Formalizing Implementation Strategies for First-Class Continuations

Olivier Danvy

BRICS, University of Aarhus, Denmark

(danvy@brics.dk)

BRICS

EWSCS, March 1, 2005

Plan

1. Implementation strategies

2. CPS programs

3. Second-class continuations

4. First-class continuations

5. Conclusion and perspectives

Folklore (1/2)

- In “normal” programs, control $\boxed{\text{can}}$ be implemented with a control stack. (“Recursive programming”, Dijkstra, 1960.)
- In programs with call/cc, control $\boxed{\text{cannot}}$ be implemented with a control stack.

Folklore (2/2)

Three implementation strategies:

1. allocate continuations in the heap;
2. allocate continuations on a stack

   and copy the stack in case of call/cc;
3. segment the stack.

But what does this all mean?

And in which sense is this correct, if at all?

Our approach

- Consider continuation-passing style (CPS) programs.
- Write one standard abstract machine.
- Formalize implementation strategies with other abstract machines.
- Prove that these machines simulate each other.

Plan

1. Implementation strategies $\surd$

2. CPS programs

3. Second-class continuations

4. First-class continuations

5. Conclusion and perspectives

BNF of direct-style terms

$$
\begin{array}{rcl}
p \in \text{DProg} & ::= & e \\
e \in \text{DExp} & ::= & e_0\, e_1 \mid t \\
t \in \text{DTriv} & ::= & \ell \mid x \mid \lambda x.e \\
\ell \in \text{Lit} & & \\
x \in \text{Ide} & &
\end{array}
$$

BNF of CPS terms

$$
\begin{array}{rcl}
p \in \text{CProg} & ::= & \lambda k.e \\
e \in \text{CExp} & ::= & t_0\, t_1\, c \mid c\, t \\
t \in \text{CTriv} & ::= & \ell \mid x \mid v \mid \lambda x.\lambda k.e \\
c \in \text{Cont} & ::= & k \mid \lambda v.e \\
\ell \in \text{Lit} & & \\
x \in \text{Ide} & & \\
k \in \text{IdeC} & & \\
v \in \text{IdeV} & &
\end{array}
$$

A closure property

Property 1 *The BNF of CPS programs is closed under* β*-reduction.*

Proof: Straightforward structural induction – a β-redex can only occur as

1. $e \ ::= \ (\lambda x.\lambda k.e')\, t\, c$

2. $e \ ::= \ (\lambda v.e')\, t$

$\square$

Left-to-right, CBV CPS transformation

$$\llbracket e \rrbracket^{\mathsf{DProg}}_{\mathsf{cps}} = \lambda k.\llbracket e \rrbracket^{\mathsf{DExp}}_{\mathsf{cps}} k$$

$$\begin{aligned}
\llbracket e_0\, e_1 \rrbracket^{\mathsf{DExp}}_{\mathsf{cps}} c &= \llbracket e_0 \rrbracket^{\mathsf{DExp}}_{\mathsf{cps}} \lambda v_0.\llbracket e_1 \rrbracket^{\mathsf{DExp}}_{\mathsf{cps}} \lambda v_1.v_0\, v_1\, c \\
\llbracket t \rrbracket^{\mathsf{DExp}}_{\mathsf{cps}} c &= c\, \llbracket t \rrbracket^{\mathsf{DTriv}}_{\mathsf{cps}}
\end{aligned}$$

$$\begin{aligned}
\llbracket \ell \rrbracket^{\mathsf{DTriv}}_{\mathsf{cps}} &= \ell \\
\llbracket x \rrbracket^{\mathsf{DTriv}}_{\mathsf{cps}} &= x \\
\llbracket \lambda x.e \rrbracket^{\mathsf{DTriv}}_{\mathsf{cps}} &= \lambda x.\lambda k.\llbracket e \rrbracket^{\mathsf{DExp}}_{\mathsf{cps}} k
\end{aligned}$$

Our standard abstract machine (1/2)

$$\vdash^{\text{CProg}}_{\text{std}} p \hookrightarrow a$$

is satisfied whenever a CPS program p evaluates to an answer a.

$$\vdash^{\text{CExp}}_{\text{std}} e \hookrightarrow a$$

is satisfied whenever a CPS expression e evaluates to an answer a.

Our standard abstract machine (2/2)

- The machine starts and stops with the initial continuation k_{init}, a distinguished fresh continuation identifier.
- $a \in$ Answer $::= \ \ell \mid \lambda x.\lambda k.e \mid$ error

$$\frac{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ e[k_{\mathsf{init}}/k] \hookrightarrow a}{\vdash^{\mathsf{CProg}}_{\mathsf{std}}\ \lambda k.e \hookrightarrow a}$$

$$\frac{}{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ \ell\ t\ c \hookrightarrow \mathsf{error}} \qquad \frac{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ e[t/x,\ c/k] \hookrightarrow a}{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ (\lambda x.\lambda k.e)\ t\ c \hookrightarrow a}$$

$$\frac{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ e[t/v] \hookrightarrow a}{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ (\lambda v.e)\ t \hookrightarrow a} \qquad \frac{}{\vdash^{\mathsf{CExp}}_{\mathsf{std}}\ k_{\mathsf{init}}\ t \hookrightarrow t}$$

The rest of this lecture.

$$\boxed{\boxed{\text{We focus on the continuation identifiers } k.}}$$

Plan

1. Implementation strategies $\surd$

2. CPS programs $\surd$

3. Second-class continuations

4. First-class continuations

5. Conclusion and perspectives

Our starting observation

1. Each direct-style expression occurs in one context.

2. Each CPS expression has one continuation.

$$k \models^{\mathrm{CExp}}_{\mathrm{2cc}} e$$

$$\frac{k \notin \mathsf{FC}(t_0) \qquad k \notin \mathsf{FC}(t_1) \qquad k \models^{\mathsf{Cont}}_{\mathsf{2cc}} c}{k \models^{\mathsf{CExp}}_{\mathsf{2cc}} t_0\, t_1\, c}$$

$$\frac{k \models^{\mathsf{Cont}}_{\mathsf{2cc}} c \qquad k \notin \mathsf{FC}(t)}{k \models^{\mathsf{CExp}}_{\mathsf{2cc}} c\, t}$$

$$\frac{k \models^{\mathsf{CExp}}_{\mathsf{2cc}} e}{k \models^{\mathsf{Cont}}_{\mathsf{2cc}} \lambda v.e} \qquad \frac{}{k \models^{\mathsf{Cont}}_{\mathsf{2cc}} k}$$

Second-class position
Second-class continuations

Definition 2 *In a continuation abstraction* $\lambda \mathrm{k}.e$, k *occurs in* second-class position *and denotes a* second-class continuation *whenever the judgment* $\mathrm{k} \models^{\mathrm{CExp}}_{2cc} e$ *is satisfied.*

More generally

$\boxed{\text{All}}$ continuation identifiers

denote second-class continuations.

$$\models^{\text{CProg}}_{\text{2cc*}} \; p$$

$$\frac{k \models^{\mathsf{CExp}}_{\mathsf{2cc^*}} e}{\models^{\mathsf{CProg}}_{\mathsf{2cc^*}} \lambda k.e}$$

$$\frac{\models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} t_0 \qquad \models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} t_1 \qquad k \models^{\mathsf{Cont}}_{\mathsf{2cc^*}} c}{k \models^{\mathsf{CExp}}_{\mathsf{2cc^*}} t_0\, t_1\, c}$$

$$\frac{k \models^{\mathsf{Cont}}_{\mathsf{2cc^*}} c \qquad \models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} t}{k \models^{\mathsf{CExp}}_{\mathsf{2cc^*}} c\, t}$$

$$\frac{}{\models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} \ell}$$

$$\frac{}{\models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} x}$$

$$\frac{}{\models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} v}$$

$$\frac{\mathrm{k} \models^{\mathsf{CExp}}_{\mathsf{2cc^*}} e}{\models^{\mathsf{CTriv}}_{\mathsf{2cc^*}} \lambda x.\lambda \mathrm{k}.e}$$

$$\frac{\mathrm{k} \models^{\mathsf{CExp}}_{\mathsf{2cc^*}} e}{\mathrm{k} \models^{\mathsf{Cont}}_{\mathsf{2cc^*}} \lambda v.e}$$

$$\frac{}{\mathrm{k} \models^{\mathsf{Cont}}_{\mathsf{2cc^*}} \mathrm{k}}$$

2Cont-validity

Definition 3 *Given a CPS program* $\mathrm{p} = \lambda \mathrm{k}.e$*, the judgment* $\models^{\text{CProg}}_{\textit{2cc*}} \mathrm{p}$ *(read: "*p *is 2Cont-valid") is satisfied whenever* $\mathrm{k} \models^{\text{CExp}}_{\textit{2cc}} e$ *is satisfied and all continuation abstractions* $\lambda \mathrm{k}'.\mathrm{e}'$ *occurring in* e *satisfy* $\mathrm{k}' \models^{\text{CExp}}_{\textit{2cc}} \mathrm{e}'$*.*

The CPS transformation yields 2Cont-valid programs

Lemma 4 *For any* $\mathfrak{p} \in$ DProg,

$$\models^{\mathsf{CProg}}_{\mathit{2cc*}} [\![\mathfrak{p}]\!]^{\mathsf{DProg}}_{\mathsf{cps}}.$$

Proof: Straightforward induction on direct-style programs. $\square$

2Cont-validity under β-reduction

Theorem 5

1. If $(\lambda x.\lambda k.e')\; t\; c$ *is 2Cont-valid, then* $e[t/x,\ c/k]$ *is 2Cont-valid.*

2. If $(\lambda v.e)\; t$ *is 2Cont-valid then* $e[t/v]$ *is 2Cont-valid.*

Proof: Using inversion on the derivation that a redex is 2Cont-valid we obtain derivations on which we prove inductively that 2Cont-validity is preserved. $\square$

And therefore...

One continuation identifier is enough!

BNF for 2CPS programs

$$
\begin{array}{rcl}
p \in \text{2CProg} & ::= & \lambda\star.e \\
e \in \text{2CExp} & ::= & t_0\, t_1\, c \mid c\, t \\
t \in \text{2CTriv} & ::= & \ell \mid x \mid v \mid \lambda x.\lambda\star.e \\
c \in \text{2Cont} & ::= & \star \mid \lambda v.e \\
\ell \in \text{Lit} & & \\
x \in \text{Ide} & & \\
\star \in \text{Token} & & \\
v \in \text{IdeV} & &
\end{array}
$$

Mappings between CPS and 2CPS

- straightforward
- homomorphic
- inverses (modulo α-conversion)

Notation: $|\cdot|_2$: CPS $\longrightarrow$ 2CPS

A stack machine

Idea: implement the bindings of $\star$ with a stack.

$$\varphi \in \text{2CStack} ::= \bullet \mid \varphi,\ \lambda v.e$$

Two judgments

$$\vdash^{\text{2CProg}}_{\text{2cc}} p \hookrightarrow a$$

is satisfied whenever $p \in$ 2CProg evaluates to an answer $a \in$ 2Answer.

$$\varphi \vdash^{\text{2CExp}}_{\text{2cc}} e \hookrightarrow a$$

is satisfied whenever $e \in$ 2CExp evaluates to an answer a, given $\varphi \in$ 2CStack.

$$\frac{\bullet \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} e \hookrightarrow a}{\vdash^{\mathsf{2CProg}}_{\mathsf{2cc}} \lambda\star.e \hookrightarrow a}$$

$$\frac{}{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} \ell\, t\, c \hookrightarrow \mathsf{error}}$$

$$\frac{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} e[t/x] \hookrightarrow a}{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} (\lambda x.\lambda\star.e)\, t \star \hookrightarrow a}$$

$$\frac{\varphi,\ \lambda v.e' \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} e[t/x] \hookrightarrow a}{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} (\lambda x.\lambda\star.e)\, t\, \lambda v.e' \hookrightarrow a}$$

$$\frac{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} e[t/v] \hookrightarrow a}{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} (\lambda v.e)\, t \hookrightarrow a}$$

$$\frac{}{\bullet \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} \star t \hookrightarrow t}$$

$$\frac{\varphi \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} e[t/v] \hookrightarrow a}{\varphi,\ \lambda v.e \vdash^{\mathsf{2CExp}}_{\mathsf{2cc}} \star t \hookrightarrow a}$$

Equivalence

Idea:

show that for each abstract machine,

the computations

are in bijective correspondence.

