Introduction to Design of Computational Calculi

I. DeFrain

1 Mathematical models of grammar

1.1 Grammar for the λ-caclulus

The grammar for the λ-calculus

$$M, N ::= \ x \mid \lambda x.M \mid MN$$

has the domain equation

$$M[X] = X + (X \times M[X]) + (M[X] \times M[X])$$

which is parametric in X (the "names" of the theory).

1.2 Grammar for the ρ-caclulus

The grammar for the ρ-calulus

$$P, Q ::= \ 0 \mid \text{for}(y \leftarrow x)P \mid x!(Q) \mid P|Q \mid {}^*x$$

$$x, y ::= \ @P$$

has the domain equation

$$P[X] = 1 + (X \times X \times P[X]) + (X \times P[X]) + (P[X] \times P[X]) + X$$

$$R = P[R]$$

which is *not* parametric in X because of "tying the recursive knot" with $R = P[R]$.

2 Names and equivalences

The term $\lambda x.M$ *binds* x in M or more generally, λ is a *binder*, it marks x as a variable.

2.1 Free and bound names

Free names for λ-calculus terms

$$\begin{aligned} \mathcal{FN}(x) &= \{x\} \\ \mathcal{FN}(\lambda x.M) &= \mathcal{FN}(M) \setminus \{x\} \\ \mathcal{FN}(MN) &= \mathcal{FN}(M) \cup \mathcal{FN}(N) \end{aligned}$$

- A name x in M is *bound* if it is not free.
- A *closed term* has no free names i.e. a compilable program.

2.2 α-equivalence

Binding and substitution for a free name y in M

$$\lambda x.M \equiv \lambda y.M\{y/x\}$$

- α-equivalence erases syntactic distinctions which make no difference for computation
- having binding operators in a computational model shifts it from being an algebra to a calculus

2.3 Structural equivalence

The structural equivalence for the ρ-calculus is given by

$$P|0 \equiv P, \quad P|Q \equiv Q|P, \quad P|(Q|R) \equiv (P|Q)|R$$

3 Operational semantics i.e. reduction rules

3.1 λ-calculus

Substitution = binding + application

$$\lambda x.MN \to M\{N/x\}$$

3.2 ρ-calculus

Comm

$$\text{for}(y \leftarrow x)P \mid x!(Q) \to P\{@Q/y\}$$

Par

$$P \to P' \implies P|Q \to P'|Q$$

Equiv

$$P \equiv P',\ P' \to Q',\ Q' \equiv Q \implies P \to Q$$

- modeling biological processes with π-calculus

4 Injecting names into the ρ-calculus

Let M be a set of names and

$$\begin{aligned} P[M] &::= 0 \mid \text{for}(m \leftarrow n)P \mid n!(Q) \mid P|Q \mid {}^*x \\ x &::= @P \\ m, n &::= x \mid M \end{aligned}$$

- Rholang is derived from ρ-calculus
- Mobile Process Calculi for Programming the Blockchain

4.1 Process annihilation

Two processes P, Q *annihilate*, denoted $P \perp Q$, means

$$P \perp Q \iff (\forall R.P|Q \to^* R \implies R \to^* 0)$$

Comm rule for annihilating processes (compositionality)

$$x_t = @T,\ x_s = @S,\ S \perp T \implies \text{for}(y \leftarrow x_t)P \mid x_s!(Q) \mid \to P\{@Q/y\}$$

To get off the ground, the stopped process annihilates itself

$$0|0 \equiv 0 \implies 0|0 \to^* 0 \implies 0 \perp 0$$

$$\text{for}(@0 \leftarrow @0)0 \mid @0!(0) \to 0$$

therefore @0 is its own co-channel i.e. $0 \perp 0$.

