```
let main =
  let swap = λx → case h@x of
    MkPair a b → let h@r = MkPair b a in r
  in let zero = Z
  in let one = S zero
  in let v = MkPair zero one
  in swap v
in main
```

Figure 18. Transformed code

```
data Pair [β1, β2, β3, β4] [α1, α2]
  = MkPair α1^(β1,β2) α2^(β3,β4)
data Nat [β1, β2] []
  = Z
  | S Nat^(β1,β2)
```

Figure 19. Annotated datatype definitions

```
let main :: (Pair [0, 0, 0, 0] [Nat [0, 0] [], Nat [0, 0] []]) 1
  main =1,1
  let swap :: (∀ {α1, α2, β1, β2, β3, β4, β5} . [] ⇒
    (Pair [β1, β2, β3, β4] [α1, α2])1,1 →
    (Pair [β3, β4, β1, β2] [α2, α1])^β5) 1
    swap =1,1 λ1 x → case h@x of
      MkPair a b → let h@r =1,1 MkPair1 b a in r
  in let zero :: (∀ {β1, β2} . [] ⇒ Nat [β1, β2] []) 0
    zero =0,ω Z0
  in let one :: (∀ {β1, β2} . [] ⇒ Nat [β1, β2] []) 0
    one =0,1 S0 zero
  in let v :: (∀ {β1, β2, β3, β4, β5, β6, β7, β8} . [] ⇒
    Pair [β1, β2, β3, β4] [Nat [β5, β6] [], Nat [β7, β8] []]) 1
    v =1,1 MkPair1 zero one
  in swap v
in main
```

Figure 20. Annotated code

```
let main =1,1
  let swap =1,1 (λx → case h@x of
    MkPair a b → let h@r =1,1 (MkPair b a)1 in r) 1
  in let zero =0,ω Z0
  in let one =0,1 (S zero)0
  in let v =1,1 (MkPair zero one)1
  in ([·] v) [swap]
in main
```

Figure 21. Evaluable code

A. Example

For reasons of conciseness, the paper proper provides only a very formal view on our work, and there is no room for a reasonable example. Recognizing that the formal development may not be immediately clear, we provide in this appendix a complete worked example, taking a our of work starting from a simple piece of code, to the results that our analysis gives. In this chapter we will take a look at a simple example that shows how everything ties together. With this example we attempt to show how the static semantics, the dynamic semantics and heap recycling work in practice.

A.1 Code

The example is very simple

$$swap\ h@(a, b) = h@(b, a)$$
$$main = swap\ (0, 1)$$

The idea behind the *swap* function is that it swaps the elements of the pair without doing any additional heap allocations, beyond the allocation of $(0, 1)$.

Unfortunately this code is not immediately usable in our system, so it has to be transformed somewhat to fit our the chosen syntax. The result of the transformation can be found in Figure 18.

The transformed code employs various datatypes that are defined in Figure 19.

A.2 Type inference

The next step is to let the type inference algorithm annotate the source code (Figure 20). Nothing uses the result of the *main* expression, so let's set the use/demand variables within its result to $\mathbb{0}$ (for simplicity's sake). Also, the outer ν has been set to $\mathbb{1}$ to drive evaluation, as required in Section 4.4. In the type of *swap* we can see that the argument is required to be unique, which is logical if we consider that the associated heap binding will be reused. Since the function does not use the values of the pair, its types and annotations are variables.

Please note that *main*, *swap* and v are all used exactly once. So, we could perform all the optimizations associated with sharing analysis (requires "at most once") and strictness analysis (requires "at least once"). The expressions *zero* and *one* are not used at all, so the optimizations for absence analysis could be used. The optimizations are described in Section **??**.

A.3 Evaluation

Evaluation uses yet another representation of the code, so lets start by transforming the result from type inference to the valid representation (Figure 21).

The valid representation can be put directly into a configuration. For completeness we included the entire evaluation sequence. Note that at any time, only a single evaluation rule can be applied, taking us from configuration to configuration.

