Computation is not Conceptually Function Application

Ulisses Ferreira

e-mail: *Ulisses.Ferreira@philosophers-fcs.org*

Abstract. Given some recent advances in global/mobile computing, the present paper shows that programs are conceptually different from functions and, accordingly, computation is conceptually different from applications of functions.

1 Introduction

In 1999, after having left Edinburgh University, the present author contacted CAPES, a Brazilian Sponsor, and sent an informal letter about the nature of computation and new ideas that were coming up, which allowed him to earn scholarship for two years. Within the following 18 months, more than 450 pages on the subject were written with minor mistakes but essentially textual mistakes, improved in 2001 in the form of closely related articles and a synthesis. One of the referred to articles that the author like most, specially for being didactic, was published in [4]. In particular, the idea of both the present paper and [2], is a consequence of [4].

Excluding this introductory section, this article is organized as follows: in 2, a few theorems are proved. After those theorems, in section 3, there is some concluding remark.

2 Ferreira's Theorem

One of the best definitions that I have seen regarding non-deterministic Turing machine is in [6], and, here, I only reproduce it:

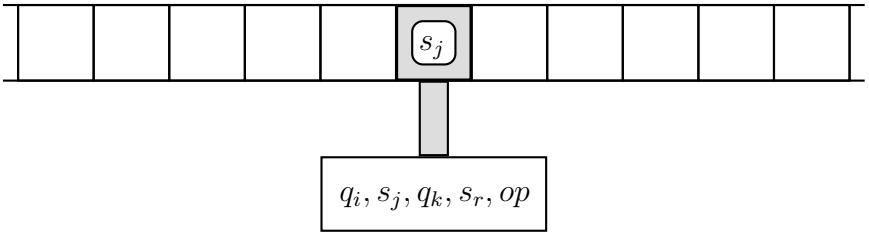
Definition 1. *A Turing machine is an ordered system* $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *where* Q *is a finite set of states,* Σ *is the input alphabet,* Γ *is the tape alphabet,* $\Gamma \cap Q = \emptyset$ *and* $\Sigma \subset \Gamma$, $q_0 \in Q$ *is the initial*

state, $B \in \Gamma - \Sigma$ *is the blank symbol,* $F \subseteq Q$ *is the set of final states and* δ *is the transition function,*

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

□

By defining the codomain of δ as above, that is $\mathcal{P}(Q \times \Gamma \times \{L, R\})$, the machine may be non-deterministic. Further, it can be shown that non-deterministic Turing machines are equivalent to deterministic Turing machines. For a deterministic version, which I use in this paper, δ can be redefined as $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$. It is known that, according to Alan Turing's analogy, a Turing machine is supplied with an infinite tape divided into squares and one write/read head. Each tape square contains one symbol in Σ.



The present paper shows that programs are conceptually different from functions and, accordingly, computation is conceptually different from applications of functions.

In contrast with Church-Turing thesis[1], the present result is the same as one in [2, 5], which is essentially based on observation over the concept of mobile agents, but here it is in the context of the theory of computability. In this way, this paper makes use of an alternative path in comparison to [5].

Given the present context, the following has been referred to as Ferreira's Theorem:

Theorem 1. *Computation is not necessarily a function application.*

[Proof] Given that computation may be mobile, e.g. by using mobile agents nowadays, computation is conceptually a physical process.

By theorem 1 in [3], the class of Turing machines is not isomorphic to the class of Turing-computable functions. Following this,

programs do not correspond to functions. Therefore, computation is not really a function application.

□

Proposition 1 *Let $k \in \mathbb{N}$. For every $k > 1$, there exists a k-level Turing-machine composition if and only if some representation of the Universal Turing machine is not in the composition.*

[Proof] Let U be a Universal Turing machine, $M, N : \mathbb{N} \longrightarrow \mathbb{N}$ be two Turing machines with corresponding Turing-computable functions $m, n : \mathbb{N} \longrightarrow \mathbb{N}$, and $x \in \mathbb{N}$, and $X : \mathbb{N}$ be the representation of x on the tape.

The Universal Turing machine, by lemma 1 in [3], guarantees the absence of unexpected effects at all levels of its parameters. I can consider the composition $M[N[X]]$. It follows that U must have *direct* control over the operations of N in such a way that, if N tries to modify the operations in the M representation, U detects this unexpected effect and intervenes, for instance, by moving physically the representation of M or N to another place on the tape, to continue the computation of the composition keeping the isomorphism between Turing machines and computable functions. Therefore, because U must have dynamic knowledge about the computation carried out by N, $U(M[N[X]])$ is not really a function application, and therefore some representation of U is not in the composition.

With respect to the converse, setting $M \neq U \wedge N \neq U$ and $X \neq U$, there exists a Turing-machine composition, e.g. respectively $M(N[X])$ above, from which U is absent.

□

Remark 1. We can capture an intuitive and precise notion of Turing machine model as follows: Let $\mathcal{M}$ be the set of all Turing machines, T be the set of Turing-computable functions, U be the Universal Turing machine and u be the Universal Turing-computable function. Let $X : \mathbb{N}$ be the null-computation Turing machine that corresponds to the 0-ary function (i.e. without any input) that always results in the same value $x \in \mathbb{N}$. Therefore, $u : \mathcal{P}(T) \longrightarrow \mathbb{N}$ (where $\mathcal{P}$ is the ordered power set of a given ordered set), in such a way that the application $m(n(x))$ is equal to $u(m(n(x)))$ and abstractly represented as $U(\{M, N, X\})$ or, more precisely, as $U(s)$ where s denotes the

string that encodes M, N and X, together with the write/read head and blank symbols, with the constraint that s does neither start nor finish with the blank symbol. Furthermore, composition is part of the notion of function, and such a representation does not capture the composition of Turing machines $U(M[N[X]])$. However, there may be applications as well as compositions involving U where there exist such representations with U, both on the tape and outside the tape. Therefore, there exist two different levels of functional abstraction in the Turing machine model of computation.

