A Type-Theoretic Foundation of Continuations and Prompts

Amr Sabry (Indiana University)

with **Zena M. Ariola** (University of Oregon)
and **Hugo Herbelin** (INRIA-Futurs)

19 September 2004

Supported by National Science Foundation grant number CCR-0204389

Outline

- Introduction: semantics of continuations and prompts
- Review: continuations and classical logic
- The essence of prompts
- Typing of continuations and prompts
- continuations and prompts and ?? logic

Basic Control operators

- $\mathcal{A}\ M$ aborts everything, returns M to the top-level

$$\#E[\mathcal{A}\ M] \mapsto \#M$$

- $\mathcal{C}\ (\lambda c.M)$ binds the entire continuation to c
- capturing the continuation aborts; calling the continuation aborts

$$\#E[\mathcal{C}\ M] \mapsto \#(M\ (\lambda x.\mathcal{A}\ E[x]))$$

- $\mathcal{C}$ can express $\mathcal{A}$: we can focus on $\mathcal{C}$

- Fairly well-understood from types and logic perspective

Minimal logic and λ-calculus

$$\begin{array}{rcl} A, B & ::= & X \mid A \to B \\ M, N & ::= & x \mid \lambda x.M \mid MN \end{array}$$

$$\frac{}{\Gamma, x\colon A \vdash x\colon A}$$

$$\frac{\Gamma, x\colon A \vdash M\colon B}{\Gamma \vdash \lambda x.M\colon A \to B}$$

$$\frac{\Gamma \vdash M\colon A \to B \qquad \Gamma \vdash N\colon A}{\Gamma \vdash MN\colon B}$$

Control operators (Griffin POPL'90)

- Type constant $\bot$ corresponding to formula *False*
  $\neg A$ abbreviates $A \to \bot$
- $\mathcal{A}$ corresponds to Ex Falso Quodlibet

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathcal{A}\,(M) : A}$$

- $\mathcal{C}$ corresponds to "proof by contradiction" (Double Negation):

$$\frac{\Gamma \vdash M : \neg\neg A}{\Gamma \vdash \mathcal{C}(M) : A}$$

- $\mathcal{A}$ "implements" intuitionistic logic
- $\mathcal{C}$ "implements" classical logic

Delimited control operators

- Fundamental idea is to allow # anywhere: (Felleisen 1988)

$$\#\ (1 + \#\ (2 + (\mathcal{C}\ (\lambda c.\ 3 + \#(4 + (c\ (\#\ (c\ 5))))))))$$

- In $(\mathcal{C}\ (\lambda c. \ldots))$, continuation c extends to the closest # only

- In $(\mathcal{A}\ M)$, expression M is returned to the closest # prompt

- $\mathcal{C}$ and # are equivalent to *shift* (abbreviated $\mathcal{S}$) and *reset* (Danvy and Filinski 1990)

In the presence of prompts . . .

- no canonical type system
  (some proposed type systems incomparable)

- no confluence results

- no strong normalization results
  (and known counterexamples in some cases)

- no expressiveness results
  (but Thielecke *et al.* have studied expressiveness of continuations, exceptions, and state)

A menagerie of control operators

- In addition to $\mathcal{A}$, $\mathcal{C}$, $\#$, *shift*, *reset*, we also have *callcc*, $shift_n$, $reset_n$, *spawn*, $\mathcal{F}$, *cupto*, *splitter*, etc.
- Design space: continuation-passing style (CPS), iterated CPS, hierarchies of control operators and delimiters, fixed set of prompt names or dynamic generation prompt names, delimiters captured in continuations or not, lifetime of delimiters, etc.

Our goal

Understand and compare these control operators using type-theory and logic.

Continuations and Classical Logic

The legacy of Griffin

- Continuations are functions from values to $\bot$
- The top-level is of type $\bot$

But ... $\mathcal{C}(\lambda c.c1) \mapsto 1$
the top-level seems to be an int !

And there are no closed terms of type $\bot$:
there are no well-typed programs !

