1 Subtyping Heaps and References

Let us consider the types of the references of the example above, given $Heap\ h_0\ ref$ for some initial heap h_0 and some reference type ref:

```
i : ref (New h0 Int) Int
s : ref (New (New h0 Int) String) String
```

The heap h_0 is the starting heap at the beginning of the program. If the program is used as a "main", that is it is just launched, then h_0 will be an empty heap; in other cases, like using our computation inside another, larger computation, will mean that h_0 will be the current heap that is available where the example is launched.
Now let us consider the two statements:

```
v ← s
x ← i
```

These two are incompatible, since the first statement forces our monad to be of type $ref\ (New\ (New\ h_0\ Int)\ String)$ and thus this type will have to be in all subsequent statements of the same monadic program, but the second statement expects our monad to be of type $ref\ (New\ h_0\ Int)$. This is absolutely reasonable on the part of the type system, but it is quite unacceptable given our circumstances: why cannot we use a reference that refers to a smaller heap where we have a larger heap available? Indeed, all the values that reference i requires are available (plus some more that are irrelevant to i) in the heap that we have when the reference s is in scope. It would be interesting to make it possible to use a reference that requires a smaller (less defined) heap when a larger (more specified) heap is available. This kind of notion is clearly a notion of subtyping between heaps and references.
To define a subtyping relationship between heaps, let us start by giving a subtyping predicate:

```
Subtype α β
```

Which for brevity we will also write

$$\alpha \leq \beta$$

The fact that α is a subtype of β implies that α is more specified than β, and as such it may be used in any context where a β is expected. For this reason we also give a casting (or coercion) function that converts from a value of type α to a value of its supertype β:

```
downcast :α → β
```

We also know that the subtyping relation is reflexive and transitive, so we will add these rules as instances of the subtyping predicate:

```
Subtype α α downcast=λ x.x
```

```
Subtype α β ∧ Subtype β γ ⇒ Subtype α γ downcast=downcast ∘ downcast
```

Since we are mostly interested in subtyping between references, when is it safe to downcast them? As already discussed, a first criterion for downcasting a reference is that the heap h that the reference manipulates is actually smaller than the current heap; this means that:

```
Heap h ref ∧ Heap h' ref ∧ h' ≤ h ⇒ ref h α ≤ ref h' α
```

This means that references are contravariant with respect to their heap. Also, the second argument of references allows for the intuitive kind of conversion:

```
α ≤ α' ⇒ ref h α ≤ ref h α'
```

This second rule shows that a reference to a value α behaves exactly like that value, thereby allowing conversions that mirror the behavior of the value that our reference is standing for.

These rules are quite hard to concretely instance. This is why we will not be able to instance these rules once and for all in a highly parametric fashion, but instead we will just give some specific instances that are of particular use to us when we are dealing with concrete heaps and references. Also, we will mostly deal with subtyping between values and not subtyping between functions.

At this point we can rewrite the "incriminated" example above so that it compiles:

```
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))
```

Now the type of ex'_2 will not only reflect the type of the result of the computation, but also the requirement of subtyping between heaps built with the New type operator:

$$\text{Heap}\ h_0\ \ \mathrm{ref}\ \wedge\ \mathrm{ref}\ (\text{New}\ h_0\ \mathbf{Int})\ \mathbf{Int} \leq \mathrm{ref}\ (\text{New}\ (\text{New}\ h_0\ \mathbf{Int})\ \mathbf{String})\ \mathbf{Int} \Rightarrow ex_2' : \mathrm{ref}\ h_0\ \mathbf{String}$$