A Graphical Representation of the Solos Calculus

CM30082 Literature Review

Adam Lassiter

October 2017

Contents

1 Literature

1.1 λ-calculus

Developed by Alonzo Church in the 1930s, the λ-calculus was the first such computational calculus and describes a mathematical representation of a computable function. While when it was first designed, it was not expected to be relevant to the newly-emerging field of theoretical Computer Science but instead Discrete Mathematics, the λ-calculus in fact forms a universal model of computation and can contain an encoding of any single-taped Turing machine. It has since become a popular formal system within which to study properties of computation.

Definition 1. The λ-calculus, as defined by Machado (2013)* consists of an expression M built from the following terms:

$$M \quad ::= \quad a \quad | \quad (\lambda x.M) \quad | \quad (MN) \tag{1}$$

where the expressions each describe a variable, an abstraction and an application of a function on an argument respectively.

From this, any computable function can be constructed and computation is achieved through a series of operations on the expression.

Definition 2. (α-substitution) Unbound variables within an expression may be substituted for any given value. This is formally expressed as:

$$x[y := P] := \begin{cases} P & \text{if } x = y \\ x & \text{otherwise} \end{cases} \tag{2}$$

$$(\lambda x.M)[y := P] := \begin{cases} \lambda x.M & \text{if } x = y \\ \lambda x.(M[y := P]) & \text{if } x \neq y \text{ and } x \notin FV(P)^\dagger \end{cases} \tag{3}$$

This operation may be thought of as variable renaming as long as both old and new names are free in the expression in which they were substituted.

Corollary. (α-equivalence) The above definition of α-substitution may be extended to give an equivalence relation on expressions, α-equivalence, defined as:

$$y \notin FV(M) \implies \lambda x.M \equiv_\alpha \lambda y\,(M[x := y]) \tag{4}$$

$$M \equiv_\alpha M' \implies \begin{cases} M\,P \equiv_\alpha M'\,P \\ P\,M \equiv_\alpha P\,M' \\ \lambda x.M \equiv_\alpha \lambda x.M' \end{cases} \tag{5}$$

Definition 3. (β-reduction) An expression may be simplified by applying one term to another through substitution of a term for a bound variable. This is formally expressed as:

$$(\lambda x.P)\,Q \to_\beta P[x := Q] \tag{6}$$

$$M \to_\beta M' \implies \begin{cases} P\,M \to_\beta PM' \\ M\,P \to_\beta M'P \\ \lambda x.M \to_\beta \lambda x.M' \end{cases} \tag{7}$$

Often β-reduction requires several steps at once and as such these multiple β-reduction steps are abbreviated to $\to_\beta^*$.

*While Church's original paper is still available, the source cited is found to be more relevant due to research in the subject area since the original paper's publication in the 1930s.

†Here, $FV(P)$ is the set of all variables x such that x is free (unbound) in P.

Corollary. (β-equivalence) The above definition of β-reduction may be extended to give an equivalence relation on expressions, *beta*-equivalence, defined as:

$$M \to_\beta^* P \text{ and } N \to_\beta^* P \implies M \equiv_\beta N \tag{8}$$

Example. The above corollaries can be seen to have desirable properties when examining whether two expressions describe equivalent computation.

$$\lambda a.x\, a[x := y] \equiv \lambda a.y\, a$$

$$\lambda x.x\, y \equiv_\alpha \lambda z.z\, y$$
$$\lambda x.x\, y \not\equiv_\alpha \lambda y.y\, y$$

$$(\lambda a.x\, a)\, y \to_\beta x\, a[a := y] \equiv x\, y$$

As computational calculus shares many parallels with modern functional programming, the following are encodings of some common functional concepts within the λ-calculus.

Definition 4. (List) Within the λ-calculus, lists may be encoded through the use of an arbitrary *cons* function that takes a head element and a tail list and of a *null* function that signifies the end of a list. The list is then constructed as a singly-linked list might be constructed in other languages:

$$[x_1, \ldots, x_n] := \lambda c.\lambda n.(c\, x_1\, (\ldots (c\, x_n\, n) \ldots)) \tag{9}$$

Definition 5. (Map) The *map* function takes two arguments — a function F that itself takes one argument and a list of suitable arguments $[x_1, \ldots, x_n]$ to this function. The output is then a list of the output of F when applied to each $x_1 \ldots x_n$.

$$map := \lambda f.\lambda l.(\lambda c.(l\, (\lambda x.c\, (f\, x)))) \tag{10}$$