Griffin adopts a convention to solve this problem

Formalizing Griffin's convention

Murthy (1992), Ariola and Herbelin (2003)

- A constant tp denotes the top-level continuation:

$$\mathcal{A}\ 6 = \mathcal{C}(\lambda\_.\mathsf{tp}\ 6) \qquad \mathcal{A}\ \mathsf{true} = \mathcal{C}(\lambda\_.\mathsf{tp}\ \mathsf{true})$$

- Implicit references to the top-level become explicit:

$$\mathcal{C}(\lambda c.\ 1 + c\ 0) \qquad \text{becomes} \qquad \mathcal{C}(\lambda c.\ \mathsf{tp}\ (1 + c\ 0))$$

- Judgments $\Gamma \vdash M : A; T$ keep track of the type T of the top-level (not restricted to $\bot$):

$$\vdash \mathcal{C}(\lambda\_.\mathsf{tp}\ 6) :\ A; \mathsf{int} \qquad \vdash \mathcal{C}(\lambda\_.\mathsf{tp}\ \mathsf{true}) :\ A; \mathsf{bool}$$

The $\lambda^{\rightarrow}_{\mathcal{C}\mathsf{tp}}$ calculus: jumps

- Special constant tp denoting the top-level continuation
- Special category J of jumps
- A symbol $\perp\!\!\!\perp$ denoting the absence of a type
- Continuations are not functions

$$\frac{\Gamma, k\colon A \to \perp\!\!\!\perp \;\vdash\; M : A; T}{\Gamma, k\colon A \to \perp\!\!\!\perp \;\vdash\; k\ M : \perp\!\!\!\perp; T} \to^{k}_{e}$$

$$\frac{\Gamma \;\vdash\; M : T; T}{\Gamma \;\vdash\; \mathsf{tp}\ M : \perp\!\!\!\perp; T} \to^{\mathsf{tp}}_{e}$$

Manipulating continuations in $\lambda^{\rightarrow}_{\mathcal{C}\mathsf{tp}}$

- One can jump if and only if one captures the continuation
- Two common patterns:

$$\mathcal{A}^{-} M \triangleq \mathcal{C}(\lambda\_.\mathsf{tp}\ M) \qquad \text{(Abbrev. 1)}$$

$$\mathit{throw}\ k\ M \triangleq \mathcal{C}(\lambda\_.k\ M) \qquad \text{(Abbrev. 2)}$$

- Type rule:

$$\frac{\Gamma, k{:}\, A \to \bot\!\!\!\bot \ \vdash\ J : \bot\!\!\!\bot; T}{\Gamma \ \vdash\ \mathcal{C}(\lambda k.J) : A; T}\ RAA$$

- Isomorphic to $\lambda\mu$ (Parigot 1992) with tp

Examples in $\lambda^{\rightarrow}_{\mathcal{C}\mathsf{tp}}$

$$\vdash (\mathcal{A}^{-}\ 5\ ==\ \texttt{"Hello"})\ :\ \mathsf{bool};\ \mathsf{int}$$

accurately predicts that the expression might return a bool or jump to the top-level with an int.

$$\vdash \lambda x.(x\ +\ \mathcal{A}^{-}\ \texttt{"Hello"})\ :\ \mathsf{int} \rightarrow \mathsf{int};\ \mathsf{string}$$

How to interpret the "top-level type" string ?

The calculus $\lambda^{\rightarrow}_{\mathcal{C}\mathbf{tp}}$ has ...

- simpler reduction rules
  (no encoding of continuations as functions)

- which are confluent

- and normalizable when simply typed

What we have so far . . .

A calculus $\lambda^{\rightarrow}_{\mathcal{C}\mathsf{tp}}$ for understanding continuations

- with better properties than Griffin's original formulation

The Essence of Prompts

Adding prompts

- A prompt is a top-level for the duration of the subexpression
- Control actions always refer to the dynamically closest prompt.
- An analysis (in the paper) shows that the dynamic aspect of the prompt is all that it contributes
- to model prompts, all we need is to change the constant `tp` to a dynamically-bound variable $\widehat{tp}$.

Forget about prompts: $\lambda^{\rightarrow}_{\mathcal{C}\widehat{tp}}$

- We can express $\#\ M$ as $\mathcal{C}(\lambda\widehat{tp}.\ \widehat{tp}\ M)$
- Semantics is just like the standard semantics for $\lambda^{\rightarrow}_{\mathcal{C}\mathsf{tp}}$ but

  – allow capture of $\widehat{tp}$ in substitution:

$$\begin{aligned} & (\lambda y.\mathcal{C}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y)))\ \hat{[}(\lambda\_.\mathcal{C}(\lambda\_.\widehat{tp}\ y))/x\hat{]} \\ = \ & \lambda y'.\mathcal{C}(\lambda\widehat{tp}.\ \widehat{tp}\ ((\lambda\_.\mathcal{C}(\lambda\_.\ \widehat{tp}\ y))\ y')) \end{aligned}$$

  – allow $\widehat{tp}$ to escape its scope:

$$\mathcal{C}(\lambda\widehat{tp}.\ \widehat{tp}\ (\lambda\_.\mathcal{C}(\lambda\_.\widehat{tp}\ y))) \rightarrow \lambda\_.\mathcal{C}(\lambda\_.\widehat{tp}\ y)$$

Important result

This is correct !

- Semantics of original control operators and prompts is preserved

What we have so far . . .

A calculus $\lambda^{\rightarrow}_{\mathcal{C}\widehat{tp}}$ for understanding continuations and prompts:

- has better properties than Griffin's original formulation
- a prompt is normal control + dynamic scope $(\mathcal{C}\lambda\widehat{tp}\ldots)$

Typing Continuations and Dynamic Scope

Typing dynamically scoped entities

- Type-and-effect system

$$\frac{}{\Gamma, x\colon A \vdash x\colon A; T}\, Ax$$

$$\frac{\Gamma, x\colon A \vdash M\colon B; T}{\Gamma \vdash \lambda x.M\colon A \to_T B; T'} \to_i$$

$$\frac{\Gamma \vdash M\colon A \to_T B; T \quad \Gamma \vdash M'\colon A; T}{\Gamma \vdash M\,M'\colon B; T} \to_e$$

- Can be extended to a sound type system for $\lambda^{\to}_{\mathcal{C}\widehat{tp}}$
- This type system is too restrictive !

Dynamic scope as environment-passing

- A good formal definition (Moreau 1998)
- Every expression (and hence every continuation) is passed the environment
- In our case: environment consists of the current top-level continuation
- Input type to continuation is the return type of the expression:

$$\frac{\Gamma, k : A \rightarrow \perp\!\!\!\perp \vdash J : \perp\!\!\!\perp ; T}{\Gamma \vdash \mathcal{C}(\lambda k.J) : A; T}\, RAA$$

- Every expression must return an environment
- The environment is really a store !

Dynamic scope as exceptions

- Exceptions are jumps to a dynamically-determined handler
- In our case: using the top-level continuation jumps to the dynamically closest handler
- Interactions of exceptions and continuations from SML/NJ:
  - the usual callcc (not the interaction we want)
  - an operator capture that we want:

$$\{H, E[\text{capture } M]\} \quad \mapsto \quad \{H, E[M\ (\lambda x.\mathcal{A}\ E[x])]\}$$

$$\text{NOT } \{H, E[M\ (\lambda x.\mathcal{A}\ \{H, E[x]\})]\}$$

- capture and exceptions can define state ! (Thielecke 2001)

Dynamic scope + control ⇒ state

$$\frac{}{\Gamma, x\colon A; T \vdash x\colon A; T}\, Ax \qquad \frac{\Gamma, x\colon A; U \vdash M\colon B; T}{\Gamma; T' \vdash \lambda x.M\colon A\ {}_{U}\!\rightarrow_{T} B; T'}\, \rightarrow_i$$

$$\frac{\Gamma; U_1 \vdash M\colon A\ {}_{U_2}\!\rightarrow_{T_1} B; T_2 \qquad \Gamma; T_1 \vdash N\colon A; U_1}{\Gamma; U_2 \vdash MN\colon B; T_2}\, \rightarrow_e$$

Still sound but more expressive: $\#\ (1 + \mathcal{S}(\lambda c.\ 2\ ==\ c\ 3))$

Olivier Danvy and Andrzej Filinski might now say:

> We could have told you so in 1989 !

Or more clearly ...

In classical logic, the following formulae are all equivalent:

$$A \wedge \neg T \rightarrow B \qquad (\widehat{tp} \text{ as an environment})$$

$$= A \rightarrow B \vee T \qquad (\widehat{tp} \text{ as an exception})$$

$$= A \wedge \neg T \rightarrow B \wedge \neg T \qquad (\widehat{tp} \text{ as a state})$$

A logic with effect annotations?

Let's just get rid of all these annotations and
have a simple elegant type system.

- Possible
- Type system is nicer
- Can be implemented in a standard type system
  (SML, Ocaml, etc)
- But we lose strong normalization (if the top-level type is not atomic)
- and we start accepting terms that may refer to non-existing prompts.

What we have so far . . .

