...looooong title... as encountered in Normalization by Evalua

Olivier Danvy

(danvy@brics.dk)

IFIP WG 2.11 August

Plan

1. Reminder on type-directed partial eva
   and normalization by evaluation.
2. Overview of correspondences
   between operational semantics for ev
3. Application to strong normalization.

Type-directed partial evaluat

- Origin: binding-time coercions in offli
- Click: Thomas Streicher's talk at DAI
  on 'reduction-free normalizatio
- Type-directed reification and reflectio
  with let insertion.

Normalization by evaluatio

- An APPSEM workshop in 1998:

  the convergence of many interests.

- Further work: denotational, operatior

- One-pass CPS transformation:

  – Danvy & Filinski, LFP’90.

  – Millikin, TFP’05.

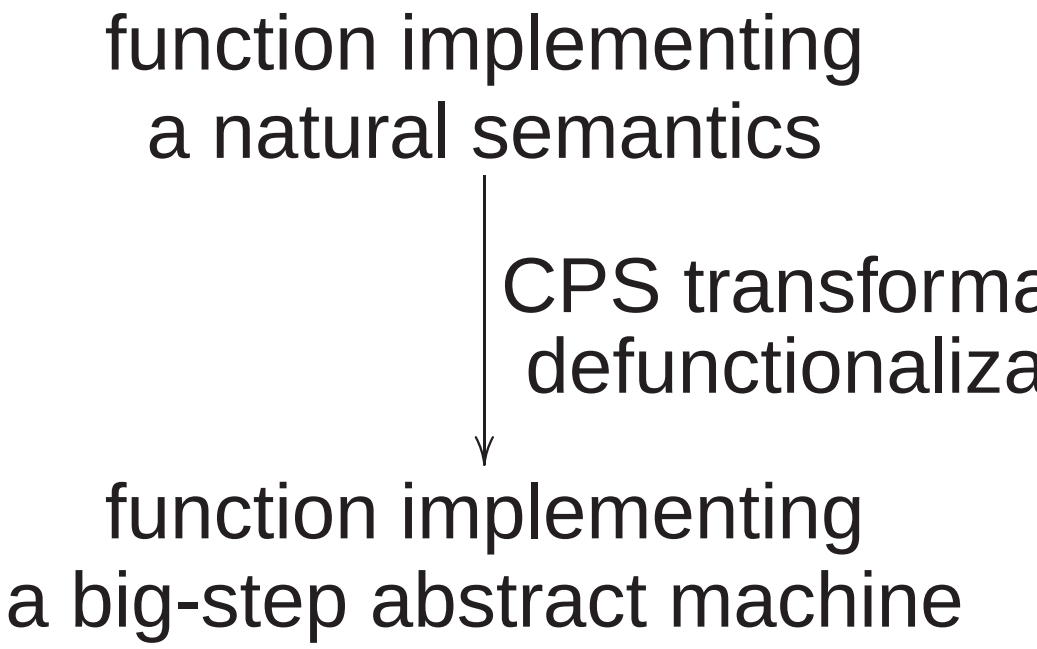
Plan

√ Reminder on type-directed partial eva
and normalization by evaluation.

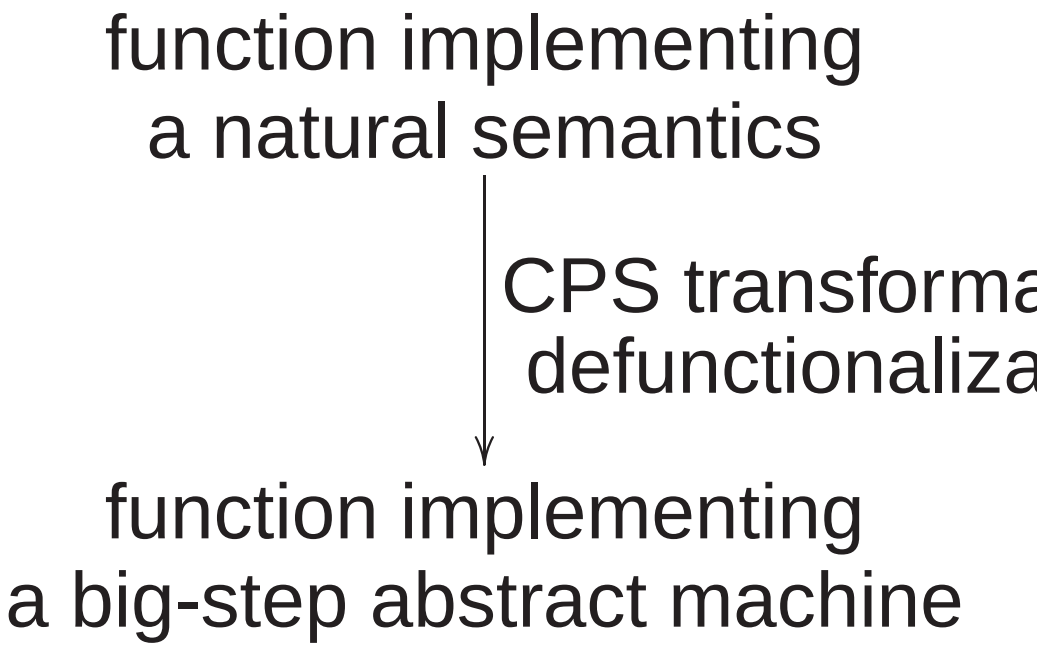
2. Overview of correspondences
between operational semantics for ev

3. Application to strong normalization.

Correspondence #1: big-step se

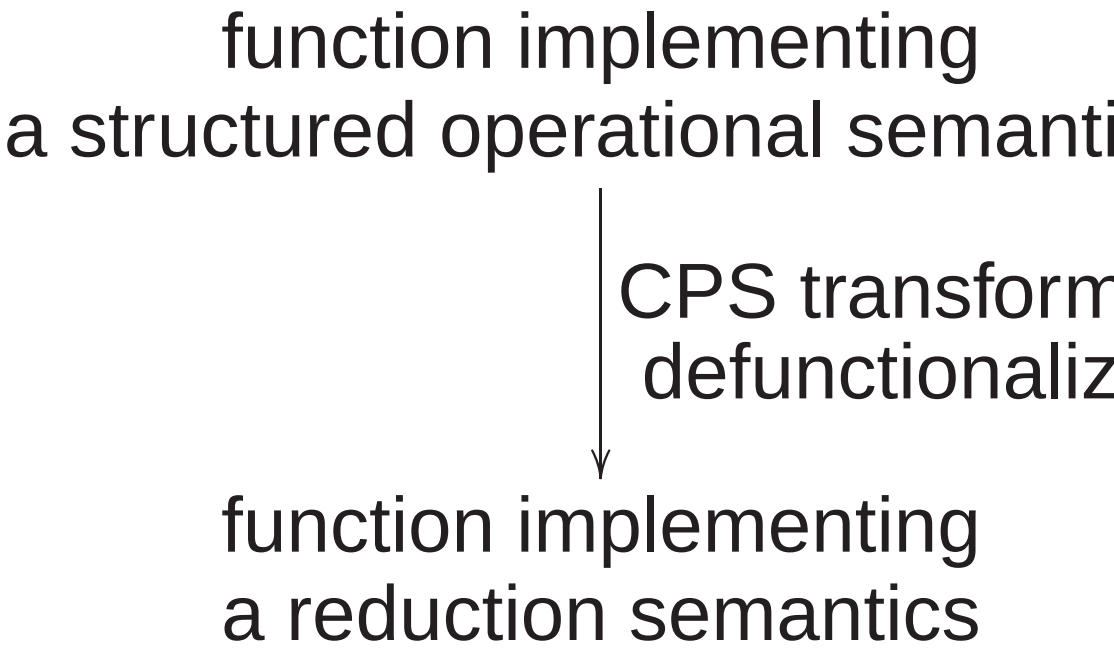


Correspondence #1: big-step se



Reync

Correspondence #2: small-step s



Correspondence #3:

function implementing
a reduction semantics

refocusing

function implementing
a small-step abstract machine

Danvy and Nielsen, BRICS

On the need for refocusing

One-step reduction function:

1. decompose a non-value term into

   a potential redex and a reduction con

2. contract the redex if it is an actual on

3. plug.

On the need for refocusing

One-step reduction function:

1. decompose a non-value term into
   a potential redex and a reduction con
2. contract the redex if it is an actual on
3. plug.

Evaluation: lather, rins

Refocusing

1. decompose
   - a value term into itself, and
   - a non-value term into
     a potential redex and a reduction c
2. contract the redex if it is an actual on
3.

Refocusing

1. decompose
   - a value term into itself, and
   - a non-value term into
     a potential redex and a reduction c
2. contract the redex if it is an actual on
3. continue the decomposition with the
   contractum and the current context.

Correspondence #3:

function implementing
a reduction semantics

refocusing

function implementing
a small-step abstract machine

Danvy and Nielsen, BRICS

Correspondence #4: abstract ma

function implementing
a small-step abstract machine

lightweight fus

function implementing
a big-step abstract machine

Ohori and Sasano,

Danvy and Millikin, BRICS

Recent progress (2007)

Formalization in Coq (Biernacka & Bie

Results (1/3): the pure cas

- reduction order $\leftrightarrow$ evaluation order
- explicit substitutions $\leftrightarrow$ environment
- pure notions of computation: SECD, CEK, ZINC, CLS, CAM, TIM, etc.

Results (2/3): notions of compu

Known and new variants of, e.g., the CP

- state
- control (exceptions, continuations, de continuations)
- stack inspection
- combinations of effects

Results (3/3): language parad

- logic
- object-oriented
- stochastic π-calculus
- imperative
- example: LILY

LILY (Rasmus Lerchedahl Pete

- The polymorphic linear lambda calcu
  recursion LILY, by Gavin Bierman, An
  Pitts, and Claudio Russo.

- From natural semantics to “a framest
  semantics that usually took a lengthy
  to show correct.”

- A calculus and a reduction strategy th
  correspond to this abstract machine.

Plan

√ Reminder on type-directed partial eva
and normalization by evaluation.

√ Overview of correspondences
between operational semantics for ev

3. Application to strong normalization.

Natural question:

what about strong normaliza

and normalization by evaluat

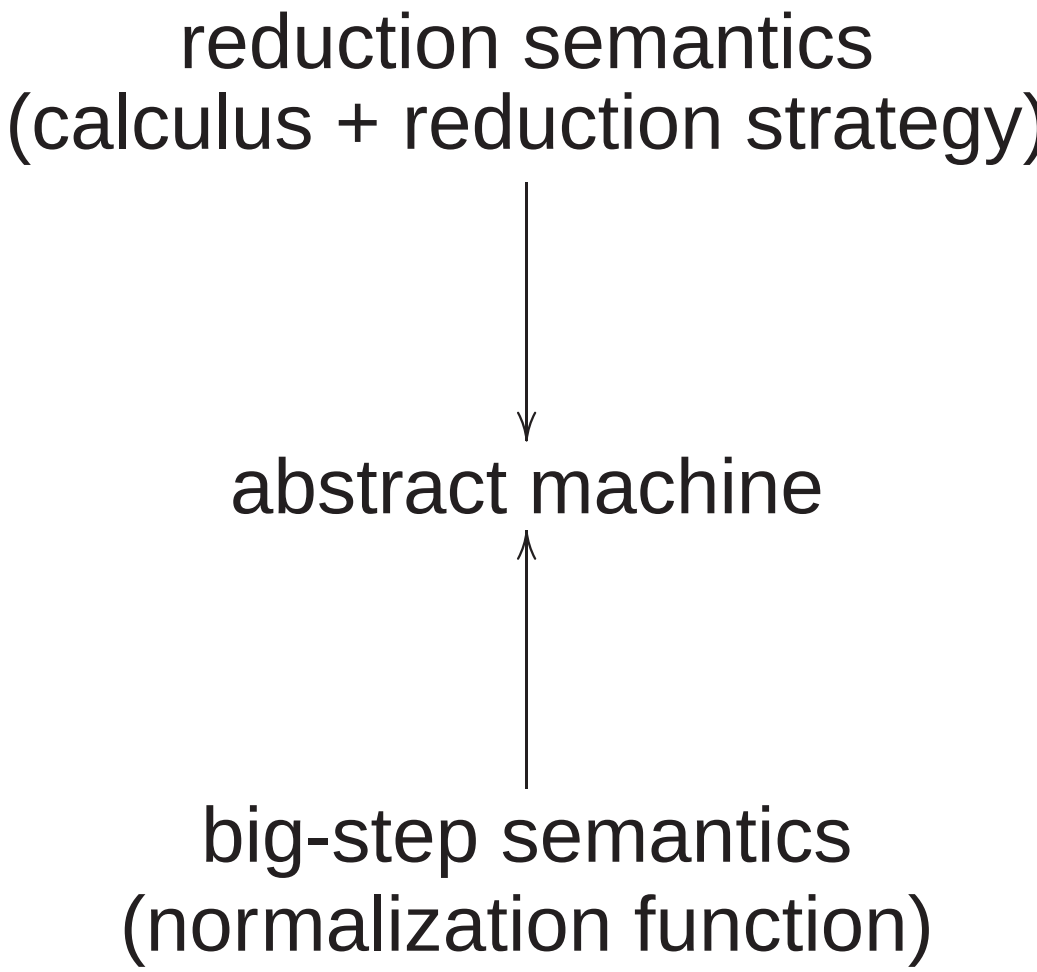
Natural question:

what about strong normaliza

and normalization by evaluat

Answer: it works too.

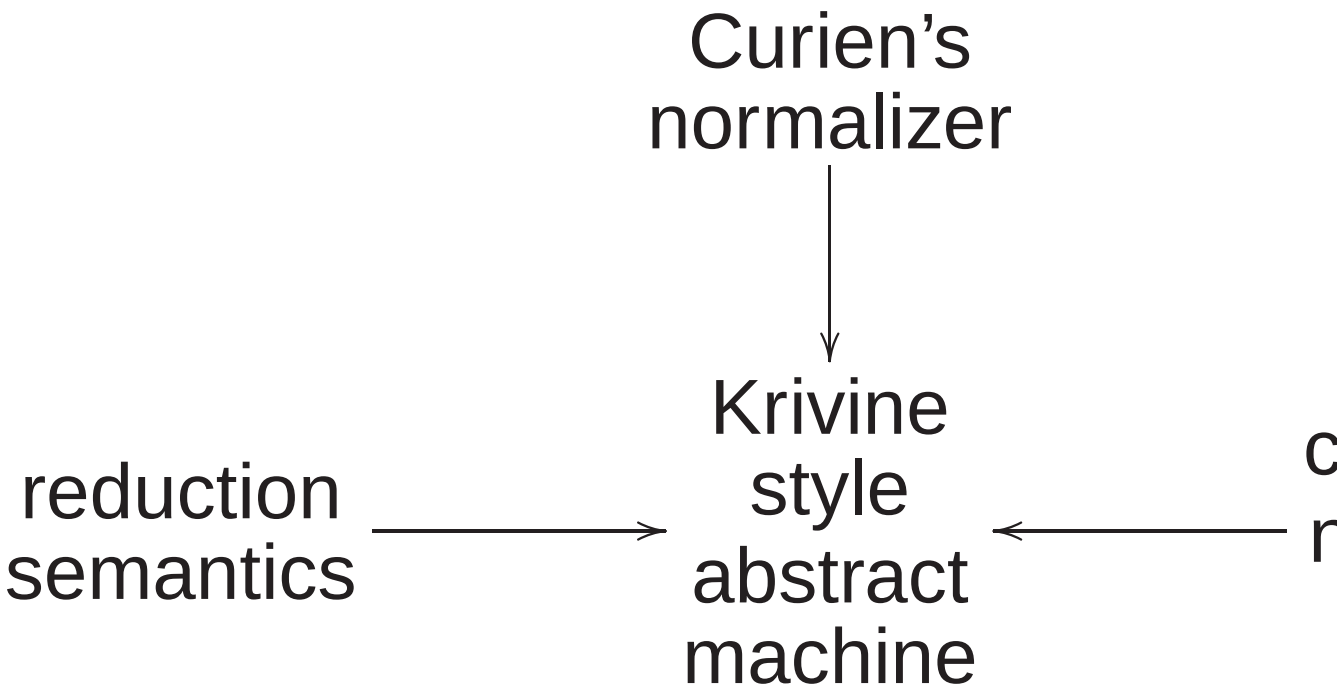
Joint work with Kevin Millikin & Johar


reduction semantics
(calculus + reduction strategy)
abstract machine
big-step semantics
(normalization function)

Our main result

Curien’s normalizer in “Categorical Com

Sequential Algorithms and Functional P



Abstract machines that fit

- McGowan's modified SECD machine (STOC'70)
- Crégut's KN machine (LFP'90)
- Lescanne's U-machine (POPL'94)
- Grégoire and Leroy's modified ZAM (
- Kluge's $\lambda\sigma$-machine ('05)

Make your own normalizer (1

Given a calculus and a strategy:

1. refocus the one-step normalizer into small-step abstract machine,
2. fuse the small-step abstract machine big-step abstract machine,

Make your own normalizer (2

3. refunctionalize the big-step abstract m
   into a normalization function, and

4. optionally, transform the normalizatio
   function into direct style.

More often than not(?), the normalizatio
function will be compositional.

Conclusion

What:

- The same elephant.

How:

- CPS.
- Defunctionalization.

Current work

- type soundness
- automated support for refactoring
- report the examples
- stress-test the whole thing

Thank you.