Programming Language Design 2023
Functional Programming and Control Structures

Hans Hüttel

20 February 2024

1 Problems that we will definitely talk about

1. *(15 minutes)* In the podcast we did not specify what should happen if a coroutine executes to the end of its body. What are some possibilities, and what are the pros and cons?

2. *(25 minutes)* Below is a Haskell program P.

```
f x = let dingo z = z + x
      in
          let
              bingo w = dingo (47-w)
          in
              g (bingo 419)

g u = let mango x = x + h u
      in
          let
              dingo k = k
          in
              mango (dingo u)

h w = 42
```

   - What is `f 14`?
   - Use the λ-lifting algorithm to transform P.

3. *(15 minutes)* Is there any need for *selective* control structures whose tests are evaluated sequentially, such as the following?

```
if i > n orelse a[i] = x then ... else ...
```

4. *(25 minutes)* In ALGOL68 there is only one loop construct, whose most elaborate form is

   for I **from** E1 **by** E2 **to** E3 **while** E4 **do** C **od**

   where E1, E2, E3 and E4 are expressions that are integer-valued and are evaluated only once. The scope of the identifier I is E4 and C.

   The only part that cannot be omitted is C.

   - How many different control structures can we express using this general loop construct?
   - What do you expect will happen if we omit the following?
     a) **for** I
     b) **from** E1
     c) **by** E2
     d) **to** E3
     e) **while** E4
   - Can we express the repeat-until loop construct using this general loop only? If yes, describe how this is done. If no, explain the difficulties and if there is a solution.
   - How might this general loop construct be extended such that we could use it as an *expression*?

2 More problems if you have time

a. In the video for today, Hans said that

> When we assume static scope rules, we need to know the following about a function when we make use of it:
>
> - The code of the function
> - The bindings that existed when the function was declared (that is, an *environment*)
>
> Such a pair
>
> $$\langle \mathsf{code}, env \rangle$$
>
> is called a *closure*. This is a central notion.

But in an earlier version of the slides, it said that we can think of a closure as a closed curried function of two arguments that has been applied to first argument, e.g. `f 5`.

That looks like a very different notion.

Why does this make sense?