A calculus $\lambda^{\rightarrow}_{\mathcal{C}\widehat{tp}}$ for understanding continuations and prompts:

- has better properties than Griffin's original formulation
- a prompt is normal control + dynamic scope $(\mathcal{C}\lambda\widehat{tp}\ldots)$
- two not-very-pretty type-and-effect systems

Subtractive Logic

Interpreting effects

- If we don't want effects, translate using monads
- Map $(k\colon A \to_U \perp\!\!\!\perp)^*$ to $(k\colon A^* \wedge \neg U^* \to \perp\!\!\!\perp)$
- LHS: restrict contexts where k can be called by requiring a top-level continuation of type U
- RHS: pass a top-level continuation of type U to k
  A continuation to the continuation is a . . . metacontinuation

Olivier Danvy and Andrzej Filinski might now say:

> We could have told you so in 1990 !

The dual of implication

What is the type $A \wedge \neg B$?

In classical (but not intuitionistic) logic, we have:

$$\begin{aligned} \neg(A \to B) &= \neg(\neg A \vee B) \\ &= \neg\neg A \wedge \neg B \\ &= A \wedge \neg B \end{aligned}$$

It is the negation of implication

This suggests we can use the dual of implication: subtraction $A - B$ (introduced at least by 1974)

To subtract or not to subtract

$A - B$ is the same as $A \wedge \neg B$ but better:

- More "abstract." A continuation k has type $A - T \to \bot\!\!\!\bot$
  - ⋆ Because $A - T$ iff $A \wedge \neg T$ continuations need metacontinuations (Danvy and Filinski 1990)
  - ⋆ Because $A - T$ iff $\neg(\neg T \to \neg A)$ delimited continuations are also the "difference of two continuations" (Queinnec and Moreau 1994)
- Makes sense even in the absence of first-class continuations (Crolard 2004)

$$\begin{aligned} even &: \mathsf{int} \to \mathsf{bool} \\ evenEncoding &: \neg\mathsf{int} \vee \mathsf{bool} \end{aligned}$$

The $\lambda_{\mathcal{C}}^{\rightarrow -}$ calculus

$$\frac{}{\Gamma, x\colon A \vdash x\colon A}\, Ax \qquad \frac{\Gamma, x\colon A \vdash M\colon B}{\Gamma \vdash \lambda x.M\colon A \to B} \to_i$$

$$\frac{\Gamma \vdash M\colon A \to B \quad \Gamma \vdash M'\colon A}{\Gamma \vdash M\,M'\colon B} \to_e$$

$$\frac{\Gamma, k\colon A \to \bot\!\!\!\bot \vdash J\colon \bot\!\!\!\bot}{\Gamma \vdash \mathcal{C}(\lambda k.J)\colon A}\, RAA \qquad \frac{\Gamma, k\colon A \to \bot\!\!\!\bot \vdash M\colon A}{\Gamma, k\colon A \to \bot\!\!\!\bot \vdash k\ M\colon \bot\!\!\!\bot} \to_e^k$$

$$\frac{\Gamma \vdash M\colon A \quad \Gamma, \Box\colon B \vdash k\ E\colon \bot\!\!\!\bot}{\Gamma \vdash (M, k\ E)\colon A - B} -_i$$

$$\frac{\Gamma \vdash M\colon A - B \qquad \Gamma, x\colon A, k\colon B \to \bot\!\!\!\bot \vdash M'\colon C}{\Gamma \vdash \mathbf{let}\ (x, k) = M\ \mathbf{in}\ M'\colon C} -_e$$

Type system is also sound

How to use $\lambda_{\mathcal{C}}^{\rightarrow -}$

- As a source language if you want
- By embedding in it other languages with control operators
  - embeddings arrange for managing the prompt (or top-level continuation or metacontinuation) using subtraction
  - $(y\ (\#\ x))$ translates as
    $y\ ((\mathcal{C}(\lambda tp'.tp'\ x), tp\ \Box))$
    The current top-level continuation is "saved" using the subtraction introduction and a new top-level continuation is used for receiving the value of x.

Important results

This is correct !

- Types and semantics of original control operators and prompts is preserved
- Well-typed terms are strongly normalizing
- Consistent with known CPS transformations of the control operators

Conclusions

Remember . . .

- Do not sweep the top-level continuation under the rug (tp)
- A continuation is not a function ($\perp\!\!\!\perp$)
- A little bit of dynamic scope can help (#)
- Regular continuations augmented with one dynamically-scoped continuation can express all other effects including state
- Type-and-effect systems or monads are essential sometimes
- Logic helps !
- Subtractive logic is the right way to think about advanced control operators (some of them at least)

Future work

. . . skipping . . .

$\mathcal{A}$