Computability and Parallelism

Andrés Sicard-Ramírez

Ciclo de Conferencias
Centro de Computación Científica Apolo
Universidad EAFIT
2018-09-19

Motivation

Question

Does parallelism increase the set of functions that can be computed?

Abstract/Outline

It is accepted that the λ-calculus is a model of computation. It is also known that Plotkin's `parallel-or` function or Church's δ function are not λ-definable. We discuss if some extensions of the λ-calculus, which these functions are definable, contradict the Church-Turing thesis.

Lambda Calculus

Alonzo Church (1903 – 1995)*

*Figures sources: History of computers, Wikipedia and MacTutor History of Mathematics .

Lambda Calculus

Some remarks

- A formal system invented by Church around 1930s.
- The goal was to use the λ-calculus in the foundation of mathematics.
- Intended for studying functions and recursion.
- Model of computation.
- A free-type functional programming language.
- λ-notation (e.g., anonymous functions and currying).

Lambda Calculus

Informally

λ-calculus	Example	Represent
Variable	x	x
Abstraction	$\lambda x.x^2 + 1$	$f(x) = x^2 + 1$
Application	$(\lambda x.x^2 + 1)3$	$f(3)$
β-reduction	$(\lambda x.x^2 + 1)3 =_\beta x^2 + 1[\,x := 3\,] \equiv 10$	$f(3) = 10$

Definition

The set of **$\boldsymbol{\lambda}$-terms** can be defined by an abstract grammar.

$$t ::= x \mid t\,t \mid \lambda x.t$$

Lambda Calculus

Conventions and syntactic sugar

- The symbol '$\equiv$' denotes the syntactic identity.
- Outermost parentheses are not written.
- Application has higher precedence, i.e.,

$$\lambda x.MN \equiv (\lambda x.(MN)).$$

- Application associates to the left, i.e.,

$$MN_1 \dots N_k \equiv (\dots((MN_1)N_1) \dots N_k).$$

- Abstraction associates to the right, i.e.,

$$\begin{aligned}\lambda x_1 x_2 \dots x_n.M &\equiv \lambda x_1.\lambda x_2.\dots.\lambda x_n.M \\ &\equiv (\lambda x_1.(\lambda x_2.(\dots(\lambda x_n.M)\dots))).\end{aligned}$$

Lambda Calculus

Example

Some λ-terms.

- xx (self-application)
- $\mathsf{I} \equiv \lambda x.x$ (identity operator)
- true $\equiv \lambda xy.x$
- false $\equiv \lambda xy.y$
- zero $\equiv \lambda fx.x$
- succ $\equiv \lambda nfx.f(nfx)$
- $\lambda f.VV$, where $V \equiv \lambda x.f(xx)$ (fixed-point operator)
- $\Omega \equiv ww$, where $\omega \equiv \lambda x.xx$.

Lambda Calculus

Definition

A variable x occurs **free** in M if x is not in the scope of λx. Otherwise, x occurs **bound**.

Notation

The result of substituting N for every free occurrence of x in M, and changing bound variables to avoid clashes, is denoted by $M[\,x := N\,]$.*

*See, e.g., Hindley and Seldin [2008, Definition 1.12].

Lambda Calculus

Definition

A **combinator** (or **closed $\boldsymbol{\lambda}$-term**) is a λ-term without free variables.

Convention

A combinator called for example succ will be denoted by succ.

Remark

The programs in a programming language based on λ-calculus are combinators.

Lambda Calculus

Conversion rules

The functional behaviour of the λ-calculus is formalised through of their conversion rules:

$$\lambda x.N =_\alpha \lambda y.(N[\,x := y\,]) \qquad (\alpha\text{-conversion})$$

$$(\lambda x.M)N =_\beta M[\,x := N\,] \qquad (\beta\text{-conversion})$$

$$\lambda x.Mx =_\eta M \qquad (\eta\text{-conversion})$$

Lambda Calculus

Example

Some examples of β-equality (or β-convertibility).

- I $M =_\beta M$
- succ zero $=_\beta \lambda fx.fx \equiv$ one
- succ one $=_\beta \lambda fx.f(fx) \equiv$ two
- $\Omega \equiv (\lambda x.xx)(\lambda x.xx) =_\beta \Omega =_\beta \Omega =_\beta \Omega \ldots$

Lambda Calculus

Definition

A $\boldsymbol{\beta}$**-redex** is a λ-term of the form $(\lambda x.M)N$.