In other words, on the one hand, we separate what is encoded on the tape from what is outside the tape, by stating that only what is outside the tape is free from unexpected effects, and hence, can be functions. On the other hand, functions do not manipulate the operations of any function. In this way, there are two different levels of abstraction: at one level, only the Universal Turing machine is function and the encoded Turing machines on the tape form a one-level parameter. At another level, there exists a Turing machine composition on the tape, and the encoded Turing machines correspond to the computable functions because the Universal Turing machine does not correspond to any function in the same space, in the sense that U is capable of managing the tape and guaranteeing absence of unexpected effects. Therefore, there exist two separate levels of function abstraction in the Turing machine model of computation.

In the next proposition, as usual, I do not regard *time* as in the concept of computation.

Proposition 2 *There exists a Turing machine that can correspond to more than one Turing-computable function.*

[Proof] Let M be some Turing machine and x be its input. Let U be a Universal Turing machine, and V be another Turing machine, which, except for the existence of unexpected effects, produces the same output as U: the only difference is that U calculates $U(U[M[X]])$, and V calculates $V(V[M[X]])$ and sometimes calculates $U(U[M[X]])$, but sometimes not, depending on the physical places where V and M rest on the tape. Because the Turing machine composition $V(V[M[X]])$ can be placed at different places on the tape at different instants and the computations receive different kinds of unexpected effects, the same running Turing machine M can produce different results

for the same input x. Each particular result from x corresponds to one Turing-computable function.

□

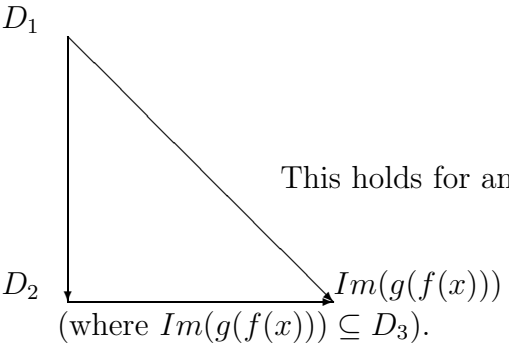
Corollary 1. *There exists a Turing machine that can avoid receiving unexpected effects from its parameter.*

[Proof] Universal Turing machines, as discussed in the theorem 1 in [3], must ensure absence of unexpected effects.

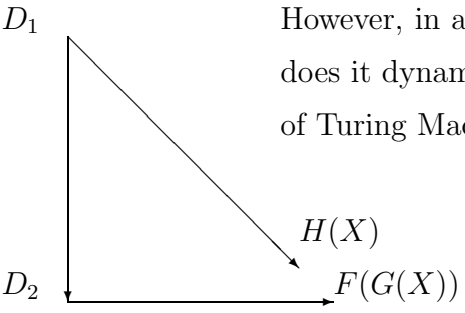
□

3 Conclusion

I can draw functions $f : D_1 \longrightarrow D_2$ and $g : D_2 \longrightarrow D_3$ as well as the corresponding composition $h : D_1 \longrightarrow D_3$, under the law $h = g(f(x))$ as in the following picture:



This holds for any function application.



However, in addition to the Tm composition, does it dynamically hold for any computation of Turing Machine composition?

For answering the question, I individually consider the correspondence between the functions f, g and h, and the Turing machines F, G and H, respectively. The answer for the question, i.e. whether the law of composition $F(G(X)) = H(X)$ holds for every computation of composition of Turing machines, depends on the absolute positions of the involved Turing machines, F and G, on the tape, as well as on whether the only Turing machine outside the tape avoids unexpected effects on the tape. However, both factors are *external* to the machines that are on the tape and play rôles in the composition. As a consequence, the global view is a property of the Universal Turing machines, in particular, it avoids what I discovered and refer to as unexpected effects.

Acknowledgements

The present author would like to thank the people who kindly helped towards the publication of this paper.

References

1. Richard L. Epstein and Walter A. Carnielli. *Computability: computable functions, logic, and the foundation of mathematics*. Thomson Learning/Wadsworth, second edition, 1999. Book with 'Computability and undecidability – a timeline (the story of the development of computable functions and the undecidability of arithmetic to 1970, Richard L. Epstein)'.
2. Ulisses Ferreira. Computation is not conceptually function application. In *Proceedings of CIC–2004 International Conference on Computing*, Mexico City, Mexico, October 2004.
3. Ulisses Ferreira. A property for church-turing thesis. In Hamid R. Arabnia, Iyad A. Ajwa, and George A. Gravvanis, editors, *Post-Conference Proceedings of the 2004 International Conference on Algorithmic Mathematics & Computer Science*, pages 507–513. CSREA Press, June 2004. Las Vegas, Nevada, USA.
4. Ulisses Ferreira. Mobility and computation. In Veljko Milutinovic, editor, *Proceedigs of IPSI-2005 HAWAII Conference*. IPSI BgD, January 2005.
5. Ulisses Ferreira. On the foundations of computing science. In David L. Hicks, editor, *Proceedings of the Metainformatics Symposium MIS'03*, number 3002 in Lecture Notes in Computer Science, pages 46–65. Springer, September 2003, published in 2004.
6. Alexandru Mateescu and Arto Salomaa. *Hanbook of Formal Languages*, volume 1, chapter Aspects of Classical Language Theory, pages 175–251. Springer-Verlag, 1997.