On the Kleene normal form

Armando B. Matos
armandobcm@yahoo.com

January 16, 2014

Abstract

[to do]

Contents

1 Introduction

In this note we transcribe and comment parts of some papers dealing with Kleene normal form. It was written mainly for personal purposes but may be of interest for other researchers. Please note that the notation is not always consistent.

It helps to have a good feeling for what can be done with primitive recursive (pr) functions. For instance, suppose that we know that the function $\text{step}(e, x)$, having as arguments the index e of a Turing machine and an integer x encoding its configuration, and returning the configuration that results from one computation step, is pr. Then, it should be clear that the function $\text{run}(e, x, n)$ that returns the configuration after n computation steps is also pr.

For the readers accustomed to computer programming, the characterization of pr functions by the LOOP language [7] programs may be useful.

2 The Kleene's normal form in the literature

The main burden in proving Kleene's normal form Theorem is to show that there are certain functions (related with "universal" or "general purpose" models of computation) that are primitive recursive. These proofs are neither difficult nor deep, but are often rather laborious. For instance, there is a *primitive recursive function* that accepts an integer y and checks things like

1. y represents a computation history of a certain Turing machine T – essentially a succession of machine configurations.
2. The state in the first configuration is the initial state.
3. The application of transtion rules of T to each configuration produces the next configuration, except the first, is obtained from the previous one with the application

Let us state the Kleene's normal form Theorem from [8], page 90.

Theorem 1 (KNF Theorem, Odifreddi) *There is a PR function $\mathcal{U}$ and (for each $n \geq 1$) PR predicates $\mathcal{T}_n$, such that for every recursive (total) function f of n variables there is a number e (called the* index *of f) for which the following hold:*

1. $\forall x_1 \ldots x_n \exists y : \mathcal{T}(e, x_1, \ldots, x_n, y)$

2. $\mathcal{U}(\mu_y \mathcal{T}_n(e, x_1, \ldots, x_n, y))$.

Theorem 1 is in fact a special case for it deals only with total functions. Also note that criterium 1. above, which characterizes the *total* character of the function, is not effective.

Most formulations of the Theorem are "general" in the sense that they give a representation or computation for any partial recursive function. A part of the statement which is often emphasised is that, to represent any partial recursive function, the operator μ needs to be used *at most once.* For instance, the theorem is stated in [1], page 94, is

Theorem 2 (KNF Theorem, Boolos, Burgess and Jeffrey) *Every recursive total or partial function can be obtained from the basic functions (zero, successor, identity) by composition, primitive recursion, and minimization,* using this last process no more than once.

Sometimes, a register language, such as WHILE, is used as the general purpose computation model. In this case, Kleene's normal form Theorem can be stated as follows. For any partial recursive function $f(\overline{x})$ there is a program of the following form (which can be slightly simplified) that computes it.

```
% Input in registers x1,... xn, e
% Output in register x0
PR function (no WHILE's)
WHILE y ≠ 0
        PR function (no WHILE's)
ENDWHILE
PR function (no WHILE's)
```

A more detailed information about a "normal form" WHILE program can be seen for instance in [2].

It is also interesting to mention Kleene's original version of the normal form Theorem, as stated in [5], page 288. The notation was slightly adapted.

Theorem 3 (KNF Theorem, Kleene) *For each $n \geq 0$: given any general recursive function $\phi(x_1, \ldots, x_n)$, a number e can be found such that*

$$\forall x_1 \ldots x_n \exists y : \mathcal{T}_n(e, x_1, \ldots, x_n, y), \tag{1}$$

$$\phi(x_1, \ldots, x_n) = \mathcal{U}(\mu_y \mathcal{T}_n(e, x_1, \ldots, x_n, y)), \tag{2}$$

$$\forall x_1 \ldots x_n y : \mathcal{T}_n(e, x_1, \ldots, x_n, y) \Rightarrow \mathcal{U}(y) = \phi(x_1, \ldots, x_n) \tag{3}$$

where $\mathcal{T}_n(e, x_1, \ldots, x_n, y)$ and $\mathcal{U}(y)$ are the particular primitive recursive predicate and function defined above.

Again, this result mentions only total (recursive) functions. Condition 1 says that the function represented by the index e is total. Condition 2 says that $\mathcal{U}(\mu_y \mathcal{T}_n(\ldots))$ is in fact the function ϕ. Finally, condition 3 says that, for any halting computational history y, the value $\mathcal{U}(y)$ is correct, that is, equal to $\phi(x_1, \ldots, x_n)$.

Regarding condition 3, we recall that the computational models or definitional systems may be non-deterministic, so that this is a kind of "normal form" theorem. Particular computations or definition sequences may of course diverge, even for total functions.

It seems that the equality of two partial recursive functions is not expressed by this result. Write as usual $f(\overline{x}) \equiv g(\overline{x})$ if, for any input $\overline{x}$ the following holds

> Either $f(\overline{x})$ and $f(\overline{x})$ are both undefined
> ...or both are defined and have the same value

and, of course, $\mu_y(P(\overline{x}, y))$, where P is a total predicate, is undefined if there is no y such that $P(\overline{x}, y)$ holds. Then (part of) Theorem 3 could be rephrased as the following function equality

$$\phi(\overline{x}) \equiv \mathcal{U}(\mu_y \mathcal{T}_n(e, \overline{x}, y))$$

3 Kleene's T predicate From Wikipedia, the free encyclopedia

In computability theory, the T predicate, first studied by mathematician Stephen Cole Kleene, is a particular set of triples of natural numbers that is used to represent computable functions within formal theories of arithmetic. Informally, the T predicate tells whether a particular computer program will halt when run with a particular input, and the corresponding U function is used to obtain the results of the computation if the program does halt. As with the smn theorem, the original notation used by Kleene has become standard terminology for the concept[1]

3.1 Definition

The definition depends on a suitable Gödel numbering that assigns natural numbers to computable functions. This numbering must be sufficiently effective that, given an index of a computable function and an input to the function, it is

[1]The predicate described here was presented in [6, 4], and this is what is usually called "Kleene's T predicate". [5] uses the letter T to describe a different predicate related to computable functions, but which cannot be used to obtain Kleene's normal form theorem.

possible to effectively simulate the computation of the function on that input. The T predicate is obtained by formalizing this simulation.