- Compositionality is fundamentally different from the λ- and π-calculi

4.2 Contexts

Contexts are of the form

$$K ::= [\,] \mid \text{for}(y \leftarrow x)K \mid x!(K) \mid K|Q$$

and we use the convention

$$K[P] := K\{P/[\,]\}$$

Communication in the context K is the comm rule

$$\text{Comm}_K : \ \text{for}(y \leftarrow x)P \mid x!(Q) \to P\{@K[Q]/y\}$$

and adapting between different protocol levels is given by composition of contexts

$$\text{Comm}_{K'} : \ \text{for}(y \leftarrow x)P \mid x!(Q) \to P\{@K'[Q]/y\}$$

$$\text{Comm}_{K \circ K'} : \ \text{for}(y \leftarrow x)P \mid x!(Q) \to P\{@K[K'[Q]]/y\}$$

5 Full abstraction

5.1 Correct-by-construction

Math becomes code and code, literally, becomes math

5.2 Full abstraction

- Hyland & Ong: full abstraction of linear logic
- Full abstraction is the gold standard for the semantics of a computational model: equivalence maps faithfully between models
- Full abstraction allows one to measure jumps in expressivity

$$\rho - \text{calculus} \xleftrightarrow{\text{expressivity}} \text{ambient calculus}$$

- Full abstraction is behavioral, not syntatic

Morris-style equivalence

$$P \approx_1 Q \iff [P] \approx_2 [Q]$$

e.g. $P \approx_1 Q$ (in λ-calculus) means P and Q have no distingugishing context (applicative bisimulation) and $[P] \approx_2 [Q]$ means (weak/strong) bisimulation.

- $\sim$ 2005: Every known process equivalence factors through a form of bisimualtion
- Full abstraction is parametric in the two behavioral equivalences

HW: Calculus presentation of arabic numerals and operational semantics for addition

6 Monoids, Monads, and Location

6.1 Arabic numerals and addition

We "factor" this into two monoids: strings with concatenation and numbers with addition

$$M[G], N[G] \ ::= \ 0 \mid G \mid M[G]@N[G]$$

$$M[G]@0 \equiv M[G] \equiv 0@M[G]$$

$$m_1@(m_2@m_3) \equiv (m_1@m_2)@m_3$$

$$G_1, G_2 \to \text{add}(G_1, G_2)$$

$$m_1 + m_2 \to m \implies (m_1 + m_2) + m_3 \to m + m_3$$

- Grammars and types are the same thing! (parsing and type checking are the same)
- Grammars are related to monads (isomorphic)

6.2 Monads

Monads $\leftrightarrow$ grammars

For a monad $T[X]$, there are the associated operations

$$\begin{array}{cc} T[X] & \text{shape} \\ \text{wrap}[X] : X \to T[X] & \text{wrap/nest (“unit”)} \\ \text{roll}[X] : T[T[X]] \to T[X] & \text{roll/flatten (“mult”)} \end{array}$$

6.3 Location and context

Contexts are the derivative of the domain equation w.r.t. the recursion variable

6.3.1 λ-calculus contexts

From quote, abstraction, and application, we get

$$T[X] = X + (X \times T[X]) + (T[X] \times T[X])$$

introducing the recursion variable

$$T[X, R] = X + X \times R + R \times R$$

and differentiating w.r.t. R and evaluating at the fixed point $R = T[X, R]$, we get

$$\frac{dT[X, R]}{dR} = (X \times 1) + (T \times 1) + (1 \times T)$$

which gives us contexts of the form

$$K \ ::= \ [\,] \mid \lambda x.K \mid TK \mid KT$$

- calculation of derivative $\implies$ one-holed context

6.3.2 Location

The location of a term Q in a term P is given by a context K and the term Q. In general, we have

$$\text{Loc}[T] = \partial T \times T$$

For the ρ-calculus with location,

$$P[X] = 1 + (X \times X \times P[X]) + (X \times P[X]) + (P[X] \times P[X]) + X$$

$$LP = P[\text{Loc}[LP]] = P[\partial LP \times LP]$$

7 Preliminaries for ρ-calculus in space

7.1 Bisimulation

8 ρ-calculus in space i.e. ρ-calculus with location

Let P, Q, R range over processes, x, y, z range over names, and K ranges over one-holed contexts

$$\begin{aligned} P, Q ::= &\ 0 \\ &\mid U \\ &\mid \text{for}(y \leftarrow x)P \\ &\mid x!(Q) \\ &\mid P|Q \\ &\mid {}^*x \\ &\mid \text{Comm}(K) \\ x, y ::= &\ @\langle K, Q\rangle \\ K ::= &\ [\,] \mid \text{for}(y \leftarrow x)K \mid x!(K) \mid K|Q \end{aligned}$$

