Recursive Function Theory and Computability

GILSON ANTONIO GIRALDI[1]

[1]LNCC–National Laboratory for Scientific Computing -
Av. Getulio Vargas, 333, 25651-070, Petropolis, RJ, Brazil
{gilson}@lncc.br

Abstract. Concepts in Computability

1 Introduction

In this material we will focus on two alternative aapraches to the classification of computable functions: machine based and the functional one. The former is based on the Turing Machine framework while the later set out to capture the concept of computability by considering the class class of functions over $\mathbf{N}$ that result from a set of basic functions and functional composition operators.

In section 4 we review the basic elements underlying this approach, developed by Gödel and Kleene, and called *Recursive Function Theory.* Before that presentation, some concepts about recursion proceedures shall be discussed.

2 Peano's Axioms

Given the set $\mathbf{N}$ of undefined objects called *natural numbers*, and a function $s : \mathbf{N} \to \mathbf{N}$,where $s(n)$ is called the successor of n. The function s satisfies the following axioms [6]:

P1) The function s in injective.

P2) $\mathbf{N} - s(\mathbf{N})$ has only one element.

P3. Induction Principle: If $X \subset \mathbf{N}$ is a subset such that $0 \in X$ and, for all $n \in X$ we have $s(n) \in X$, then $X = \mathbf{N}$.

The Induction Principle can be also stated in another way which is more suitable in practice. Before to re-write it, let us consider any property P concerning natural numbers as a function:

$$P : \mathbf{N} \to \{0, 1\},$$

such that $P(n) = 1$ if P is true for a given n and $P(n) = 0$ otherwise. So, we can re-write the axiom P3) in the following way:

Induction Principle: Let us consider P as a property related to natural numbers. If $P(0) = 1$, and starting from the hypotesis that $P(k) = 1$, it is possible to show that $P(k + 1) = 1$ also, then we can conclude that $P(n) = 1, \forall n \in \mathbf{N}$.

Deomonstration by Mathematical Induction: It is any demonstration in which the Induction Principle is applied.In this case, the demonstration follows the next steps:

(a) We show that $P(0) = 1$,

(b) As the *inductive hypothesis*, we assume that $P(k) = 1$. Then, using this hypothesis we try to prove that $P(k + 1) = 1$ also.

(c) From (a), (b), and by the Induction Principle we conclude that $P(n) = 1, \forall n \in \mathbf{N}$.

Example: Prove by mathematical induction that:

$$(a - 1)\left(1 + a + a^2 + ... + a^n\right) = a^{n+1} - 1.$$

Theorem: The set $\mathbf{N}$ is well-ordered; that is, every nonempty subset of $\mathbf{N}$ has a least element.

3 Axioms and Computer Science

The Peano 's Axioms, plus set $\mathbf{N}$, composes a formal (axiomatic) system. We can say that in an Axiomatic System there are *objects,* which existence is assumed, and a finite number of statements about the objects, the axioms [4].

We always hope that an axiomatic system has the following properties:

(A1) Complete: If and only if for all statement S **at least** one of S, $\tilde{}S$ is a theorem;

(A2) Consistent: If and only if for all statement S **at most** one of S, $\tilde{}S$ is a theorem;

Until the beginning of the XX century it was a belief that we could solve any well defined problem. So, if anyone did not have success to demonstrate some property, we should expect that the chosen hypotheses were not suitable or some mistake was made. Such belief in the axiomatic method was the starting point for Hilbert 's work in computer science. Hilbert asked whether or not there existed some algorithm which could be used, in principle, to solve all the problems of mathematics. He expected that the answer to this question would be yes. However, the answer to Hilbert's problem turned out to be no: there is no algorithm to solve all mathematical problems. Such demonstration was due Church and Turing and is a remarkable work in the theory of algorithms, and consequently, for the modern theory of computer science (see [7], Chapter 3)

Besides, the development of a precise formulation of the concept of mathematical proof, culminated with the Kurt Gödel 's results on completeness and consistency in forma theories. These establish the existence, in any sufficiently powerful theory, of mathematical statements that are (self-evidently) *true* but can not be proved to be so. Gödel 's work was also motivated by the Hilbert's problem. In 1931 Gödel published his famous Incompleteness Theorem, that essentially shows that no set of axioms with properties (A1) and (A2) above could exist ; and thus, Hilbet's algorithm could not be written.

Interested reader shall see references, [7], Chapter 3, and [3] to complete this discussion.

4 Recursive Function Theory

Many of the sets involved in applications, such as formal language generation, are infinite. Then, we must be able to define an infinite set in a manner that allows its members to be *efficiently* constructed and manipulated. This can be done recursively.

