Programming Languages HW 2

Jiraphon Yenphraphai (jy3694)

June 26, 2022

1 Activation Records and Lifetimes

1.1 1

Since each argument for the printf function might require different memory space: int uses 4 bytes, double uses 8 bytes. This is a problem for C when accessing a variable in stack as we don't know a constant frame pointer offset.

What if we fix the spaces for each variable to store to be the maximum space that it could be. For example, rather than using 4 bytes for int, we uses 12 bytes. Likewise for float and double, we uses 12 bytes instead of the standard 4 and 8 bytes. From Fig. 1, let say we have format string printf("this is %d a format %d string", x, y). We want to access actual x, we can do so by adding fix amount of bytes to frame pointer. In this case, it is $12 * 2 = 24$ bytes. We can access each variable by adding offset of size which is multiples of 12 bytes. The number of arguments can be counted by the multiples of 12. To access format string, we usually store format string in runtime stack and refer to this string by its location.

```
printf("this␣is␣%d␣a␣format␣%d␣string", x, y);
```

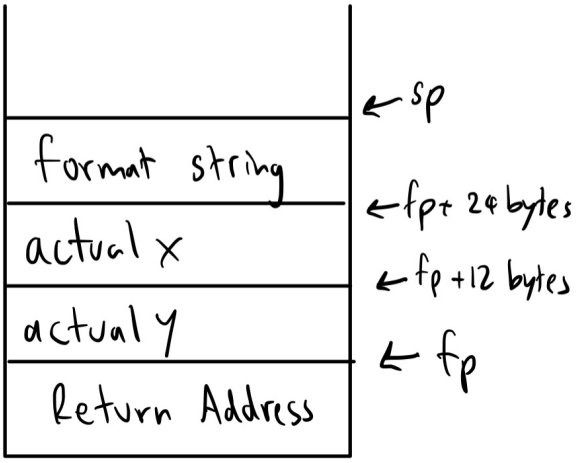

Figure 1: Runtime stack of the proposed method

1.2 2

Normally, in the language supporting recursion, it needs stack-based allocation as the number of object of variables may be undetermined. To retrieve local variables, code can do so by adding positive offsets to the frame pointer. Since recursion is not supported in "NoRec", the subroutine can be active not more than once at a time. The local variables can be statically allocated, and different function calls can refer to this location; thus, stack-based allocation is not needed. That is, we don't need frame pointer and its offset as we can access a local variable from the static-allocated location.

Normally in recursion, the return address changes depending on where subroutine is called. In our case, global and local variables' locations are statically allocated statically the compile time. At the end of subroutine, we can access the address location of where it should jump back after the call from static-allocated location. The return address is not needed.

1.3 3

Heap-allocated objects

The language runtime automatically deallocated objects when they are unreachable. To do so, the language runtime use a garbage collection to identify, reclaim and call destructor of objects.

Stack-allocated objects

The object of the class is first alive when the subroutine is called. When the subroutine is done, the epilogue will restore callee-save registers. The object has dynamic size and is allocated in runtime stack. If the epilogue needs to access and clean up the object, it needs to refer to the object's address and call destructor for the object of this address. Then, it pops this frame and jumps back to return address.

1.4 4

Aliveness and visibility of a variable are 2 different things. Even if it is hidden inside inner scope, it still exists. In this code, we have int a, b, c, d, g, h which use $4 * 6 = 24$ bytes. double a is defined inside the inner-most scope. It hides int a from outer scope, but these variables (double a, int a) are 2 different local variables. The scope of int a is everywhere except the inner-most scope. The scope of double a is only in inner-most scope. Therefore, we need memory to store double a which is another 8 bytes. In total, the minimal amount of space is $24 + 8 = 32$ bytes.

1.5 5

Line marked with an asterisk can see A,C which are formal parameters. Notice that A which is formal parameter hides A from outer procedure P. R.Z, which is the local variable in procedure R, is obviously visible. X and B from outer procedure P are also visible. Procedure Q is also visible. In total, R.A, R.C, R.Z, OUTER P.X, OUTER P.B, and procedure Q are the names which are visible from the specified location.

Note that the variable and its scope that is referred to by procedure followed by name (procedure.name).

