The NORMAL programming language

A Constructive Three-Step Proof of Turing-Completeness

The NORMAL community

1. Abstract

We introduce the NORMAL programming language that is built upon and relies on fundamental mathematical numbers and theorems to provide a disruptively new coding experience. Programs in NORMAL consist of a number that is mapped to a position within the decimal expansion of π. This digit – and recursively following digits – are decoded into expressions of an untyped lambda calculus that is subsequently interpreted to compute the programs resulting value or diverge.

Furthermore we present a nearly-complete proof of the most probable Turing-completeness of NORMAL.

The proof consists of three conceptually straightforward steps: (1) show the normality of π, (2) invoke the classical Turing-completeness of the untyped lambda calculus, and (3) conclude that every lambda term's number must eventually occur in π and is therefore programmable in NORMAL.

We show that most of these steps are either trivial or have already been proven.

2. Introduction

Programming languages have historically suffered from excessive syntactic structure, explicit semantics, and an unhealthy dependence on programmer intent. NORMAL addresses these shortcomings by replacing conventional source code with a single integer[1]

Given an input number $n \geq 0$, the NORMAL interpreter examines the n-th decimal digit of $\pi \approx 3.14159265\ldots$ This digit is interpreted as a grammar symbol for a recursively expanding grammar defining the untyped lambda calculus with conditionals and integer constants (see below). Whenever a non-terminal symbol is encountered, subsequent digits of π are consumed until a complete lambda expression is produced.

Example programs include:

Program	corresponding λ-expression	Semantics
1	$\lambda x.1$	A quine
2	$\lambda x.0$	Constant zero function
4	$\lambda x.x$	Identity function, i.e. returns its input
47	$\lambda x.x + 1$	Successor function
388	$\lambda x.$ if x then 1 else (if 0 then 1 else x)	Sign function

[1] The NORMAL programming language, http://normal-lang.kölzer.eu/

`988899`	$\lambda x.1+1+1+1$	xkcd random number[2]
`131825471`	$\lambda x.(\lambda x.xx)(\lambda x.xx)$	Divergent program

Programming with lambda calculus has therefore never been simpler.

3. Definition of NORMAL

More formally, to parse and run a NORMAL program, run these steps:

1. Parse the code as non-negative integer in base 10 as n. Ignore preceding or trailing whitespace and an optional $+$ sign, i.e. NORMAL code is valid, iff it matches the following regular expression: `^\s*\+?([1-9][0-9]*)\s*$`[3]
2. Use the intermediate grammar shown in Table 1 to derive a word by replacing the non-terminal M by the production corresponding to n.
3. While there is a non-terminal M, take the subsequent digit n' of π and replace the leftmost M by the production corresponding to n'.
4. Let NORMAL be the resulting term. Then evaluate (NORMAL arg), where arg is the input to the program, using call-by-value semantics and return the result. Non-negative integers are interpreted as natural numbers, binary operations $+$ and $*$ are interpreted as sum resp. product of the corresponding numbers (in $\mathbb{Z}$).

   If a variable is not bound, the left-hand side of an application cannot be evaluated to a lambda expression or any operand of a binary operation cannot be evaluated to an integer, the program fails. (Note that a program might also diverge.)

	$\mathbf{NORMAL} ::= (\lambda x.M)$	A NORMAL program
(0)	$M ::= (MM)$	Application
(2)	$M ::= (\lambda x.M)$	Lambda Expression
(3)	$M ::= (\lambda y.M)$	Lambda Expression
(5)	$M ::= x$	Variable
(6)	$M ::= y$	Variable
(4)	$M ::= \mathbf{0}$	Integer constant
(1)	$M ::= \mathbf{1}$	Integer constant
(9)	$M ::= \textbf{if } M \textbf{ then } M \textbf{ else } M$	If-Statement
(7)	$M ::= (M + M)$	Sum in $\mathbb{Z}$
(8)	$M ::= (M * M)$	Product in $\mathbb{Z}$

Table 1: Intermediate Grammar (context free grammar, CFG) with start symbol NORMAL and terminal symbols λ, ., (,), 0, 1, x, y, if, then, else, + and *. The first column show the digit that is mapped to the respective production rule.

4. Turing-Completeness

NORMAL's Turing-completeness follows from normality of π, a surjective mapping into the lambda calculus and Turing-completeness of the lambda calculus. Our prove naturally decomposes into several lemmas.

[2] Randall Munroe: xkcd 221, https://xkcd.com/221/

[3] see PCRE2 Project, https://www.pcre.org/

4.1. Turing-Completeness of the Untyped Lambda Calculus

We start with the latter step, which is a classic.

Lemma 1. The untyped lambda calculus is Turing-complete.

Proof Note that NORMAL's intermediate grammar defines a superset of the pure untyped lambda calculus that has been proven to be Turing-complete by uncounted graduate students around the world. So the lemma follows trivially from results by Church, Kleene and several others[4].

This step is therefore considered entirely uncontroversial. ∎

4.2. Universality of NORMAL

We now show that every finite lambda expression can be expressed in the NORMAL programming language.

Lemma 2. Assume π is normal. Then every finite syntactic object over the NORMAL intermediate grammar occurs somewhere within the decimal expansion of π.

Proof is straightforward by induction over the structure of the term and examining the top-most production rule.

Consequently, for every lambda term M, there exists some natural number n such that the NORMAL interpreter beginning at offset n decodes precisely M.

Thus every lambda program is representable within NORMAL. ∎

4.3. Turing-completeness of NORMAL

Combining the two observations, we can now formulate the main result of our work.

Conjecture 3. NORMAL is Turing-complete.

Proof (incomplete sketch) From the lemmas above we can conclude that it remains to be shown that π is normal in base 10.

Substantial evidence suggests that the result should be validate in the near future. Numerical investigations strongly indicate statistical uniformity in the decimal expansion of π, and many mathematicians privately believe the statement to be true while waiting for somebody else to prove it. (At least, we do!)

Furthermore, related constants have already been shown to be normal. For example, the Champernowne constant

$$0.12345678910111213\ldots$$

is known to be normal in base 10[5]. Similar constructions by Copeland and Erdős establish normality results for several digit-concatenation constants.

Consequently, we regard this step as essentially solved modulo technical details. But for the time being, this remains unproven.

5. Future Work

Several important research directions remain open:

- Proving the normality of π.

[4] Shane Steinert-Threlkeld, Lambda Calculi, https://iep.utm.edu/lambda-calculi/, Stanford University, USA

[5] Champernowne, D. G. (1933), "The construction of decimals normal in the scale of ten", Journal of the London Mathematical Society, 8 (4): 254–260, https://doi.org/10.1112/jlms/s1-8.4.254

- Designing practical debuggers for transcendental execution traces.
- Locating useful enterprise software within the first 10^{18} digits of π.
- Determining whether a Brainf*ck interpreter already exists in sufficiently distant digits.

We believe these goals are achievable within a modest timeframe.

6. Conclusion

We have presented a simple and elegant three-step proof sketch for the Turing-completeness of NORMAL. Two of the three steps have already been completed. The remaining step appears mostly administrative.