Conversion from `callcc` to Continuation Passing Style (CPS)

Tom Nikken
Based on slides from Xavier Leroy, INRIA

Type Theory and Coq
May 8th 2018

Conversion to continuation-passing style (CPS)

- **Goal:** make explicit the handling of continuations.
- **Input:** a call-by-value functional language with `callcc`.
- **Output:** a pure functional language (no `callcc`).
- **Uses:** compilation of `callcc`, semantics, programming with continuations in Scheme, Haskell, ...

Illustration of CPS in Javascript

We need to save the continuation, in case we encounter a `callcc`
Idea: turn everything into a function from its continuation to its value

```
function id(x) {
  return x;
}
```

Illustration of CPS in Javascript

We need to save the continuation, in case we encounter a `callcc`
Idea: turn everything into a function from its continuation to its value

```
function id(x, cont) {
  cont(x);
}
```

Illustration of CPS in Javascript

```
function fact(n,cont) {
  if (n == 0)
    cont(1) ;
  else
    fact(n-1, function (t0) {
     cont(n * t0) }) ;
}
```

Illustration of CPS in Javascript

```
function fact(n,cont) {
  if (n == 0)
    cont(1) ;
  else
    fact(n-1, function (t0) {
     cont(n * t0) }) ;
}

fact (5, function (n) {
  console.log(n) ;
})
```

Now in the lambda calculus

Now in the lambda calculus

$[\![...]\!] : (\lambda + \texttt{callcc}) \to \lambda$

If $a : \tau$, then $[\![a]\!] : ([\![\tau]\!] \to \mathit{answer}) \to \mathit{answer}$, where:
$[\![\tau]\!] = \tau$ for base types
and $[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \to ([\![\tau_2]\!] \to \mathit{answer}) \to \mathit{answer}$

$$
\begin{aligned}
[\![N]\!] &= \lambda k.\ k\ N \\
[\![x]\!] &= \lambda k.\ k\ x \\
[\![\lambda x.\ a]\!] &= \lambda k.\ k\ (\lambda x.\ [\![a]\!]) \\
[\![f\ a]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_f.\ [\![a]\!]\ (\lambda v_a.\ v_f\ v_a\ k))
\end{aligned}
$$

$$
\begin{aligned}
[\![\texttt{callcc}\ f]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_f.\ v_f\ k\ k) \\
[\![\texttt{throw}\ a\ b]\!] &= \lambda k.\ [\![a]\!]\ (\lambda v_a.\ [\![b]\!]\ (\lambda v_b.\ v_a\ v_b))
\end{aligned}
$$

Now in the lambda calculus

$[\![...]\!] : (\lambda + \texttt{callcc}) \to \lambda$

If $a : \tau$, then $[\![a]\!] : ([\![\tau]\!] \to \mathit{answer}) \to \mathit{answer}$, where:
$[\![\tau]\!] = \tau$ for base types
and $[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \to ([\![\tau_2]\!] \to \mathit{answer}) \to \mathit{answer}$

$$
\begin{aligned}
[\![N]\!] &= \lambda k.\ k\ N \\
[\![x]\!] &= \lambda k.\ k\ x \\
[\![\lambda x.\ a]\!] &= \lambda k.\ k\ (\lambda x.\ [\![a]\!]) \\
[\![f\ a]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_f.\ [\![a]\!]\ (\lambda v_a.\ v_f\ v_a\ k))
\end{aligned}
$$

$$
\begin{aligned}
[\![\texttt{callcc}\ f]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_f.\ v_f\ k\ k) \\
[\![\texttt{throw}\ a\ b]\!] &= \lambda k.\ [\![a]\!]\ (\lambda v_a.\ [\![b]\!]\ (\lambda v_b.\ v_a\ v_b))
\end{aligned}
$$

Remarks

- `callcc` conversion duplicates k: one as argument to f and one to propagate the current continuation.
- Throw ignores k.
- Argument of application is always a value, so works for CBV and CBN.