2 Nested Subprograms

2.1 a

MAIN $\rightarrow$ BIGSUB $\rightarrow$ SUB2 $\rightarrow$ SUB3 $\rightarrow$ SUB1

When SUB2 is called in BIGSUB, the actual parameter is 7. Apart from SUB2, there are no actual parameter and formal parameter passing as procedure BIGSUB, SUB1, SUB3 don't need an input.

2.2 b

Assume that it is statically scope. Define the variable and its scope that is referred to by procedure followed by name (procedure.name). MAIN.X, BIGSUB.A, BIGSUB.B, BIGSUB.C, SUB1.D, SUB2.B, SUB2.X, SUB3.C never change in its scope.

2.3 c

Now that, it is dynamic scoping. Variable gets update by the most recent declaration. It changes the behavior of the program.

- Variable X in MAIN (MAIN.X) changes from 3 to 7 in procedure SUB2 during binding formal parameter with actual.
- Variable A in BIGSUB (BIGSUB.A) is assigned by B+C in line 1, which means A will change to $3 + 4 = 7$. A also changes in SUB2 where it is assigned by E.
- Variable B in BIGSUB (BIGSUB.B) changes from 3 to 1 in SUB2.
- Variable C in BIGSUB (BIGSUB.C) changes from 4 to 9 in SUB3.
- Variable A in SUB1 (SUB1.A) changes because of line 1.
- Variable D in SUB1 (SUB1.D) never changes.
- Variable B in SUB2 (SUB2.B) never changes.

- Variable E in SUB2 (SUB2.E) is assigned by B+C in SUB3, which means E's value will change from 8 to 10.
- Variable X in SUB2 (SUB2.X) never changes
- Variable C in SUB3 (SUB3.C) never changes.

In total, SUB1.D, SUB2.B, SUB2.X, and SUB3.C have the same value when it is dynamic scoping.

2.4 d

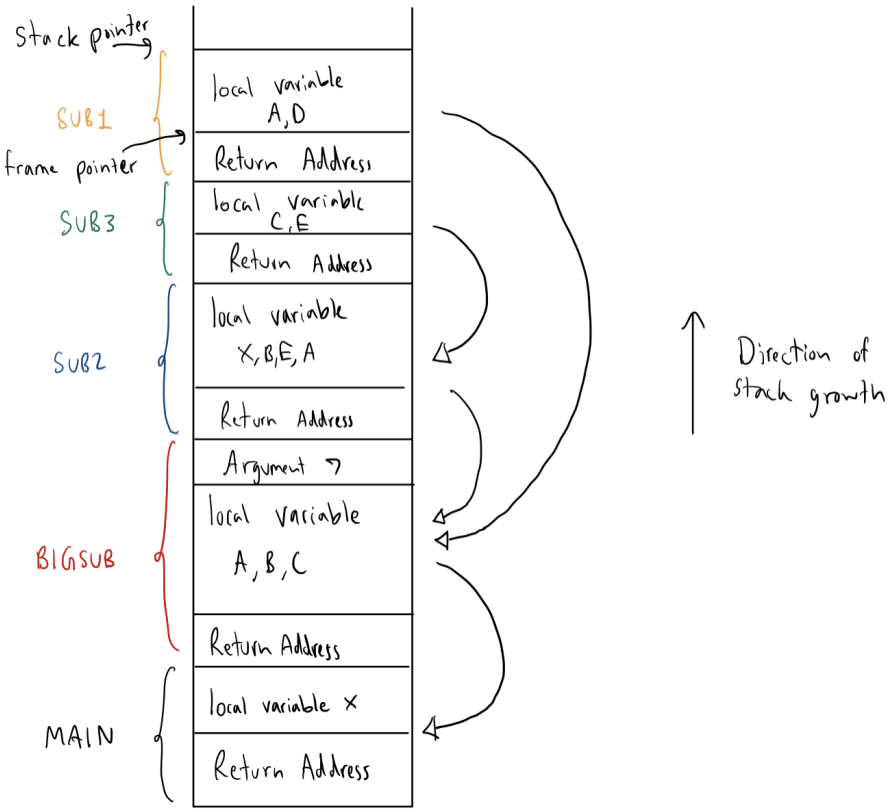

Figure 2: Runtime stack after line marked (1)

Fig. 2 shows the runtime stack of the program after line 1. There is no subroutine that is done at line 1. It means, that every subroutine is still in the stack. Notice that, when calling some procedure, there is no arguments passed through the next subroutine as some procedure doesn't need an input.

