Programming Paradigms 2022
Session 14

Continuations; Static and dynamic typing

Hans Hüttel

12 December 2023

Our plan for today

1. The learning goals
2. Presentations of the discussion problems
3. Problem no. 1
4. Problem no. 2
5. Break
6. Problem no. 3
7. Problem no. 4
8. If time allows: More problems at your own pace.
9. We evaluate the course as a whole

Learning goals

- To be able to explain and use the notion of continuations in Haskell.
- To understand the difference between static and dynamic typing and their respective advantages.

Discussion problem - Tail recursion

Here is the exptwo function.

```
exptwo 0 = 1
exptwo n = 2 * (exptwo (n-1))
```

Why is this function not tail-recursive? Write a version of the function that uses continuation-passing and is tail-recursive.

An answer from us

The function is not tail-recursive, as the recursive part of the definition has a multiplication that must be carried out after the recursive call.

Here is a tail-recursive version. The continuation tells us what the caller of the function wants to do with the result.

```
exptwo (0,c) = c 1
exptwo (n,c) = exptwo (n-1, \z -> c (z*2))
```

In the recursive call, the outside caller on the right-hand side is exptwo (n,c) and the definition must therefore tell us what we must give to it.

Discussion problem - Slack

Find an example of slack in the Haskell type system that does not use the if-then-else or case constructs and would be safe, if Haskell were dynamically typed.

An answer from us

Unreachable code can never result in a run-time error.

```
bingo x = x + 1
                where f y = y + True
```

Problem 1: The length function – again

Here is the function that computes the length of a list.

```
lgd []     = 0
lgd (x:xs) = 1 + lgd xs
```

Define it again in continuation-passing style and find an appropriate test case for your new definition.

Making the length function tail-recursive

The argument is a pair in which the second component is the continuation.

```
lgd ([],c) = c 0
lgd ((x:xs),c) = lgd (xs, \n -> (c n) +1)
```

Problem 2: The Fibonacci function revisited

Here is the definition of the function that computes Fibonacci numbers.

```
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Define it again in continuation-passing style and find an appropriate test case for your new definition.

We nest the calls now

Because we use continuations, we have to make up our minds as to in which order we want the recursive calls. Since the $n-1$th Fibonacci numbers is defined from the $n-2$th, there is only one sensible choice.

```
fib (0,c) = c 0
fib (1,c) = c 1
fib (n,c) = fib (n-1, \a -> fib (n-2, \b -> c
    (a+b)))
```

Problem 3: Slack in Haskell!

Find an example of slack in the Haskell type system that does *not* involve conditional expressions or unreachable subexpressions and only contains function definition and a single function application.

A simple example

```
(\x -> x (x)) id
```

This would not lead to a run-time error – we would get the identity function as the returned value. But a self-application $x(x)$ cannot be typed in Haskell, for we then x would have to have type $a \to b$ as well as a where $a = a \to b$.

Problem 4: Javascript (!!)

The very last problem in the very last problem set in this course is about – Javascript! Describe what the results of the following operations are in Javascript:

```
3 + "2"
3 * "2"
"3" * "2"
3 + {}
{} + 2
{3}
3+{2}
{3}+2
```

Can you make sense of them?

What Javascript says

According to the Javascript REPL in macOS

```
3 + "2"    evaluates to '32'}
3 * "2"    evaluates to 6
"3" * "2"  evaluates to 6
3 + {}     evaluates to '3[object Object]'
{} + 2     evaluates to '[object Object]2'
{3}        evaluates to 3
3+{2}      evaluates to Uncaught SyntaxError: Unexpected tok
{3}+2      evaluates to 2
```

Type conversion in Javascript

Javascript is a scripting language that is used for generating dynamic web content – and textual content in particular.

When a number is added to a string, the number is converted to a string before concatenation.

Brackets can be used to specify the order of evaluation.

(but there is more . . .)

Now the course is nearly over

- What did you find difficult?
- What surprised you?
- What went well?
- What could be improved?

Some final thoughts from Hans

- There were a lot of responsible people who presented solutions to the discussion problems. Thank you! How can we make sure that it becomes the norm to work on these small problems before the session?
- A miniproject would be useful but everyone is busy – and when a miniproject is not assessed, what is the incentive? Good ideas would be highly appreciated.
- The rooms have been really useful for this course (I think), but the management of them has been silly. The "Linux silence" is a disgrace. Let the Department of Computer Science take over these rooms and lease them to sociology!
- Other Danish degree programmes in computing have functional programming as part of the undergraduate degree. (Parts of) this course ought to appear earlier at AAU but people have other interests here.