Functional modularity in the lambda calculus

Chung-chieh Shan
Cornell University

28 December 2011
FROM UNDERSTANDING COMPUTATION TO
UNDERSTANDING NEURAL CIRCUITRY

by

D. Marr and T. Poggio

Abstract: The CNS needs to be understood at four nearly independent levels of description: (1) that at which the nature of a computation is expressed; (2) that at which the algorithms that implement a computation are characterized; (3) that at which an algorithm is committed to particular mechanisms; and (4) that at which the mechanisms are realized in hardware. In general, the nature of a computation is determined by the problem to be solved, the mechanisms that are used depend upon the available hardware, and the particular algorithms chosen depend on the problem and on the available mechanisms. Examples are given of theories at each level.
On the Design and Development of Program Families

DAVID L. PARNAS

On the Design and Development of Program Families

One may consider a program development to be good, if the early decisions exclude only uninteresting, undesired, or unnecessary programs. The decisions which remove desired programs would be either postponed until a later stage or confined to a well delimited subset of the code. Objective criticism of a program’s structure would be based upon the fact that a decision or assumption which was likely to change has influenced too much of the code either because it was made too early in the development or because it was not confined to an information hiding module.

Clearly this is not the only criterion which one may use in evaluating program structures. Clarity (e.g., ease of understanding, ease of verification) is another quite relevant consideration. Although there is some reason to suspect that the two measures are not completely unrelated, there are no reasons to suspect that they conflict. For example, the “easy
Functional modularity

A module is a part of a description of a system

- Modularity should be invariant under physically entangled emulation with dye pack
- Modularity makes a theory more concise, comprehensible
- ‘Functional structure’ (Gallistel)/
  ‘Wirkungsgefüge’ (behavioral physiology)/source code

\[ a : \varepsilon \quad a : a \quad a : \varepsilon \]
\[ b : \varepsilon \quad b : b \quad b : \varepsilon \]
\[ \otimes \, a b b = \]
\[ a : \varepsilon \quad a : a \quad a : \varepsilon \]
\[ b : \varepsilon \quad b : b \quad b : \varepsilon \]
\[ a \quad b \quad a \quad b \quad a \]
Functional modularity

A module is a part of a description of a system

- Modularity should be invariant under physically entangled emulation with dye pack
- Modularity makes a theory more concise, comprehensible
- ‘Functional structure’ (Gallistel)/‘Wirkungsgefüge’ (behavioral physiology)/source code

Good decomposition helps reuse when environment changes

- Utterances need not re-conventionalize
- Organisms need not re-learn
- Species need not re-evolve
- Researchers need not re-discover
Lambda the ultimate

The essence of reuse: a module is a sub-expression. Binding. Higher-order abstractions.

Types classify terms. Polymorphism circumscribes information flow.

1. Expressions and interpretations in Abstract Categorial Grammar
2. Layers of monads for quantification and state
Lambda the ultimate

The essence of reuse: a module is a sub-expression. Binding. Higher-order abstractions.

Types classify terms. Polymorphism circumscribes information flow.

1. Expressions and interpretations in Abstract Categorial Grammar
2. Layers of monads for quantification and state
Abstract Categorial Grammar

\[ e = \lambda \langle \text{john, mary, like, } r_1, r_2 \rangle. \]
\[ r_1 \text{ john } (r_2 \text{ like mary}) \]
Abstract Categorial Grammar

\[
e = \lambda\langle \text{john, mary, like, } r_1, r_2 \rangle. \\
  r_1 \text{john} (r_2 \text{like mary})
\]

\[
\begin{align*}
\text{EN} &= \langle '\text{John}', \\
&\quad '\text{Mary}', \\
&\quad '\text{likes}', \\
&\quad \lambda s. \lambda v. s'v, \\
&\quad \lambda v. \lambda o. v'o \rangle \\
\text{Sem} &= \langle j', \\
&\quad m', \\
&\quad l', \\
&\quad \lambda s. \lambda v. vs, \\
&\quad \lambda v. \lambda o. vo \rangle
\end{align*}
\]

