Homoiconic Lisp

Yosuke Fukuda
Graduate School of Informatics, Kyoto University

The goal of this work and the current status

- **Ultimate goal** To give a reasonably minimal condition of "program semantics" to define various program semantics
- **Current status** A Lisp implementation that can define a lot of programming language features for it, in the language itself

The notion of "Homoiconicity"

According to Alan Kay [Kay 1969], a language is called *homoiconic* if its internal and external representations are essentially the same.

Homoiconic Lisp (HLisp)

Feature

- HLisp is a small fragment of Scheme with a first class macro mechanism
- It can write a lot of constructs (e.g. if-expression, recursive function definition, and quasi-quotation) as user-defined programs
- It is based on a simple extension of *SECD machine* [Landin 1964] with a few primitives

GitHub Repository

The implementation of Homoiconic Lisp is available at `https://github.com/yf-fyf/hlisp`

Macro closure, a way to achieve homoiconicity

The notion of *macro closure* is a function closure to manipulate program, designed analogously to the closure of λ-abstraction

Intuition of its computation

$$((\textbf{macro}\ (x_1\ \cdots\ x_n)\ M)\ N_1\ \cdots\ N_n) \rightsquigarrow (\textbf{eval}\ M[x_1 := (\textbf{quote}\ N_1), \cdots, x_n := (\textbf{quote}\ N_n)])$$

Example The first program shows a usage of macro closure (macro abstraction), that corresponds to the second one

```
((macro (x y) x) (print 123) (print 456))
; => 123 (as a side-effect)
```

```
(eval ((lambda (x y) x) '(print 123) '(print 456)))
; => 123 (as a side-effect)
```

An extended SECD machine with macro closure, ESECD

Syntactic category

Constant	$c ::= f \mid \overline{f} \mid \textbf{quote} \mid \textbf{eval}$
Term	$M, N ::= c \mid x \mid \lambda x.M \mid \overline{\lambda} x.M \mid @M\ N \mid \lceil M \rceil$
SECD Value	$U ::= c \mid \langle(\lambda x.M), E\rangle \mid \langle(\overline{\lambda} x.M), E\rangle \mid \lceil M \rceil$
Stack	$S ::= \textbf{nil} \mid U :: S$
Environment	$E ::= \textbf{nil} \mid \langle x, U\rangle :: E$
Control string	$C ::= \textbf{ret} \mid \textbf{back} \mid \textbf{call} :: C \mid M :: C$
Dump	$D ::= \textbf{halt} \mid \langle S, E, C, D\rangle \mid \langle M, C, D\rangle$

Transition rules (with some omissions)

