

Program Transformation and Analysis

Assignment 1

Benjamin Brandt Ohrt, zpn492

May 1, 2019

1 Introduction

This is the first of five weekly assignments in the course Program Transformation and Analysis (PAT) at Copenhagen University. The course professor is Robert Glück. The course is held in block 4, 2019.

In this assignment the focus is on Programs as Data Objects and Reversible Computing.

Before we start on the assignment we introduce an inverse interpreter *invint* and a program inverter *invtrans*. For this introduction we need to define the following: Let p be a program written in a language L , x an input into p and y an output from p . We define p^{-1} as the inverse of p such that given y as input to p^{-1} produces x .

Inverse Interpreter

An inverse interpreter can be defined with the following definition [1, p. 271]:

$$[[invint]][p, y] = x \quad (1)$$

Program Inverter

A program inverter can be defined with the following definition [1, p. 271]:

$$\begin{aligned} [[invtrans]][p] &= p^{-1} \\ [[p^{-1}]] [y] &= x \end{aligned} \quad (2)$$

2 Assignment

Exercise 1

Define a trivial program inverter given an inverse interpreter.

$$[[invtrans']] = \lambda p. \lambda y. [[invint]] [p, y] \quad (3)$$

[1, p. 271]

Exercise 2

Define a trivial inverse interpreter given a program inverter.

$$[[invint']] = \lambda [p, y]. [[Lint]] [[[[invtrans]] p], y] \quad (4)$$

[1, p. 271]

Where Lint is an L-interpreter.

Exercise 3

Run the Janus-program fib forward section 3 by hand, where $n = 4$, $x1 = 0$ and $x2 = 0$. Trace the store $(n, x1, x2)$ after each statement.

Recursion	State	n	x1	x2	branch
0	initial	4	0	0	else
1	initial	3	0	0	else
2	initial	2	0	0	else
3	initial	1	0	0	else
4	initial	0	0	0	then
4	terminating	0	1	1	fi:true
3	terminating	0	1	2	fi:false
2	terminating	0	2	3	fi:false
1	terminating	0	3	5	fi:false
0	terminating	0	5	8	fi:false

Exercise 4

Run the Janus-program fib backward section 3 by hand where $n = 0$, $x1 = 5$ and $x2 = 8$. Trace the store $(n, x1, x2)$ after each statement.

Recursion	State	n	x1	x2	branch
0	initial	0	5	8	else
1	initial	0	3	5	else
2	initial	0	2	3	else
3	initial	0	1	2	else
4	initial	0	1	1	then
4	terminating	0	0	0	fi:true
3	terminating	1	0	0	fi:false
2	terminating	2	0	0	fi:false
1	terminating	3	0	0	fi:false
0	terminating	4	0	0	fi:false

Exercise 5

Describe formally but concisely, how you interpreted each statement ($+=$, $-$, $<=>$, *if*, *call*) in the Janus-program fib in the forward and backward direction.

forward (p)	$+=$	$-$	$<=>$	<i>if</i>	<i>fi</i>	<i>call</i>
backward (p^{-1})	$-$	$+=$	$<=>$	<i>if</i>	<i>fi</i>	<i>call</i>

Switching sourcecode between a Janus-program p and its inverse p^{-1} can be done by interpret statements using above table. After the interpretation you just swap the conditions in *if* and *fi*. and reverse the order of operations within the *then* and *else* statement. Ex. a list l of operations in p where $l := \{x1+=1; x2+=1\}$ is a list l' in p^{-1} where $l' := \{x2-=1; x1-=1\}$

One could formalize this interpretation with three lists and an inverse function: Let $lTable$ be a list of tuples t in a Janus-program p , where the first item is a statement $stmt \in \{if, fi, then, else\}$, second is a list of expressions $exps \in \{+=, -, <=>, call\}$ and third is a boolean telling if a pair has been handled. First element in t can be accessed as $t[0]$, second $t[1]$ and so forth. Let $interpretTable$ be a list of tuples containing the inverse expression. Let $stmtTable$ be a list of tuples, first element is statement of same type as

stmt, and second element is an string str, where $\text{str} \in \{\text{"reverse"}, \text{"swap"}, \text{" "}\}$. Let inverse be a function taking three arguments lTable, interpretTable, stmtTable. Then this function can inverse the fib sourcecode to fib^{-1} and from fib^{-1} to fib.

$$lTable := [(stmt, [e1, \dots, eN], h), \dots, (stmt, [e1, \dots, eN], h)] \quad (5)$$

$$interpretTable := [(+ =, - =), (<=>, <=>), (if, if), (fi, fi), (call, call)] \quad (6)$$

$$stmtTable := [(if, \text{"swap"}), (fi, \text{"swap"}), (then, \text{"reverse"}), (else, \text{"reverse"})] \quad (7)$$

```
function inverse(lTable , interpretTable , stmtTable)
  // Reset handled
  for all pairs in lTable
    set pairs[2] = false

  for all pairs in lTable
    //change all expressions to their inverse
    for all exps in pairs[1]
      for expss in exps
        set expss =
          lookup (expss , interpretTable)[1]

    // Search for the inverse statement
    let inverse_stmt = lookup (pairs[0] , stmtTable)[1]

    let inverse_action = lookup (pairs[0] , stmtTable)[2]

    // We found a then or else statement
    // Reverse all expressions
    if inverse_action = "reverse" then
      set pairs[1] = reverse (pairs[1])
```

```

// We found an if or fi statement
// Find the corresponding if/fi statement
// and swap expressions.
// and mark corresponding statements as handled.
if inverse_action = "swap" then
    let tmp = pairs[1]
    let corr = lookup (inverse_stmt, lTable)
    set pairs[1] = corr[1]
    set corr[1] = tmp
    set corr[2] = true
// Mark pair as handled
set pairs[2] = true

function lookup (idx, table)
    for all elm in table
        if idx = elm[0] then
            elm
        Error: "No such element"

function reverse(table)
    let j = length(table)
    while (j > 1)
        let idx1 = 0
        let idx2 = 1
        while (idx2 <= j - 1)
            swap (idx1, idx2, 1)
            idx1 += 1
            idx2 += 1
        j -= 1

function swap(idx1, idx2, table)
    set tmp = table[idx1]
    set table[idx1] = table[idx2]
    set table[idx2] = tmp

```

3 Janus-programs

The procedures below has been found in the Janus-Playground [2], and is used in exercise 3, 4 and 5.

Fibonacci forward

```
procedure fib
  if n=0 then x1 += 1
              x2 += 1
            else n -= 1
              call fib
              x1 += x2
              x1 <=> x2
fi x1=x2
```

Fibonacci backward

```
procedure fib{-1}
  if x1=x2 then x2 -= 1
              x1 -= 1
            else x1 <=> x2
              x1 -= x2
              call fib{-1}
              n += 1
fi n=0
```

References

- [1] Mogensen, Torben, Schmidt, David, Sudborough, I. Hal (Eds.).
The Essence of Computation - Complexity, Analysis, Transformation.
Springer-Verlag, Berlin Heidelberg, 2002.
- [2] Claus Skou Nielsen, Michael Budde.
Janus-playground - Fibonacci example
<http://topps.diku.dk/pirc/janus-playground/#examples/fib>