2.5 e

MAIN cannot invoke SUB3 as it is not visible in MAIN, because SUB3 is nested inside BIGSUB and SUB2. SUB3 cannot invoke MAIN as MAIN is in the surrounding scope.

3 Parameter Passing

3.1 1

All parameters are passed by call-by-name semantics. The expression is evaluated every time.
Iteration 1
$i = 1, j = 0$
$a_1 = a_2 + a_4$ where a_2 is $i + 1$ and a_4 is j. Thus, $a_1 = (i + 1) + j = (1 + 1) + 0 = 2$. Value of a_1 is bound with i. Thus, $i = 1 \rightarrow 2$.

$i = 2, j = 0$
$a_4 = a_4 + 1$ where a_4 is j. Thus, $a_4 = j + 1 = 0 + 1 = 0$. Value of a_4 is bound with j. Thus, $j = 0 \rightarrow 1$.

$i = 2, j = 1$
$a_5 = (a_3 + a_2) * 2$ where a_3 is $i * 4$ and a_2 is $i + 1$. Thus, $a_5 = (i * 4 + i + 1) * 2 = (2 * 4 + 2 + 1) * 2 = 22$. Value of a_5 is bound with j. Thus, $j = 1 \rightarrow 22$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(i, i + 1, i * 4, j, j) = (2, 3, 8, 22, 22)$.

Iteration 2
$i = 2, j = 22$
$a_1 = a_2 + a_4$ where a_2 is $i + 1$ and a_4 is j. Thus, $a_1 = (i + 1) + j = (2 + 1) + 22 = 25$. Value of a_1 is bound with i. Thus, $i = 2 \rightarrow 25$.

$i = 25, j = 22$
$a_4 = a_4 + 1$ where a_4 is j. Thus, $a_4 = j + 1 = 22 + 1 = 23$. Value of a_4 is bound with j. Thus, $j = 22 \rightarrow 23$.

$i = 25, j = 23$
$a_5 = (a_3 + a_2) * 2$ where a_3 is $i * 4$ and a_2 is $i + 1$. Thus, $a_5 = (i * 4 + i + 1) * 2 = (25 * 4 + 25 + 1) * 2 = 252$. Value of a_5 is bound with j. Thus, $j = 23 \rightarrow 252$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(i, i + 1, i * 4, j, j) = (25, 26, 100, 252, 252)$.

Iteration 3
$i = 25, j = 252$
$a_1 = a_2 + a_4$ where a_2 is $i + 1$ and a_4 is j. Thus, $a_1 = (i + 1) + j = (25 + 1) + 252 = 278$. Value of a_1 is bound with i. Thus, $i = 25 \rightarrow 278$.

$i = 278, j = 252$
$a_4 = a_4 + 1$ where a_4 is j. Thus, $a_4 = j + 1 = 252 + 1 = 253$. Value of a_4 is bound with j. Thus, $j = 252 \rightarrow 253$.

$i = 278, j = 253$
$a_5 = (a_3 + a_2) * 2$ where a_3 is $i * 4$ and a_2 is $i + 1$. Thus, $a_5 = (i * 4 + i + 1) * 2 = (278 * 4 + 278 + 1) * 2 = 2782$. Value of a_5 is bound with j. Thus, $j = 253 \rightarrow 2782$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(i, i + 1, i * 4, j, j) = (278, 279, 1112, 2782, 2782)$.

3.2 2

Parameters (a_1, a_4, a_5) are passed by call-by-reference semantics, and (a_2, a_3) are passed by call-by-need semantics.
Iteration 1
$i = 1, j = 0$
$a_1 = a_2 + a_4$ where a_2 is $i + 1$ and a_4 is j. Thus, $a_1 = (i + 1) + j = (1 + 1) + 0 = 2$. $a_2 = 2$ is evaluated the first time only. Value of a_1 is bound with i. Thus, $i = 1 \rightarrow 2$.

