A computable expression of closure to efficient causation

Matteo Mossio, Giuseppe Longo, John Stewart

A computable expression of closure to efficient causation

[Penultimate draft: to appear in the *Journal of Theoretical Biology*]

Matteo Mossio[i]

Institut d'Histoire et de Philosophie des Sciences et des Techniques, CNRS/Université Paris 1/ENS, 13 rue du Four, 75006, France. Matteo.Mossio@ens.fr.

Giuseppe Longo
Laboratoire d'Informatique, CNRS/Ecole Normale Supérieure, 45, Rue d'Ulm, 75005, Paris, France. http://www.di.ens.fr/users/longo/. Giuseppe.Longo@ens.fr.

John Stewart
COSTECH, Université de Technologie de Compiègne, Centre Pierre Guillaumat, BP 60.319 60206, Compiègne, France. johnmacgregorstewart@gmail.com.

Abstract

In this paper, we propose a mathematical expression of closure to efficient causation in terms of λ-calculus; we argue that this opens up the perspective of developing principled computer simulations of systems closed to efficient causation in an appropriate programming language. An important implication of our formulation is that, by exhibiting an expression in λ-calculus, which is a paradigmatic formalism for computability and programming, we show that there are no conceptual or principled problems in realizing a computer simulation or model of closure to efficient causation. We conclude with a brief discussion of the question whether closure to efficient causation captures all relevant properties of living systems. We suggest that it might not be the case, and that more complex definitions could indeed create some obstacles to computability.

Keywords

Closure to efficient causation; λ-calculus; Computability; Impredicativity; (M, R) systems; Robert Rosen.

1. Introduction

All fully-fledged scientific objects, from atoms to black holes, are constituted *in theory*. If contemporary biology is excessively focussed on genes, as a number of critical commentators have suggested (Fox-Keller, 2000; Oyama, 1985; Lewontin, 1984), this is nothing but a logical consequence of the fact that genes are indeed constituted in theory (Jacob, 1970), whereas (to date) living organisms as such are not. Consequently, it is common to adopt a merely common-sense definition of life, and then to develop models of specific aspects of living organisms. According to an increasing number of researchers, however, this scientific approach to biological systems is missing the point

in an important sense: we are not modelling the organism as living, we are just treating it as thought it were not alive. Robert Rosen (1991, pp 111-112), referring to the life work of his mentor Rashevsky, writes: "No collection of separate models, however comprehensive, could be pasted together to capture the organism itself". In this sense, the heart of the question lies in the construction of a theoretical (preferably mathematical) model, which captures what should be considered the key properties of living systems.

At the present time, one of the most prominent proposals aimed at providing a theoretical characterization of life is Rosen's definition in terms of *closure to efficient causation* (Rosen, 1991). Rosen's proposal is thus, potentially, of the greatest importance for biology as a whole, since it could contribute to a better balance between Genetics and a Biology of Organisms (Stewart, 2004). However, as noted with perspicacity by Letelier et al. (2006), Rosen's work has been very diversely appreciated. Some authors consider that Rosen is indeed the "Newton of biology" (Mikulecky, 2001) and some others have tried to apply Rosen's framework to build a mathematical model of metabolism (Letelier et al., 2006). Wolkenhauer & Hofmeyr (2007) developed an abstract cell model inspired by Rosen's ideas. Moreover, a considerable amount of work has been recently undertaken to clarify the conceptual relations between the concept of closure to efficient causation and that of autopoïesis (Nomura, 2007; Letelier et al., 2003; Zaretzky & Letelier, 2002). Yet, despite its strong theoretical interest, Rosen's work has had so far regrettably little impact on the mainstream of contemporary biology.

Among the possible reasons for this lack of influence (to date), the one we wish to focus on in this paper has already been pointed out by other authors (Letelier et al., 2006): this is the fact that Rosen's original formulation in terms of Category Theory, although intuitively understandable, was not easily biologically interpretable, nor operationally generative. The purpose of this paper is to propose an interpretation of Rosen's closure to efficient causation in terms of (type-free) lambda-calculus. The importance of our proposal lies in the fact that lambda-calculus lies at the very heart of modern definitions of "computability". In this sense, above and beyond the interest of the lambda-calculus formulation itself, our proposal thus opens up the perspective of developing computer simulations of closure to efficient causation in an appropriate programming language.