Exercise

$$[\![f\ x]\!] =_\beta ?$$

Exercise

$$[\![f\ x]\!] = \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k))$$

Exercise

$$
\begin{aligned}
[\![f\ x]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&= \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda k_2.\ k_2\ x)\ (\lambda v_b.\ v_a\ v_b\ k))
\end{aligned}
$$

Exercise

$$
\begin{aligned}
[\![f\ x]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&= \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda k_2.\ k_2\ x)\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda v_b.\ v_a\ v_b\ k)\ x)
\end{aligned}
$$

Exercise

$$
\begin{aligned}
[\![f\ x]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&= \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda k_2.\ k_2\ x)\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda v_b.\ v_a\ v_b\ k)\ x) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ v_a\ x\ k)
\end{aligned}
$$

Exercise

$$
\begin{aligned}
[\![f\ x]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&= \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda k_2.\ k_2\ x)\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda v_b.\ v_a\ v_b\ k)\ x) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ v_a\ x\ k) \\
&\to_\beta \lambda k.\ (\lambda v_a.\ v_a\ x\ k)\ f
\end{aligned}
$$

Exercise

$$
\begin{aligned}
[\![f\ x]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&= \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda k_2.\ k_2\ x)\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda v_b.\ v_a\ v_b\ k)\ x) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ v_a\ x\ k) \\
&\to_\beta \lambda k.\ (\lambda v_a.\ v_a\ x\ k)\ f \\
&\to_\beta \lambda k.\ f\ x\ k
\end{aligned}
$$

Exercise

$$
\begin{aligned}
[\![f\ x]\!] &= \lambda k.\ [\![f]\!]\ (\lambda v_a.\ [\![x]\!]\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&= \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda k_2.\ k_2\ x)\ (\lambda v_b.\ v_a\ v_b\ k)) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ (\lambda v_b.\ v_a\ v_b\ k)\ x) \\
&\to_\beta \lambda k.\ (\lambda k_1.\ k_1\ f)\ (\lambda v_a.\ v_a\ x\ k) \\
&\to_\beta \lambda k.\ (\lambda v_a.\ v_a\ x\ k)\ f \\
&\to_\beta \lambda k.\ f\ x\ k
\end{aligned}
$$

Perform β reductions at transformation time to eliminate the "administrative redexes" introduced by the translation.

Example

To show the effect of the translation.

$[\![f(f\ x)]\!] =_\beta \lambda k.\ f\ x\ (\lambda v.\ f\ v\ k)$

Example

To show the effect of the translation.

$$[\![f(f\ x)]\!] =_\beta \lambda k.\ f\ x\ (\lambda v.\ f\ v\ k)$$

$$\begin{aligned}&[\![\mu fact.\ \lambda n.\ \texttt{if}\ n\ =\ 0\ \texttt{then}\ 1\ \texttt{else}\ n\ *\ fact(n-1)]\!]\\ &=_\beta \lambda k_0.\ k_0(\\ &\quad \mu fact.\ \lambda n.\ \lambda k.\ \texttt{if}\ n = 0\ \texttt{then}\ k\ 1\ \texttt{else}\ fact(n-1)(\lambda v.\ k\ (n\ *\ v)))\end{aligned}$$

Example

To show the effect of the translation.

$$[\![f\,(f\ x)]\!] =_\beta \lambda k.\ f\ x\ (\lambda v.\ f\ v\ k)$$

$$\begin{aligned}&[\![\mu fact.\ \lambda n.\ \texttt{if}\ n\ =\ 0\ \texttt{then}\ 1\ \texttt{else}\ n\ *\ fact(n-1)]\!]\\ &=_\beta \lambda k_0.\ k_0(\\ &\quad \mu fact.\ \lambda n.\ \lambda k.\ \texttt{if}\ n = 0\ \texttt{then}\ k\ 1\ \texttt{else}\ fact(n-1)(\lambda v.\ k\ (n\ *\ v)))\end{aligned}$$

$$[\![\texttt{callcc}\ (\lambda k.a)]\!] =_\beta \lambda k.\ [\![a]\!]\ k$$

Example

To show the effect of the translation.