\[
e(\text{EN}) = '\text{John likes Mary}' \quad e(\text{Sem}) = lmj
\]
Abstract Categorial Grammar

\[ e = \lambda \langle \text{john}, \text{mary}, \text{like}, r_1, r_2 \rangle. \]

\[ r_1 \text{john} (r_2 \text{like mary}) \]

\[ \text{JA} = \langle \ldots \rangle \]

\[ \text{EN} = \langle '\text{John}', \text{‘Mary’}, \text{‘likes’}, \lambda s. \lambda v. s ^ ' v, \lambda v. \lambda o. v ^ ' o \rangle \]

\[ e(\text{JA}) = \text{‘ジョンさんはメリさんのことが好きだ’} \]

\[ e(\text{EN}) = \text{‘John likes Mary’} \]

\[ e(\text{Sem}) = \text{l}mjs \]
Interpretations
Interpretation transformers

\[
\text{Sem} = \langle j', m', l', \lambda s. \lambda v. vs, \lambda v. \lambda o. vo \rangle
\]
Semantics

Sem = \lambda \langle j, m, l, \@, \neg, \land, \ldots \rangle.
\langle j, m, l, \lambda s. \lambda v. \@ vs, \lambda v. \lambda o. \@ vo \rangle
**Interpretation transformers**

\[ \text{Sem} = \lambda \langle j, m, l, @, \neg, \land, \ldots \rangle. \]
\[ \langle j, m, l, \lambda s. \lambda v. @vs, \lambda v. \lambda o. @vo \rangle \]

\[ R = \langle j', m', l', \lambda f. \lambda x. fx, \ldots \rangle \]
\[ C = \langle 'j', 'm', 'l', \lambda f. \lambda x. f'^{('^{x}')}'), \ldots \rangle \]
\[ P = \lambda \langle j, m, l, @, \neg, \land, \ldots \rangle. \langle \ldots \rangle \]
Interpretation transformers

Semantics

Symantics

Lambda

Sem

R

C

P

Sem = \[ \lambda \langle j, m, l, @, \neg, \land, \ldots \rangle. \]
\[ \langle j, m, l, \lambda s. \lambda v. @vs, \lambda v. \lambda o. @vo \rangle \]

R = \[ \langle j', m', l', \lambda f. \lambda x. fx, \ldots \rangle \]

C = \[ \langle 'j', 'm', 'l', \lambda f. \lambda x. f \langle ^{\wedge}x^{\wedge} \rangle, \ldots \rangle \]

P = \[ \lambda \langle j, m, l, @, \neg, \land, \ldots \rangle. \langle \ldots \rangle \]
Interpretation transformers

\[ \lambda\langle\text{john, mary, like, } r_1, r_2, \text{ every, some, } r_4, r_5\rangle. \]
\[ \langle\text{john, mary, like, } r_1, r_2\rangle \]
Dynamic logic

Symantics

Sem

Lambda

R

C

P

Quantifier

States

JA

EN

JA

EN

EN
Dynamic logic

Symantics

Quantifier

States

Dynamics

Lambda

Sem

R

C

P

EN

JA
Dynamic logic
Expression transformers

Macros are maps from expressions to expressions.

\[ \lor = \lambda(j, m, l, @, \neg, \land, \ldots). \]
\[ \quad \lambda e_1. \lambda e_2. \neg(\land(\neg e_1)(\neg e_2)) \]

Also for analyzing unquotation.

*Ralph warned that he has ‘long suspected that [Ortcutt] is a spy’.*

*Ralph warned that he has ‘long suspected that [Ortcutt’s beach alias] is a spy’.*
Lambda the ultimate

The essence of reuse: a module is a sub-expression. Binding. Higher-order abstractions.

Types classify terms. Polymorphism circumscribes information flow.