This is a key issue, both theoretically and operationally. As a matter of fact, one of Rosen's best known theses, supported with a mathematical demonstration, is that *closure to efficient causation has no computable models* (Rosen, 1991 p. 235-243). We may call this "Rosen's conjecture" (Stewart & Mossio, 2007). At present, the status of this conjecture is uncertain and controversial. Whereas some studies have claimed that Rosen's purported proof of the conjecture is flawed (Chu & Ho, 2007a; Chu & Ho, 2007b; Chu & Ho, 2006; Landauer & Bellman, 2002), their conclusions have been contested as wrong and irrelevant by advocates of Rosen's thesis (Louie, 2007 and 2006), and the logic underlying Rosen's conjecture has been restated and defended (Chemero & Turvey, 2007; Kercel, 2007).

In addition, it should be noted that some studies have recently tried to spell out some relevant implications of the (supposed) non-computability of closure to efficient causation, as if Rosen's demonstration were correct. In particular, Letelier and co-workers have recently argued that autopoietic systems are a subset of (M,R)-systems and that therefore they inherit the property of being non-computable (Letelier et al., 2003). During the last thirty years, a specific line of research in field of Artificial Life has developed computational simulations of autopoietic systems (see McMullin, 2004 for a recent review). If Letelier's thesis is correct, this would have a major impact on the relevance of these studies: we should conclude that, whatever organization they simulate, they are not (and could not) properly simulating autopoïesis.The expression of closure to efficient causation in terms of lambda-calculus proposed in this paper may constitute a useful contribution to this debate, since it shows that there are no conceptual problems in realizing computational simulations of closure on the basis of the classical definition of computability. We consider that this result revitalizes Rosen's proposal by opening up a whole new vista of possible expressions through *principled* computer simulations, going beyond mere *ad hoc* tinkering to capture (at least some) relevant properties of "life itself", at least as defined by Rosen's formalism.

At this point, it may be useful to interject a clarification that we have discussed at greater length in (Stewart & Mossio, 2007). Rosen (1991) makes a clear and important distinction between a *modelling* relationship and a *simulation.* In the modelling relationship (p.60), a natural system N is put in relation with a formal system F. In order for this to succeed, F must itself have a structure of efficient causes that is potentially

congruent with that of the natural system. In a Turing machine (or any formal equivalent thereof), there is only one efficient cause: the reading-head. Since there is not a human-built Turing machine such that its operation actually *produces* its own reading-head, nothing in the operation of a computer can be adequate to authentically *model* CTEC. In this sense, "life itself" is definitely not computable.

This does not, however, foreclose the discussion; because the question remains open as to whether it may not be possible to *simulate* CTEC. Rosen (1991, pp. 191-193) gives a very clear discussion and definition of "simulation". In causal terms, simulation involves the conversion of efficient cause, the hardware of that which is being simulated, into material cause in the simulator. In such a computational universe, the most that can be done is to set up a system of structures that construct and maintain each other in the circular fashion exemplified by CTEC. Even this remains a far from trivial exercise; but we claim that our lambda-calculus formulation shows that it is indeed possible.

The structure of this article is the following. We will first recall (section 2) the conceptual framework necessary for qualitative expression of the notion of closure to efficient causation. We next discuss (sections 3 and 4) the question of which mathematical tools are most adequate and fruitful for expressing this concept. This leads us (section 5) to our central proposition, a formulation of closure to efficient causation in terms of lambda-calculus. Since our formulation implies, in contrast with Rosen's own claim, the computability of closure to efficient causation, we next try to spell out (section 6) the reasons for the divergence with respect to Rosen's conclusions, as well as (section 7) other authors' interpretations. We then provide some preliminary guidelines (section 8) on how the closure to efficient causation expressed in terms of λ-calculus could be implemented as a computer simulation preserving the essential properties of the formal model. Finally (section 9), we conclude with a brief discussion of the question whether closure to efficient causation captures all relevant properties of living systems. We suggest that this it might not be the case, and that more complex definitions could indeed create some obstacles to computability.