$$[\![f\,(f\ x)]\!] =_\beta \lambda k.\ f\ x\ (\lambda v.\ f\ v\ k)$$

$$\begin{aligned}&[\![\mu fact.\ \lambda n.\ \texttt{if}\ n\ =\ 0\ \texttt{then}\ 1\ \texttt{else}\ n\ *\ fact(n-1)]\!]\\ &=_\beta \lambda k_0.\ k_0(\\ &\quad \mu fact.\ \lambda n.\ \lambda k.\ \texttt{if}\ n = 0\ \texttt{then}\ k\ 1\ \texttt{else}\ fact(n-1)(\lambda v.\ k\ (n\ *\ v)))\end{aligned}$$

$$[\![\texttt{callcc}\ (\lambda k.a)]\!] =_\beta \lambda k.\ [\![a]\!]\ k$$

Still confused?

The types can help.

$$[\![N]\!] = \lambda k.\ k\ N$$

$[\![N]\!]$:

Still confused?

The types can help.

$$[\![N]\!] = \lambda k.\ k\ N$$

$$[\![N]\!] : (nat \to answer) \to answer$$
$$k :$$

Still confused?

The types can help.

$$[\![N]\!] = \lambda k.\ k\ N$$

$$[\![N]\!] : (nat \to answer) \to answer$$
$$k : nat \to answer$$

Still confused?

$$[\![\lambda x.\ a]\!] = \lambda k.\ k\ (\lambda x.\ [\![a]\!])$$

$$[\![\lambda x.\ a]\!] : ([\![x \to a]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$[\![a]\!] : ([\![a]\!] \to \mathit{answer}) \to \mathit{answer}$$

Still confused?

$$[\![\lambda x.\ a]\!] = \lambda k.\ k\ (\lambda x.\ [\![a]\!])$$

$$[\![\lambda x.\ a]\!] : ([\![x \to a]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$[\![a]\!] : ([\![a]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$k : [\![x \to a]\!] \to \mathit{answer}$$

Still confused?

$$[\![\lambda x.\ a]\!] = \lambda k.\ k\ (\lambda x.\ [\![a]\!])$$

$$[\![\lambda x.\ a]\!] : ([\![x \to a]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$[\![a]\!] : ([\![a]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$k : [\![x \to a]\!] \to \mathit{answer}$$
$$[\![x \to a]\!] = [\![x]\!] \to ([\![a]\!] \to \mathit{answer}) \to \mathit{answer}$$

Still confused?

$$[\![\lambda x.\ a]\!] = \lambda k.\ k\ (\lambda x.\ [\![a]\!])$$

$$\begin{aligned}
[\![\lambda x.\ a]\!] &: ([\![x \to a]\!] \to \mathit{answer}) \to \mathit{answer} \\
[\![a]\!] &: ([\![a]\!] \to \mathit{answer}) \to \mathit{answer} \\
k &: [\![x \to a]\!] \to \mathit{answer} \\
[\![x \to a]\!] &= [\![x]\!] \to ([\![a]\!] \to \mathit{answer}) \to \mathit{answer} \\
\lambda x.\ [\![a]\!] &: [\![x]\!] \to ([\![a]\!] \to \mathit{answer}) \to \mathit{answer}
\end{aligned}$$

Still confused?

$$[\![f\ a]\!] = \lambda k.\ [\![f]\!]\ (\lambda v_f.\ [\![a]\!]\ (\lambda v_a.\ v_f\ v_a\ k))$$

with $f : a \to b$ and $a : a$

$$[\![f\ a]\!] : ([\![b]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$[\![f]\!] : ([\![a \to b]\!] \to \mathit{answer}) \to \mathit{answer}$$
$$[\![a]\!] : ([\![a]\!] \to \mathit{answer}) \to \mathit{answer}$$

