University of Sussex
Informatics

Spring 2020

Limits of Computation

Feedback Exercises 3

WHILE-Semantics, Extensions, Programs-As-Data

(covers Lectures 3–6)

Dr Bernhard Reus

WHILE-Programs

1. Consider the programs p1 in Figure 1 and p2 in Figure 2.

```
p1 read XY {
   R := 0;
   X := hd XY;
   Y := hd tl XY;
   while X {
      R := <p2> [ Y, R ];
      X := tl X
    }
}
write R
```

Figure 1: WHILE-program p1

```
p2 read XY {
  X := hd XY;
  Y := hd tl XY;
   while X {
     Y := cons nil Y;
     X := tl X
    }
}
write Y
```

Figure 2: WHILE-program p2

(a) Which of the programs is not written in pure (*core*) WHILE?
Answer: Only p2 is in core WHILE.

(b) What is $[\![\mathtt{p1}]\!]^{\mathtt{WHILE}}(\ulcorner[2,5]\urcorner)$ according to the definition of p1 and p2 and $[\![\_]\!]^{\mathtt{WHILE}}$?
Answer: Since p2 is addition (on encodings of natural numbers), p1 is multiplication, so $[\![\mathtt{p1}]\!]^{\mathtt{WHILE}}(\ulcorner[2,5]\urcorner) = \ulcorner 10\urcorner$.

(c) Translate the program p2 into its data representation (programs as data) using the definition of the encoding presented in Lecture 6. Encode the variables starting from number 0 in the order of appearance.
Answer:

```
[0,
[ [:=,1,[hd,[var,0]]],
  [:=,2,[hd,[tl,[var,0]]]],
  [while,[var,1],
            [ [:=,2,[cons,[quote,nil],[var,2]]],
              [:=,1,[tl,[var,1]]]
            ]
   ]
],
2]
```

The most common mistakes here are:

- Forgetting that a block is translated into a list of command encodings. So the corresponding list brackets are often omitted.
- Translating a variable expression into just a number, e.g. `0` instead of `[var,0]`. Note that when translating expressions it is important that each encoding is a list with the first element indicating the operator in use.
- Forgetting some commas that separate list elements.

Note that the numbers chosen for the variables in general is arbitrary as long as its consistent. For this question, however, it was required that one starts with 0 in the order of appearance!

2. Consider the tree t depicted on the exercise sheet.

   (a) Why is t a correct tree in $\mathbb{D}$ although items like `1`, `0`, `cons`, `:=`, `quote`, `var` appear at its leaves rather than just nil?

   Answer:
   Because these symbols at the leaves are

   i. additional atoms, either encoded as numbers (so one can unfold them into a list of nil's) or added to the definition of $\mathbb{D}$.

   ii. or numbers which are also encoded as lists of nils.

   We could unfold their definitions and get an even larger tree that only contains nil atoms at its leaves.

   (b) Write tree t in list notation.

   Answer:

```
[0,
    [while,[cons,[quote,nil],[quote,nil]],
              [:=,1,[var,0]]
    ]
,1]
```

(c) Does t correctly encode a `WHILE`-program?

Answer: No.

We need to apply minimal corrections to t such that the resulting new tree t' encodes a `WHILE`-program p. Two sets of brackets `[ ]` were missing (it's the ones that are most often forgotten, see feedback to Question 1).

Here is the fixed version:

```
[0,
    [  [while,[cons,[quote,nil],[quote,nil]],
             [  [:=,1,[var,0]]  ]
       ]
    ]
,1]
```

So we can write p in concrete (textual) syntax (so that you can run it in `hwhile`) as follows. Let us choose variable name `X` for variable number `0` and `Y` for number `1`:

```
p read X {
  while cons nil nil {
     Y:= X
   }
}
write Y
```

(d) What does the corrected program from 2(c) do? Give its semantics!

Answer: $[\![\mathrm{p}]\!]^{\texttt{WHILE}}(d) = \bot$ for all $d \in \mathbb{D}$.

3. Find out how atoms `while`, `doWhile` and `quote` are encoded in `hwhile`. Why is there no `doQuote`?

Answer: Just write a `WHILE`-program that returns a list of those atoms, e.g.

```
print read X {
 X:= [@while,@doWhile,@quote]
}
write X
```

and use the `-Li` flag to display them as numbers. You will get:

```
[5, 7, 19]
```

There is no `doQuote` as `quote` d can be evaluated in one go as no intermediate results have to be produced first. We know that d must already be a binary tree (evaluated) so we can directly push it on the result stack.