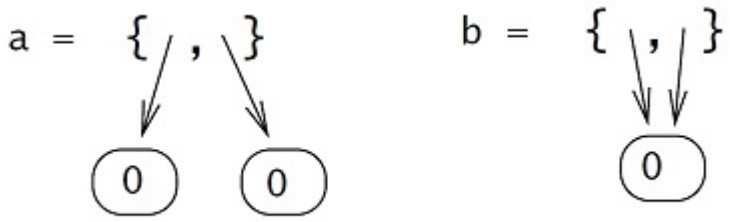
1. Solution 13.1.1: $a = \{\text{ref } 0, \text{ref } 0\}$ and $b = (\lambda x : \mathsf{Ref\ Nat}.\ \{x, x\})(\text{ ref } 0)$. In the case of a is created with two different references, while b is created with the same reference.



2. Solution 13.1.2: The new update would not behave the same way. Calls to lookup with any index other than the one given to update will now diverge. The point is that we need to make sure we look up the value stored in a before we overwrite it with the new function. Otherwise, when we get around to doing the lookup, we will find the new function, not the original one.

3. Solution 13.1.3:

   - let $r =$ ref 0 in
   - let $s = r$ in
   - free r;
   - let $t =$ ref $true$ in
   - $t := false$;
   - succ $(!s)$

   If the language provides a primitive free that takes a reference cell as argument and releases its storage so that (say) the very next allocation of a reference cell will obtain the same piece of memory. Then the program will evaluate to the stuck term succ $false$. Note that aliasing plays a crucial role here in setting up the problem, since it prevents us from detecting invalid deallocations by simply making free r illegal if the variable r appears free in the remainder of the expression.

4. Solution 13.4.1:

   - let $r_1 =$ ref $(\lambda x : \mathsf{Nat}.\ 0)$ in
   - let $r_2 =$ref $(\lambda x : \mathsf{Nat}.(!r_1)\ x)$ in
     $(r_1 := (\lambda x :\mathsf{Nat}.(!r_2)\ x);$
     $r_2);$

5. Solution 13.5.2: Let μ be a store with a single location l
   $\mu = (l \mapsto \lambda x : \mathsf{Unit}.\ (!l)(x))$,
   and Γ the empty context. Then μ is well typed with respect to both of the following store typings:
   $\Sigma_1 = l$:Unit→Unit
   $\Sigma_2 = l$: Unit→ (Unit→Unit).

6. Solution 13.5.8: There are well-typed terms in this system that are not strongly normalizing. For example:

   - $t1 = \lambda r : \mathsf{Ref}\ (\mathrm{Unit} \to \mathrm{Unit}).$
     $(r := (\lambda x : \mathrm{Unit}.(!r)\ x);$
     $(!r)\ \mathsf{unit});$
   - $t_2 = \mathsf{ref}\ (\lambda x : \mathrm{Unit}.x);$

   The term $(t_1\ t_2)$ yields a (well-typed) divergent term. Its type is Unit.