Messing with the Future
or the Art of Continuations

Marco Devesas Campos
(maf58@cam.ac.uk)
A *Continuation* is a description of what happens next in the execution of a program.
With Continuations we can…

… be more efficient
… do exceptions
… do backtracking
… do multitasking
… do concurrent programming
… do… everything!*

*… minus anything that involves interacting with the hardware itself…
Continuations are everywhere

\[(1 + 1) + 2\]
Continuations are really everywhere

$$(1 + 1) + 2$$

<table>
<thead>
<tr>
<th>Program State</th>
<th>Continuation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>$(1+1) + 2$</td>
</tr>
<tr>
<td>$(1 + 1)$</td>
<td>$(1) + 2$</td>
</tr>
<tr>
<td>Finish</td>
<td>“Done!”</td>
</tr>
</tbody>
</table>
Continuations are a Dynamic Concept

```c
int f(int *x, int *y){
    *x = *x + *y;
}

int main( void){
    int x=1, y=1;
    f(&x, &y);
    print("%d\n", x);
    return 0
}
```

Continuation
Continuations are a Dynamic Concept

```
int f(int *x, int *y)
{
    *x = *x + *y;
}

int main(void)
{
    int x=1, y=1;
    f(&x, &y);
    printf("%d\n", x);
    return 0
}
```
Continuations are a Dynamic Concept

```
int f(int *x, int *y){
    *x = *x + *y;
}

int main(void){
    int x=1, y=1;
    f(&x, &y);
    y = 2;
    f(&x, &y);
    print("%d\n", x);
    return 0
}
```
Part 1

Continuation Passing Style
The slogan of CPS

“Parametrize on the continuation and apply values to it to return.”
Summing a List (Freshers)

\[
\text{sumList} \; [] = 0 \\
\text{sumList} \; (x:xs) = x + (\text{sumList} \; xs)
\]
Summing a List (Postgrads)

\[
\begin{align*}
\text{sumCPS} & \quad [ ] \quad k = k \ 0 \\
\text{sumCPS} & \quad (x:xs) \quad k = \text{sumCPS} \ xs \ (\n \rightarrow \ k \ (x + n))
\end{align*}
\]
sumList Call Stack

call sumList

call sumList

call sumList

call sumList
sumList Call Stack

- return 0
- 0 + 3
- return 3
- 3 + 2
sumList Call Stack

5
RP
SP
[1, 2, 3]

5 + 1

6
RP
SP
[1, 2, 3]

return 6

6
sumCPS Call Stack

call sumCPS

call sumCPS

call sumCPS (2x)

Friday, 2 May 14
sumCPS Optimised Call Stack

call sumCPS
jump to sumCPS
jump to sumCPS
jump to sumCPS

<table>
<thead>
<tr>
<th></th>
<th>RP</th>
<th>SP</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>[1,2,3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>RP</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>-&gt;</td>
<td>k</td>
</tr>
<tr>
<td></td>
<td>n+1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>RP</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>-&gt;</td>
</tr>
<tr>
<td></td>
<td>n+2</td>
<td>k'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>RP</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>-&gt;</td>
</tr>
<tr>
<td></td>
<td>n+3</td>
<td>k''</td>
</tr>
</tbody>
</table>

[ ]
sumCPS Optimised Call Stack

```
<table>
<thead>
<tr>
<th>do k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ret</td>
</tr>
<tr>
<td>RP</td>
</tr>
<tr>
<td>SP</td>
</tr>
<tr>
<td>n -&gt; k'' (n+3)</td>
</tr>
<tr>
<td>[]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ret</td>
</tr>
</tbody>
</table>
```
Almost anything goes inside a Continuation

```
fibCPS 0  k = k 1
fibCPS 1  k = k 1
fibCPS n k = fibCPS (n - 1) (  
   \fibn1 -> fibCPS (n - 2)  
       (\fibn2 -> k (fibn1 + fibn2))  
   )
```

Friday, 2 May 14
A different problem

Are there three positive integer numbers $a$, $b$ and $c$ s.t. $a^2 = b^2 + c^2$?
A Non-deterministic Algorithm (NDA)

Go through the alternatives...

