alpha.(\mathbb{N} \to \alpha \to \alpha) \to \alpha \to \alpha$
- `push`, `peek` and `pop` are defined the same as in the exam booklet, over the (concrete) `Stack` type.

Question 2

$$Y = \lambda y.(\lambda f.t(\lambda z.ffz))(\lambda f.t(\lambda z.ffz))$$

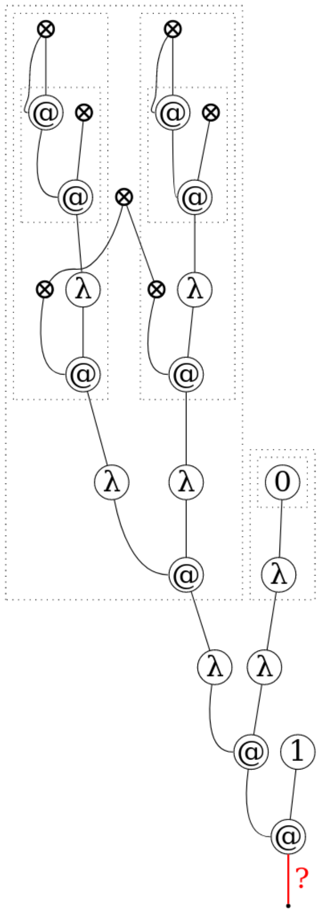
(a) The following ASG was generated using SPARTAN and the following code:

```
            LAMBDA(;t.APP(
    LAMBDA(; f.
      APP(t, LAMBDA(; z.
                     APP(APP(f,f),z))
          )
    ),
    LAMBDA(; f.
      APP(t, LAMBDA(; z.
                     APP(APP(f,f),z))
          )
    )
))
```

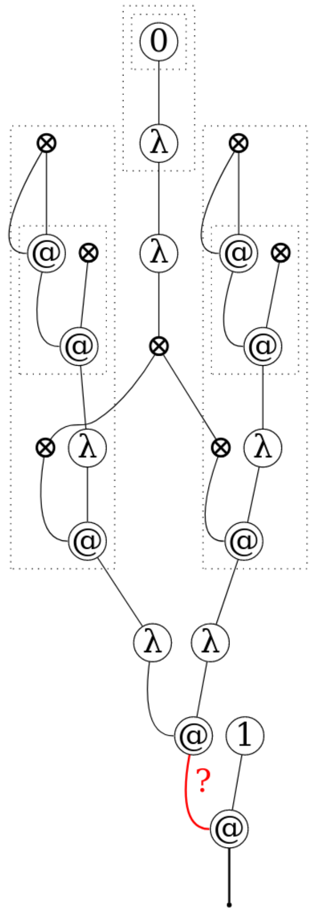


(b) (i)

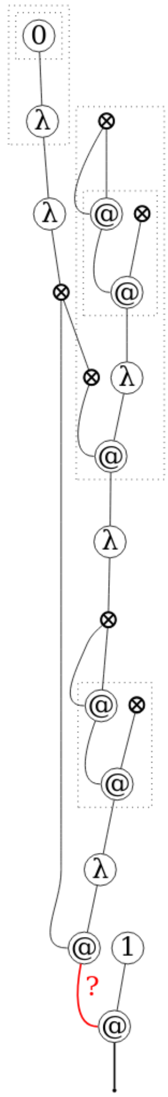
$$Y(\lambda f.\lambda x.0)1$$



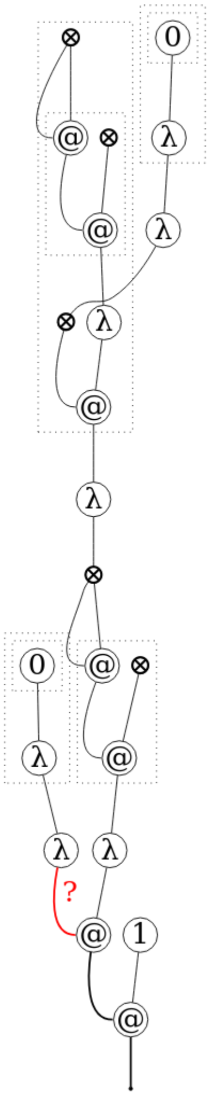
Firstly, the ASG evaluate the LHS, finding values on either side of the first application, it performs a reduction, replacing $\lambda t.$ with $\lambda f.\lambda x.0$:



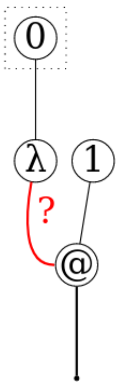
It goes on to attempt to further evaluate the LHS, again finding two values either side of an application, the LH thunk is expanded attaching the RH value as its parameter:



Next, the machine performs a rewrite of a shared reference of $\lambda f.\lambda x.0$:



Again finding two values on the LHS application, a reduction is performed, stripping the outer λ from $\lambda f.\lambda x.0$, discarding the Y combinator:



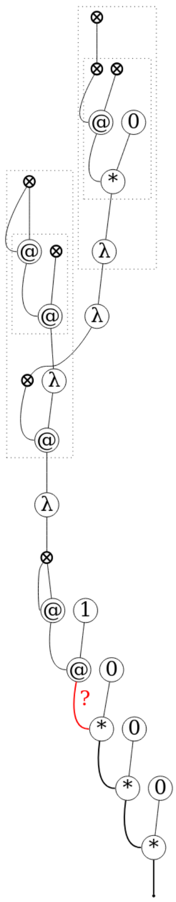
Finally, finding 2 values either side of our application the system removes the next λ, leaving just 0, our final result from this computation:

(ii)

$$Y(\lambda f.\lambda x.f(x) * 0)1$$

This expression will diverge. This occurs as the second operand of the $*$ operator (1) is is never evaluated as our Y combinator will infinitely expand over the first function argument $\lambda f.\lambda x.f(x) * 0$

After 91 steps of execution the ASG abstract machine will be in the following state:



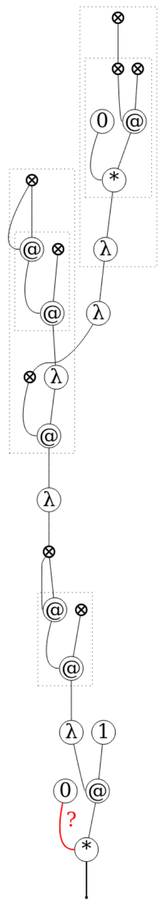
Intuitively, you can see that the second operand (1) is being pushed up as the Y combinator duplicates the operation and first operand (0) infinitely.

(iii) If we were to swap our operands to form an expression:

$$Y(\lambda f.\lambda x.0 * f(x))1$$

This would terminate, due to the nature of our $*$ *shortcut* operator which does not need to evaluate a second argument in the case where the first operand is 0.

After 26 steps of execution we reach the following state:



Here you can see that we have a state where we are attempting to evaluate $0 * @$, with a traditional (eager) multiplication operator, we would find a value on the left and an expression on the right, forcing us to evaluate the RHS until we reach a value. However, our *shortcut* operator allows us to instead skip this evaluate and return 0, eliminating the hang the Y combinator would otherwise cause. Our final result would be:

(c) (i) My answer to b)i) would not be affected. Nor would my answer to b)ii) as the cause of the infinite expansion was the first argument, not the second.

For b)iii) however, our answer would change. The program would no longer terminate as when it reached the state shown above, where $*$ is applied to a value (0) and an expression (@), it would be forced to expand the RHS application and create a similar infinite expansion as in b)ii) due to the use of the *Y* combinator without a terminating condition.

(ii) *I feel like this is some sort of trick question, so I am going to explain my entire thought process*

From a purely mathematical point of view, both $Y(\lambda f.\lambda x.f(x)*0)1$ and $Y(\lambda f.\lambda x.0* f(x))1$ can be optimised to 0. However, I doubt any impure language would implement this optimisation. The possible side effects of executing $f(x)$ would be lost if it were removed. For instance, take the following code in Rust:

```
fn f(_x:i32) -> i32{
  println!("Some desired side effect");
  2
}

fn main() {

  let x = 2;
  let res = 0*f(x);
  println!("{}",res);

}
```

Here we can see f to have desirable side effects. The code produces the result `Some desired side effect` followed by 0, indicating that this optimisation is not made, keeping our desirable side effects.

Our question however, is concerning the λ-Calculus, a mathematically pure language. Therefore, this optimisation would not lose any desirable side effects, as they are impossible, allowing us to optimise both expressions to 0.

(iii) Yes $Y(\lambda f.\lambda x.0)1$ can be optimised to 0. This is the case as the result of evaluating the expression is not dependant on either of the parameters f or x.

Question 3

(a) (i) a is true whenever b is true. This is an example of an invariant, (a subset of *safety properties*) as it can only be verified by checking $b \rightarrow a$ in each state individually. It can be represented in LTL as:

$$\Box(b \rightarrow a)$$

(ii) a and b are simultaneously true only a finite number of times.

This is a *liveness property* as only an infinite path could satisfy $\Box(a \wedge b)$ and given a path which violates it, we could easily extend it with a single state which satisfied $\neg(a \wedge b)$ to satisfy this property. In LTL it can be represented as:

$$\Diamond\Box\neg(a \wedge b)$$

(iii) every b is immediately followed by an a

This is an example of a *safety property* as any trace violating this property would have a finite prefix in which a state satisfying b is (immediately) followed by a state satisfying $\neg a$. Such a trace cannot be extended to satisfy the property. It can be written in LTL as

$$\Box(b \rightarrow \bigcirc a)$$

(iv) exactly one of a or b (but not both) is eventually true.