Definition

A λ-term which contains no β-redex is in $\boldsymbol{\beta}$**-normal form** (β-nf).

Definition

A λ-term N **is a** $\boldsymbol{\beta}$**-nf of** M (or M **has the** $\boldsymbol{\beta}$**-nf** M) iff N is a β-nf and $M =_\beta N$.

Lambda Calculus

Theorem

Church [1935, 1936] proved that the set

$$\{M \in \lambda\text{-term} \mid M \text{ has a } \beta\text{-normal form}\}$$

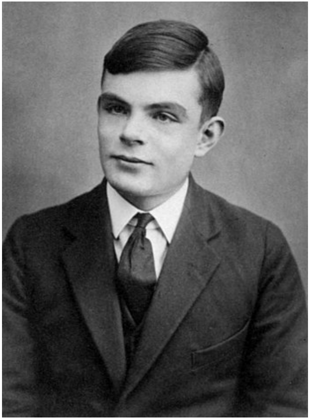
is not computable.* This was the first not computable (undecidable) set ever.†

*We use the term ‘computable‘ rather than ‘recursive‘ following to Soare [1996].

†See also Barendregt [1990].

The Church-Turing Thesis

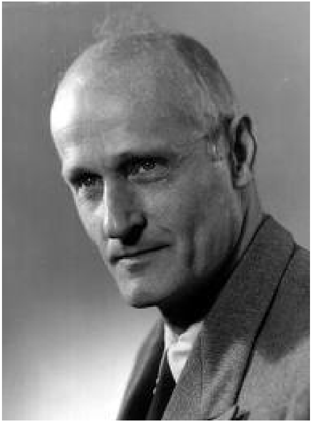
Alan Mathison Turing (1912 – 1954)*

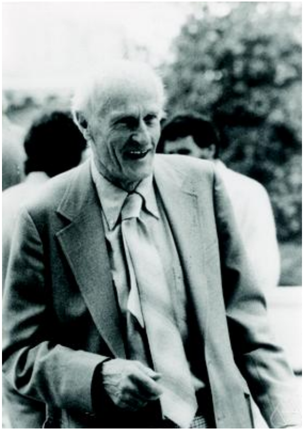


*Figures sources: Wikipedia and National Portrait Gallery .

The Church-Turing Thesis

Stephen Cole Kleene (1909 – 1994)*





*Figures sources: MacTutor History of Mathematics and Oberwolfach.

The Church-Turing Thesis

Theorem

The following sets are coextensive:

i) λ-definable functions,

ii) functions computable by a Turing machine and

iii) general recursive functions.

The Church-Turing Thesis

Common versions of the Church-Turing thesis

> *"A function is computable (effectively calculable) if and only if there is a Turing machine which computes it." [Galton 2006, p. 94]*

> *"The unprovable assumption that any general way to compute will allow us compute only the partial-recursive functions (or equivalently, what Turing machines or modern-day computers can compute) is know as Church's hypothesis or the Church-Turing thesis." [Hopcroft, Motwani and Ullman 2007, p. 236]*

The Church-Turing Thesis

Historical remark

The Church-Turing thesis was not stated by Church nor Turing (they stated definitions) but by Kleene.*

An imprecision

Church [1936] and Turing [1936] definitions were in relation to a computor (human computer).

*See, e.g., Soare [1996] and Copeland [2002].

The Church-Turing Thesis

A better version of the Church-Turing thesis

> *"Any procedure than can be carried out by an idealised human clerk working mechanically with paper and pencil can also be carried out by a Turing machine." [Copeland and Sylvan 1999]*

The Church-Turing Thesis

A better version of the Church-Turing thesis

> *"Any procedure than can be carried out by an idealised human clerk working mechanically with paper and pencil can also be carried out by a Turing machine." [Copeland and Sylvan 1999]*

Question

Why are we talking about "versions" of the Church-Turing thesis?

The Church-Turing Thesis

A better version of the Church-Turing thesis

> *"Any procedure than can be carried out by an idealised human clerk working mechanically with paper and pencil can also be carried out by a Turing machine." [Copeland and Sylvan 1999]*

Question

Why are we talking about "versions" of the Church-Turing thesis?