2. Closure to efficient causation

Rosen's whole conceptual scheme is based on a rehabilitation and reinterpretation of the Aristotelian categories of causality: material cause, efficient cause, and (under certain conditions, but we will not discuss this point here) final cause. Rosen presents the Aristotelian categories as different ways of answering the question "why?" Given a mathematical function, b = f(a), there are two answers to the question "why b?": (i) "because a", i.e. the argument of the function, which Rosen interprets as the "material cause"; and (ii) "because f" where the function f is interpreted as the "efficient cause". In set-theoretical terms, if a and b are in the domain and co-domain of f, respectively, then f maps a to b. Applied to the case of state-determined dynamic systems (SDDS), the mapping is an endomorphism from x(t), the state-vector at time t, to the "next state" x(t+dt). The whole art of finding a mathematical expression of SDDS is to choose the state variables in such a way that the state x(t+dt) is a function *only* of the state x(t) (Rosen, 1991, pp. 89-98).

This formulation enables Rosen to express what is, in his view, the difference between physics and biology. In physics, we can ask questions about the state of a dynamic system. "Why x(t)?" – (i) "because $x(t_0)$", the state at any reference time t_0; this is the "material cause"; and (ii) "because f", the dynamic law; this is the "efficient cause" (in more common terms: f is the evolution function of the dynamics). But if we ask the question "why f?", within physics there is not really any answer, other than that this just is a natural law. Rosen's proposition is that this is where biology is different: for a living organism, the question "why f?" has a non-trivial answer from within the functioning of the system itself. Let us look at this a little closer.

There is fairly wide agreement that *metabolism* is at the core of living organisms. In Rosen's formulation (Rosen, 1991, p. 249), this is expressed by the equation:

$$B = f(A) \quad (E1)$$

where A is the "material cause" of the metabolism, f is the "efficient cause" of the metabolism, and B is the result. To give a rough-and-ready interpretation, A corresponds to the input materials and energy; f may be associated with the set of enzymes which are necessary to catalyze the biochemical reactions, but also the cell membrane, necessary to avoid loss of reactants by diffusion, and probably other features of cell organization as well; and B corresponds to the total resulting biochemical

network. We will return later to the question of biological interpretations of these formulae.

What characterizes living organisms is that the maintenance, and indeed the ongoing production of this "metabolism" function, are themselves ensured by the functioning of the organism. In Rosen's formalism, this is expressed by a second function, Φ, which takes B as material cause and which produces f; Rosen (1991, p. 250) calls this function *repair*:

$f = \Phi(B)$ (E2)

Now by an iteration of the same argument, we must now ask: "why Φ?" As before, we can introduce a new function, b, which Rosen (1991, p. 250) calls *replication*:

$\Phi = \beta(f)$ (E3')

The point is that we now see clearly the threat of an incipient infinite regress. On the face of it we will require another function for the production of β, and then yet another function for the production of this function, and so on indefinitely. We come now to the key point: Rosen makes the crucial observation that the infinite regress can be avoided by introducing a circularity: β can be identified with B, which is already produced by the system (Equation [E1]). Thus (E3') becomes:

$\Phi = B(f)$ (E3)

We thus arrive at the situation which Rosen (1991, p. 251) calls « closure to efficient causation », in which each efficient cause is materially produced within the system, as illustrated in Figure 1. The three efficient causes – f, the metabolism function; Φ, the repair function; and B, the replication function – are all produced by the operation of the system itself. Rosen considers that "closure to efficient causation" – hereafter "CTEC" – is *the* essential defining property of "life itself" (Rosen, 1991, p. 244).

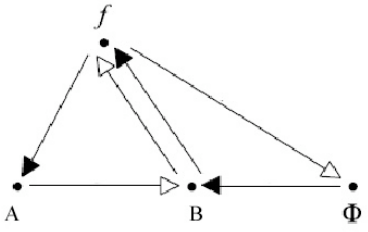

Figure 1. Rosen's relational model of closure to efficient causation. White arrows represent relations of material causation; black arrows represent relations of efficient causation.

In the following section we will see how CTEC can be easily related to Cartesian Closed Categories (CCC), which are suitable structures to deal at once with mathematical objects and transformations on these objects, a key point in Rosen's approach, as extensively explained in Letelier et al. (2006). We will interpret Rosen's formalism in CCC by relying on the strong relations existing between CCC and lambda-calculus.