$i = 2, j = 0$
$a_4 = a_4 + 1$ where a_4 is j. Thus, $a_4 = j + 1 = 0 + 1 = 0$. Value of a_4 is bound with j. Thus, $j = 0 \rightarrow 1$.

$i = 2, j = 1$
$a_5 = (a_3 + a_2) * 2$ where a_3 is $i * 4$ and $a_2 = 2$. Thus, $a_5 = (i * 4 + a_2) * 2 = (2 * 4 + 2) * 2 = 20$. $a_3 = 8$ is evaluated the first time only. Value of a_5 is bound with j. Thus, $j = 1 \rightarrow 20$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(i, a_2, a_3, j, j) = (2, 2, 8, 20, 20)$.

Iteration 2
$i = 2, j = 20$
$a_1 = a_2 + a_4$ where $a_2 = 2$ and a_4 is j. Thus, $a_1 = a_2 + j = 2 + 20 = 22$. Value of a_1 is bound with i. Thus, $i = 2 \rightarrow 22$.

$i = 22, j = 20$
$a_4 = a_4 + 1$ where a_4 is j. Thus, $a_4 = j + 1 = 20 + 1 = 21$. Value of a_4 is bound with j. Thus, $j = 20 \rightarrow 21$.

$i = 22, j = 21$
$a_5 = (a_3 + a_2) * 2$ where $a_3 = 8$ and $a_2 = 2$. Thus, $a_5 = (a_3 + a_2) * 2 = (8 + 2) * 2 = 20$. Value of a_5 is bound with j. Thus, $j = 21 \rightarrow 20$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will

be $(i, a_2, a_3, j, j) = (22, 2, 8, 20, 20)$.

Iteration 3
$i = 22, j = 20$
$a_1 = a_2 + a_4$ where $a_2 = 2$ and a_4 is j. Thus, $a_1 = a_2 + j = 2 + 20 = 22$. Value of a_1 is bound with i. Thus, $i = 22 \rightarrow 22$.

$i = 22, j = 20$
$a_4 = a_4 + 1$ where a_4 is j. Thus, $a_4 = j + 1 = 20 + 1 = 21$. Value of a_4 is bound with j. Thus, $j = 20 \rightarrow 21$.

$i = 22, j = 21$
$a_5 = (a_3 + a_2) * 2$ where $a_3 = 8$ and $a_2 = 2$. Thus, $a_5 = (a_3 + a_2) * 2 = (8 + 2) * 2 = 20$. Value of a_5 is bound with j. Thus, $j = 21 \rightarrow 20$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(i, a_2, a_3, j, j) = (22, 2, 8, 20, 20)$.

3.3 3.3

All parameters $(a_1, a_2, a_3, a_4, a_5)$ are passed by call-by-value semantics. These formal parameters $(a_1, a_2, a_3, a_4, a_5)$ are $(1, 2, 4, 0, 0)$.
Iteration 1
$a_1 = a_2 + a_4$ where $a_2 = 2$ and $a_4 = 0$. Thus, $a_1 = 2 + 0 = 2$.
$a_4 = a_4 + 1$ where $a_4 = 0$. Thus, $a_4 = 0 + 1 = 1$.
$a_5 = (a_3 + a_2) * 2$ where $a_3 = 4$ and $a_2 = 2$. Thus, $a_5 = (4 + 2) * 2 = 12$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(2, 2, 4, 1, 12)$.

Iteration 2
$a_1 = a_2 + a_4$ where $a_2 = 2$ and $a_4 = 1$. Thus, $a_1 = 2 + 1 = 3$.
$a_4 = a_4 + 1$ where $a_4 = 1$. Thus, $a_4 = 1 + 1 = 2$.
$a_5 = (a_3 + a_2) * 2$ where $a_3 = 4$ and $a_2 = 2$. Thus, $a_5 = (4 + 2) * 2 = 12$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(3, 2, 4, 2, 12)$.

Iteration 3
$a_1 = a_2 + a_4$ where $a_2 = 2$ and $a_4 = 2$. Thus, $a_1 = 2 + 2 = 4$.
$a_4 = a_4 + 1$ where $a_4 = 2$. Thus, $a_4 = 2 + 1 = 3$.
$a_5 = (a_3 + a_2) * 2$ where $a_3 = 4$ and $a_2 = 2$. Thus, $a_5 = (4 + 2) * 2 = 12$. When showing $(a_1, a_2, a_3, a_4, a_5)$ at the end of this iteration, it will be $(4, 2, 4, 3, 12)$.