1. Expressions and interpretations in Abstract Categorial Grammar

2. Layers of monads for quantification and state
Generalization to the worst case

extensional

john = j
mary = m
\( r_1 = \lambda s. \lambda v. vs \)
\( r_2 = \lambda v. \lambda o. vo \)

possible worlds

john = \( \lambda w. j \)
mary = \( \lambda w. m \)
\( r_1 = \lambda s. \lambda v. \lambda w. vw(sw) \)
\( r_2 = \lambda v. \lambda o. \lambda w. vw(ow) \)

alternative sets

john = \{ j \}
mary = \{ m \}
\( r_1 = \lambda s. \lambda v. \{ fx | x \in s, f \in v \} \)
\( r_2 = \lambda v. \lambda o. \{ fx | f \in v, x \in o \} \)
Generalization to the worst case

extensional
  john = j
  mary = m
  \( r_1 = \lambda s. \lambda v. vs \)
  \( r_2 = \lambda v. \lambda o. vo \)

state
  john = \( \lambda i. \langle i, j \rangle \)
  mary = \( \lambda i. \langle i, m \rangle \)
  \( r_1 = \lambda s. \lambda v. \lambda i. \langle i'', f x \rangle \) where \( \langle i', x \rangle = si \) \( \langle i'', f \rangle = vi' \)
  \( r_2 = \lambda v. \lambda o. \lambda i. \langle i'', f x \rangle \) where \( \langle i', f \rangle = vi \) \( \langle i'', x \rangle = oi' \)

continuations
  john = \( \lambda c. cj \)
  mary = \( \lambda c. cm \)
  \( r_1 = \lambda s. \lambda v. \lambda c. s\lambda x. v\lambda f. c(f x) \)
  \( r_2 = \lambda v. \lambda o. \lambda c. v\lambda f. o\lambda x. c(f x) \)
Generalization of the worst case

Three components of a monad:

\[ M, \quad \eta : \alpha \to M\alpha, \quad \star : M\alpha \to (\alpha \to M\beta) \to M\beta \]

\[
\begin{align*}
\text{john} &= \eta(j) \\
\text{mary} &= \eta(m) \\
\text{r}_1 &= \lambda s. \lambda v. s \star \lambda x. v \star \lambda f. \eta(fx) \\
\text{r}_2 &= \lambda v. \lambda o. v \star \lambda f. o \star \lambda x. \eta(fx)
\end{align*}
\]

- **extensional**
  \[ M\alpha = \alpha \]
  \[ \eta(\alpha) = \alpha \]
  \[ m \star q = qm \]

- **possible worlds**
  \[ M\alpha = s \to \alpha \]
  \[ \eta(\alpha) = \lambda w. \alpha \]
  \[ m \star q = \lambda w. q(mw)w \]

- **alternative sets**
  \[ M\alpha = \alpha \to t \]
  \[ \eta(\alpha) = \{a\} \]
  \[ m \star q = \bigcup_{a \in m} qa \]

- **state**
  \[ M\alpha = i \to (i \times \alpha) \]
  \[ \eta(\alpha) = \lambda i. \langle i, \alpha \rangle \]
  \[ m \star q = \lambda i. qai' \text{ where } \langle i', a \rangle = mi \]

- **continuations**
  \[ M\alpha = (\alpha \to r) \to r \]
  \[ \eta(\alpha) = \lambda c. ca \]
  \[ m \star q = \lambda c. m\lambda a. qac \]
Generalization of the worst case

Three components of a monad:

\[ M, \quad \eta : \alpha \to M\alpha, \quad \star : M\alpha \to (\alpha \to M\beta) \to M\beta \]

extensional

\[ M\alpha = \alpha \]
\[ \eta(a) = a \]
\[ m \star q = qm \]

possible worlds

\[ M\alpha = s \to \alpha \]
\[ \eta(a) = \lambda w. a \]
\[ m \star q = \lambda w. q(mw)w \]

alternative sets

\[ M\alpha = \alpha \to t \]
\[ \eta(a) = \{a\} \]
\[ m \star q = \bigcup_{a \in m} qa \]

state

\[ M\alpha = i \to (i \times \alpha) \]
\[ \eta(a) = \lambda i. \langle i, a \rangle \]
\[ m \star q = \lambda i. qai' \text{ where } \langle i', a \rangle = mi \]

continuations

\[ M\alpha = (\alpha \to r) \to r \]
\[ \eta(a) = \lambda c. ca \]
\[ m \star q = \lambda c. m\lambda a. qac \]
Generalization of the worst case

