CS 164 Programming Languages and Compilers **Spring 2019**

Discussion Week of 4/15: Midterm 2 Review

Instructor: Paul N. Hilfinger GSIs: Nikhil Athreya, Vivant Sakore

1 Type Unification

1. Compute the types of the following functions:

```
fun
  f g x = if (g x) then (h g x) else (f g x)
and
  h i y = if (i y) then (f i y) else (h i y)
```

```
f |-> 'x1 -> 'x2 -> 'x3
h |-> 'x4 -> 'x5 -> 'x6

g |-> 'x1
x |-> 'x2
'x1 |-> 'x7 -> 'x8
'x2 |-> 'x7
'x8 |-> bool
'x1 |-> 'x4
'x2 |-> 'x5
'x3 |-> 'x6

'x1 |-> 'x7 -> bool
'x2 |-> 'x7
'x3 |-> 'x6

i |-> 'x4
y |-> 'x5
'x4 |-> 'x11 -> 'x12
'x5 |-> 'x11
'x12 |-> bool
'x4 |-> 'x1
'x5 |-> 'x2
```

```
'x6 |-> 'x3

'x4 |-> 'x7 -> bool
'x5 |-> 'x7
'x6 |-> 'x6

f |-> ('x7 -> bool) -> 'x7 -> 'x6
h |-> ('x7 -> bool) -> 'x7 -> 'x6
```

2. Is it possible to consider each of their types separately? In other words, just knowing that h is a function which takes two arguments and returns something, can we deduce the correct type for f and vice versa?

   No, this is because the types of the parameters of f and h need to match, which can only be done when we consider the entire program.

2 Operational Semantics

Suppose we have already defined the operational semantics of basic arithmetic expressions involving the integers, variables, and booleans as follows (respectively):

$$\frac{\ldots}{\Gamma \vdash e_1 \ : \ n, \ \Gamma}$$

and

$$\frac{\ldots}{\Gamma \vdash \ b_1 \ : \ b, \ \Gamma}$$

Γ here represents our mapping between variables and their values. We use the convention e for arithemtic expressions and b for boolean expressions. Note that as of now, expressions do not modify the state of our program. Write the operational semantics for the following types of constructs:

1. Assignment to an arithmetic expression as follows: $x := e$. Note an assignment returns void.

   $$\frac{\Gamma \vdash \ e \ : \ n, \ \Gamma}{\Gamma \vdash \ x \ := \ e \ : \ \mathit{void}, \ \Gamma[x \ := \ n]}$$

   The new state needs to have a mapping to the variable x.

2. A sequence of statements s_1 and s_2 as follows: s_1; s_2. Note a sequence returns void.

   $$\frac{\Gamma \vdash \ s_1 \ : \ \mathit{void}, \ \Gamma' \ \wedge \ \Gamma' \vdash \ s_2 \ : \ \mathit{void}, \ \Gamma''}{\Gamma \vdash \ s_1 \ ; \ s_2 \ : \ \mathit{void}, \ \Gamma''}$$

3. An if statement as follows: if b then s_1 else s_2. Note an if statement must only execute its corresponding branch.

$$\frac{\Gamma \vdash b : true,\ \Gamma \wedge \Gamma \vdash s_1 : void,\ \Gamma'}{\Gamma \vdash if\ b\ then\ s_1\ else\ s_2 : void,\ \Gamma'}$$

$$\frac{\Gamma \vdash b : false,\ \Gamma \wedge \Gamma \vdash s_2 : void,\ \Gamma'}{\Gamma \vdash if\ b\ then\ s_1\ else\ s_2 : void,\ \Gamma'}$$

We need two operational semantic definitions to account for needing to execute only one of the statements in different cases. This allows us to pattern-match the true case and the false case.

3 Runtime support for functions

1. Consider the following program:

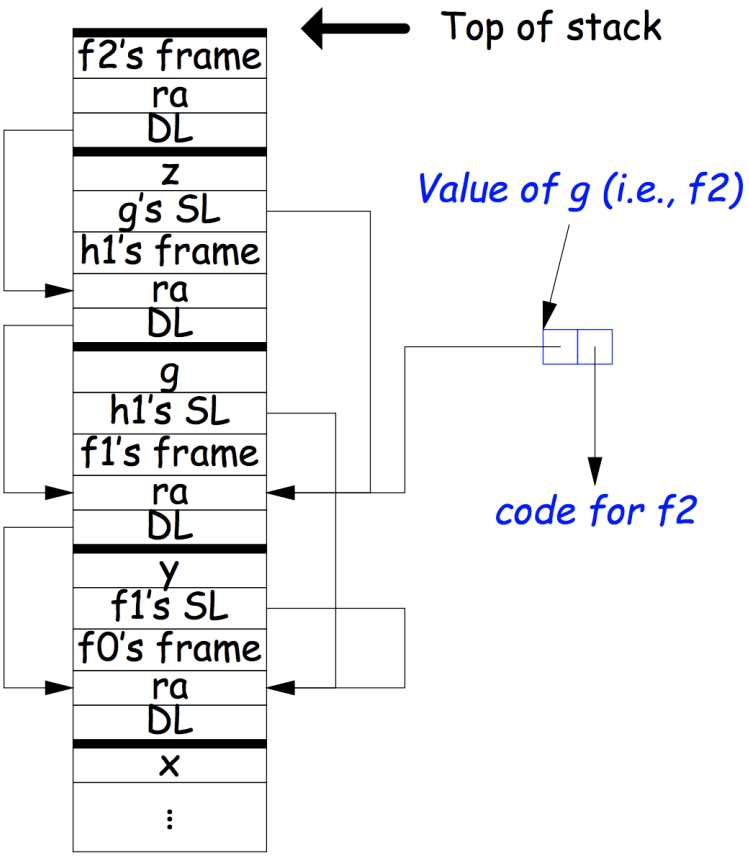
```
def f0 (x):
    def f1 (y):
        def f2 (z):
            return x * y * z
        def h1 (g):
            return g(3)
        print(h1(f2))
    f1(2)

if __name__ == "__main__":
    f0(5)
```

What will the above code print? Give the stack representation of function calls.

Answer: 30

2. What is continuation? Can it be handled by just using stack?

Answer: A continuation is the function that does whatever is supposed to happen after instruction pointer returns from function (and exits program). Continuations can be used to implement exceptions, threads etc. They cannot be handled by just using stack. It requires all the ativation records to be stored on heap instead.


Top of stack
f2's frame
ra
DL
z
g's SL
h1's frame
ra
DL
g
h1's SL
f1's frame
ra
DL
y
f1's SL
f0's frame
ra
DL
x
Value of g (i.e., f2)
code for f2