Note that the interpretation of Rosen's formalism is far from obvious (and is not necessarily unique). In *Life Itself*, Φ and f appear to be morphisms. Yet, and this is the challenge, in the *diagrams* they are also objects, that is they are also sources and targets of morphisms, while B, an object, may act as a morphism on f, say, as expressed in the equational writing above. As shown below, the λ-calculus makes it possible to deal with such an apparent type-theoretic mismatch; in particular, by constructing the CCC of "finitary projections" (Amadio et al., 1986) out of a type-free model. In this category, both types (as objects) and morphisms are "elements" of the type-free universe, thus they can freely act one on the other. We will not spell out the finitary projection interpretation in full detail, as this would require a lengthy and highly technical exposition. We restrict ourselves here to noting that the free use of B, Φ and f both as morphisms and as objects may be fully mathematically justified (see Amadio et al., 1986).

3. Lambda calculus

In order to formulate the theoretical concept of CTEC, Rosen employed the mathematical formalism of mappings and abstract block diagrams (Rosen 1991, p. 123ff). This formalism is perfectly adequate for its primary purpose, which is to express the qualitative concept of CTEC as such. However, as we have already mentioned in the introduction, it would seem that in practice theoretical biologists have not been able to actually *use* this formalism to generate detailed models of living organisms and/or testable predictions (see also Letelier et al., 2006 on this point). This raises the question of a possible alternative formalism, in order to give Rosen's proposal a more

operationally fruitful mathematical expression. It seems to us that the essential requirement to attain this objective is the following: given a mathematical function, b = f(a), we need a formalism in which the *same* entity can occupy the three roles of argument (a), function (f) and result (b); moreover, the notation must also be such that it is perfectly clear at each point *which* role is being played. As clearly pointed out by Fontana & Buss (1994) in a related context, λ-calculus meets these requirements exactly.

Type-free λ-calculus (Church, 1932/1933; Barendregt, 1984) is a formal theory of functional abstraction and application. In Mathematics the notation f(x) is indeed ambiguous: does it denote the mapping from x to f(x), or the value of f at x? Let us then write f(x) only for the expression or *value* of f on x, and denote λx.f(x) for the *mapping* from x to f(x) (the operation is called "λ-abstraction" or functional abstraction). So, λ "binds" variables and makes explicit the functional dependence of functions on variables. More generally, λx.f(x,y), where x is bound and y is free, is the mapping from x to f(x,y), which differs of course from λy.f(x,y), where the explicit functional dependence is on y. This also allows us to explicitly formalize the evaluation of functions on arguments, by (functional) "application": for instance, from $(\lambda x.f(x,y))\cdot 5$ one obtains f(5,y). So, $(\lambda x.(\lambda y.(x^2+y))\cdot 4)\cdot 5$ gives first $(\lambda y.(25+y))\cdot 4$), then 25+4 (parentheses are very important in λ-calculus, as in the many derived programming languages such as LISP). The calculus is *type-free*, which means that it contains no constraints on what may be a function and what an argument; yet, their role in each term is specified by the *order* and *the use of parentheses*. Thus, one may apply x to y or y to x or even write $x\cdot x$ and then, for instance, abstract: $\lambda x.x\cdot x$. The consistency of this very expressive calculus (it computes all Turing-computable functions) is assured by a fundamental theorem due to Church and Rosser (Barendregt, 1984; Hindley & Seldin, 1986).

In more formal terms, both the formal theory and the mathematical semantics contain:

- a sign λ (or a semantic operator) for denoting (or interpreting) a functional operator;
- a sign "·" (or a semantic application) for denoting (or interpreting) functional application.

That is, it is a formal applicative and non-commutative structure $(X, \lambda, \cdot)$, where:

- $\lambda x.M$ forms a function of x from any formal expression M (we say that λ binds x in $\lambda x.M$; if a variable y occurs and it is not bound in M, then it is free in M);
- $M \cdot N$ forms the application of M to N.

Formal expressions, or terms, are made out of variables: x, y, z...; parentheses: "(" and ")"; and the operators λ and $\cdot$ (usually, one writes (MN) for $M \cdot N$ and omit "$\cdot$"). Thus, if M and N are terms, $\lambda x.M$ and (MN) are terms (and nothing else is a term). Note that one can form xy *and* yx, which are thus both legal terms, *but* that $xy \neq yx$ (non commutative).

To this, one has to add the usual axioms for equality, "=" (as a symmetric, reflexive transitive and substitutive relation) and one key axiom:

(β) $(\lambda x.M)N = [N/x]M$

By this axiom the left term is equated to the replacement of the free occurrences of x by N in M. The renaming of bound variables may be needed, when replacing x by N in M. Equality of λ-terms is handled by the usual congruence rules. λ-calculus is a paradigmatic "rewriting system": all that it does is to replace strings by strings. Nevertheless, it has the same expressive power as Turing Machines or any other complete formalism for computability (Gödel's Recursion in Arithmetic, Kleene's equations... see Barendregt, 1984; Hindley & Seldin, 1986). Actual computer languages compute at most these computable functions. LISP is an implemented version of type-free λ-calculus.

The expressive power of type-free λ-calculus is due to the fixed-point operator:

$Y = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$.

By applying several times the (β) axiom above, it is easy to show that for *any* term M, one has, by replacing y by M:

$YM = (\lambda x.M(xx))(\lambda x.M(xx))$

and then, by replacing x *in the first* M(xx) by *the second* $\lambda x.M(xx)$:

$YM = M((\lambda x.M(xx))(\lambda x.M(xx)))$

Thus:

$YM = M(YM)$

since $(\lambda x.M(xx))(\lambda x.M(xx)) = YM$, by the first equation.

Thus, Y produces, uniformly and effectively, a fixed point YM for M, as we have shown for YM = M(YM). In short, given any recursive definition of a function f, which is usually given under the form f = Mf for some *definiens* term M, one can compute f by setting f = (YM).

A turning moment in the scientific role of λ-calculus was the invention of its mathematical (categorical) semantics (Scott, 1972), i.e. the construction of a "mathematical" (geometric/topological…) structure, *independent* of the formal syntax above, where signs and variables could be interpreted (that is, given a mathematical – geometric/algebraic – meaning, in terms, for instance, of elements of a topological space, functions on it etc). The semantic difficulty, of course, lies in the type-free syntax: terms may act as functions and as arguments. The construction of a reflexive object in a category of topological spaces (a CCC, see below) yields a non trivial object D, such that $(D \rightarrow D) < D$, that is of an isomorphic embedding (or, more precisely, a retraction) of its function space into D itself. More formally, $I : (D \rightarrow D)$ into D and $J: D$ onto $D \rightarrow D$, such that $J(I(f)) = f$, which clearly allows, by the (obviously injective) embedding I to interpret functions as arguments. The isomorphic embedding (or, also, in some cases, an isomorphism, i.e. with also I(J(d)) = d) is not an identity: this will be relevant for the discussion below. This work started the broad areas of the mathematical semantics of programming languages, (Stoy, 1977; Amadio, & Curien, 1993), and it has been already applied in a discussion of Rosen's approach (see Letelier et al., 2006).

4. CTEC, Cartesian Closed Categories and Differential Dynamical Systems

Before developing an expression of CTEC in terms of λ-calculus, a relevant mathematical implication should be clarified. By solving Rosen's equations in (type-free) λ-calculus, we will interpret CTEC as the constructability of a suitable reflexive object[ii] in a Closed Cartesian Category (CCC)[iii]. We will then discuss some of the relations of these categories with the usual mathematical modelling of physics in smooth manifolds, an issue raised by many authors (quoted on place) concerning

Rosen's approach.

A category **C** is a CCC if it has all finite products (and hence has a terminal object); and for every pair of objects, say A and B, there exists an exponential or map object A→B, which represents, within the category, the collection of all maps from A to B, **C**(A,B). Very importantly, products and exponentials are related by a (natural) isomorphism **C**(_xA,B) ≈ **C**(_,A→B) (see (Asperti & Longo, 1991), among others).

