let main =
let swap = λx → case h@x of
MkPair a b → let h@r = MkPair b ain r
in letzero = Z
in letone = S zero
in letv ... =0 ,ω··· ,
one =0 ,1 ··· ,
v =1 ,1 ··· ;
(·) v) [swap ;
ϵ⟩

Figure 18. Transformed code

data Pair [β1, β2, β3, β4] [α1, α2]
= MkPair α1β1,β2 α2β3,β4

data Nat [β1, β2] []
= Z
| S Natβ1,β2

Figure 19. Annotated datatype definitions

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

\[
\text{swap } h \circ (a, b) = h \circ (b, a) \\
\text{main } = \text{swap } (0, 1)
\]

The idea behind the \text{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 de-

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 \text{main}
expression, so let’s set the use/demand variables within its result to
\(0\) (for simplicity’s sake). Also, the outer \(v\) has been set to \(\hat{\alpha}\) to
drive evaluation, as required in Section 4.4. In the type of \text{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 \text{main}, \text{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 \(\text{zero}\) and \(\text{one}\) are not used at
all, so the optimizations for absence analysis could be used. The
optimizations are described in Section ??.

let main =
let swap = λx → case h@x of
MkPair a b → let h@r = MkPair b ain r
in letzero = Z
in letone = S zero
in letv ... =0 ,ω··· ,
one =0 ,1 ··· ,
v =1 ,1 ··· ;
(·) v) [swap ;
ϵ⟩

Figure 20. Annotated code

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:

\[
\langle \rangle; \\
\text{let } \text{swap } =^{1,4} \ldots \text{in} \\
\text{let } \text{zero } =^{0,ω} \ldots \text{in} \\
\text{let } \text{one } =^{0,α} \ldots \text{in} \\
\text{let } v =^{1,α} \ldots \text{in} \\
(\langle \rangle v) [\text{swap } ;
\epsilon)
\]

Apply LET multiple times:

\[
(\text{swap } =^{1,4} \ldots , \\
\text{zero } =^{0,ω} \ldots , \\
\text{one } =^{0,α} \ldots , \\
v =^{1,α} \ldots , \\
\langle \rangle v) [\text{swap } ;
\epsilon)
\]

Apply UNWIND:
(swap =\cdot \ldots ,
zero =\cdot \ldots ,
one =\cdot \ldots ,
v =\cdot \ldots ;
swap;
[\cdot v,\epsilon])

Apply LOOKUP:
⟨zero =\cdot \ldots ,
one =\cdot \ldots ,
v =\cdot \ldots ;
swap;\[\cdot v,\epsilon⟩

Apply UPDATE:
⟨zero =\cdot \ldots ,
one =\cdot \ldots ,
v =\cdot \ldots ;
(\lambda x \rightarrow \cdot \ldots )^\dagger ;
#^0^0\mbox{ swap,}[\cdot v,\epsilon⟩

Apply REDUCE, APP:
⟨zero =\cdot \ldots ,
one =\cdot \ldots ,
v =\cdot \ldots ;
(\lambda x \rightarrow \cdot \ldots )^\dagger ;
\mbox{ case } h@v \mbox{ of }
\mbox{ MkPair a b } \rightarrow \cdot \ldots ;
\epsilon⟩

Apply UNWIND:
⟨zero =\cdot \ldots ,
one =\cdot \ldots ,
v =\cdot \ldots ;
\mbox{ case } [\cdot v] \mbox{ of } \mbox{ MkPair a b } \rightarrow [v/h] \cdot \ldots ,
\epsilon⟩

Apply LOOKUP:
⟨zero =\cdot \ldots ,
one =\cdot \ldots ,
(MkPair zero one)^\dagger ;
#^0^0\mbox{ v,}\mbox{ case } [\cdot v] \mbox{ of } \mbox{ MkPair a b } \rightarrow [v/h] \cdot \ldots ,
\epsilon⟩

Apply UPDATE:
⟨zero =\cdot \ldots ,
one =\cdot \ldots ,
(MkPair zero one)^\dagger ;
\mbox{ case } [\cdot v] \mbox{ of } \mbox{ MkPair a b } \rightarrow [v/h] \cdot \ldots ,
\epsilon⟩

Apply REDUCE: