

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 can be implemented with a control stack.
(“Recursive programming”, Dijkstra, 1960.)
- In programs with call/cc, control 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 ✓
2. CPS programs
3. Second-class continuations
4. First-class continuations
5. Conclusion and perspectives

BNF of direct-style terms

$p \in D\text{Prog} ::= e$

$e \in D\text{Exp} ::= e_0 e_1 \mid t$

$t \in D\text{Triv} ::= \ell \mid x \mid \lambda x.e$

$\ell \in \text{Lit}$

$x \in \text{Ide}$

BNF of CPS terms

$p \in \text{CProg} ::= \lambda k.e$

$e \in \text{CExp} ::= t_0 t_1 c \mid c t$

$t \in \text{CTriv} ::= \ell \mid x \mid v \mid \lambda x.\lambda k.e$

$c \in \text{Cont} ::= k \mid \lambda v.e$

$\ell \in \text{Lit}$

$x \in \text{Ide}$

$k \in \text{IdeC}$

$v \in \text{IdeV}$

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



Left-to-right, CBV CPS transformation

$$\llbracket e \rrbracket_{\text{cps}}^{\text{DProg}} = \lambda k. \llbracket e \rrbracket_{\text{cps}}^{\text{DExp}} k$$

$$\llbracket e_0 e_1 \rrbracket_{\text{cps}}^{\text{DExp}} c = \llbracket e_0 \rrbracket_{\text{cps}}^{\text{DExp}} \lambda v_0. \llbracket e_1 \rrbracket_{\text{cps}}^{\text{DExp}} \lambda v_1. v_0 v_1 c$$

$$\llbracket t \rrbracket_{\text{cps}}^{\text{DExp}} c = c \llbracket t \rrbracket_{\text{cps}}^{\text{DTriv}}$$

$$\llbracket \ell \rrbracket_{\text{cps}}^{\text{DTriv}} = \ell$$

$$\llbracket x \rrbracket_{\text{cps}}^{\text{DTriv}} = x$$

$$\llbracket \lambda x. e \rrbracket_{\text{cps}}^{\text{DTriv}} = \lambda x. \lambda k. \llbracket e \rrbracket_{\text{cps}}^{\text{DExp}} k$$

Our standard abstract machine (1/2)

$$\vdash_{\text{std}}^{\text{CProg}} p \hookrightarrow a$$

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

$$\vdash_{\text{std}}^{\text{CExp}} e \hookrightarrow a$$

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.
- $a \in \text{Answer} ::= \ell \mid \lambda x. \lambda k. e \mid \text{error}$

$$\frac{\vdash_{\text{std}}^{\text{CExp}} e[\mathbf{k}_{\text{init}}/\mathbf{k}] \hookrightarrow \mathbf{a}}{\vdash_{\text{std}}^{\text{CProg}} \lambda \mathbf{k}. e \hookrightarrow \mathbf{a}}$$

$$\frac{}{\vdash_{\text{std}}^{\text{CExp}} \ell \ t \ c \hookrightarrow \text{error}} \quad \frac{\vdash_{\text{std}}^{\text{CExp}} e[t/x, c/k] \hookrightarrow \mathbf{a}}{\vdash_{\text{std}}^{\text{CExp}} (\lambda x. \lambda k. e) \ t \ c \hookrightarrow \mathbf{a}}$$

$$\frac{\vdash_{\text{std}}^{\text{CExp}} e[t/v] \hookrightarrow \mathbf{a}}{\vdash_{\text{std}}^{\text{CExp}} (\lambda v. e) \ t \hookrightarrow \mathbf{a}}$$

$$\frac{}{\vdash_{\text{std}}^{\text{CExp}} \mathbf{k}_{\text{init}} \ t \hookrightarrow t}$$

The rest of this lecture.

We focus on the continuation identifiers k .

Plan

1. Implementation strategies ✓
2. CPS programs ✓
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.

$$k \Vdash_{2cc}^{CExp} e$$

$$\frac{k \notin \text{FC}(t_0) \quad k \notin \text{FC}(t_1) \quad k \models_{2\text{cc}}^{\text{Cont}} c}{k \models_{2\text{cc}}^{\text{CExp}} t_0 t_1 c}$$

$$\frac{k \models_{2\text{cc}}^{\text{Cont}} c \quad k \notin \text{FC}(t)}{k \models_{2\text{cc}}^{\text{CExp}} c t}$$

$$\frac{k \models_{2\text{cc}}^{\text{CExp}} e}{k \models_{2\text{cc}}^{\text{Cont}} \lambda v. e}$$

$$\frac{}{k \models_{2\text{cc}}^{\text{Cont}} k}$$

Second-class position

Second-class continuations

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

More generally

All continuation identifiers
denote second-class continuations.

$$\models_{2cc^*}^{CProg} p$$

$$\frac{k \Vdash_{2cc^*}^{\text{CExp}} e}{\Vdash_{2cc^*}^{\text{CProg}} \lambda k.e}$$

$$\frac{\Vdash_{2cc^*}^{\text{CTriv}} t_0 \quad \Vdash_{2cc^*}^{\text{CTriv}} t_1 \quad k \Vdash_{2cc^*}^{\text{Cont}} c}{k \Vdash_{2cc^*}^{\text{CExp}} t_0 t_1 c}$$

$$\frac{k \Vdash_{2cc^*}^{\text{Cont}} c \quad \Vdash_{2cc^*}^{\text{CTriv}} t}{k \Vdash_{2cc^*}^{\text{CExp}} c t}$$

$$\overline{\vDash_{2cc^*}^{CTriv} \ell}$$

$$\overline{\vDash_{2cc^*}^{CTriv} \chi}$$

$$\overline{\vDash_{2cc^*}^{CTriv} \nu}$$

$$\frac{k \vDash_{2cc^*}^{CExp} e}{\vDash_{2cc^*}^{CTriv} \lambda x. \lambda k. e}$$

$$\frac{k \vDash_{2cc^*}^{CExp} e}{k \vDash_{2cc^*}^{Cont} \lambda \nu. e}$$

$$\overline{k \vDash_{2cc^*}^{Cont} k}$$

2Cont-validity

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

The CPS transformation yields 2Cont-valid programs

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

$$\models_{2cc^*}^{\text{CProg}} \llbracket p \rrbracket_{\text{cps}}^{\text{DProg}}.$$

Proof: Straightforward induction on direct-style programs. □

2Cont-validity under β -reduction

Theorem 5

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

$p \in 2CProg ::= \lambda\star.e$

$e \in 2CExp ::= t_0 t_1 c \mid c t$

$t \in 2CTriv ::= \ell \mid x \mid v \mid \lambda x.\lambda\star.e$

$c \in 2Cont ::= \star \mid \lambda v.e$

$\ell \in Lit$

$x \in Ide$

$\star \in Token$

$v \in IdeV$

Mappings between CPS and 2CPS

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

Notation: $|\cdot|_2: \text{CPS} \rightarrow \text{2CPS}$

A stack machine

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

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

Two judgments

$$\vdash_{2cc}^{2CProg} p \hookrightarrow a$$

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

$$\varphi \vdash_{2cc}^{2CExp} e \hookrightarrow a$$

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

$$\frac{\bullet \vdash_{2cc}^{2CExp} e \hookrightarrow a}{\vdash_{2cc}^{2CProg} \lambda\star.e \hookrightarrow a} \qquad \frac{}{\varphi \vdash_{2cc}^{2CExp} \ell t c \hookrightarrow \text{error}}$$

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

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

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

$$\bullet \vdash_{2cc}^{2CExp} \star t \hookrightarrow t$$

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

Equivalence

Idea:

show that for each abstract machine,
the computations
are in bijective correspondence.

Key technique

A “stack substitution” $\left\{ \begin{array}{l} e\{\varphi\}_2 \\ c\{\varphi\}_2 \end{array} \right.$

Stack substitution

$$|t_0 t_1 c|_{\{\varphi\}} = t_0 t_1 (|c|_{\{\varphi\}})$$

$$|c t|_{\{\varphi\}} = (|c|_{\{\varphi\}}) t$$

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

$$*{\bullet}|_{\{\varphi\}} = k_{\text{init}}$$

$$*{\varphi, |\lambda v. e|_{\{\varphi\}}}|_{\{\varphi\}} = \lambda v. (|e|_{\{\varphi\}})$$

Lemma 6

1. For any $e \in \text{CExp}$ satisfying $k \Vdash_{2\text{cc}^*}^{\text{CExp}} e$ for some k and for any stack of 2Cont continuations φ ,

$$k_{\text{init}} \Vdash_{2\text{cc}^*}^{\text{CExp}} | e |_{\{ \varphi \}}.$$

2. For any $c \in \text{Cont}$ satisfying $k \Vdash_{2\text{cc}^*}^{\text{Cont}} e$ for some k and for any stack of 2Cont continuations φ ,

$$k_{\text{init}} \Vdash_{2\text{cc}^*}^{\text{Cont}} | c |_{\{ \varphi \}}.$$

Proof: By structural induction. □

Control-stack substitution

Lemma 7

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

2. For any $e \in 2\text{CExp}$, for any $t' \in \text{CTriv}$ satisfying

$\models_{2cc^*}^{CTriv} t'$, for any identifier i in Ide or in $IdeV$, and
for any stack of $2Cont$ continuations φ ,
 $e[\![t' \]\!/\!i]\{\varphi\} = e\{\varphi\}[t'/i]$.

Proof: By structural induction. □

Theorem 8 (Simulation)

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

$$\vdash_{std}^{CProg} p \hookrightarrow a \text{ if and only if } \vdash_{2cc}^{2CProg} |p|_2 \hookrightarrow |a|_2.$$

2. For any CPS expression e satisfying $k \models_{2cc^*}^{CExp} e$

for some k and for any stack of 2Cont

continuations φ , $\vdash_{std}^{CExp} |e|_2\{\varphi\}_2 \hookrightarrow a$ if and only

if $\varphi \vdash_{2cc}^{2CExp} |e|_2 \hookrightarrow |a|_2$.

Proof

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

Let us see the case of tail calls.

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

where \mathcal{E}_1 names the derivation ending in

$$\varphi \vdash_{2cc}^{2CExp} e[t/x] \hookrightarrow a.$$

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

$$\vdash_{\text{std}}^{\text{CExp}} e[t/x]\{\varphi\}_2 \stackrel{\mathcal{E}'_1}{\hookrightarrow} a'$$

such that $a = | a' |_2$.

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

- there exists a CPS expression e' satisfying

$$k \Vdash_{2cc^*}^{CExp} e' \text{ for some } k, \text{ and}$$

- there exists a CPS trivial expression t' satisfying

$$\Vdash_{2cc^*}^{CTriv} t'$$

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

By Lemma 7,

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

because t' has no free k

and φ has no free x .

By inference,

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

Now by definition of stack substitution,

$$\begin{aligned} & (\lambda x. \lambda k. e') t' (\star\{\varphi\}_2) \\ &= | (\lambda x. \lambda k. e) t k' |_2 \{\varphi\}_2, \text{ for some } k'. \end{aligned}$$

In other words, there exists a derivation

$$\frac{\mathcal{E}'_1 \quad \vdash_{\text{std}}^{\text{CExp}} | e[t/x] |_2 \{ \varphi \}_2 \hookrightarrow a'}{\vdash_{\text{std}}^{\text{CExp}} | (\lambda x. \lambda k. e) t k' |_2 \{ \varphi \}_2 \hookrightarrow a'}$$

which is what we wanted to show. □

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 ✓
2. CPS programs ✓
3. Second-class continuations ✓
4. First-class continuations
5. Conclusion and perspectives

call/cc

$$\llbracket \text{call/cc } e \rrbracket_{\text{cps}}^{\text{DExp}} c$$

$$= \llbracket e \rrbracket_{\text{cps}}^{\text{DExp}} \lambda f. f (\lambda x. \lambda k. c \ x) \ c$$

where f , x , and k are fresh.

Key observation

Continuation identifiers now occur
“out of order”.

$$k \models_{1cc}^{CExp} e$$

$$\frac{k \in \text{FC}(t_0)}{k \models_{1\text{cc}}^{\text{CExp}} t_0 t_1 c} \quad \frac{k \in \text{FC}(t_1)}{k \models_{1\text{cc}}^{\text{CExp}} t_0 t_1 c} \quad \frac{k \models_{1\text{cc}}^{\text{Cont}} c}{k \models_{1\text{cc}}^{\text{CExp}} t_0 t_1 c}$$

$$\frac{k \models_{1\text{cc}}^{\text{Cont}} c}{k \models_{1\text{cc}}^{\text{CExp}} c t} \quad \frac{k \in \text{FC}(t)}{k \models_{1\text{cc}}^{\text{CExp}} c t}$$

$$\frac{k \models_{1\text{cc}}^{\text{CExp}} e}{k \models_{1\text{cc}}^{\text{Cont}} \lambda v. e}$$

First-class position

First-class continuations

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

“Sub-closure” property

Example:

$$\lambda k. (\lambda x. \lambda k'. k x) \ell k$$

$$\beta$$
$$\lambda k. k \ell$$

Key idea

Single out

continuation identifiers in first-class position

BNF for 1CPS programs

$p \in \text{CProg} ::= \lambda\star.e \mid \lambda^1k.e$

$e \in \text{CExp} ::= t_0 t_1 c \mid c t$

$t \in \text{CTriv} ::= \ell \mid x \mid v \mid \lambda x.\lambda\star.e \mid \lambda x.\lambda^1k.e$

$c \in \text{Cont} ::= \lambda v.e \mid \star \mid k$

$\ell \in \text{Lit}$

$x \in \text{Ide}$

$\star \in \text{Token}$

$k \in \text{IdeC}$

$v \in \text{IdeV}$

From CPS to 1CPS (1/4)

$$\llbracket t_0 \ t_1 \ c \rrbracket_{\text{ann}}^{\text{CExp}} \mathbf{k} = \llbracket t_0 \rrbracket_{\text{ann}}^{\text{CTriv}} \llbracket t_1 \rrbracket_{\text{ann}}^{\text{CTriv}} (\llbracket c \rrbracket_{\text{ann}}^{\text{Cont}} \mathbf{k})$$

$$\llbracket c \ t \rrbracket_{\text{ann}}^{\text{CExp}} \mathbf{k} = (\llbracket c \rrbracket_{\text{ann}}^{\text{Cont}} \mathbf{k}) \llbracket t \rrbracket_{\text{ann}}^{\text{CTriv}}$$

$$\llbracket \ell \rrbracket_{\text{ann}}^{\text{CTriv}} = \ell$$

$$\llbracket x \rrbracket_{\text{ann}}^{\text{CTriv}} = x$$

$$\llbracket v \rrbracket_{\text{ann}}^{\text{CTriv}} = v$$

$$\llbracket \lambda v. e \rrbracket_{\text{ann}}^{\text{Cont}} \mathbf{k} = \lambda v. \llbracket e \rrbracket_{\text{ann}}^{\text{CExp}} \mathbf{k}$$

From CPS to 1CPS (2/4)

$$\begin{aligned} \llbracket \lambda k. e \rrbracket_{\text{ann}}^{\text{CProg}} = & \\ \begin{cases} \lambda^! k. \llbracket e \rrbracket_{\text{ann}}^{\text{CExp}} k & \text{if } k \Vdash_{1\text{cc}}^{\text{CExp}} e \\ \lambda^* . \llbracket e \rrbracket_{\text{ann}}^{\text{CExp}} k & \text{otherwise} \end{cases} \end{aligned}$$

From CPS to 1CPS (3/4)

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

From CPS to 1CPS (4/4)

$$\llbracket k' \rrbracket_{\text{ann}}^{\text{Cont}} k = \begin{cases} \star & \text{if } k = k' \\ k' & \text{otherwise} \end{cases}$$

From 1CPS to CPS

Straightforward – just replace \star by a fresh k .

Lemma 10 (Inverseness)

$$\llbracket \cdot \rrbracket_{\text{unann}}^{1\text{CProg}} \circ \llbracket \cdot \rrbracket_{\text{ann}}^{\text{CProg}} = \textit{identity}_{\alpha}.$$

A stack machine for 1CPS programs

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

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

Stack substitution

$$t_0 \ t_1 \ c\{\varphi\}_h = \llbracket t_0 \rrbracket_{\text{unann}}^{1\text{CTriv}} \llbracket t_1 \rrbracket_{\text{unann}}^{1\text{CTriv}} (c\{\varphi\}_h)$$

$$c \ t\{\varphi\}_h = (c\{\varphi\}_h) \llbracket t \rrbracket_{\text{unann}}^{1\text{CTriv}}$$

$$(\lambda v. e)\{\varphi\}_h = \lambda v. (e\{\varphi\}_h)$$

$$\star\{\bullet\}_h = k_{\text{init}}$$

$$\star\{\varphi, \lambda v. e\}_h = \lambda v. (e\{\varphi\}_h)$$

$$k\{\varphi\}_h = k$$

$$(\text{swap } \varphi')\{\varphi\}_h = \star\{\varphi'\}_h$$

The abstract machine

$$\vdash_{1cc}^{1CProg} p \hookrightarrow a$$

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

$$\varphi \vdash_{1cc}^{1CExp} e \hookrightarrow a$$

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

The domain of answers: 1Answer

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

$$\frac{\bullet \vdash_{1cc}^{1CExp} e \hookrightarrow a}{\vdash_{1cc}^{1CProg} \lambda^*.e \hookrightarrow a}$$

$$\frac{\bullet \vdash_{1cc}^{1CExp} e[\text{swap } \bullet / k] \hookrightarrow a}{\vdash_{1cc}^{1CProg} \lambda^! k.e \hookrightarrow a}$$

$$\frac{}{\varphi \vdash_{1cc}^{1CExp} \ell \text{ t } c \hookrightarrow \text{error}}$$

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

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

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

$$\frac{\varphi, \lambda v. e' \vdash_{1\text{cc}}^{1\text{CExp}} e[t/x, \text{swap } (\varphi, \lambda v. e')/k] \hookrightarrow a}{\varphi \vdash_{1\text{cc}}^{1\text{CExp}} (\lambda x. \lambda! k. e) t \lambda v. e' \hookrightarrow a}$$

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

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

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

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

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

$$\frac{}{\varphi \vdash_{1cc}^{1CExp} \text{swap} \bullet t \hookrightarrow t}$$

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

Theorem 11 (Simulation)

1. $\vdash_{std}^{CProg} p \hookrightarrow a$ if and only if

$$\vdash_{1cc}^{1CProg} \llbracket p \rrbracket_{ann}^{CProg} \hookrightarrow \llbracket a \rrbracket_{ann}^{Answer}.$$

2. $\vdash_{std}^{CExp} \llbracket e \rrbracket_{ann}^{CExp} k \{ \varphi \} \hookrightarrow a$ if and only if

$$\varphi \vdash_{1cc}^{1CExp} \llbracket e \rrbracket_{ann}^{CExp} k \hookrightarrow \llbracket a \rrbracket_{ann}^{Answer},$$

for some k .

Proof: Similar to the proof of Theorem 8. □

Practical assessment

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

Alternative

Segment the stack.

$$\Phi ; \varphi \vdash_{1cc'}^{\text{CExp}} e \hookrightarrow a$$

On-the-fly garbage collection of unshared continuations

$$b ; \Phi ; \varphi \vdash_{1cc''}^{CExp} e \hookrightarrow a$$

Plan

1. Implementation strategies ✓
2. CPS programs ✓
3. Second-class continuations ✓
4. First-class continuations ✓
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 -calculus.

And to finish with a pun

The lambda-calculus has many applications.

(Mayer Goldberg)