Language Implementation

Assignment 1

Breno Fernando Guerra Marrão

April 4, 2024

Contents

1 Introduction

This report discusses the extension of a compiler for the SECD machine to incorporate operations for pairs and lists. The extension involves adding functions for pair manipulation (fst and snd) and list operations (head, tail, and null).

2 Implementation

2.1 Adittions to the fun language

To incorporate pairs and lists into the fun language, I extended its data structure with new terms specifically designed to handle these data types. Below are the alterations made to the fun language data structure:

2.1.1 Pairs

I introduced a new construct 'Pair' to represent pairs of values, as well as the two functions 'Fst' and 'Snd' to get the first element of the pair and the second one.

```
| Pair Term Term          -- Pair
| Fst Term                -- First term of pair
| Snd Term                -- Snd Term of pair
```

2.1.2 Lists

For the lists, I introduced `Empty` to represent an empty list, and `:$` to denote the concatenation of elements in a list. Additionally, I defined functions `MyNull` to determine if a list is empty, `MyHead` to retrieve the first element of a list, and `MyTail` to obtain the remaining elements of a list.

```
| Empty                  -- Representation of empty List
| Term :$ Term           -- Representation of a cons list
| MyNull Term            -- yield 0 if it is null and 1 if it isn't
| MyHead Term            -- yields the head of a list
| MyTail Term            -- yields the tail of a list
```

2.2 Additions to the SECD compiler

Because I utilized church encondings to express these new terms, this means that I can just recompile to the lambda form and call compile again recursively

```
compile (MyTrue) sym
  =  compile (Lambda "x" (Lambda"y" (Var "x"))) sym

compile (MyFalse) sym
  = compile (Lambda "x" (Lambda"y" (Var "y"))) sym

compile (Fst e) sym
  = compile (App(Lambda "p" (App (Var "p")  MyTrue)) e) sym

compile (Snd e) sym
  = compile (App(Lambda "p" (App (Var "p")  MyFalse)) e) sym

compile (Pair e1 e2) sym
  = compile (Lambda "x" (App (App (Var "x") e1) e2)) sym

compile (Empty) sym
  = compile (Pair (Const 0) (Const 0)) sym

compile (MyNull e) sym
  = compile (Fst e) sym

compile (e1 :$ e2) sym
  = compile (Pair (Const 1) (Pair e1 e2)) sym

compile (MyHead e) sym
  = compile (Fst (Snd (e))) sym

compile (MyTail e) sym
  = compile (Snd (Snd (e))) sym
```

3 Examples

3.1 append

```
append = (Fix
      (Lambda "f"
       (Lambda "l"
        (Lambda "n"
          (
           IfZero (MyNull  (Var"l"))
                  ((Var "n") :$ Empty)
                  ((MyHead (Var "l")) :$
                           (App(App (Var "f") (MyTail (Var "l"))) (Var "n"))
                  ) )
        ))))
exList = (((Const 0) :$((Const 2) :$ ((Const 1):$ Empty))))
exAppend = (App (App append (exList))(Const 5))
```

3.2 length

```
tamanho = (Fix
      (Lambda "f"
       (Lambda "l"
         (
          IfZero (MyNull  (Var"l"))
                          (Const 0)
                          ((Const 1) :+ (App (Var "f") (MyTail (Var "l"))))
         )
         )))
exTamanho1 =  (App tamanho exList)
exTamanho2 = (App tamanho (App (App append (exList))(Const 5)))
```

3.3 zip

```
myzip = (Fix
      (Lambda "f"
       (Lambda "l1"
        (Lambda "l2"
          (
           IfZero (MyNull  (Var"l1"))
                  (Empty)
                  (IfZero(MyNull (Var "l2"))
                         (Empty)
                         ((Pair (MyHead (Var"l1"))(MyHead (Var"l2"))))
                         :$
                         (App(App(Var"f") (MyTail (Var "l1")))(MyTail (Var "l2")))
          )
        )))))
exZip = App (App myzip exList ) (exList)
exTamanho3 =  (App tamanho (exZip)
```

3.4 map

```
mymap = (Fix
      (Lambda "f"
       (Lambda "l"
        (Lambda "func"
          (
           IfZero
           (MyNull  (Var"l"))
           (Empty)
           ((App (Var "func")
                 ( MyHead (Var "l")))
            :$
            (App(App(Var"f") (MyTail (Var "l")))(Var "func")))
        )))))

exMap1 = (App (App mymap(exZip))(somaPar))
exMap2 =(App (App mymap(exAppend))(ex3))
exTamanho4 =  (App tamanho (exMap2))
```

3.5 Pair Addition

```
somaPar = (Lambda "x" ((Fst (Var "x")) :+ (Snd (Var "x"))))
par = (Pair (Const 1) (Const 2))
exSomaPar = (App (somaPar) (par))
```

3.6 mysum

```
mysum = (Fix
      (Lambda "f"
       (Lambda "l"
         (
          IfZero
          (MyNull  (Var"l"))
          (Const 0)
          ((MyHead (Var"l")) :+ (App (Var "f") (MyTail (Var "l"))))  )
         )))


exSum = (App mysum (exAppend))
exSum2 = (App mysum (exMap1))
exTamanho4 =  (App tamanho (exMap2))
```

3.7 Reverse

```
myreverse = (Lambda "l" (App(App(myReserveAux) (Var "l"))(Empty)))

myReserveAux = Fix
                (Lambda "f"
                  (Lambda "l1"
                    (Lambda "l2"
                      (
                      IfZero (MyNull  (Var"l1")) (Var "l2") ((App(App(Var"f") (MyTail (Var '
                ))))

exReverse = (App myreverse (exAppend))
exTamanho5 = (App tamanho (exReverse))
exSum3 = (App mysum (exAppend))
```

4 conclusion

In conclusion, this report extends the SECD machine compiler to integrate pair and list operations into the fun language, introducing constructs like Pair, Fst, Snd, and list operations such as null, empty, cons. Examples include pair addition and essential functions such as append, length, zip, map, reverse and sum, showcasing recursive list traversal.