Below is an example for $n := 3$:

$$\begin{aligned}
map\, F\, [x_1, x_2, x_3] &\equiv_\alpha \lambda f.\lambda l.(\lambda c.(l\, (\lambda x.c\, (f\, x)))\, F\, \lambda c.\lambda n.(c\, x_1\, (c\, x_2\, (c\, x_3\, n))) \\
&\to_\beta^* \lambda c.(\lambda c.\lambda n.(c\, x_1\, (c\, x_2\, (c\, x_3\, n)))\, (\lambda x.c\, (F\, x))) \\
&\to_\beta^* \lambda c.(\lambda n.((\lambda x.c\, (F\, x))\, x_1\, ((\lambda x.c\, (F\, x))\, x_2\, ((\lambda x.c\, (F\, x))\, x_3\, n)))) \\
&\to_\beta^* \lambda c.(\lambda n.(c\, (F\, x_1)\, (c\, (F\, x_2)\, (c\, (F\, x_3)\, n)))) \\
&\equiv_\alpha [F\, x_1, F\, x_2, F\, x_3]
\end{aligned}$$

1.1.1 Evaluation

While the λ-calculus has been successful and been studied by various areas of academia outside of Computer Science, it is limited in application by its fundamentally 'single-process' model and struggles to describe multiple systems working and communicating together.

1.2 Calculus of Communicating Systems

1.2.1 Background

Xu (2009)

1.2.2 Definitions

1.2.3 Evaluation

1.3 π-calculus

1.3.1 Background

Parrow (2001) Similar to the λ-calculus as described in 1.1, there exists the π-calculus for the study of concurrent computation.

1.3.2 Definitions

The π-calculus is constructed from the recursive definition of an expression P:

$$P \quad ::= \quad P|P \quad | \quad c(x).P \quad | \quad \bar{c}(x).P \quad | \quad vx.P \quad | \quad !P \quad | \quad 0$$

where the expressions describe each of concurrency, waiting for input, providing output, name binding (similar to $\lambda x.P$), replication (which itself is $!P :- P|!P$) and process termination respectively.

1.3.3 Examples

1.3.4 Evaluation

While it provides the expressiveness required for Turing-completeness, it does not lend itself to understandability nor clarity of the problem encoding when presented as a standalone expression.

1.4 Fusion Calculus

1.4.1 Background

Miculan (2008)

1.4.2 Definitions

1.4.3 Examples

1.4.4 Evaluation

1.5 Solos Calculus

1.5.1 Background

Ehrhard and Laurent (2010)

1.5.2 Definitions

As defined by Laneve and Victor (1999), the Solos calculus is constructed from *solos* ranged over by $\alpha, \beta, \ldots$ and *agents* ranged over by $P, Q, \ldots$ as such:

$$\begin{array}{lll} \alpha ::= & u\tilde{x} & \text{(input)} \\ & \bar{u}\tilde{x} & \text{(output)} \\ \\ P ::= & 0 & \text{(inaction)} \\ & \alpha & \text{(solo)} \\ & Q|R & \text{(composition)} \\ & (x)Q & \text{(scope)} \\ & [x = y]Q & \text{(match)} \end{array}$$

1.5.3 Examples

1.5.4 Solos Diagrams

This was further developed by Laneve et al. (2001) to provide a one-to-one correspondence between these expressions and 'diagram-like' objects. This provides a strong analog to real-world systems and an applicability to be used as a modelling tool for groups of communicating systems.

2 Technology

2.1 Calculus Reduction

2.2 Diagram Visualisation

Graf et al. (2008)

3 References

Ehrhard, Thomas and Laurent, Olivier. Acyclic Solos and Differential Interaction Nets. In *Logical Methods in Computer Science*, vol. 6(3):1–36 (2010). ISSN 18605974.

Graf, Sabine, Lin, Taiyu and Kinshuk, Taiyu. The relationship between learning styles and cognitive traits Getting additional information for improving student modelling. In *Computers in Human Behavior*, vol. 24(2):122–137 (2008). ISSN 0747-5632.

Laneve, Cosimo, Parrow, Joachim and Victor, Björn. Solo Diagrams. In *Theoretical Aspects of Computer Software: 4th International Symposium, TACS 2001 Sendai, Japan, October 29–31, 2001 Proceedings*, (pp. 127–144). Springer Berlin Heidelberg (2001). ISBN 978-3-540-45500-4.

Laneve, Cosimo and Victor, Björn. Solos in Concert. In *Automata, Languages and Programming: 26th International Colloquium, ICALP'99 Prague, Czech Republic, July 11–15, 1999 Proceedings*, (pp. 513–523). Springer Berlin Heidelberg (1999). ISBN 978-3-540-48523-0.

Machado, Rodrigo. An Introduction to Lambda Calculus and Functional Programming. In *Theoretical Computer Science (WEIT), 2013 2nd Workshop-School on*, (pp. 26–33). IEEE (2013). ISBN 978-1-4799-3057-9.

Miculan, Marino. A Categorical Model of the Fusion Calculus. In *Electronic Notes in Theoretical Computer Science*, vol. 218(1):275–293 (2008). ISSN 1571-0661.

Parrow, Joachim. An Introduction to the π-Calculus. In *Handbook of Process Algebra*, (pp. 479–543). Elsevier Science (2001). ISBN 1-281-03639-0.

Xu, Xian. Expressing First-Order -Calculus in Higher-Order Calculus of Communicating Systems. In *Journal of Computer Science and Technology*, vol. 24(1):122–137 (2009). ISSN 1000-9000.