along with the rewrites

$$\text{Comm}(K) \mid \text{for}(y \leftarrow x)P \mid x!(Q) \rightarrow P\{@\langle K, Q\rangle/y\}$$

$$U \mid {}^*@\langle K, Q\rangle \rightarrow \text{Comm}(K)$$

- terms are data structures which we want to be able to place processes into

A variant of the ρ-calculus in space has the location update $U(x)$ depend on a name x, along with the new rewrite rule

$$U(x) \mid {}^*@\langle K, Q\rangle \rightarrow \text{Comm}(K) \mid x!(Q)$$

which preserves the process Q and the name x.

A simple calculation shows that

$$\text{for}(@\langle K, Q\rangle \leftarrow @\langle[\,], 0\rangle){}^*@\langle K, Q\rangle \mid @\langle[\,], 0\rangle!(P) \mid \text{Comm}(K') \rightarrow {}^*@\langle K, Q\rangle\{@\langle K', P\rangle/@\langle K, Q\rangle\} = K'[P]$$

If K' is of the form

$$\text{for}(@\langle K, Q\rangle \leftarrow @\langle[\,], 0\rangle){}^*@\langle K, Q\rangle \mid @\langle[\,], 0\rangle!([\,]) \mid \ldots$$

then P moves through the context.

- Recursive Einstein field equations?
- Notion of derivative is closely related to space (discrete)
- Quantum mechanics in terms of process calculi?
- Knots $\leftrightarrow$ Processes (processes are knot-invariants)
- Intrinsic geometry for process calculi?

9 LADL: generating formulae for bisimulation

ρ-calculus in space

- The notion of space is a position in a data structure
- Term structures $\leftrightarrow$ tree structures
- Zippers $\leftrightarrow$ locations

Concurrent equivalent of the Y combinator $\implies$ calculus must be higher-order and reflective

$$[\![-]\!] : \text{Form} \Rightarrow \text{BoolAlg} \circ \text{Mon}$$

$[\![\top]\!] = L(M)$	$[\![E]\!] = \{m \in L(M) \mid m \equiv e\}$
$[\![\neg\varphi]\!] = L(M) \setminus [\![\varphi]\!]$	$[\![G_i]\!] = \{m \in L(M) \mid m \equiv g_i\}$
$[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$	$[\![\varphi^* \psi]\!] = \{m \in L(M) \mid m \equiv m_1^* m_2, m_1 \in [\![\varphi]\!], m_2 \in [\![\psi]\!]\}$

10 Modalities

- Modal logic: world $\leftrightarrow$ program state, reachability/transitions $\leftrightarrow$ rewrites
- Kripke semantics
  (possibility) $\diamond\varphi \implies \exists$ world reachable from current world where $\varphi = \mathit{true}$
  (necessity) $\Box\varphi \implies \forall$ reachable worlds from current world, $\varphi = \mathit{true}$
- Proof theory vs. Model theory (Hennessy-Milner)

$$\text{Proof theory} \ \leftrightarrow \ \text{possibility} \ \diamond$$

$$\text{Model theory} \ \leftrightarrow \ \text{necessitiy} \ \Box$$

In the π-calculus, we have the *Milner satisfaction relation* $\models$

$$m \models \varphi \iff m \in [\![\varphi]\!]$$

$$A \models \langle\alpha\rangle\varphi \iff \exists A' \text{ s.t. } A \succ \alpha.A' \text{ and } A' \models \varphi$$

where $A \succ \alpha.A'$ means that $A \equiv \alpha.A' \mid \ldots$

- α's are barbs, not full prefix
- de Morgan's law for the modal operators:

$$\Box\varphi := \neg \diamond (\neg\varphi)$$

- natural way to label transitions with contexts $\rightarrow$ minimal contexts $\rightarrow$ rewrites $\rightarrow$ observations for bisimulation $\rightarrow$ bismulation is a congruence relation w.r.t. rewrite system

Q: Logical formulae for λ-calculus from LADL with Set (arrow type $\leftrightarrow$ modal type)

Q: How to search for a smart contract based on shape and functionality?
Hoogle: search Haskell types ("not interesting")