$$\langle S, E, c :: C, D\rangle \rightsquigarrow \langle c :: S, E, C, D\rangle$$
$$\langle S, E, x :: C, D\rangle \rightsquigarrow \langle U :: S, E, C, D\rangle \quad \text{if } U = \mathit{Lookup}_x(E)$$
$$\langle S, E, (\lambda x.M) :: C, D\rangle \rightsquigarrow \langle\langle(\lambda x.M), E\rangle :: S, E, C, D\rangle$$
$$\langle S, E, (\overline{\lambda} x.M) :: C, D\rangle \rightsquigarrow \langle\langle(\overline{\lambda} x.M), E\rangle :: S, E, C, D\rangle$$
$$\langle S, E, (@M\ N) :: C, D\rangle \rightsquigarrow \langle S, E, M :: \textbf{back}, \langle N, C, D\rangle\rangle$$
$$\langle\langle\lambda x.M', E'\rangle :: S, E, \textbf{back}, \langle N, C, D\rangle\rangle \rightsquigarrow \langle\langle\lambda x.M', E'\rangle :: S, E, N :: \textbf{call} :: C, D\rangle$$
$$\langle\langle\overline{\lambda} x.M', E'\rangle :: S, E, \textbf{back}, \langle N, C, D\rangle\rangle \rightsquigarrow \langle\lceil N \rceil :: \langle\overline{\lambda} x.M', E'\rangle :: S, E, \textbf{call} :: C, D\rangle$$
$$\langle U :: \langle(\lambda x.M'), E'\rangle :: S, E, \textbf{call} :: C, D\rangle \rightsquigarrow \langle \textbf{nil}, \langle x, U\rangle :: E', M' :: \textbf{ret}, \langle S, E, C, D\rangle\rangle$$
$$\langle\lceil N \rceil :: \langle(\overline{\lambda} x.M'), E'\rangle :: S, E, \textbf{call} :: C, D\rangle \rightsquigarrow \langle \textbf{nil}, \langle x, \lceil N \rceil\rangle :: E', M' :: \textbf{ret}, D'\rangle$$
$$\langle U :: \textbf{quote} :: S, E, \textbf{call} :: C, D\rangle \rightsquigarrow \langle U :: S, E, C, D\rangle$$
$$\langle\lceil M \rceil :: \textbf{eval} :: S, E, \textbf{call} :: C, D\rangle \rightsquigarrow \langle S, E, M :: C, D\rangle$$
$$\langle U :: \textbf{eval} :: S, E, \textbf{call} :: C, D\rangle \rightsquigarrow \langle U :: S, E, C, D\rangle \quad \text{if } U \text{ is not a code}$$
$$\langle U :: S, E, \textbf{ret}, \langle S', E', C', D'\rangle\rangle \rightsquigarrow \langle U :: S', E', C', D'\rangle$$

Example

$(\overline{\lambda} x.1)\,(\textbf{print}\,0) \Downarrow 1$ (without any printing effect)

$$\langle S, E, (@(\overline{\lambda} x.1)\,(\textbf{print}\,0)) :: \textbf{ret}, \textbf{halt}\rangle$$
$$\rightsquigarrow \langle S, E, (\overline{\lambda} x.1) :: \textbf{back}, \langle(\textbf{print}\,0), \textbf{ret}, \textbf{halt}\rangle\rangle$$
$$\rightsquigarrow \langle\langle(\overline{\lambda} x.1), E\rangle :: S, E, \textbf{back}, \langle(\textbf{print}\,0), \textbf{ret}, \textbf{halt}\rangle\rangle$$
$$\rightsquigarrow \langle\lceil \textbf{print}\,0 \rceil :: \langle(\overline{\lambda} x.1), E\rangle :: S, E, \textbf{call} :: \textbf{ret}, \textbf{halt}\rangle$$
$$\rightsquigarrow \langle \textbf{nil}, \langle x, \lceil \textbf{print}\,0 \rceil\rangle :: E, 1 :: \textbf{ret}, \langle \textbf{eval} :: S, E, \textbf{call} :: \textbf{ret}, \textbf{halt}\rangle\rangle$$
$$\rightsquigarrow \langle 1 :: \textbf{nil}, \langle x, \lceil \textbf{print}\,0 \rceil\rangle :: E, \textbf{ret}, \langle \textbf{eval} :: S, E, \textbf{call} :: \textbf{ret}, \textbf{halt}\rangle\rangle$$
$$\rightsquigarrow \langle 1 :: \textbf{eval} :: S, E, \textbf{call} :: \textbf{ret}, \textbf{halt}\rangle$$
$$\rightsquigarrow \langle 1 :: S, E, \textbf{ret}, \textbf{halt}\rangle$$

Property of ESCD

The "correctness" of ESCD is shown through a λ-calc. as in [Plotkin 1975]

Thm If M is a closed term, then TFAE:

- $M \Downarrow V$ in λ_H (Note: λ_H is an extension of λ-calc. with macro closure)
- $\langle \textbf{nil}, \textbf{nil}, M :: \textbf{ret}, \textbf{halt}\rangle \Downarrow U$ in ESECD

where V and U denote the "same" value (formally, defined as in [Plotkin 1975])

Future work

- Fill the gap between ESECD and HLisp, since the former has no primitive that produces side-effect
- Extend the theory and implementation to cover other evaluation strategies (Adding a hook operation to variable lookup may achieve this)