Key technique

$$\text{A "stack substitution"} \begin{cases} e\{\varphi\}_2 \\ c\{\varphi\}_2 \end{cases}$$

Stack substitution

$$| t_0\, t_1\, c |_2 \{\varphi\}_2 \;=\; t_0\, t_1\, (| c |_2 \{\varphi\}_2)$$

$$| c\, t |_2 \{\varphi\}_2 \;=\; (| c |_2 \{\varphi\}_2)\, t$$

$$| \lambda v.e |_2 \{\varphi\}_2 \;=\; \lambda v.(| e |_2 \{\varphi\}_2)$$

$$\star\{\bullet\}_2 \;=\; k_{\text{init}}$$

$$\star\{\varphi,\, | \lambda v.e |_2\}_2 \;=\; \lambda v.(| e |_2 \{\varphi\}_2)$$

Lemma 6

1. For any $e \in \mathrm{CExp}$ *satisfying* $k \models^{\mathrm{CExp}}_{2cc^*} e$ *for some* k *and for any stack of 2Cont continuations* φ, $k_{init} \models^{\mathrm{CExp}}_{2cc^*} |e|_2\{\varphi\}_2$.

2. For any $c \in \mathrm{Cont}$ *satisfying* $k \models^{\mathrm{Cont}}_{2cc^*} e$ *for some* k *and for any stack of 2Cont continuations* φ, $k_{init} \models^{\mathrm{Cont}}_{2cc^*} |c|_2\{\varphi\}_2$.

Proof: By structural induction. □

Control-stack substitution

Lemma 7

1. *For any* $e' \in$ CExp *satisfying* $k \models^{\text{CExp}}_{\textit{2cc*}} e'$ *for some* k *and for any stack of 2Cont continuations* φ*,* $| e' |_2\{\varphi\}_2 = e'[\star\{\varphi\}_2/k]$.

2. *For any* $e \in$ 2CExp, *for any* $t' \in$ CTriv *satisfying*

$\models^{\text{CTriv}}_{\mathit{2cc*}} t'$, *for any identifier* i *in* Ide *or in* IdeV, *and for any stack of 2Cont continuations* φ,

$e[| t' |_2 / i]\{\varphi\}_2 = e\{\varphi\}_2 [t'/i]$.

Proof: By structural induction. $\square$

Theorem 8 (Simulation)

1. For any 2Cont-valid CPS program p*,*

$\vdash^{\mathit{CProg}}_{\mathit{std}} \mathrm{p} \hookrightarrow \mathrm{a}$ *if and only if* $\vdash^{\mathit{2CProg}}_{\mathit{2cc}} |\,\mathrm{p}\,|_2 \hookrightarrow |\,\mathrm{a}\,|_2$.

2. For any CPS expression e *satisfying* $\mathrm{k} \models^{\mathit{CExp}}_{\mathit{2cc*}} e$ *for some* k *and for any stack of 2Cont continuations* φ*,* $\vdash^{\mathit{CExp}}_{\mathit{std}} |\,\mathrm{e}\,|_2\{\varphi\}_2 \hookrightarrow \mathrm{a}$ *if and only if* $\varphi \vdash^{\mathit{2CExp}}_{\mathit{2cc}} |\,\mathrm{e}\,|_2 \hookrightarrow |\,\mathrm{a}\,|_2$.

Proof

By induction over the structure of the derivations, frequently relying on Lemma 7.

Let us see the case of tail calls.

Case $\mathcal{E} = \dfrac{\begin{matrix}\mathcal{E}_1 \\ \varphi \vdash_{2cc}^{\text{2CExp}} e[t/x] \hookrightarrow a\end{matrix}}{\varphi \vdash_{2cc}^{\text{2CExp}} (\lambda x.\lambda \star.e)\, t \star \hookrightarrow a}$

where $\mathcal{E}_1$ names the derivation ending in $\varphi \vdash_{2cc}^{\text{2CExp}} e[t/x] \hookrightarrow a$.