The relevance of CCCs for representing CTEC consists in the fact that they allow the introduction of maps acting on maps (more precisely, maps' objects), which is a key feature of Rosen's proposal. In particular, they provide a suitable formalism for modelling typed and type-free λ-calculus and, more specifically, "reflexive objects". As a matter of fact, given a reflexive object in any category, one can construct out of the object a (non trivial) CCC, as a (full) subcategory of the given category (Longo & Moggi, 1990). Thus, by these further steps, our treatment of Rosen's equations is done in a CCC, since once the CTEC equations are solved in a type-free model, this automatically yields a CCC. It also happens that reflexive objects may be found in Cartesian Closed Categories of effective objects, in the sense of computability theory; in particular, in Hyland's Effective Topos (Longo & Moggi, 1991)[iv]. Moreover, the entire construction can be carried on in fully effective (computable) topoi.

Wolkenhauer & Hofmeyr (2007) make several interesting remarks concerning the "difficult" interplay between purely mathematical modelling using (partial) differential equations on one hand, and computability structures (such as the λ-calculus models we present here) on the other. In particular, they mention the fact that there is in principle no way to have CCCs over smooth manifolds, the latter being the natural frame for differential equations. The λ-calculus solutions to CTEC we present below necessarily involve reflexive objects and hence yield CCCs; it follows that they are not compatible with smooth manifolds.

It may be useful to further elaborate on the remarks in Wolkenhauer & Hofmeyr (2007). In a CCC **D**, by definition, one has a (natural) isomorphism between **D**(AxB, C), the morphisms from AxB to C, and **D**(A, B→C), where B→C is the exponent object representing D(B,C) within the category **D**. An immediate consequence of this isomorphism over topological spaces is that any component-wise continuous function is

globally continuous. This is false, in general, over smooth manifolds. Finally (and even more crucially), in a reflexive object A, the embedding of A→A into A also implies that AxA can be isomorphically embedded into A (Longo & Moggi, 1990). This is strictly in contrast with the notion of "dimension as a topological invariant"[v], which holds in smooth manifolds with the "natural" or interval topology. This notion of dimension is crucial in Physics; and these smooth manifolds are the mathematical structures typically used to model physical processes in general, and SDDS in particular[vi].

5. The expression of closure in lambda calculus

We are now in a position to address the core of this article: an expression of closure to efficient causation in λ-calculus terms. In type-free λ-calculus notation, Rosen's equations (E1, E2 and E3) become:

$(fA) = B$ (L1)

$(\Phi B) = f$ (L2)

$(Bf) = \Phi$ (L3)

Thus, by replacing B in (L2) and (L3) and then Φ in (L2), one has:

$((fA)f)(fA) = f$

How can such an f be constructed in type-free λ-calculus, once A is given? Let now:

$G = \lambda x.((xA)x)(xA)$

and

$Y = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$ (the fixed point operator).

As shown above, for *any* term M, one has M(YM) = YM, that is, Y produces a fixed point for M. In this particular case then, G(YG) = YG, that is for:

$f = YG$

one has:

$f = Gf = ((fA)f)(fA)$

and then:

$((YG)A) = B$

$(B(YG)) = \Phi$

or also:

$f = YG$

$B = (YGA)$

$\Phi = ((YGA)(YG)))$

Hence, given A and once defined $G = \lambda x.((xA)x)(xA)$, one has the result that f, B and Φ are all defined in terms of A and of the fixed point operator Y[vii].

Self-application is a crucial "circularity feature" of type-free λ-calculus: it makes it possible to define recursion by a strong form of fixed point, the Y operator above (this operator is definable thanks to the "xx" occurrence, a key type-free term). As we showed, it is then possible to encode Rosen's diagram in the formalism of λ-calculus by generating terms, which can work alternatively as functions or arguments. Note though that the λ-calculus algebra is *non-commutative*, which means $xy \neq yx$, in general. Accordingly, x on the left has not the same role as the x on the right, and this clearly shows up both in the operational and in the mathematical semantics. In xx, for instance, the first x is interpreted as a function acting on the argument x (or, the "same" x is interpreted as function *or* argument, *according to its position*). In other words, the position and the parentheses give different *mathematical meanings* to terms, and the reading of a term makes it possible to reconstruct the different roles played by its sub-terms. Following the above topological interpretation, à la Scott, if x is interpreted by the element d of D, then (xx) is interpreted by J(d)(d); dually, if x is interpreted by the function g in $(D \rightarrow D)$, then (xx) is interpreted by g(I(g)). So, for example, in $(fA) = B$ one understands that f acts, as a function, on argument A to produce output B. The type-free structure though, makes it possible to change the role and obtain a different term, (Af) or (Bf) or whatever, both legal, as we have no typing constraints, but with *different meanings* (in the precise sense of different mathematical interpretations: (Af) is different from (fA), yet both are possible). In conclusion, the operational meaning of the efficient cause f as acting on the material cause A is fully preserved by the non-commutative structure of the λ-calculus and its mathematical interpretations[viii]. In other

words, there is an *interpretation* of the formalism we gave above which is consistent with the diagram depicted in Figure 1.