A/ Because the term 'Church-Turing thesis' was first named, but not defined, by Kleene in 1952 [Jay and Vergara 2004].

Plotkin's `parallel-or` Function

Definition

Let A be a type and let f and $\perp$ be a terminating and a non-terminating function from a to a, respectively. Plotkin [1977] **`parallel-or` function** has the following behaviour:

$$
\begin{aligned}
&\mathrm{pOr} :: (a \to a) \to (a \to a) \to a \to a \\
&\mathrm{pOr}\ f\ \perp = f \\
&\mathrm{pOr}\ \perp\ f = f \\
&\mathrm{pOr}\ \perp\ \perp = \perp
\end{aligned}
$$

* `http://hackage.haskell.org/package/unamb` .

Plotkin's `parallel-or` Function

Definition

Let A be a type and let f and $\bot$ be a terminating and a non-terminating function from a to a, respectively. Plotkin [1977] **`parallel-or` function** has the following behaviour:

$$
\begin{aligned}
&\mathrm{pOr} :: (a \to a) \to (a \to a) \to a \to a \\
&\mathrm{pOr}\ f\ \bot = f \\
&\mathrm{pOr}\ \bot\ f = f \\
&\mathrm{pOr}\ \bot\ \bot = \bot
\end{aligned}
$$

Haskell implementation

See the `unamb` function from the unambiguous choice library.*

*`http://hackage.haskell.org/package/unamb` .

Plotkin's `parallel-or` Function

Definition

From Sun's Multithreaded Programming Guide:*

> "***Parallelism:*** *A condition that arises when at least two threads are executing simultaneously.*"
>
> "***Concurrency:*** *A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism.*"

* `https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032b/index.html` .

Plotkin's `parallel-or` Function

Definition

From Sun's Multithreaded Programming Guide:*

> "***Parallelism:*** *A condition that arises when at least two threads are executing simultaneously.*"
>
> "***Concurrency:*** *A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism.*"

Question

Are we talking about a parallel or concurrent function?

---

* https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032b/index.html .

Plotkin's `parallel-or` Function

Theorem

The `parallel-or` function is an effectively calculable function which is not λ-definable [Plotkin 1977].*

*See, also, Turner [2006].

Church's δ Function

Definition

Let M and N be combinators in β-nf. **Church's $\boldsymbol{\delta}$** function is defined by

$$\delta MN = \begin{cases} \text{true}, & \text{if } M \equiv N; \\ \text{true}, & \text{if } M \not\equiv N. \end{cases}$$

Theorem

Church's δ function is not λ-definable [Barendregt 2004, Corollary 20.3.3, p. 520].

Extensions of Lambda Calculus

Jay and Vergara [2017] wrote (emphasis is ours):

> *"For over fifteen years, the lead author has been developing calculi that are more expressive than λ-calculus, beginning with the constructor calculus [8], then pattern calculus [2,7,3], SF-calculus [6] and now λSF-calculus [5]...*
>
> *[The] λSF-calculus is able to query programs expressed as λ-abstractions, as well as combinators, something that is beyond pure λ-calculus.*
>
> *In particular, we have proved (and verified in Coq [4]) that equality of closed normal forms is definable within λSF-calculus."*

Extensions of Lambda Calculus

Jay and Vergara [2017] also stated the following corollaries:

1. Church's δ is λSF-definable.
2. Church's δ is λ-definable.
3. Church's δ is not λ-definable.

Discussion

Question

Do Plotkin's `parallel-or` function or Church's δ function—which are effectively calculable functions but they are not λ-definable functions—contradict the Church-Turing thesis?

Discussion

Question

Do Plotkin's `parallel-or` function or Church's δ function—which are effectively calculable functions but they are not λ-definable functions—contradict the Church-Turing thesis?

A/ No! But we need a better version of the Church-Turing thesis.

Discussion

Definition

A function f is a **number-theoretical function** iff

$$f : \mathbb{N}^k \to \mathbb{N}, \text{ with } k \in \mathbb{N}.$$

Discussion

Definition

A function f is a **number-theoretical function** iff

$$f : \mathbb{N}^k \to \mathbb{N}, \text{ with } k \in \mathbb{N}.$$

Theorem

The following sets are coextensive:

i) λ-definable number-theoretical functions,

ii) number-theoretical functions computable by a Turing machine and

iii) general recursive functions.