By applying the induction hypothesis to $\mathcal{E}_1$, we obtain a derivation

$$\frac{\mathcal{E}_1'}{\vdash^{\mathsf{CExp}}_{\mathsf{std}} e[t/x]\{\varphi\}_2 \hookrightarrow a'}$$

such that $a = \lfloor a' \rfloor_2$.

Since $e[t/x]$ is a 2CPS expression,

- there exists a CPS expression e' satisfying $k \models^{\text{CExp}}_{\text{2cc*}} e'$ for some k, and

- there exists a CPS trivial expression t' satisfying $\models^{\text{CTriv}}_{\text{2cc*}} t'$

such that $e = | e' |_2$ and $t = | t' |_2$.

By Lemma 7,

$$
\begin{aligned}
\lfloor e' \rfloor_2 [\lfloor t' \rfloor_2 / x]\{\varphi\}_2 &= \lfloor e' \rfloor_2 \{\varphi\}_2 [t'/x] \\
&= e'[\star\{\varphi\}_2/k][t'/x] \\
&= e'[t'/x, \star\{\varphi\}_2/k]
\end{aligned}
$$

because t' has no free k

and φ has no free x.

By inference,

$$\frac{\vdash^{\text{CExp}}_{\text{std}} e'[t'/x,\ \star\{\varphi\}_2/k] \hookrightarrow a'}{\vdash^{\text{CExp}}_{\text{std}} (\lambda x.\lambda k.e')\, t'\, (\star\{\varphi\}_2) \hookrightarrow a'}$$

Now by definition of stack substitution,

$$(\lambda x.\lambda k.e')\, t'\, (\star\{\varphi\}_2)$$

$= \lfloor (\lambda x.\lambda k.e)\, t\, k' \rfloor_2\{\varphi\}_2$, for some k'.

In other words, there exists a derivation

$$\frac{\begin{array}{c}\mathcal{E}_1' \\ \vdash^{\mathsf{CExp}}_{\mathsf{std}} \mid e[t/x] \llbracket \{\varphi\} \rrbracket \hookrightarrow a'\end{array}}{\vdash^{\mathsf{CExp}}_{\mathsf{std}} \mid (\lambda x.\lambda k.e)\; t\; k' \llbracket \{\varphi\} \rrbracket \hookrightarrow a'}$$

which is what we wanted to show. $\square$

First conclusion

Two folklore theorems about CPS programs with second-class continuations:

1. One continuation identifier is enough.

2. Control can be implemented with a control stack.

Plan

1. Implementation strategies $\surd$

2. CPS programs $\surd$

3. Second-class continuations $\surd$

4. First-class continuations

5. Conclusion and perspectives

call/cc

$$\llbracket \text{call/cc}\ e \rrbracket^{\text{DExp}}_{\text{cps}} c$$
$$= \llbracket e \rrbracket^{\text{DExp}}_{\text{cps}} \lambda f.f\,(\lambda x.\lambda k.c\,x)\,c$$

where f, x, and k are fresh.

Key observation

Continuation identifiers now occur

"out of order".

$$k \models^{\mathrm{CExp}}_{\mathrm{1cc}} e$$

$$\frac{k \in \mathsf{FC}(t_0)}{k \models^{\mathsf{CExp}}_{\mathsf{1cc}} t_0\, t_1\, c} \qquad \frac{k \in \mathsf{FC}(t_1)}{k \models^{\mathsf{CExp}}_{\mathsf{1cc}} t_0\, t_1\, c} \qquad \frac{k \models^{\mathsf{Cont}}_{\mathsf{1cc}} c}{k \models^{\mathsf{CExp}}_{\mathsf{1cc}} t_0\, t_1\, c}$$

$$\frac{k \models^{\mathsf{Cont}}_{\mathsf{1cc}} c}{k \models^{\mathsf{CExp}}_{\mathsf{1cc}} c\, t} \qquad \frac{k \in \mathsf{FC}(t)}{k \models^{\mathsf{CExp}}_{\mathsf{1cc}} c\, t}$$

$$\frac{k \models^{\mathsf{CExp}}_{\mathsf{1cc}} e}{k \models^{\mathsf{Cont}}_{\mathsf{1cc}} \lambda v.e}$$