The ternary relation $T_1(e, i, x)$ takes three natural numbers as arguments. The triples of numbers (e, i, x) that belong to the relation (the ones for which T1(e,i,x) is true) are defined to be exactly

> the triples in which x encodes a computation history of the computable function with index e when run with input i, and the program halts as the last step of this computation history.

That is, T1:

- Asks whether x is the Gödel number of a finite sequence $\langle x_j \rangle$ of complete configurations of the Turing machine with index e, running a computation on input i.
- If so, T_1 then asks if this sequence begins with the starting state of the computation and each successive element of the sequence corresponds to a single step of the Turing machine.
- If it does, T1 finally asks whether the sequence $\langle x_j \rangle$ ends with the machine in a halting state.

If all three of these questions have a positive answer, then $T_1(e, i, x)$ holds (is true). Otherwise, $T_1(e, i, x)$ does not hold (is false).

There is a corresponding function U such that if $T(e, i, x)$ holds then $U(x)$ returns the output of the function with index e on input i.

Because Kleene's formalism attaches a number of inputs to each function, the predicate T_1 can only be used for functions that take one input. There are additional predicates for functions with multiple inputs; the relation

$$T_k(e, i_1, \ldots, i_k, x)$$

holds if x encodes a halting computation of the function with index e on the inputs $i_1, \ldots, i_k$.

3.2 Normal form theorem

The T predicate can be used to obtain Kleene's normal form theorem for computable functions (see [9], page 15). This states there exists a primitive recursive function U such that a function f of one integer argument is computable if and only if there is a number e such that for all n one has

$$f(n) \simeq U(\mu x\, T(e, n, x)),$$

where

1. μ is the μ operator, namely $\mu x\, \phi(x)$ is the smallest natural number x' such that $\phi(x')$ holds and, for every $0 \leq x < x'$, $\phi(x)$ is defined but does not hold (is false). Note that the operator μ can be effectively implemented by, say, a Turing machine.
2. $\simeq$ holds if both sides are undefined or if both are defined and they are equal.

Here U is a universal operation (it is independent of the computable function f) whose purpose is to extract, from the number x (encoding a complete computation history) returned by the operator μ, just the value $f(n)$ that was found at the end of the computation.

3.3 Formalization

The T predicate is primitive recursive in the sense that there is a primitive recursive function that, given inputs for the predicate, correctly determine the truth value of the predicate on those inputs. Similarly, the U function is primitive recursive.

Because of this, any theory of arithmetic that is able to represent every primitive recursive function is able to represent T and U. Examples of such arithmetical theories include Robinson arithmetic and stronger theories such as Peano arithmetic.

3.4 Arithmetical hierarchy

In addition to encoding computability, the T predicate can be used to generate complete sets in the arithmetical hierarchy. In particular, the set

$$K = \{e \mid \exists x : T(e, 0, x)\}$$

which is of the same Turing degree as the halting problem, is a Σ_1^0 complete unary relation (see [9], pages 28, 41). More generally, the set

$$K_{n+1} = \{\langle e, a_1, \ldots, a_n\rangle \mid \exists x : T(e, a_1, \ldots, a_n, x)\}$$

is a Σ_1^0 complete $(n+1)$-ary predicate. Thus, once a representation of the T predicate is obtained in a theory of arithmetic, a representation of a Σ_1^0-complete predicate can be obtained from it.

This construction can be extended higher in the arithmetical hierarchy, as in Post's theorem (compare with [3], page 397). For example, if a set $A \subseteq \mathbb{N}^{k+1}$

is Σ^0_n complete then the set

$$\{\langle a_1, \ldots, a_k\rangle \mid \forall x(\langle a_1, \ldots, a_k, x) \in A)\}$$

is Π^0_{n+1} complete.

References

[1] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, 2007. Fifth Edition.

[2] Computer Science Courses. Course CS 20a, Solution for homework #6, 2002. `http://courses.cms.caltech.edu/cs20/a/hw/hw6/solution/sol6.pdf`.

[3] Peter G. Hinman. *Recursion-theoretic Hierarchies*. Perspectives in mathematical logic. Springer-Verlag, 1978.

[4] Stephan Cole Kleene. *Introduction to Metamathematics*. North-Holland, 1952. Reprinted by Ishi press, 2009.

[5] Stephan Cole Kleene. *Mathematical Logic*. John Wiley, 1967. Reprinted by Dover, 2001.

[6] Stephen Cole Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53(1):41–73, 1965.

[7] A. R. Meyer and D. M. Ritchie. The complexity of loop programs. *Proceedings of 22nd National Conference of the ACM*, pages 465–469, 1967.

[8] Piergiorgio Odifreddi. *Classical Recursion Theory – The Theory of Functions and Sets of Natural Numbers*. Studies in Logic and the Foundations of Mathematics. Elsevier North Holland, first edition, 1989. Second impression.

[9] Robert I. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer, 1987.