4 Lambda Calculus

4.1 Rewrite the expression using parentheses

4.1.1 b

$$\begin{aligned}
\lambda x\ .\ \lambda y\ .\ \lambda z\ .\ z\ y\ x &\equiv \lambda x\ .\ (\lambda y\ .\ \lambda z\ .\ z\ y\ x) \\
&\equiv \lambda x\ .\ (\lambda y\ .\ (\lambda z\ .\ z\ y\ x)) \\
&\equiv \lambda x\ .\ (\lambda y\ .\ (\lambda z\ .\ (z\ y\ x))) \\
&\equiv \lambda x\ .\ (\lambda y\ .\ (\lambda z\ .\ ((z\ y)\ x)))
\end{aligned}$$

4.1.2 c

$$\begin{aligned}
\lambda x\ .\ x\ y\ \lambda z\ .\ w\ \lambda w\ .\ w\ x\ z &\equiv \lambda x\ .\ (x\ y\ \lambda z\ .\ w\ \lambda w\ .\ w\ x\ z) \\
&\equiv \lambda x\ .\ ((x\ y)\ \lambda z\ .\ w\ \lambda w\ .\ w\ x\ z) \\
&\equiv \lambda x\ .\ ((x\ y)\ \lambda z\ .\ (w\ \lambda w\ .\ w\ x\ z)) \\
&\equiv \lambda x\ .\ ((x\ y)\ \lambda z\ .\ (w\ (\lambda w\ .\ w\ x\ z))) \\
&\equiv \lambda x\ .\ ((x\ y)\ \lambda z\ .\ (w\ (\lambda w\ .\ (w\ x\ z)))) \\
&\equiv \lambda x\ .\ ((x\ y)\ \lambda z\ .\ (w\ (\lambda w\ .\ ((w\ x)\ z))))
\end{aligned}$$

4.1.3 d

$$\begin{aligned} x\ \lambda z\ .\ x\ \lambda w\ .\ w\ z\ y &\equiv x\ (\lambda z\ .\ x\ \lambda w\ .\ w\ z\ y) \\ &\equiv x\ (\lambda z\ .\ (x\ \lambda w\ .\ w\ z\ y)) \\ &\equiv x\ (\lambda z\ .\ (x\ (\lambda w\ .\ w\ z\ y))) \\ &\equiv x\ (\lambda z\ .\ (x\ (\lambda w\ .\ (w\ z\ y)))) \\ &\equiv x\ (\lambda z\ .\ (x\ (\lambda w\ .\ ((w\ z)\ y)))) \end{aligned}$$

4.1.4 e

$$\begin{aligned} \lambda z.((\lambda s.s\ q)\ (\lambda q.q\ z))\ \lambda z.z\ z &\equiv (\lambda z.((\lambda s.s\ q)\ (\lambda q.q\ z)))\ (\lambda z.z\ z) \\ &\equiv (\lambda z.((\lambda s.(s\ q))\ (\lambda q.(q\ z))))\ (\lambda z.(z\ z)) \end{aligned}$$

4.2 Circle all the free variables

4.2.1 a

$$\begin{aligned} \lambda z\ .\ z\ x\ \lambda y\ .\ y\ z &\equiv \lambda z\ .\ z\ x\ \lambda y\ .\ y\ z \\ &\equiv \lambda z\ .\ (z\ x\ \lambda y\ .\ y\ z) \\ &\equiv \lambda z\ .\ ((z\ x)\ (\lambda y\ .\ y\ z)) \\ &\equiv \lambda z\ .\ ((z\ \textcircled{x})\ (\lambda y\ .\ (y\ z))) \end{aligned}$$

4.2.2 b

$$(\lambda x.x)\ (\lambda x.x\ (\lambda y.y))\ z \equiv ((\lambda x.x)\ (\lambda x.(x\ (\lambda y.y))))\ \textcircled{z}$$

