ASSIGNMENT 9 — MACHINES

COMP 3010 — ORGANIZATION OF PROGRAMMING LANGUAGES

1. Call Stacks

Exercise 1. Do *Concepts In Programming Languages* Exercise 7.2 on tail recursion and iteration (page 193).

You may write your answer in any language of your choosing from this list: Standard ML, Ruby, Scheme, Python, or C/C++. No matter what language you choose to write your answer in, make sure you write a *non-recursive* program using a "while" or "for" loop that is equivalent to the function `mult` given in Exercise 7.2 (that is to say, your non-recursive function should always return the same number as the recursively-defined `mult` for any pair of arguments.)

Hint: The tail recursive function `mult` in Exercise 7.2 can be simplified by removing the special case where `a = 1` sharing the constant variable `b` like so:

```
fun mult(a, b) =
    let fun mult1(a, result) =
            if a = 0
            then result
            else mult1(a-1, result+b)
    in
        mult1(a, 0)
    end;
```

You may base your answer off of this simplified definition of `mult`.

Exercise 2. Do *Concepts In Programming Languages* Exercise 7.8 on static and dynamic scope (pages 196-197).

Extra Credit Exercise 1. Do *Concepts In Programming Languages* Exercise 7.11 on λ-calculus and scope (page 198).

Exercise 3. Do *Concepts In Programming Languages* Exercise 7.12 on function calls and memory management (pages 198-199).

Extra Credit Exercise 2. Do *Concepts In Programming Languages* Exercise 7.13 on function returns and memory management (pages 199-200).

Date: Summer 2022.

2. Abstract Machines

Recall this definition of an abstract machine for calculating the answers to arithmetic problems:

$$n ::= 0 \mid 1 \mid 2 \mid 3 \mid \dots$$
$$A ::= \underline{n} \mid \texttt{plus}(A, A) \mid \texttt{minus}(A, A) \mid \texttt{times}(A, A)$$

$$K ::= \texttt{done} \mid F; K$$
$$F ::= \texttt{plus}(\square, A) \mid \texttt{plus}(n, \square) \mid \texttt{minus}(\square, A) \mid \texttt{minus}(n, \square) \mid \texttt{times}(\square, A) \mid \texttt{times}(n, \square)$$
$$S ::= \mathit{Eval}\langle A \mid K\rangle \mid \mathit{Ret}\langle n \mid K\rangle$$

$$\begin{aligned}
\mathit{Eval}\langle \texttt{plus}(A_1, A_2) \mid K\rangle &\mapsto \mathit{Eval}\langle A_1 \mid \texttt{plus}(\square, A_2); K\rangle \\
\mathit{Eval}\langle \texttt{minus}(A_1, A_2) \mid K\rangle &\mapsto \mathit{Eval}\langle A_1 \mid \texttt{minus}(\square, A_2); K\rangle \\
\mathit{Eval}\langle \texttt{times}(A_1, A_2) \mid K\rangle &\mapsto \mathit{Eval}\langle A_1 \mid \texttt{times}(\square, A_2); K\rangle \\
\mathit{Eval}\langle \underline{n} \mid K\rangle &\mapsto \mathit{Ret}\langle n \mid K\rangle
\end{aligned}$$

$$\begin{aligned}
\mathit{Ret}\langle n \mid \texttt{plus}(\square, A); K\rangle &\mapsto \mathit{Eval}\langle A \mid \texttt{plus}(n, \square); K\rangle \\
\mathit{Ret}\langle n \mid \texttt{plus}(n', \square); K\rangle &\mapsto \mathit{Ret}\langle n' + n \mid K\rangle \\
\mathit{Ret}\langle n \mid \texttt{minus}(\square, A); K\rangle &\mapsto \mathit{Eval}\langle A \mid \texttt{minus}(n, \square); K\rangle \\
\mathit{Ret}\langle n \mid \texttt{minus}(n', \square); K\rangle &\mapsto \mathit{Ret}\langle n' - n \mid K\rangle \qquad (\text{if } n' \geq n) \\
\mathit{Ret}\langle n \mid \texttt{times}(\square, A); K\rangle &\mapsto \mathit{Eval}\langle A \mid \texttt{times}(n, \square); K\rangle \\
\mathit{Ret}\langle n \mid \texttt{times}(n', \square); K\rangle &\mapsto \mathit{Ret}\langle n' \times n \mid K\rangle
\end{aligned}$$

Exercise 4. Using the above rules for executing programs in the abstract machine, show the step-by-step calculating the final state of the machine command $\mathit{Eval}\langle \texttt{times}(\underline{2}, \texttt{minus}(\texttt{plus}(\underline{3}, \underline{4}), \underline{5})) \mid \texttt{done}\rangle$.

Hint: Beginning with the starting machine state (S_0) above, keep applying the machine steps listed above on at a time to transition between states

$$\begin{aligned}
S_0 &\mapsto S_1 \\
&\mapsto S_2 \\
&\mapsto \dots \\
&\mapsto S_{\texttt{done}}
\end{aligned}$$

until you reach a final state $S_{\texttt{done}}$ that should look like either $\mathit{Ret}\langle n \mid \texttt{done}\rangle$ or $\mathit{Ret}\langle b \mid \texttt{done}\rangle$. Each machine state has *at most one* transition rule that could possibly apply, so you will never have to choose between multiple options.