First-class position

First-class continuations

Definition 9 *In a continuation abstraction* $\lambda \mathrm{k}.e$*,* k *occurs in* first-class position *and denotes a* first-class continuation *whenever the judgment* $\mathrm{k} \models^{\mathrm{CExp}}_{\mathit{1cc}} e$ *is satisfied.*

“Sub-closure” property

Example:

$$\begin{array}{c} \lambda k.(\lambda x.\lambda k'.k\,x)\,\ell\,k \\ \Big\downarrow \beta \\ \lambda k.k\,\ell \end{array}$$

Key idea

Single out

continuation identifiers in first-class position

BNF for 1CPS programs

$$
\begin{array}{rcl}
p \in \text{CProg} & ::= & \lambda\star.e \mid \lambda^1 k.e \\
e \in \text{CExp} & ::= & t_0\, t_1\, c \mid c\, t \\
t \in \text{CTriv} & ::= & \ell \mid x \mid v \mid \lambda x.\lambda\star.e \mid \lambda x.\lambda^1 k.e \\
c \in \text{Cont} & ::= & \lambda v.e \mid \star \mid k \\
\ell \in \text{Lit} & & \\
x \in \text{Ide} & & \\
\star \in \text{Token} & & \\
k \in \text{IdeC} & & \\
v \in \text{IdeV} & &
\end{array}
$$

From CPS to 1CPS (1/4)

$$
\begin{aligned}
[\![t_0\ t_1\ c]\!]^{\mathsf{CExp}}_{\mathsf{ann}} k &= [\![t_0]\!]^{\mathsf{CTriv}}_{\mathsf{ann}}\ [\![t_1]\!]^{\mathsf{CTriv}}_{\mathsf{ann}}\ ([\![c]\!]^{\mathsf{Cont}}_{\mathsf{ann}} k) \\
[\![c\ t]\!]^{\mathsf{CExp}}_{\mathsf{ann}} k &= ([\![c]\!]^{\mathsf{Cont}}_{\mathsf{ann}} k)\ [\![t]\!]^{\mathsf{CTriv}}_{\mathsf{ann}} \\
[\![\ell]\!]^{\mathsf{CTriv}}_{\mathsf{ann}} &= \ell \\
[\![x]\!]^{\mathsf{CTriv}}_{\mathsf{ann}} &= x \\
[\![v]\!]^{\mathsf{CTriv}}_{\mathsf{ann}} &= v \\
[\![\lambda v.e]\!]^{\mathsf{Cont}}_{\mathsf{ann}} k &= \lambda v.[\![e]\!]^{\mathsf{CExp}}_{\mathsf{ann}} k
\end{aligned}
$$

From CPS to 1CPS (2/4)

$$[\![\lambda k.e]\!]^{\mathsf{CProg}}_{\mathsf{ann}} =$$

$$\begin{cases} \lambda^{1} k.[\![e]\!]^{\mathsf{CExp}}_{\mathsf{ann}} k & \text{if } k \models^{\mathsf{CExp}}_{\mathsf{1cc}} e \\ \lambda \star.[\![e]\!]^{\mathsf{CExp}}_{\mathsf{ann}} k & \text{otherwise} \end{cases}$$

From CPS to 1CPS (3/4)

$$\llbracket \lambda x.\lambda k.e \rrbracket^{\mathsf{CTriv}}_{\mathsf{ann}} =$$
$$\begin{cases} \lambda x.\lambda^{1} k.\llbracket e \rrbracket^{\mathsf{CExp}}_{\mathsf{ann}} k & \text{if } k \models^{\mathsf{CExp}}_{\mathsf{1cc}} e \\ \lambda x.\lambda \star.\llbracket e \rrbracket^{\mathsf{CExp}}_{\mathsf{ann}} k & \text{otherwise} \end{cases}$$

From CPS to 1CPS (4/4)

$$[\![\mathrm{k}']\!]^{\mathsf{Cont}}_{\mathsf{ann}}\mathrm{k} = \begin{cases} \star & \text{if } \mathrm{k} = \mathrm{k}' \\ \mathrm{k}' & \text{otherwise} \end{cases}$$

From 1CPS to CPS

Straightforward – just replace $\star$ by a fresh k.