The crucial implication of our demonstration is that, to the extent that λ-calculus is a canonical formalism for computability and programming, there are no conceptual or principled problems in realizing (programming) our formalism in a physical machine. In particular, as we will discuss in more detail in section 8, there are no conceptual problems in writing an application in which i) the three terms f, B and Φ work as *programs*; ii) each of them is a *result* produced by one of the other programs. Overall, this creates a situation where Rosen's "closure to efficient causation" is deployed as a computer program f which writes a program B which writes a program Φ which writes the program f…. a quite ordinary circularity in functional programming. As a matter of fact, this is just the functional core of general recursion (Barendregt, 1984).

6. Rosen's conjecture revisited

According to Rosen (1991), no model of closure to efficient causation (CTEC) could be Turing-computable: in a previous publication (Stewart & Mossio 2007) this is what we have called "Rosen's conjecture". Since the expression of closure to efficient causation in terms of λ-calculus implies that CTEC *is* computable, in a mathematically well-defined sense, it follows immediately that there is a conceptual divergence between our proposal and Rosen's conjecture, which clearly calls for comment. In order to identify the possible reasons for this difference, let us recapitulate the logic of Rosen's demonstration.

Rosen's analysis of the computability of life phenomena may be split into two main aspects. Firstly, Rosen makes a fully general (and fundamental) remark:

> "The assertion that formalizations suffice in the expression of Natural Law, and hence, that causal entailment is to be reflected entirely in algorithms, is a form of *Church's Thesis*... If it were true, the consequences that follow from its truth would clearly have the most staggering implications for all aspects of human thought. For good or ill, however, it is not true, not even in mathematics itself." (Rosen, 1991, p. 191).

The form of Church's Thesis mentioned here by Rosen is usually called the "physical Church Thesis", and we entirely agree with Rosen's claim about its failure[ix]. The second aspect arises when Rosen continues, on the next page:

> "If f is simulable, then there is a Turing machine T such that, for any word w in the domain of f, suitably inscribed on an input tape to T, and for a suitably chosen initial state of T, the machine will halt after a finite number of steps, with f(w) in its output tape" (Rosen, 1991, p. 192).

Rosen proceeds by proving that CTEC, which he takes as a key property of life, is a Natural Law, which does not satisfy the (physical) Church Thesis. In other words, no model of closure to efficient causation could be Turing-computable. Rosen correctly points out that the cycles defined by the diagrammatic approach to CTEC produce a regression to infinity. More precisely, Rosen's demonstration of the theorem is based on the idea that if a (natural) system has a model which is Turing-computable, then its elements are fractionable. This means that different occurrences of the same element correspond to different states of the system, which have to be physically separated. Rosen offers then a *reductio ad absurdum* argument showing that that if we try to build a closed path of efficient causation with fractionable elements, we fall into an infinite regress in the definition, because iterated fractioning requires an operationally infinite behaviour (see below)[x]. Therefore we do not obtain a simulable function (Rosen, 1991, p. 238-241).

In contrast to Rosen's argument, we have shown in the previous section that CTEC can be expressed in the form of equations (as can all diagrams in Category Theory), and that those equations do have adequate computable (i.e. algorithmically representable, or "reflected in algorithms", in Rosen's terms) solutions through (partial) algorithms, as given by the λ–calculus implementation of General Recursion, by a strong fixed-point operator. Crucially, *circular* processes such as CTEC may give rise to non-halting computable cycles, which are an unavoidable component of general recursion, beyond primitive recursion (see Rosen's sound distinction in the Appendix, footnote xxi). These cycles are beautifully represented by λ-calculus, which computes recursion (and, thus, also diverging computations) by the fixed-point operator. This makes it possible to write programs – *finite* strings of signs, no more no less than Rosen's equations – which

formally describe the limit process of causal closure in Rosen's sense. In a word, Rosen's *definitional* infinite regress is perfectly handled by recursion, in particular as formalized in λ-calculus. At the same time, those programs simulating the closure may potentially activate an operationally infinite behaviour[xi].