Still confused?

$$[\![f\ a]\!] = \lambda k.\ [\![f]\!]\ (\lambda v_f.\ [\![a]\!]\ (\lambda v_a.\ v_f\ v_a\ k))$$

with $f : a \to b$ and $a : a$

$$\begin{aligned} [\![f\ a]\!] &: ([\![b]\!] \to \mathit{answer}) \to \mathit{answer} \\ [\![f]\!] &: ([\![a \to b]\!] \to \mathit{answer}) \to \mathit{answer} \\ [\![a]\!] &: ([\![a]\!] \to \mathit{answer}) \to \mathit{answer} \\ k &: [\![b]\!] \to \mathit{answer} \end{aligned}$$

Still confused?

$$[\![f\ a]\!] = \lambda k.\ [\![f]\!]\ (\lambda v_f.\ [\![a]\!]\ (\lambda v_a.\ v_f\ v_a\ k))$$

with $f : a \to b$ and $a : a$

$$\begin{aligned}
[\![f\ a]\!] &: ([\![b]\!] \to \mathit{answer}) \to \mathit{answer} \\
[\![f]\!] &: ([\![a \to b]\!] \to \mathit{answer}) \to \mathit{answer} \\
[\![a]\!] &: ([\![a]\!] \to \mathit{answer}) \to \mathit{answer} \\
k &: [\![b]\!] \to \mathit{answer} \\
v_f &: [\![a \to b]\!] = [\![a]\!] \to ([\![b]\!] \to \mathit{answer}) \to \mathit{answer}
\end{aligned}$$

Still confused?

$$[\![f\ a]\!] = \lambda k.\ [\![f]\!]\ (\lambda v_f.\ [\![a]\!]\ (\lambda v_a.\ v_f\ v_a\ k))$$

with $f : a \to b$ and $a : a$

$$\begin{aligned}
[\![f\ a]\!] &: ([\![b]\!] \to \mathit{answer}) \to \mathit{answer} \\
[\![f]\!] &: ([\![a \to b]\!] \to \mathit{answer}) \to \mathit{answer} \\
[\![a]\!] &: ([\![a]\!] \to \mathit{answer}) \to \mathit{answer} \\
k &: [\![b]\!] \to \mathit{answer} \\
v_f &: [\![a \to b]\!] = [\![a]\!] \to ([\![b]\!] \to \mathit{answer}) \to \mathit{answer} \\
v_a &: [\![a]\!]
\end{aligned}$$

Execution of CPS-converted programs

Execute *prog* by giving it the initial continuation $\lambda x.x$:

$$[\![prog]\!](\lambda x.x)$$

Theorem

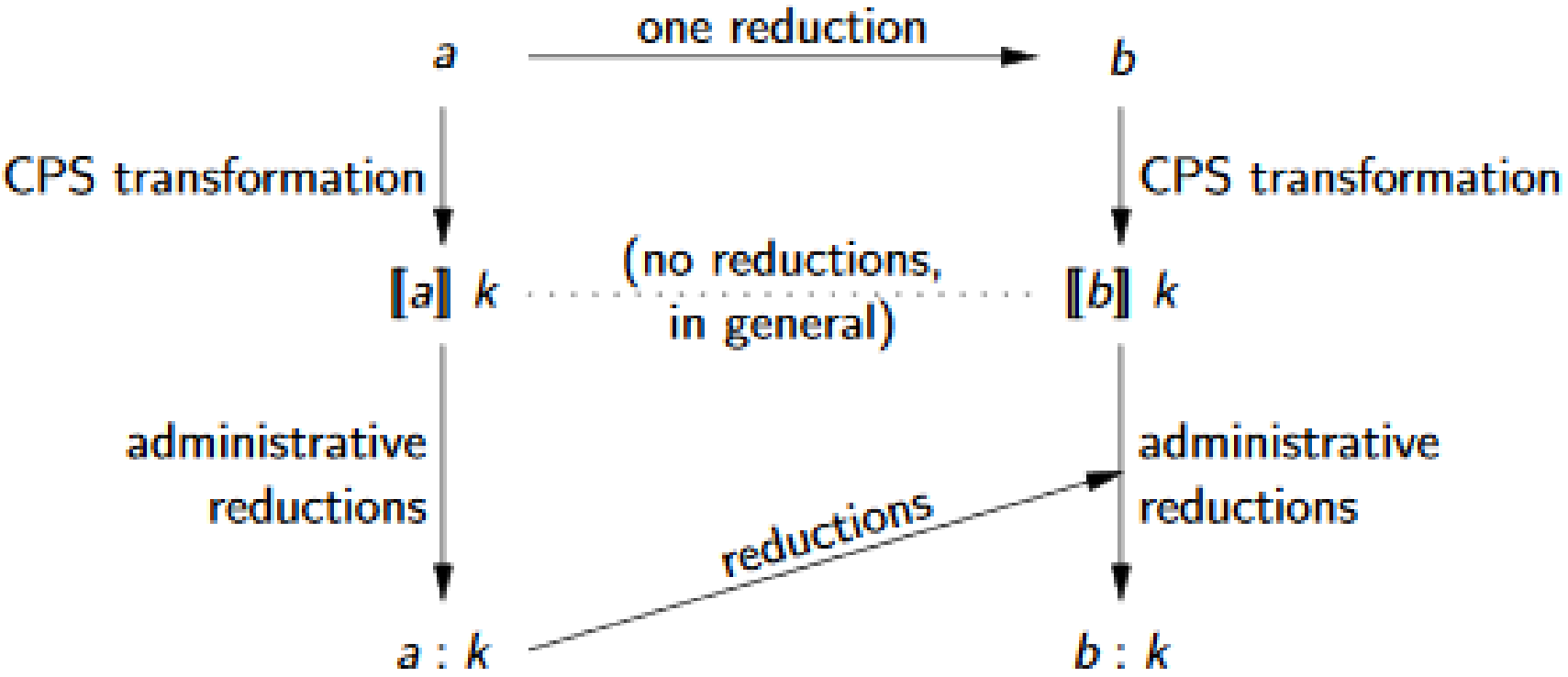
If $a \to^* v$, then $[\![a]\!](\lambda x.x) \to^* [\![v]\!]_v$. If a diverges, so does $[\![a]\!](\lambda x.x)$

Where the CPS translation of values v is:

$$[\![N]\!]_v = N \qquad [\![\lambda x.a]\!]_v = \lambda x.[\![a]\!]$$

Plotkin's proof

A difficult proof based on the following simulation diagram:



CPS terms

The λ-terms produced by the CPS transformation have a very specific shape, described by the following grammar:

CPS atom: $\quad atom ::= x \mid N \mid \lambda v.\ body \mid \lambda x.\lambda k.\ body$

CPS body: $\quad body ::= atom \mid atom_1\ atom_2 \mid atom_1\ atom_2\ atom_3$

$[\![a]\!]$ is an *atom*, and $[\![a]\!]\ (\lambda x.x)$ is a *body*.

Reduction of CPS terms

CPS atom: $atom ::= x \mid N \mid \lambda v.\ body \mid \lambda x.\lambda k.\ body$

CPS body: $body ::= atom \mid atom_1\ atom_2 \mid atom_1\ atom_2\ atom_3$

Note that all applications (unary or binary) are in tail-position and at application-time, their arguments are closed atoms, that is, values.

The following reduction rules suffice to evaluate CPS-converted programs:

$$\begin{aligned} (\lambda x.\lambda k.\ body)\ atom_1\ atom_2 &\rightarrow body[x \leftarrow atom_1,\ k \leftarrow atom_2] \\ (\lambda v.\ body)\ atom &\rightarrow body[v \leftarrow atom] \end{aligned}$$

These reductions are always applied at the top of the program — there is no need for reduction under a context!