Examples of machines

Krivine’s machine

CEK

CLS

SECD

ZAM

JVM

VEC
What is the difference between an abstract machine and a virtual machine?

A virtual machine has an instruction set. An abstract machine does not.
This work: From Interpreter to Compiler and Virtual Machine

Related work

Lots of it
This talk

- A methodology.

- Two examples:
  - an evaluation function for the λ-calculus;
  - a normalization function for the λ-calculus.

Methodology

- Defunctionalization.

- Closure conversion.

- CPS transformation.

- Factorization.
Defunctionalization (Reynolds, 1972)

Who inhabits a function space?

Answer: instances of lambda abstractions.

Let us enumerate them
and represent a function space as a sum:

- function declaration: sum constructor;
- function application: case.

A simple example (1/2)

(* aux : (int -> int) -> int *)
fun aux f
    = f 100

fun main x
    = aux (fn n => n+1) +
      aux (fn n => x+n)
A simple example (2/2)

datatype lam = L0 | L1 of int
fun apply (L0, n) = n+1
    | apply (L1 x, n) = x+n
fun aux f = apply (f, 100)
fun main x
    = aux L0 + aux (L1 x)

Closure conversion

• Special case of defunctionalization:
  – only one summand;
  – apply function inlined.
CPS transformation

1. Names intermediate results.

2. Sequentializes their computation.

3. Introduces continuations.

Or again: transforms into monadic normal form and picks the continuation monad.

Or again: $\lambda$-counterpart of double negation.

Factorization

- $[\cdot]$ maps source program to endofunction.

- We factorize into composition of combinators and recursive calls to $[\cdot]$.

- We name combinators.
Example

Before factorization:

eval (LAM t) =
(fn (e, (c, e’) :: s)
 => eval t ((c, e’) :: e, s))

After factorization:

eval (LAM t) = (eval t) o grab
val grab = (fn (e, (c, e’) :: s)
 => ((c, e’) :: e, s))

Evaluation function, call-by-name
Evaluation function, call-by-name

- Closure convert.
- Call-by-name CPS transform.
- Defunctionalize continuations.
- Factorize.

Result: compiler and transition system.

... Krivine’s machine.
Evaluation function, call-by-value

• Closure convert.

• Call-by-value CPS transform.

• Defunctionalize continuations.

• Factorize.

Result: compiler and transition system.
Evaluation function, call-by-value

• Closure convert.
• Call-by-value CPS transform.
• Defunctionalize continuations.
• Factorize.

Result: compiler and transition system.

... CEK machine.

Assessment

Krivine: the simplest derivation we know of.

CEK: new reconstruction.
Assessment

Krivine: the simplest derivation we know of.

CEK: new reconstruction.

They are two sides of the same coin.

Normalization function, call-by-name
Normalization function, call-by-name

- Closure convert.
- Call-by-name CPS transform.
- Defunctionalize continuations.
- Factorize.

Result: compiler and transition system.

... a generalization of Krivine’s machine.
Normalization function, call-by-value

- Closure convert.
- Call-by-value CPS transform.
- Defunctionalize continuations.
- Factorize.

Result: compiler and transition system.
Normalization function, call-by-value

- Closure convert.
- Call-by-value CPS transform.
- Defunctionalize continuations.
- Factorize.

Result: compiler and transition system.

... a generalization of the CEK machine.

Assessment

- Two new virtual machines for strong normalization.
- Derived rather than discovered or invented.
Reynolds, 1972

Self-interpreters depend on the evaluation order of their meta-language.

Case in point: Krivine’s machine and CEK machine.
Reynolds, 1972

Self-interpreters depend on the evaluation order of their meta-language.

Case in point: Krivine’s machine and CEK machine.

Well, the same holds for normalization functions.

Thank you.