Remark

The above theorem is historically precise as pointed out in [Jay and Vergara 2004].

Discussion

A better version of the Church-Turing thesis

We should define the Church-Turing thesis by:

Any number-theoretical function than can be computed by an idealised human clerk working mechanically with paper and pencil can also be computed by a Turing machine.

Discussion

A better version of the Church-Turing thesis

We should define the Church-Turing thesis by:

Any number-theoretical function than can be computed by an idealised human clerk working mechanically with paper and pencil can also be computed by a Turing machine.

Remark

Jay and Vergara [2004, 2017] also negatively answer the question under discussion stating other versions of the Church-Turing thesis.

References

Barendregt, H. P. (2004). *The Lambda Calculus. Its Syntax and Semantics*. Revised edition, 6th impression. Vol. 103. Studies in Logic and the Foundations of Mathematics. Elsevier (cit. on p. 28).

Barendregt, Henk (1990). 'Functional Programming and Lambda Calculus'. In: *Handbook of Theoretical Computer Science*. Ed. by van Leeuwen, J. Vol. B. Formal Models and Semantics. MIT Press. Chap. 7. DOI: 10.1016/B978-0-444-88074-1.50012-3 (cit. on p. 14).

Church, Alonzo (1935). 'An Unsolvable Problem of Elementary Number Theory. Preliminar Report (Abstract)'. In: *Bull. Amer. Math. Soc.* 41.5, pp. 332–333. DOI: 10.1090/S0002-9904-1935-06102-6 (cit. on p. 14).

— (1936). 'An Unsolvable Problem of Elementary Number Theory'. In: *American Journal of Mathematics* 58.2, pp. 345–363. DOI: 10.2307/2371045 (cit. on pp. 14, 19).

Copeland, B. Jack (2002). 'Hypercomputation'. In: *Minds and Machines* 12.4, pp. 461–502. DOI: 10.1023/A:1021105915386 (cit. on p. 19).

References

Copeland, B. Jack and Sylvan, Richard (1999). 'Beyond the Universal Turing Machine'. In: *Australasian Journal of Philosophy* 77.1, pp. 44–66. DOI: 10.1080/00048409912348801 (cit. on pp. 20–22).

Galton, Antony (2006). 'The Church-Turing Thesis: Still Valid after All These Years?' In: *Applied Mathematics and Computation* 178.1, pp. 93–102. DOI: 10.1016/j.amc.2005.09.086 (cit. on p. 18).

Hindley, J. R. and Seldin, J. (2008). *Lambda-Calculus and Combinators. An Introduction.* Cambridge University Press (cit. on p. 9).

Hopcroft, John E., Motwani, Rajeev and Ullman, Jefferey D. (2007). *Introduction to Automata theory, Languages, and Computation*. 3rd ed. Pearson Education (cit. on p. 18).

Jay, Barry and Vergara, Jose (2004). 'Confusion in the Church-Turing Thesis'. Draft version. URL: https://arxiv.org/abs/1410.7103 (cit. on pp. 20–22, 33–36).

— (2017). 'Conflicting Accounts of λ-Definability'. In: *Journal of Logical and Algebraic Methods in Programming* 87, pp. 1–3. DOI: 10.1016/j.jlamp.2016.11.001 (cit. on pp. 29, 30, 35, 36).

References

- Plotkin, G. D. (1977). 'LCF Considered as a Programming Language'. In: *Theoretical Computer Science* 5.3, pp. 223–255. DOI: 10.1016/0304-3975(77)90044-5 (cit. on pp. 23, 24, 27).
- Soare, Robert I. (1996). 'Computability and Recursion'. In: *The Bulletin of Symbolic Logic* 2.3, pp. 284–321. DOI: 10.2307/420992 (cit. on pp. 14, 19).
- Turing, Alan M. (1936). 'On Computable Numbers, with an Application to the Entscheidungsproblem'. In: *Proc. London Math. Soc.* 42, pp. 230–265. DOI: 10.1112/plms/s2-42.1.230 (cit. on p. 19).
- Turner, David (2006). 'Church's Thesis and Functional Programming'. In: *Church's Thesis After 70 Years*. Ed. by Olszewski, Adam, Woleński, Jan and Janusz, Robert. Ontos Verlag, pp. 518–544. DOI: 10.1515/9783110325461.518 (cit. on p. 27).

Thanks!