Modern SECD Machine

University of Calgary

October 31, 2014

Modern SECD Machine

1. An improvement over the SECD machine .
2. Doesn't have a dump . Stack doubles up as a dump.

Features of Modern SECD

1. a code pointer c (the instructions yet to be executed)
2. a stack s (holding intermediate result and pending function calls)
3. an environment e (giving values to the variables)
4. Modern SECD uses De-Bruijn's Indices to look into the environment.

λ-expression to De Bruijn's notation

Example

Lets consider $(\lambda x.xx)\ (\lambda x.x)$ and $(\lambda x\ .\lambda y\ .(xy))\ (\lambda x\ .x)\ (\lambda y\ .y)$

De Bruijn's notation for the above lambda expression are as below
$(\lambda x.xx)\ (\lambda x.x) \Rightarrow (\lambda.(\#1\ \#1))(\lambda.\ \#1)$

$(\lambda x\ .\lambda y\ .(xy))\ (\lambda x\ .x)\ (\lambda y\ .y) \Rightarrow (\lambda.\ \lambda.(\#2\ \#1))(\lambda.\#1)(\lambda.\#1)$

The De Bruijn Indices have been assumed to start from 1 here.

Instruction Set in Modern SECD Machine

Instruction Set

Const - push the constant on the stack

Clo(c) - push closure of code c with current environment

Ret - terminate current function, jump back to caller

App - pop function closure and argument, perform application

Add - pop two arguments from the top of the stack and add

Access(n) - push n-th field of the environment

Compilation Scheme of Modern SECD Machine

Compilation Scheme

Lambda Terms	Compilation
$[\![k]\!]$	$Const\ (k)$
$[\![\lambda a]\!]$	$Clo([\![a]\!] :\ Ret)$
$[\![MN]\!]$	$[\![N]\!] : [\![M]\!] : App$
$[\![a + b]\!]$	$[\![b]\!] : [\![a]\!] : Add$
$[\![\#n]\!]$	$Access\ (n)$

Example of Compilation

Compilation

Lets have a look at the compilation of source term:

$$(\lambda\ x\ .x\ +\ 1)\ 2\ \Rightarrow (\lambda\ .\#1\ +\ 1)\ 2$$

Compiled Code:

$$Const(2) : Clo[Const(1) : Access(1) : Add : Ret] : App$$

Machine Transitions in Modern SECD Machine

Before			After		
Code	Env	Stack	Code	Env	Stack
$Const\ k : c$	e	s	c	e	$k : s$
$Clo(c') : c$	e	s	c	e	$Clos(c', e) : s$
$App : c$	e	$Clos(c', e') : v : s$	c'	$v : e'$	$Clos(c, e) : s$
$Ret : c$	e	$v : Clos(c', e') : s$	c'	e'	$v : s$
$Add : c$	e	$n : m : s$	c	e	$(n + m) : s$
$Access(n); c$	e	s	c	e	$e(n) : s$

$Clos(c, e)$ denotes closure of code "c" with environment "e"

Start and End conditions in Modern SECD Machine

Initial State

$Code = c \ ; \ Environment = Nil \ ; \ Stack = Nil$

Final State

$Code = Nil \ ; \ Environment = Nil \ ; \ Stack = v$

Example of evaluation in Modern SECD Machine

Lets try evaluating $Const(2) : Clo[Const(1) : Access(1) : Add : Ret] : App$ which is compilation of $(\lambda\ x\ .\ x\ +\ 1)\ 2$

Let $c = Const(1) : Access(1) : Add : Ret$

Code	Env	Stack
$Const(2) : Clo(c) : App$	Nil	Nil
$Clo(c) : App$	Nil	$Const\ 2$
App	Nil	$Clos(c, Nil) : 2$
$Const(1) : Access(1) : Add : Ret$	2	$Clos(Nil, Nil)$
$Access(1) : Add : Ret$	2	$1 : Clos(Nil, Nil)$
$Add : Ret$	2	$2 : 1 : Clos(Nil, Nil)$
Ret	2	$3 : Clos(Nil, Nil)$
Nil	Nil	3