... and return first match
NDAs in CPS Form

To generate potential solutions:

```haskell
try f [] (nay, yay) = nay
try f (x:xs) (nay, yay) = f x (try f xs (nay, yay), yay)
```

To test potential solutions:

```haskell
test p a (nay, yay) = if (p a) then yay a else nay
```
The CPS solution

The test:

\[
\text{isRightTriangle} (a, b, c) = a^2 = b^2 + c^2
\]

To find (potential) solutions:

\[
\text{findRightTriangle} l = \\
\text{try} (\lambda a \rightarrow \\
\text{try} (\lambda b \rightarrow \\
\text{try} ( \\
\text{try} (\lambda c \rightarrow \text{test isRightTriangle} (a,b,c) \\
\quad \quad \quad \quad \quad \quad \quad \text{) l \\
\quad \quad \quad \quad \}) \text{ l \\
\text{) l \\
\text{) l}
\]

Friday, 2 May 14
Part II

First Class Continuations
Support (at least) in...

... ML
... LISP
... Scheme
... C (sort of)
... Python (sort of)
... Ruby
Primitives

\texttt{throw k v}: aborts the current computation and proceeds with continuation \( k \) applied to \( v \)

\texttt{callcc f}: captures the current continuation \( (k) \) and continues as \( f \ k \)
Example 3

\texttt{callcc(\k \rightarrow 1 + 2)}
Example 2

callcc(\k \rightarrow 1 + (\text{throw } k \ 2))
Example 1

callcc(\k -> (throw k 1) + (throw k 2))
Example 0

callcc (\k ->
    (throw k (callcc (\k' -> 1 + (throw k 0))))
    + (throw k 2))

callcc (\k ->
    (throw k (callcc (\k' -> 1 + (throw k' 0))))
    + (throw k 2))
find e l = callcc' (\return ->
  let find' e l = if l == [] then throw return Nothing
               else if (head l) == e
               then throw return (Just e)
               else find' e (tail l)
     in find' e l)
Exceptions

\[
\text{withExc block handler = callcc' (} \lambda k \to \text{block} \\
\qquad (\lambda v \to \text{throw } k (\text{handler } v)))
\]

\[
\text{raise exc } v = \text{exc } v
\]

\[
\text{eight'} = \text{withExc } (\lambda \text{exc} \to 1 + (\text{raise exc } 2)) \\
\qquad (\lambda v \to v \times 4) \quad \text{// the handler}
\]
Part III

Concluding Remarks
Continuations are Powerful

• local state (i.e. memory)
• co-routines
• threads
• processes
• reduction orders
• …
Continuations are Hard

Google search results for "ml callcc"

- **ML with callcc is unsound**
  - To: types; Subject: ML with callcc is unsound; From: Robert.Harper@gottlob.tip.cs.cmu.edu; Date: Mon, 8 Jul 91 16:22:52 EDT; Reply-To: Robert Harper ...

- A page about call/cc - Madore.org
  - [www.madore.org/~david/computers/callcc.html](http://www.madore.org/~david/computers/callcc.html)
  - The Standard ML of New Jersey also has a call/cc function, that is called SMLofNJ.Cont.callcc (it differs a bit from the Scheme version in that continuations are ... What is call/cc? What is this ... - A first introduction - Who invented call/cc?

- Specifying ECMAScript via ML | Lambda the Ultimate
  - [lambda-the-ultimate.org/node/1784](http://lambda-the-ultimate.org/node/1784)
  - 20 Oct 2006 - 34 posts - 17 authors
  - Of course, one has to wonder why ML and not (Haskell or Scheme)... By Ehud .... Incidentally, another problem with call/cc in OCaml is that it...
We’ve merely scratched the surface

- Delimited Continuations
- Compiler Optimisation
- “The mother of all Monads”
- Classical Logic (!)
- …
If You Want to Know More...


“Compiling With Continuations”, Appel (1992)


“A Formulae-as-types Notion of Control”, Griffin (1990)

“Call-by-name, Call-by-value and the Lambda Calculus”, Plotkin (1975)

“Representing Control—a study of the CPS transformation”, Danvy and Filinski (1992)