Start with:

$$\langle \emptyset;\ \mathbf{let}\ swap =^{\mathbb{1},\mathbb{1}} \cdots \mathbf{in}\ \mathbf{let}\ zero =^{\mathbb{0},\omega} \cdots \mathbf{in}\ \mathbf{let}\ one =^{\mathbb{0},\mathbb{1}} \cdots \mathbf{in}\ \mathbf{let}\ v =^{\mathbb{1},\mathbb{1}} \cdots \mathbf{in}\ ([\cdot]\ v)\ [swap];\ \epsilon \rangle$$

Apply LET multiple times:

$$\langle swap =^{\mathbb{1},\mathbb{1}} \cdots,\ zero =^{\mathbb{0},\omega} \cdots,\ one =^{\mathbb{0},\mathbb{1}} \cdots,\ v =^{\mathbb{1},\mathbb{1}} \cdots;\ ([\cdot]\ v)\ [swap];\ \epsilon \rangle$$

Apply UNWIND:

$$\langle swap =^{\mathbb{1},\mathbb{1}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{1},\mathbb{1}} \cdots ;\\ swap;\\ [\cdot]\ v, \epsilon\rangle$$

Apply LOOKUP:

$$\langle zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{1},\mathbb{1}} \cdots ;\\ (\lambda x \to \cdots)\ ^{\mathbb{1}};\\ \#^{\mathbb{0},\mathbb{0}}\ swap, [\cdot]\ v, \epsilon\rangle$$

Apply UPDATE:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{1},\mathbb{1}} \cdots ;\\ (\lambda x \to \cdots)\ ^{\mathbb{1}};\\ [\cdot]\ v, \epsilon\rangle$$

Apply REDUCE, APP:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{1},\mathbb{1}} \cdots ;\\ \mathbf{case}\ h@v\ \mathbf{of}\\ \quad MkPair\ a\ b \to \cdots ;\\ \epsilon\rangle$$

Apply UNWIND:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{1},\mathbb{1}} \cdots ;\\ v;\\ \mathbf{case}\ [\cdot]\ \mathbf{of}\ MkPair\ a\ b \to [^{v}/_{h}] \cdots ,\\ \epsilon\rangle$$

Apply LOOKUP:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ (MkPair\ zero\ one)^{\mathbb{1}};\\ \#^{\mathbb{0},\mathbb{0}}\ v,\\ \mathbf{case}\ [\cdot]\ \mathbf{of}\ MkPair\ a\ b \to [^{v}/_{h}] \cdots ,\\ \epsilon\rangle$$

Apply UPDATE:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{0},\mathbb{0}} \cdots ;\\ (MkPair\ zero\ one)^{\mathbb{1}};\\ \mathbf{case}\ [\cdot]\ \mathbf{of}\ MkPair\ a\ b \to [^{v}/_{h}] \cdots ,\\ \epsilon\rangle$$

Apply REDUCE:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{1},\mathbb{1}} MkPair\ one\ zero;\\ v;\\ \epsilon\rangle$$

Apply LOOKUP:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ;\\ MkPair\ one\ zero;\\ \#^{\mathbb{0},\mathbb{0}}\ v, \epsilon\rangle$$

Apply UPDATE:

$$\langle swap =^{\mathbb{0},\mathbb{0}} \cdots ,\\ zero =^{\mathbb{0},\omega} \cdots ,\\ one =^{\mathbb{0},\mathbb{1}} \cdots ,\\ v =^{\mathbb{0},\mathbb{0}} MkPair\ one\ zero,\\ MkPair\ one\ zero;\\ \epsilon\rangle$$

Since we evaluated to a value and the stack is empty, this is where the program terminates. Note that the termination condition does not hold. This is because we cheated a little bit during type inference. The generalized type of $main$ contained demand variables (which is not allowed according to Section 5). These were all set to $\mathbb{0}$ for simplicity. The demand variables express the demand on $zero$ and one, and these are exactly the values that make the termination condition fail.

So, we should not have evaluated this program in this way to begin with. However, $swap$ and v are indeed correctly annotated with $\mathbb{0}, \mathbb{0}$.