1 Running the Examples

Having a concrete implementation means that now we can try and run the examples of which we only listed the code and gave the type. Let us start with:

```
ex₁ = eval i >>= (λv. i := (v × 10) >> eval i)
```

We wish to evaluate the term $runST\ ex_1\ (5 : Nil)$, assuming that:

```
i=Reference (λ h.(hRead h (_:Z),h)) (λ v.λ h.( (),hWrite v h (_:Z)))
```

the resulting evaluation is:

```
= runST (eval i >>= (λv. ... )) (5:Nil) \
= runST ((ST λh.(hRead h (_:Z),h)) >> =(λv. ... )) (5:Nil) \
= runST (i := 5 × 10 >> ... ) (5:Nil) \
= runST ((ST λh.((),hWrite 50 h (_:Z))) >> ... ) (5:Nil) \
= runST (eval i) (50:Nil) \
= runST (ST λh.(hRead h (_:Z),h)) (50:Nil) \
= 50
```

This is exactly the result we would have expected. Also notice that for all practical purpose our programs act as if exactly one heap is accessible at all times. Depending on the capabilities of the runtime this might also be essentially true, as most of the parts of a heap could be shared thanks to the immutability of the data structures of the various heaps which are allocated with various shared parts.

Let us now consider our second example:

```
ex'₂ =
    do 10 >>= (λi.
    do "hello␣" >>= (λs.
    do s *= (λx.x++"world")
       let i' = downcast i
       v←eval s
       x←eval i'
       return v ++ show x))
```

We will evaluate this as if it were a program launched all by itself (that is with an empty heap and no external references), by evaluating the term $runST\ ex_2'\ Nil$; we will evaluate this term with a (simpler) small-step semantics that refers to the effect our statements have on the heap and the various bound variables:

Statement	i	s	x	v	Heap
ex_2ˆ'=do 10≫+(λi.	-	-	-	-	10::Nil
do(_ ˆ")helloˆ"≫+(λs.	10	"Hello"	-	-	"Hello"::10::Nil
do s*=(λx.x++ˆ" worldˆ")	10	"Hello World"	-	-	"Hello World"::10::Nil
let iˆ'=downcast i	10	"Hello World"	-	-	"Hello World"::10::Nil
v?eval s	10	"Hello World"	10	"Hello World"	"Hello World"::10::Nil
x?eval i'	10	"Hello World"	10	"Hello World"	"Hello World"::10::Nil
return v++show x))	-	-	-	-	Nil

The returned result is (as expected) "$10 Hello\ World$" .