A Functional Correspondence between Evaluators and Abstract Machines

Mads Sig Ager, Dariusz Biernacki,

Olivier Danvy, and Jan Midtgaard

BRICS, University of Aarhus, Denmark

BRICS

Nottingham, March 28, 2003

Our message

A correspondence between evaluators and abstract machines by

- closure conversion,
- CPS transformation, and
- defunctionalization.

Our results

- Krivine’s machine,
- the CEK machine and variants,
- the CLS machine,
- the SECD machine,
- the Categorical Abstract Machine,

and more.

My presentational problem

Continuations.

This talk

A new attempt to explain continuations.

BRICS

The Two Disguises of Continuations

Olivier Danvy and Lasse R. Nielsen
BRICS, University of Aarhus, Denmark
(danvy@brics.dk)

Nottingham, March 28, 2003

What are continuations?

Take 1: A functional representation

of the evaluation context.

What are continuations?

Take 2: A functional representation

of the rest of the computation.

Together

Take 1: A functional representation

of the evaluation context.

Take 2: A functional representation

of the rest of the computation.

Goal of this talk

To reconcile these two views:

- continuations as evaluation contexts, and
- continuations as the rest of the computation.

Tool: Syntactic Theories

A small-step operational semantics

with an explicit representation

of the evaluation context.

Ideal to specify control operators.

- Notion of term, notion of value.
- Evaluation contexts: decomposition, plugging.
- Unique-decomposition lemma

  (for deterministic languages).
- Redexes and reduction rules.
- One-step reduction:

  decompose, contract, and plug.

Syntactic theories: Quo Vadis?

- A small-step operational semantics with evaluation contexts.
- Origin: Felleisen's PhD thesis (1987).
- Largely used since.
- Practical challenges: evaluation contexts and unique-decomposition lemma.

A programming experiment (1/2)

Write a one-step reduction function:

$$exp \rightarrow val + exp$$

- Compositional, first order, direct style.
- Recursive descent:

  calls decompose, returns (re)construct.

A programming experiment (2/2)

- CPS-transform the one-step reducer:

$$exp \times (val + exp \rightarrow \alpha) \rightarrow \alpha$$

- Defunctionalize the continuation into:
  - a data type, and
  - an apply function.

And then a miracle happens

- The data type is that of evaluation contexts.

  (And in hindsight what else could it be?)

- The apply function is the plug function.

And then a miracle happens

- The data type is that of evaluation contexts.

  (And in hindsight what else could it be?)

- The apply function is the plug function.

So continuations do represent

evaluation contexts.

Byproducts

- Mechanically deriving a syntactic theory from a one-step reduction function.
- If the reduction function is compositional then the unique-decomposition lemma is guaranteed to hold.

Evaluation

Evaluation: The transitive closure

of one-step reduction.

$$(\mathtt{decompose}, \mathtt{contract}, \mathtt{plug})^*$$

Deforestation

Replace $\mathtt{decompose}, \mathtt{contract}, \mathtt{plug},$
$\mathtt{decompose}, \mathtt{contract}, \mathtt{plug},$
$\ldots$

by $\mathtt{decompose}, \mathtt{contract},$
$\mathtt{refocus}, \mathtt{contract},$
$\mathtt{refocus}, \mathtt{contract},$
$\ldots$

Reference

Olivier Danvy and Lasse R. Nielsen

“Syntactic Theories in Practice”

BRICS RS-02-04

And then another miracle happens

The data type of evaluation contexts and

the refocus function

are in defunctionalized form.

And then another miracle happens

The data type of evaluation contexts and

the refocus function

are in defunctionalized form.

The corresponding higher-order program

is a compositional evaluator in CPS.

And then another miracle happens

The data type of evaluation contexts and

the refocus function

are in defunctionalized form.

The corresponding higher-order program

is a compositional evaluator in CPS.

So continuations do represent

the rest of the computation.

Byproducts

- Mechanically deriving a syntactic theory from an evaluation function.
- Mechanically deriving an abstract machine from an evaluation function.

Syntactic theories for the λ-calculus

1. Pick your favorite monad.
2. Write the corresponding functional evaluator (by inlining the monad in an evaluator for the computational λ-calculus).
3. Mechanically construct the corresponding syntactic theory and its abstract machine.

Conclusions

- Continuations represent evaluation contexts for one-step reduction.
- Continuations represent the rest of the computation for evaluation.
- Syntactic theories can be constructed from one-step reduction functions and from evaluation functions.