Lemma 10 (Inverseness)

$[\![\cdot]\!]^{\text{1CProg}}_{\text{unann}} \circ [\![\cdot]\!]^{\text{CProg}}_{\text{ann}} = \textit{identity}_{\alpha}$.

A stack machine for 1CPS programs

We handle first-class continuations by extending 1CPS with a new syntactic form:

$$c \in \text{1Cont} \;::=\; \lambda v.e \;\mid\; \star \;\mid\; k \;\mid\; \boxed{\text{swap } \varphi}$$

Stack substitution

$$
\begin{aligned}
t_0\, t_1\, c\{\!\{\varphi\}\!\} &= [\![t_0]\!]^{\text{1CTriv}}_{\text{unann}}\, [\![t_1]\!]^{\text{1CTriv}}_{\text{unann}}\, (c\{\!\{\varphi\}\!\}) \\
c\, t\{\!\{\varphi\}\!\} &= (c\{\!\{\varphi\}\!\})\, [\![t]\!]^{\text{1CTriv}}_{\text{unann}} \\
(\lambda v.e)\{\!\{\varphi\}\!\} &= \lambda v.(e\{\!\{\varphi\}\!\}) \\
\star\{\!\{\bullet\}\!\} &= k_{\text{init}} \\
\star\{\!\{\varphi, \lambda v.e\}\!\} &= \lambda v.(e\{\!\{\varphi\}\!\}) \\
k\{\!\{\varphi\}\!\} &= k \\
(\text{swap } \varphi')\{\!\{\varphi\}\!\} &= \star\{\!\{\varphi'\}\!\}
\end{aligned}
$$

The abstract machine

$$\vdash^{\mathsf{1CProg}}_{\mathsf{1cc}} p \hookrightarrow a$$

is satisfied whenever $p \in$ 1CProg evaluates to an answer $a \in$ 1Answer.

$$\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e \hookrightarrow a$$

is satisfied whenever $e \in$ 1CExp evaluates to an answer a, given $\varphi \in$ 1CStack.

The domain of answers: 1Answer

$$a \ ::= \ \ell \ | \ \lambda x.\lambda\star.e \ | \ \lambda x.\lambda^1 k.e \ | \ \text{error}$$

$$\frac{\bullet \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e \hookrightarrow a}{\vdash^{\mathsf{1CProg}}_{\mathsf{1cc}} \lambda\star.e \hookrightarrow a}$$

$$\frac{\bullet \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[\mathsf{swap}\,\bullet/k] \hookrightarrow a}{\vdash^{\mathsf{1CProg}}_{\mathsf{1cc}} \lambda^{1}k.e \hookrightarrow a}$$

$$\frac{}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} \ell\; t\; c \hookrightarrow \mathsf{error}}$$

$$\frac{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[t/x] \hookrightarrow a}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} (\lambda x.\lambda \star.e)\, t \star \hookrightarrow a}$$

$$\frac{\varphi,\ \lambda v.e' \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[t/x] \hookrightarrow a}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} (\lambda x.\lambda \star.e)\, t\, \lambda v.e' \hookrightarrow a}$$

$$\frac{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[t/x,\ \mathsf{swap}\,\varphi/k] \hookrightarrow a}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} (\lambda x.\lambda^{l} k.e)\, t \star \hookrightarrow a}$$

$$\frac{\varphi,\ \lambda v.e' \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[t/x,\ \mathsf{swap}\,(\varphi,\ \lambda v.e')/k] \hookrightarrow a}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} (\lambda x.\lambda^{l} k.e)\, t\, \lambda v.e' \hookrightarrow a}$$

$$\frac{\varphi' \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[t/x] \hookrightarrow a}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} (\lambda x.\lambda\star.e)\, t\, (\mathsf{swap}\ \varphi') \hookrightarrow a}$$

$$\frac{\varphi' \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} e[t/x,\ \mathsf{swap}\ \varphi'/k] \hookrightarrow a}{\varphi \vdash^{\mathsf{1CExp}}_{\mathsf{1cc}} (\lambda x.\lambda^{\mathsf{l}} k.e)\, t\, (\mathsf{swap}\ \varphi') \hookrightarrow a}$$