Recall the Krivine machine for calculating answers to λ-calculus programs (extended with numbers $n ::= 0 \mid 1 \mid 2 \mid \dots$):

$$M ::= n \mid x \mid M\ M \mid \lambda x.M$$
$$\Gamma ::= \varepsilon \mid x = [\Gamma]M, \Gamma$$
$$\Pi ::= \texttt{done} \mid [\Gamma]M; \Pi$$

$$\begin{aligned}
\langle M\ M' \mid \Gamma \mid \Pi \rangle &\mapsto \langle M \mid \Gamma \mid [\Gamma]M; \Pi \rangle \\
\langle \lambda x.M \mid \Gamma \mid [\Gamma']M'; \Pi \rangle &\mapsto \langle M \mid x = [\Gamma']M', \Gamma \mid \Pi \rangle \\
\langle x \mid \Gamma \mid \Pi \rangle &\mapsto \langle M' \mid \Gamma' \mid \Pi \rangle \qquad \text{(if } (x = [\Gamma']M') \in \Gamma\text{)}
\end{aligned}$$

Exercise 5. (a) Show the step-by-step β-reduction in the small-step semantics of the λ-calculus to simplify the expression

$$(\lambda x.(\lambda y.(\lambda x.y\ 30)\ 20)\ (\lambda z.x))\ 10$$

down to a final answer of the form of a number n or function $\lambda x'.M'$ with some parameter x' and some body M'.

(b) Using the above rules of the Krivine machine and starting from the initial machine state of

$$\langle (\lambda x.(\lambda y.(\lambda x.y\ 30)\ 20)\ (\lambda z.x))\ 10 \mid \varepsilon \mid \texttt{done} \rangle$$

show the step-by-step transition sequence for calculating the final state of the machine, which should have the form of either $\langle \lambda z.M \mid \Gamma \mid \texttt{done} \rangle$ or $\langle n \mid \Gamma \mid \texttt{done} \rangle$ for some environment Γ.

(c) Compare the answer you get to both previous steps. Are they the same? Should they be the same? Why or why not?

Recall that the local names in an expression can be eliminated by translating to de Bruijn indexes like so:

$$\begin{aligned}
i &::= 0 \mid 1 \mid 2 \mid \dots \\
M &::= var_i \mid M\ M \mid \lambda M
\end{aligned}$$

$$\begin{aligned}
[\![x]\!] &= x \\
[\![M\ M']\!] &= [\![M]\!]\ [\![M']\!] \\
[\![\lambda x.M]\!] &= \lambda([0/x][\![M]\!])
\end{aligned}$$

$$\begin{aligned}
[i/x]x &= var_i \\
[i/x]y &= y \qquad \text{(if } x \neq y\text{)} \\
[i/x](M\ M') &= ([i/x]M)\ ([i/x]M') \\
[i/x]\lambda M &= \lambda([i+1/x]M)
\end{aligned}$$

These nameless expression can be run inside of a de Bruijn machine that only uses numeric positional offsets, rather than looking up variables by name:

$$\begin{aligned}
\Gamma &::= \varepsilon \mid [\Gamma]M, \Gamma \\
\Pi &::= \texttt{done} \mid [\Gamma]M; \Pi
\end{aligned}$$

$$\begin{aligned}
\langle M\ M' \mid \Gamma \mid \Pi \rangle &\mapsto \langle M \mid \Gamma \mid [\Gamma]M'; \Pi \rangle \\
\langle \lambda M \mid \Gamma \mid [\Gamma']M'; \Pi \rangle &\mapsto \langle M \mid [\Gamma']M', \Gamma \mid \Pi \rangle \\
\langle var_i \mid \Gamma \mid \Pi \rangle &\mapsto \langle M' \mid \Gamma' \mid \Pi \rangle \qquad \text{(if } \Gamma \text{ at index } i = [\Gamma']M'\text{)}
\end{aligned}$$

Notice how in this machine, both the environment Γ and the stack Π are just plain arrays/lists of closures $[\Gamma]M$.

Extra Credit Exercise 3.

(a) Draw a syntax tree of the λ-calculus expression $(\lambda x.(\lambda y.(\lambda x.y\ 30)\ 20)\ (\lambda z.x))\ 10$. Then, draw an arrow from each leaf which references a variable to the λ that introduces the variable it refers to. In other words, walk up the tree from the reference leaf containing a variable like x and stop at the first λ you encounter which introduces that variable (x) as a parameter; this is the parameter the leaf refers to.

Next to each variable reference leaf, write down the number of λs that were *skipped* between each reference to a variable and the λ-parameter it points to. For example, if a variable reference leaf points to the nearest λ up the tree (even if there are other nodes like application in the middle), write a 0 next to it. If there is only one other λ node between a variable leaf and the λ introducing a parameter of the same name as the variable, write a 1. And so on.

(b) Show how to translate the λ-calculus expression $(\lambda x.(\lambda y.(\lambda x.y\ 30)\ 20)\ (\lambda z.x))\ 10$ to its nameless de Bruijn form by following the definition of

$$[\![(\lambda x.(\lambda y.(\lambda x.y\ 30)\ 20)\ (\lambda z.x))\ 10]\!]$$

Note, in the base case where you get to a number n (in this case, $n = 10$ or 20 or 30), assume that $[\![n]\!] = n$.

Hint: The index i you get at each var_i in the translated expression should correspond to the number of skipped λs you found in the syntax tree from part (a). In other words, if you take your syntax tree from part (a) and erase all the variable names, you should get a new nameless syntax tree that corresponds to your answer to this question.

(c) Given that your answer to part (b) is the term M which no longer has any variables on it, use the above rules for the de Bruijn machine and start from the initial machine state $\langle M \mid \varepsilon \mid \texttt{done}\rangle$ to show the step-by-step transition sequence for calculating the final state of the machine, which should have the form of either $\langle \lambda M \mid \Gamma \mid \texttt{done}$ or $\langle n \mid \Gamma \mid \texttt{done}$ for some environment array Γ.