Language Implementation

Assignment 1

Breno Fernando Guerra Marrão - UP202308171

April 4, 2024

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

In this section, I will detail the implementation of the additions to the Fun language and to the SECD compiler.

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 or second element of a pair.

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

2.1.2 Lists

For the lists, `Empty` represents an empty list, and `:$` denotes 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

In this definition of the SECD compiler with these additions, it wasn't necessary to add new instructions. Because I utilized Church encodings to define the new terms, I can simply recompile with the Church encoding.

2.2.1 True and false

Here I defined MyTrue as a function that giving two inputs chooses the first one, and MyFalse that chooses the second one, I defined these functions because they were in the church encondings on other definitons such as Fst and Snd.

$$\text{True} = \lambda xy.x$$
$$\text{False} = \lambda xy.y$$

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

2.2.2 Pairs

The definitions of Fst, Snd, and Pair are the following:

$$\text{Pair} = \lambda xyz.zxy$$
$$\text{Fst} = \lambda p.p \text{ MyTrue}$$
$$\text{Snd} = \lambda p.p \text{ MyFalse}$$

In the definition of Pair, because I already have e_1 and e_2, I decided to already apply the lambda for x and y and just do z that I represented as x.

```
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
```

2.2.3 Lists

The church definitions of Empty, cons, null , hd, tl are the following:

$$\text{Empty} = \text{pair true true}$$
$$\text{cons} = \lambda xy.\text{pair false(pair } x\ y)$$
$$\text{null} = \text{fst}$$
$$\text{hd} = \lambda z.\text{fst(snd } z)$$
$$\text{tl} = \lambda z.\text{snd(snd } z)$$

But I had to make some alterations to the Church encodings because I didn't have an if function defined, just ifZero. So instead of using true or false, I decided to put (Const 0) and (Const 1) to represent whether it is an empty list. In null, head, and tail, I already applied the lambdas.

```
compile (Empty) sym
  = compile (Pair (Const 0) (Const 0)) sym
compile (e1 :$ e2) sym
  = compile (Pair (Const 1) (Pair e1 e2)) sym
compile (MyNull e) sym
  = compile (Fst e) 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 Pairs

```
somaPar = (Lambda "x" ((Fst (Var "x")) :+ (Snd (Var "x"))))
par = (Pair (Const 1) (Const 2))
exFst = (Fst par)
exSnd = (Snd par)
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 "l1")))((MyHead(Var"l1"))
                      :$ (Var "l2"))))
                ))))

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

4 conclusion

In conclusion, in this assignment I extended 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.