$$\frac{\varphi \vdash_{\text{1cc}}^{\text{1CExp}} e[t/v] \hookrightarrow a}{\varphi \vdash_{\text{1cc}}^{\text{1CExp}} (\lambda v.e)\, t \hookrightarrow a}$$

$$\frac{}{\bullet \vdash_{\text{1cc}}^{\text{1CExp}} \star\, t \hookrightarrow t}$$

$$\frac{\varphi \vdash_{\text{1cc}}^{\text{1CExp}} e[t/v] \hookrightarrow a}{\varphi,\ \lambda v.e \vdash_{\text{1cc}}^{\text{1CExp}} \star\, t \hookrightarrow a}$$

$$\frac{}{\varphi \vdash^{\text{1CExp}}_{\text{1cc}} \text{swap} \bullet \mathrm{t} \hookrightarrow \mathrm{t}}$$

$$\frac{\varphi' \vdash^{\text{1CExp}}_{\text{1cc}} e[\mathrm{t}/v] \hookrightarrow a}{\varphi \vdash^{\text{1CExp}}_{\text{1cc}} \text{swap}\,(\varphi',\, \lambda v.e)\,\mathrm{t} \hookrightarrow a}$$

Theorem 11 (Simulation)

1. $\vdash^{\mathsf{CProg}}_{\mathit{std}}\ \mathrm{p} \hookrightarrow \mathrm{a}$ *if and only if*

$\vdash^{\mathsf{1CProg}}_{\mathit{1cc}}\ [\![\mathrm{p}]\!]^{\mathsf{CProg}}_{\mathsf{ann}} \hookrightarrow [\![\mathrm{a}]\!]^{\mathsf{Answer}}_{\mathsf{ann}}$.

2. $\vdash^{\mathsf{CExp}}_{\mathit{std}}\ [\![\mathrm{e}]\!]^{\mathsf{CExp}}_{\mathsf{ann}}\mathrm{k}\{\varphi\}_{\mathsf{1}} \hookrightarrow \mathrm{a}$ *if and only if*

$\varphi \vdash^{\mathsf{1CExp}}_{\mathit{1cc}}\ [\![\mathrm{e}]\!]^{\mathsf{CExp}}_{\mathsf{ann}}\mathrm{k} \hookrightarrow [\![\mathrm{a}]\!]^{\mathsf{Answer}}_{\mathsf{ann}}$,

for some k*.*

Proof: Similar to the proof of Theorem 8. $\square$

Practical assessment

- Programs without call/cc do not pay for them.
- Programs with call/cc pay for them – a lot.

Alternative

Segment the stack.

$$\Phi\,;\varphi \vdash^{\mathsf{CExp}}_{\mathsf{1cc}'} e \hookrightarrow a$$

On-the-fly garbage collection of unshared continuations

$$\mathsf{b}\,;\Phi\,;\varphi \vdash^{\mathsf{CExp}}_{\mathsf{1cc}''} e \hookrightarrow \mathsf{a}$$

Plan

1. Implementation strategies $\surd$

2. CPS programs $\surd$

3. Second-class continuations $\surd$

4. First-class continuations $\surd$

5. Conclusion and perspectives

Conclusion

We have formalized

some implementation strategies

for first-class continuations.

Further work

(with Frank Pfenning and Belmina Dzafic)

Stackability not only for continuation identifiers

but also for parameters of continuations

(i.e., intermediate results).

Formalized in Elf

for second-class continuations.

Further work (Frank Pfenning)

Intuitionistic Non-Commutative Linear Logic

Reference

“Formalizing Implementation Strategies

for First-Class Continuations”

ESOP 2000

BRICS RS-99-51 (extended version)

General observation

Continuations provide:

- A source of inspiration.
- A fruitful playground.
- Not necessarily a final solution,
  more like a stepping stone.

Why continuations?

A (typed) λ-encoding of control.

Their success is the success of λ-calculi.

And to finish with a pun

The lambda-calculus has many applications.

(Mayer Goldberg)