4.2.3 c

$$\begin{aligned} \lambda p.(\lambda z.f\ \lambda x.z\ y)\ p\ x &\equiv (\lambda p.(\lambda z.f\ \lambda x.z\ y))\ (p\ x) \\ &\equiv (\lambda p.((\lambda z.f\ \lambda x.z)\ y))\ (p\ x) \\ &\equiv (\lambda p.((\lambda z.(\textcircled{f}\ (\lambda x.z)))\ \textcircled{y}))\ (\textcircled{p}\ \textcircled{x}) \end{aligned}$$

4.2.4 d

$$\begin{aligned} \lambda x\ .\ x\ y\ \lambda x\ .\ y\ x &\equiv \lambda x\ .\ (x\ y\ \lambda x\ .\ y\ x) \\ &\equiv \lambda x\ .\ ((x\ y)\ \lambda x\ .\ y\ x) \\ &\equiv \lambda x\ .\ ((x\ \textcircled{y})\ \lambda x\ .\ (\textcircled{y}\ x)) \end{aligned}$$

4.2.5 e

$$\lambda x\ .\ x\ (x\ y) \equiv \lambda x\ .\ (x\ (x\ \textcircled{y}))$$

4.3 α-conversion

4.3.1 a

$$(\lambda xy.zx)(\lambda x.x\textcircled{y})$$

This expression has y as the free variable which is bound in $\lambda xy.zx$. We need to do α-conversion before legally do β-reduce.

The correct way

$$\begin{aligned} (\lambda xy.zx)(\lambda x.xy) &\rightarrow_\alpha (\lambda xt.zx)(\lambda x.xy) \\ &\rightarrow (\lambda x.\lambda t.zx)(\lambda x.xy) \\ &\rightarrow_\beta [x \rightarrow \lambda x.xy](\lambda t.zx) \\ &\rightarrow_\beta \lambda t.z\lambda x.xy \end{aligned}$$

The incorrect way

$$\begin{aligned}(\lambda xy.zx)(\lambda x.xy) &\to (\lambda x.\lambda y.zx)(\lambda x.xy)\\ &\to_\beta [x \to \lambda x.xy](\lambda y.zx)\\ &\to_\beta \lambda y.z\lambda x.xy\end{aligned}$$

We can see that the two methods don't reduce to the same expression. y in the first one is a free variable, whereas the second one is bound in $\lambda y.z\lambda x.xy$.

4.3.2 b

$$(\lambda x.\lambda yz.xyz)(\lambda z.zx)$$

This expression has x as the free variable which is bound in $\lambda x.\lambda yz.xyz$. We need to do α-conversion before legally do β-reduce.
The correct way

$$\begin{aligned}(\lambda x.\lambda yz.xyz)(\lambda z.zx) &\to (\lambda x.\lambda y.\lambda z.xyz)(\lambda z.zx)\\ &\to_\alpha (\lambda t.\lambda y.\lambda z.tyz)(\lambda z.zx)\\ &\to_\beta [t \to \lambda z.zx](\lambda y.\lambda z.tyz)\\ &\to_\beta \lambda y.\lambda z.(\lambda z.zx)yz\end{aligned}$$

The incorrect way

$$\begin{aligned}(\lambda x.\lambda yz.xyz)(\lambda z.zx) &\to (\lambda x.\lambda y.\lambda z.xyz)(\lambda z.zx)\\ &\to_\beta [x \to \lambda z.zx](\lambda y.\lambda z.xyz)\\ &\to_\beta \lambda y.\lambda z.(\lambda z.zx)yz\end{aligned}$$

We can see that the two methods reduce to the same expression.

4.3.3 c

$$(\lambda x.xz)(\lambda xz.xy)$$

This expression has y as the free variable which is not bound in $\lambda x.xz$.We don't need to do α-conversion.

$$\begin{aligned}(\lambda x.xz)(\lambda xz.xy) &\to_\beta [x \to \lambda xz.xy](xz)\\ &\to_\beta \lambda xz.xyz\end{aligned}$$

4.3.4 d

$$(\lambda x.xy)(\lambda x.y)$$

This expression has y as the free variable which is bound in $\lambda x.xy$. We need to do α-conversion before legally do β-reduce.
The correct way

$$\begin{aligned}(\lambda x.xy)(\lambda x.y) &\to_\alpha (\lambda x.xt)(\lambda x.y)\\ &\to_\beta [x \to y](xt)\\ &\to_\beta yt\end{aligned}$$

The incorrect way

$$\begin{aligned}(\lambda x.xy)(\lambda x.y) &\to_\alpha (\lambda x.xy)(\lambda x.y)\\ &\to_\beta [x \to y](xy)\\ &\to_\beta yy\end{aligned}$$

We can see that the two methods don't reduce to the same expression.

4.4 β-reduce to normal form

4.4.1 a

$$\begin{aligned}((\lambda y\ .\ z\ y)\ x)(\lambda x\ .\ x\ y) &\to_\beta ([y \to x](z\ y))(\lambda x\ .\ x\ y)\\ &\to_\beta (z\ x)(\lambda x\ .\ x\ y)\end{aligned}$$

4.4.2 b

$$\begin{aligned}(\lambda x\ .\ x\ x\ x)(\lambda x\ .\ x\ x\ x) &\to_\beta [x \to \lambda x\ .\ x\ x\ x](x\ x\ x)\\ &\to_\beta (\lambda x\ .\ x\ x\ x)(\lambda x\ .\ x\ x\ x)(\lambda x\ .\ x\ x\ x)\end{aligned}$$

We can see that it reduces to itself. Application rule can be applied infinitely. This expression doesn't have the normal form.

4.4.3 c

$$\begin{aligned}(\lambda x\ .\ x)(\lambda y\ .\ x\ y)(\lambda z\ .\ x\ y\ z) &\to_\alpha (\lambda x\ .\ x)(\lambda t\ .\ x\ t)(\lambda z\ .\ x\ y\ z)\\ &\to_\beta (\lambda x\ .\ x)[t \to \lambda z\ .\ x\ y\ z](x\ t)\\ &\to_\beta (\lambda x\ .\ x)(x\ \lambda z\ .\ x\ y\ z)\\ &\to_\alpha (\lambda t\ .\ t)(x\ \lambda z\ .\ x\ y\ z)\\ &\to_\beta [t \to x\ \lambda z\ .\ x\ y\ z](t)\\ &\to_\beta x\ \lambda z\ .\ x\ y\ z\end{aligned}$$

4.4.4 d

$$\begin{aligned}\mathrm{MULT}\lceil 0\rceil\lceil 3\rceil &\to (\lambda mnf.m(nf))(\lambda fx.x)(\lambda fx.f(f(fx)))\\ &\to_\beta ([m \to \lambda fx.x](\lambda nf.m(nf)))\ (\lambda fx.f(f(fx)))\\ &\to_\beta (\lambda nf.(\lambda fx.x)\ (nf)))\ (\lambda fx.f(f(fx)))\\ &\to_\beta [n \to \lambda fx.f(f(fx))](\lambda f.(\lambda fx.x)\ (nf)))\\ &\to_\beta \lambda f.(\lambda fx.x)\ (\lambda fx.f(f(fx))\ f))\\ &\to \lambda f.(\lambda fx.x)\ (\lambda fx.f(f(fx)))\\ &\to_\beta [f \to (\lambda fx.f(f(fx)))](\lambda fx.x)\\ &\to_\beta \lambda fx.x\\ &\equiv 0\end{aligned}$$

4.4.5 e

$$\begin{aligned}\mathrm{EXP}\lceil 2\rceil\lceil 1\rceil &\to (\lambda mn.nm)(\lambda fx.f(fx))(\lambda fx.fx)\\ &\to_\beta ([m \to \lambda fx.f(fx)](\lambda n.nm))(\lambda fx.fx)\\ &\to_\beta (\lambda n.n\ (\lambda fx.f(fx)))(\lambda fx.fx)\\ &\to_\beta [n \to \lambda fx.fx](n\ (\lambda fx.f(fx)))\\ &\to_\beta (\lambda fx.fx)\ (\lambda fx.f(fx))\\ &\to_\beta [f \to \lambda fx.f(fx)](\lambda x.fx)\\ &\to_\beta \lambda x.(\lambda fx.f(fx)\ x)\\ &\to_\eta \lambda fx.f(fx)\\ &\equiv 2\end{aligned}$$