A **recursive definition** of a set X specifies a method for constructing the elements of the set by using two components: the basis elements and a finite set of rules or operators. The basis consists of a set of elements that are explicitly given as member of the set X. The rules (or operators) are used to construct new elements of the set from the previously defined members. The reader will certainly remember the recursive programing technique in which a computational procedure calls itself. The link between such programing technique and the recursive definition is that we can implement the later by using the former.

For instance, we can recursively define the set $\mathbf{N}$, following the Peano 's Axioms, through the following scheme:

i) Basis: $0 \in \mathbf{N}$,

ii)Recursive Step: If $n \in \mathbf{N}$, then $s\left(n\right) \in \mathbf{N}$,

iii) Closure: $n \in \mathbf{N}$ only if it can be obtained from 0 by a finite number of applications of the operation s.

More interesting recursive definitions can be found in the context of formal languages. So, let an alphabet X, for example, $X = \{a, b, c, d, .., x, y, z\}$. A string over X is a finite sequence of elements from X. Strings are the fundamental objects used in the definition of languages. In order to establish the properties of strings, the set of strings over an alphabet is defined recursively. The basis consists of the string that contains no elements, the **null string**, denoted by λ. The definition is as follows:

Definition: Let Σ be an alphabet. The set Σ^*,of strings over Σ, is defined by:

i) Basis: $\lambda \in \Sigma^*$,

ii) Recursive step: If $w \in \Sigma^*$ and $a \in \Sigma$ then $wa \in \Sigma^*$,

iii) Closure: $w \in \Sigma^*$ only if it can be obtained from λ by a finite number of applications of the operation in ii).

A **language** over an alphabet Σ is a subset of Σ^*. Set operations can be used to specify languages [9]. Automata theory can be used to define a class of abstract machines whose computation determine if a string belongs to a language or not [9, 3]. For instance, such theory found a fundamental and beautiful application in the field of DNA computing framework [8]. Computational modeling of insect societies is also another field with interesting applications of frameworks, like Stochastic Process Algebras, derived from the relationships between automata and languages [5].

5 Recursive Function as a Model of Computation

We shall observe that everything recursively defined can be computed; in particular it can be implemented in a computer. However, what is a computable function? From a practical viewpoint, we may say that a computable function is the one that can be computed, in finite time, in some computer hardware.

Such point of view link a theoretical universe, composed by procedures and algorithms, and the physical realization of an abstract machine that can be the host of the computation. For instance, up to now, we can not really compute with real numbers. In fact, we can say that the available hardwares can only compute with countable quantities. The relationship between computational theory and natural numbers received a more fundamental role when Church and Turing (1936) stated the conjecture that the limitation to compute with real number is not just a matter of the state-of-the-art in hardwares but it is the consequence of an universal concept (Church-Turing Thesis) [7]. Such thesis can be stated as follows, which was presented by Deutch [2] in his famous work on the universal quantum computer:

”Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means”.

From the viewpoint of the Quantum Mechanics, the basis for such statement may be the influence of noise and the discrete nature of the universe inherent in the Quantum Mechanics postulates [1].

Now, we must return to the initial question: what is a computable function? Would be any function $f : \mathbf{N}^k \rightarrow \mathbf{N}$ a computable one?

To answer this question, we will follow a way inspired in the recursive definition presented above. So, the key idea is based on the following elements:

a) The set $\mathbf{N}$; which elements can be represented in a computer;

b) A set of basic (computable) functions,

c) Operations over the basic set, also computables..

Every function of such set will be computable. If the so defined set of functions is enough general to encompass the intuitively computable functions we are close to solve the initial question. In fact, that is the case. The so called Partial Recursive Functions (**PRF**), developed by Gödel and Kleene, set out to capture the concept of computability by considering the classes of functions over $\mathbf{N}$ that result from a set of basic functions and functional composition operators. Such theory is develope next.

6 Partial Recursive Functions

a) Basic Function. The following functions are partial recursive:

$$zero : \mathbf{N} \rightarrow \mathbf{N}; \quad zero(n) = 0, \forall n \in \mathbf{N}, \tag{1}$$

$$succ : \mathbf{N} \rightarrow \mathbf{N}; \quad succ(n) = n + 1, \tag{2}$$

$$proj\_x_i : \mathbf{N}^n \rightarrow \mathbf{N}; \quad proj\_x_i(x_1, .., x_n) = x_i. \tag{3}$$

b)Composition. If the following functions are partial recursive:

$g : \mathbf{N}^k \rightarrow N$,

$f_i : \mathbf{N}^m \rightarrow N, \quad i \in \{1, 2, ..., k\}$,

then, the function defined by the composition of these functions:

$$\begin{aligned} h &: \mathbf{N}^m \rightarrow \mathbf{N}, \\ h(x) &= g(f_1(x), f_2(x), ..., f_k(x)), \end{aligned} \tag{4}$$

is also partial recursive function.

c) Recursion. If the following functions are partial recursive:

$f : \mathbf{N}^m \rightarrow \mathbf{N}$,

$g : \mathbf{N}^{m+2} \rightarrow \mathbf{N}$,

then, the function $h : \mathbf{N}^{m+1} \rightarrow \mathbf{N}$ defined as:

$$h(x_1, ..., x_n, 0) = f(x_1, ..., x_n), \tag{5}$$

$$h(x_1, ..., x_n, y + 1) = g(x_1, ..., x_n, y, h(x_1, ..., x_n, y)), \tag{6}$$

is said to be defined by recursion from f and g, and is also partial recursive.

d) Minimization: If the function $f : \mathbf{N}^{m+1} \to \mathbf{N}$ is partial recursive, then the function $h : \mathbf{N}^m \to \mathbf{N}$ defined by:

$$h(x_1, ..., x_n) = \min\{y; \quad f(x_1, ..., x_n, y) = 0 \quad and \quad if \quad z \leq y \Rightarrow f(x_1, ..., x_n, z) \quad exists\} \tag{7}$$

is said to be defined by minimization from f and is also partial recursive.

The reader can find more details in [3]

We shall made some considerations about the minimization. We are going to show that we need this concept in order to include the factorial function in the PRF set. So, let us consider the partial recursive function:

$$f : \mathbf{N} \to \mathbf{N},$$

$$\begin{aligned} f(n) &= 1 - n, \quad n \leq 1, \\ f(n) &= 0, \quad otherwise. \end{aligned} \tag{8}$$

Then, the function

$$h = \min\{y; \quad f(y) = 0 \quad and \quad if \quad z \leq y \Rightarrow f(z) \quad exists\}, \tag{9}$$

is well-defined in the context of PRFs, despite of the fact that it does not have a domain! In fact, this function can be identified with the number "1", and written as:

$$\begin{aligned} h &: \cdot \to \mathbf{N}, \\ h &= 1. \end{aligned} \tag{10}$$

From this fact, using also the axiom that successor is a PRF and the fact that the product function is a PRF (Exercise), it follows that the function:

$$g : \mathbf{N}^2 \to \mathbf{N},$$

$$g(x, y) = succ(x) \cdot y, \tag{11}$$

is also a PRF one (Exercise). Therefore, once functions h, g are PRFs we can use the recursion operation, defined in axiom (c) above, to show that the factorial function is PRF. The corresponding scheme defines a function $f : \mathbf{N}^{0+1} \to \mathbf{N}$, such that:

$$f(0) = h, \tag{12}$$

$$f(k+1) = g(k, f(k)) = (k+1) \cdot f(k), \tag{13}$$

which is a PRF function that implements the factorial (Exercise).

7 Exercises

1) By using induction, prove that:

1.a) for all $n \in \mathbf{N}$

$$1 + 2 + 3 + ... + n = \frac{n}{2}(n+1).$$

1.b) for all positive integer n, $5^n - 1$ is divided by 4.

1.c) for all $n \in \mathbf{N}$

$$1 + x + x^2 + ... + x^n = \frac{x^{n+1} - 1}{x - 1}.$$

1.d)

$$\frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \frac{1}{3 \cdot 4 \cdot 5} + ... \frac{1}{n \cdot (n+1) \cdot (n+2)} = \frac{n (n+3)}{4 (n+1) (n+2)}$$

2) Based on the axioms of the PRFs, demonstrate that the following functions are PRF:
2.a) $f : \mathbf{N} \to \mathbf{N}; \quad f(n) = 1$ (constant "1" function)
2.b) Addition of two natural numbers;
2.c) Product of two natural numbers;
2.d) $Power(n, m) = n^m$;
3) Prove that the function given by expression (11) is a PRF.
4) Show, by induction, that the scheme given by expressions (12)-(13) defines the factorial $f(k) = k!$, $k \in \mathbf{N}$ and $k \geq 0$.

References

[1] S. Bains and J. Johnson. Noise, physics, and non-turing computation. *www.sunnybains.com/jcis2000.pdf*, 2000.

[2] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London Ser. A*, A400:97–117, 1985.

[3] P.E. Dunne. *Computability Theory: Concepts and Applications*. Ellis Horwood, 1991.

[4] R. Garnier and J. Taylor. *Discrete Mathematics for New Technology*. Adam Hilger, 1992.

[5] Gilson A. Giraldi, Adilson V. Xavier, Antonio L. Apolinario Jr., and Paulo S. Rodrigues. Lattice gas cellular automata for computational fluid animation. Technical report, 2005.

[6] E. L. Lima. *Curso de Análise*, volume 2. Livros Técnicos e Científicos Editora S.A., 1985.

[7] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press., December 2000.

[8] G. Paun, G. Rozenberg, and A. Salomaa. *DNA computing : new computing paradigms*. Springer, Berlin, New York, 1998.

[9] T.A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley Publishing Company, INC., 1988.