Three components of a monad:

\[ M, \quad \eta : \alpha \to M\alpha, \quad \star : M\alpha \to (\alpha \to M\beta) \to M\beta \]

**extensional**

\[ M\alpha = \alpha \]
\[ \eta(a) = a \]
\[ m \star q = q m \]

**possible worlds**

\[ M\alpha = s \to \alpha \]
\[ \eta(a) = \lambda w. a \]
\[ m \star q = \lambda w. q(mw)w \]

**alternative sets**

\[ M\alpha = \alpha \to t \]
\[ \eta(a) = \{a\} \]
\[ m \star q = \bigcup_{a \in m} qa \]

**continuations**

\[ M\alpha = (\alpha \to r) \to r \]
\[ \eta(a) = \lambda c. ca \]
\[ m \star q = \lambda c. m\lambda a. qac \]

john = \eta(j)
mary = \eta(m)
\[ r_1 = \lambda s. \lambda v. s \star \lambda x. v \star \lambda f. \eta(fx) \]
\[ r_2 = \lambda v. \lambda o. v \star \lambda f. o \star \lambda x. \eta(fx) \]
her = \lambda i. \langle i, i(5) \rangle

state

\[ M\alpha = i \to (i \times \alpha) \]
\[ \eta(a) = \lambda i. \langle i, a \rangle \]
\[ m \star q = \lambda i. qai' \text{ where } \langle i', a \rangle = mi \]
Generalization of the worst case

Three components of a monad:

\[
\mathbb{M}, \quad \eta : \alpha \to \mathbb{M}\alpha, \quad \star : \mathbb{M}\alpha \to (\alpha \to \mathbb{M}\beta) \to \mathbb{M}\beta
\]

john = \eta(j)
mary = \eta(m)

\[
r_1 = \lambda s. \lambda v. s \star \lambda x. v \star \lambda f. \eta(f x)
\]
\[
r_2 = \lambda v. \lambda o. v \star \lambda f. o \star \lambda x. \eta(f x)
\]

john and mary = \lambda c. cj \land cm

extensional

\[
\mathbb{M}\alpha = \alpha
\]
\[
\eta(\alpha) = a
\]
\[
m \star q = qm
\]

possible worlds

\[
\mathbb{M}\alpha = s \to \alpha
\]
\[
\eta(\alpha) = \lambda w. a
\]
\[
m \star q = \lambda w. q(mw)w
\]

alternative sets

\[
\mathbb{M}\alpha = \alpha \to t
\]
\[
\eta(\alpha) = \{a\}
\]
\[
m \star q = \bigcup_{a \in m} qa
\]

state

\[
\mathbb{M}\alpha = i \to (i \times \alpha)
\]
\[
\eta(\alpha) = \lambda i. \langle i, a \rangle
\]
\[
m \star q = \lambda i. qai' \text{ where } \langle i', a \rangle = mi
\]

continuations

\[
\mathbb{M}\alpha = (\alpha \to r) \to r
\]
\[
\eta(\alpha) = \lambda c. ca
\]
\[
m \star q = \lambda c. m\lambda a. qac
\]
Semantic cartography

**state monad**

\[ M\alpha = i \to (i \times \alpha) \]

\[ \eta(a) = \cdots \]

\[ m \times q = \cdots \]

\[ \text{her} = \ell(\lambda i. \langle i, i(5) \rangle) \]

**continuation monad transformer**

\[ M'\alpha = (\alpha \to Mr) \to Mr \]

\[ \eta'(a) = \cdots \]

\[ m \times' q = \cdots \]

\[ \ell(m) = \lambda c. m \times c \]

\[ j\&m = \lambda c. cj \times \lambda x. cm \times \lambda y. \eta(x \land y) \]
Semantic cartography

state monad

\[ M \alpha = i \rightarrow (i \times \alpha) \]
\[ \eta(a) = \cdots \]
\[ m \star q = \cdots \]
\[ \text{her} = \ell(\lambda i. \langle i, i(5) \rangle) \]

continuation monad transformer

\[ M' \alpha = (\alpha \rightarrow Mr) \rightarrow Mr \]
\[ \eta'(a) = \cdots \]
\[ m \star' q = \cdots \]
\[ \ell(m) = \lambda c. m \star c \]
\[ j&m = \lambda c. cj \star \lambda x. cm \star \lambda y. \eta(x \land y) \]
Semantic cartography

**state monad**

\[ M\alpha = i \rightarrow (i \times \alpha) \]

\[ \eta(a) = \cdots \]

\[ m \star q = \cdots \]

\[ \text{her} = \ell (\lambda i. \langle i, i(5) \rangle) \]

**continuation monad transformer**

\[ M'\alpha = (\alpha \rightarrow Mr) \rightarrow Mr \]

\[ \eta'(a) = \cdots \]

\[ m \star' q = \cdots \]

\[ \ell(m) = \lambda c. m \star c \]

\[ \text{j&m} = \lambda c. cj \star \lambda x. cm \star \lambda y. \eta(x \land y) \]
Semantic cartography

**state monad**

\[ M\alpha = i \to i \times \alpha \]
\[ \eta(a) = \ldots \]
\[ m \star q = \ldots \]
\[ \text{her} = \ell(\lambda i. \langle i, i(5) \rangle) \]

**continuation monad transformer**

\[ M'\alpha = (\alpha \to i \to i \times r) \to i \to i \times r \]
\[ \eta'(a) = \ldots \]
\[ m \star' q = \ldots \]
\[ \ell(m) = \lambda c. m \star c \]
\[ j&m = \lambda c. cj \star \lambda x. cm \star \lambda y. \eta(x \land y) \]

\[ M'\alpha = (\alpha \to i \to i \times r) \to i \to (i \times r) \]
\[ \text{her} = \lambda c. \lambda i. c(i(5))i \]
\[ j&m = \lambda c. \lambda i. \langle i'', x \land y \rangle \quad \text{where} \quad \langle i', x \rangle = cj i \quad \langle i'', y \rangle = cm i' \]
Semantic cartography

state monad

\[ M\alpha = i \to (i \times \alpha) \]
\[ \eta(a) = \cdots \]
\[ m \star q = \cdots \]
\[ \text{her} = \ell(\lambda i. \langle i, i(5) \rangle) \]

continuation monad transformer

\[ M'\alpha = (\alpha \to M\xi) \to M\xi \]
\[ \eta'(a) = \cdots \]
\[ m \star' q = \cdots \]
\[ \ell(m) = \lambda c. m \star c \]
\[ j&m = \lambda c. cj \star \lambda x. cm \star \lambda y. \eta(x \land y) \]

\[ M'\alpha = (\alpha \to i \to (i \times r)) \to i \to (i \times r) \]

\[ \text{her} = \lambda c. \lambda i. c(i(5))i \]
\[ \neq \lambda c. \lambda i. \langle i, c(i(5))i \lor c(i(6))i \rangle \]
\[ j&m = \lambda c. \lambda i. \langle i'', x \land y \rangle \quad \text{where} \quad \langle i', x \rangle = cj \quad \langle i'', y \rangle = cmi' \]
\[ \neq \lambda c. \lambda i. \langle i', x \land y \rangle \quad \text{where} \quad \langle i', x \rangle = cj \quad \langle i'', y \rangle = cmi' \]
Summary

Functional modules are description parts that can be reused in the face of change

- Expressions
- Interpretations
- Side effects
- Lexical entries
- ... 

Types enforce information hiding