This is not a *liveness property* as given a trace violating this property by including a state in which a and b we cannot add a suffix s.t. it no longer violates the property. Equally, it is not *safety property* as an infinite path in which neither a or b is true does not satisfy the property and yet has no bad prefix.

$$(\Diamond a \vee \Diamond b) \wedge (a \rightarrow \Box\neg b) \wedge (b \rightarrow \Box\neg a)$$

(v) if a ever becomes true, then it remains true forever, and this is immediately preceded by a state where b was true.

This is a *safety property* as any violation would have a trace in which either the first occurrence of a was not immediately preceded by a b or the first occurrence of a is eventually followed by a state in which a is not true.

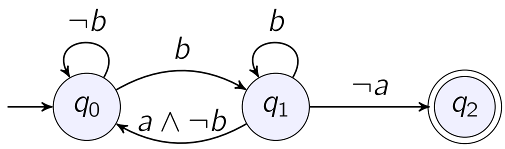
$$((b \wedge \bigcirc a) \rightarrow \bigcirc\bigcirc\Box a) \vee \neg(\neg b \wedge \bigcirc a)$$

(b) The first stage of LTL model checking is to negate the LTL formula $\Psi = \Box(b \rightarrow \bigcirc a)$:

$$\begin{aligned}
\neg\Psi &= \neg\Box(b \rightarrow \bigcirc a) \\
&= \Diamond\neg(b \rightarrow \bigcirc a) \\
&= \Diamond\neg(\neg b \vee \bigcirc a) \\
&= \Diamond(b \wedge \bigcirc\neg a)
\end{aligned}$$

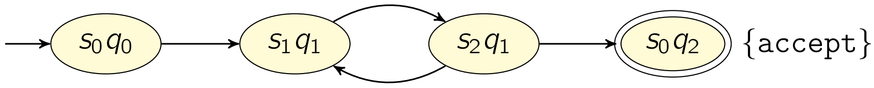
i.e. to prove that *"every b is immediately followed by an a"* does not hold, one must find a trace whereby a b is immediately followed by $\neg a$.

We then construct an NFA, $\mathcal{A}_\Psi$ of this formula:



We then take the product of the LTS given in the question and the NFA $\mathcal{A}_\Psi$, defined above, to produce:

$\mathcal{A}_\Psi \otimes M$, where M is the LTS given in the question:



Above you can see that the product of the NFA and LTS has an *accepting* state, meaning there exists a trace that satisfies the negation of the property, i.e. the property does not hold for all possible traces.

(c) The revised property can be expressed as:

$$\Box(a \rightarrow \Diamond^{\leq k} b)$$

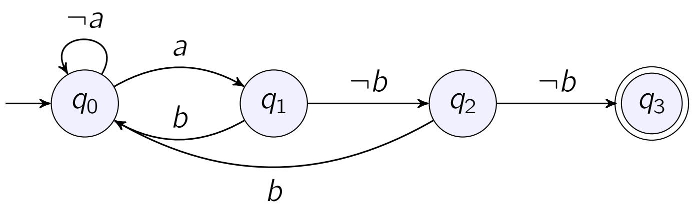
A simple negation of this property, for checking, could be:

$$\Diamond(a \wedge \neg\Diamond^{\leq k} b)$$

In order to check this property over the LTS in the question, we would require a mechanism of *counting* occurrences of non-b states following an a.

A simple implementation of this would be, for a fixed k, to add k extra states to our NFA, where in each state $(1..k-1)$ we could either continue waiting or return to the initial state. The final state could either find a b and return to the initial state or terminate in an error state.

An example for $k = 2$ could be:



The product of the above NFA and the given LTS would show whether the property could be violated.

In a more general sense, the $\Diamond^{\leq k}\psi$ operator could be incorporated into LTL model checking by constructing it's negation as a series of k linear states followed by a terminating state in which if ψ was true the NFA returned to the initial state and if ψ was not true it would continue to the next state until it reached the final, accepting, position.

Statement of good academic conduct
By submitting this assignment, I understand that I am agreeing to the following statement of good academic conduct.

- I confirm that this assignment is my own work and I have not worked with others in preparing this assignment.
- I confirm this assignment was written by me and is in my own words, except for any materials from published or other sources which are clearly indicated and acknowledged as such by appropriate referencing.
- I confirm that this work is not copied from any other person's work (published or unpublished), web site, book or other source, and has not previously been submitted for assessment either at the University of Birmingham or elsewhere.
- I confirm that I have not asked, or paid, others to prepare any part of this work for me.
- I confirm that I have read and understood the University regulations on plagiarism.