



Type Theory & Coq

On the notion of *Continuations*

Jan Martens

8th of May 2018





Outline

Recap reductions, Felleisen approach

Notion of Continuations

Continuation passing style





Recap reductions

- Structural-operational style, Felleisen contexts
- Call-by-name **CBN**, call-by-value **CBV**
- Encoding CBN in a CBV world using Thunks
- Some terms evaluate in CBN but not in CBV, $(\lambda x.1)\omega$.





Felleisen approach Recap

Felleisen contexts $E ::= [] \mid E b \mid v E$

Head reduction (on top of a term)

$$(\lambda x.a)v \xrightarrow{\varepsilon} a[x \leftarrow v] \quad (\beta_v)$$

Head reduction within a reduction context:

$$\frac{a \xrightarrow{\varepsilon} a'}{E[a] \rightarrow E[a']}$$

Example 1

$$(\lambda x.\lambda y.y x)((\lambda x.x) 1)(\lambda x.x)$$

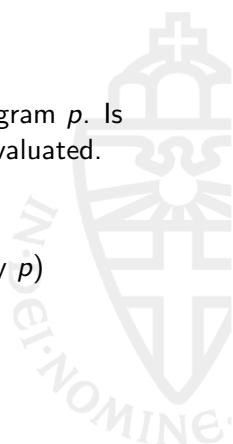


Notion of continuations

The **Continuation** of a sub-expression a in a given program p . Is the computations that remain to be done, after a is evaluated.

Intuitively viewed as a function:

(Value produced by a) \mapsto (Value produced by p)





Continuations

- Concept used in many languages, (Continuation-passing style)
- Usefull when reasoning about the control-flow of program

Example 2

Given a (functional) program $p = (1 + 2) * (3 + 4)$

What is the continuation of the subexpressions a, b

Continuation of $a = (1 + 2)$?

Continuation of $b = (3 + 4)$?



Continuations

- Concept used in many languages, (Continuation-passing style)
- Usefull when reasoning about the control-flow of program

Example 2

Given a (functional) program $p = (1 + 2) * (3 + 4)$

What is the continuation of the subexpressions a, b

Continuation of $a = (1 + 2)$? $\lambda x.x * (3 + 4)$

Continuation of $b = (3 + 4)$?



Continuations

- Concept used in many languages, (Continuation-passing style)
- Usefull when reasoning about the control-flow of program

Example 2

Given a (functional) program $p = (1 + 2) * (3 + 4)$

What is the continuation of the subexpressions a, b

Continuation of $a = (1 + 2)$? $\lambda x.x * (3 + 4)$

Continuation of $b = (3 + 4)$? $\lambda x.3 * x$



Continuations

Example Continuations

$$(\lambda x. \lambda y. y \ x) ((\lambda x. x) \ 1) (\lambda x. x)$$



Notion of continuations

Example in javascript

```
1 // Not using continuation passing
2 var data = getData(id)
3 doStuff(data);
4
5 // Using continuation passing
6 getData(id,
7     function(data) {
8         doStuff(data);
9     }
10 );
```



Notion of continuations

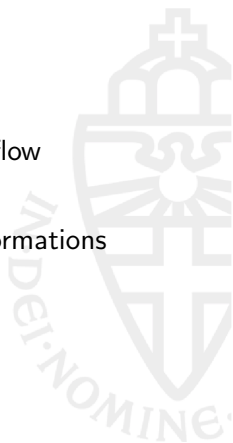
Example implementation in javascript

```
1 // Not using continuation passing
2 function id(x) {
3   return x;
4 }
5 // Using continuation passing
6 function id(x, cc) {
7   cc(x);
8 };
```



Continuation passing style

- Pass continuations explicitly, to describe control flow
- Return by calling given continuation
- Frequently used in compilers and program transformations

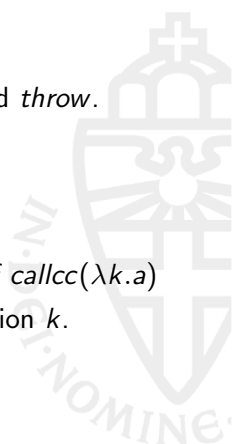




Call-with-current-continuation

In the Scheme language there is a primitive *callcc*, and *throw*.

- *callcc* has the type $(\alpha \text{ Cont} \rightarrow \alpha) \rightarrow \alpha$
- *throw* has the type $\alpha \text{ Cont} \rightarrow \alpha \rightarrow \beta$
- *callcc*($\lambda k.a$) gets its continuation as argument
- The value obtained by evaluating *a* is the value of *callcc*($\lambda k.a$)
- *throw* *k v* in order to *backtrack* to the continuation *k*.





Call-with-current-continuation

- *callcc* uses current context to pass continuation
- *throw* removes current context
- Note the non-linear use of the context

Call/cc and throw reduction rules

$$E[\text{callcc } v] \rightarrow E[v (\lambda x. E[x])]$$

$$E[\text{throw } k v] \rightarrow k v$$



Callcc and Throw Example

Example in functional programming

```
1  (* list_iter: (a -> unit) -> a list -> unit *)
2
3  let rec list_iter f l =
4      match l with
5      | [] -> ()
6      | head :: tail -> f head; list_iter f tail
```



Callcc and Throw Example

Example in functional programming

```
1 let find pred lst =  
2   callcc (λk.  
3     list_iter  
4       (λx. if pred x then throw k (Some x) else ()))  
5     lst;  
6   None)
```



Callcc and Throw Example

Example with backtrack

```
1 let find pred lst =
2   callcc (λk.
3     list_iter
4       (λx. if pred x
5           then callcc (λk'. throw k (Some(x, k')))
6           else ())
7     )
8   lst;
9   None)
```



Callcc and Throw Example

Example print all numbers

```
1 let printall pred lst =  
2   match find pred lst with  
3     | None -> ()  
4     | Some(x, k) -> print_string x; throw k ()
```



Continuation-passing style

- The "goto of functional program languages"
- Can be hard to use, easy to make mistakes
- Possibility of control manipulation
 - Exceptions
 - Backtracking
 - Checkpointing and replay
- Implemented in many languages