Moreover, thanks to the richness of the formalism, in λ-calculus the (potentially infinite) operational nature of λ–terms is fully displayed (and explained) by the notion of the *Böhm-tree* of a λ-term (a rather complex definition, see Barendregt, 1984). A Böhm-tree may be an infinite tree. In particular, when λ-terms encode general recursion, which includes partial functions, the corresponding Böhm-trees may be infinite, as they display the operationally infinite behaviour of the intended computations. Yet, *infinite* Böhm-trees associated to λ-terms are recursively enumerable and effectively generated, i.e. the regression is effectively given (by a program if desired). In particular, one may have finitely branching infinite trees, which implement Rosen's fractionability[xii]. Böhm-trees have been used in the semantic analysis of programs, precisely because they display their computational behaviour as being possibly circularly infinite (Barendregt & Longo, 1980).

A further aspect concerning Rosen's own demonstration may be worthy of comment. His demonstration that a computational implementation of CTEC leads to an infinite regress (Rosen, 1991, pp 238-241) is based on a rather peculiar version of closure to efficient causation, in which a single term is the efficient cause of two different objects. Accordingly instead of equation L3: $(Bf) = \Phi$, Rosen used a variant L3': $(fB) = \Phi$[xiii]. This is in no way necessary for closure to efficient causation and it simply means that f has value B on A and Φ on B. Of course, we can deal also with this variant in terms of λ-calculus:

$(fA) = B$

$(\Phi B) = f$

$(fB) = \Phi$.

Consider:

$(\Phi B)B = \Phi$,

from the last two equations. Then, $G = \lambda y.\lambda x.(xy)y$ gives:

$\Phi = Y(GB)$.

Similarly, from the first two equations, $H_A = \lambda y.\lambda x.(yx)A$ gives:

$B = Y(H_A\Phi)$.

Thus, one obtains:

$\Phi = Y(G(Y(H_A\Phi))$.

By a further application of the fixed-point method to $L = \lambda x.Y(G(Y(H_Ax))$, one has :

$\Phi = YL$

thus:

$B = Y(H_A(YL))$

and:

$f = \Phi B$.

7. Recursion vs. Impredicativity

In recent years, several contributions have tried to restate and justify the claim of the (non-) computability of Rosen's diagram (Chemero & Turvey, 2007; Chemero & Turvey, 2006; Louie, 2007; Louie & Kercel, 2007; Louie, 2006). In his last work, Rosen himself claimed that "there is no algorithm for building something that is impredicative" (Rosen, 2000, p. 294). Indeed, according to Louie and co-workers, the crucial point is that the closed path of efficient causation described by Rosen forms a hierarchical cycle of containment in the natural system, which corresponds to an "impredicative" cycle of inferential entailment in the formal model. And impredicativity, these authors argue, is (supposedly) not compatible with computability. In a similar vein, Chemero and Turvey claimed that models of systems closed under efficient causation "contain impredicativities, and, therefore, are not computable" (Chemero & Turvey, 2006, p. 13. See also Chemero & Turvey, 2007)[xiv].

With respect to this claim, we would develop two arguments. The first one is that Rosen's closed path of efficient causation is not an impredicative cycle. The second one

is that, even if it were the case, an impredicative cycle would still be computable. Let us briefly discuss the two issues.

An impredicative definition defines sets or types or elements of a set (or of a type) in terms of the set (or type), which is being defined[xv]. Accordingly, Rosen's definitions are circular in the usual sense of Recursion Theory (or of non-well-founded Set-Theories, see below), but not impredicative, because the circularity (or apparent regression to infinity) shows up only at the level of the *terms* and their mutual definition, but not at the level of the set (or type), which is being defined. The mutual definitions in equations E1, E2 and E3 – and the condensed form: f = f(f) = ff, in proper λ-calculus notation, mentioned by (Letelier et al. 2003) – are circular, and indeed recursive, but they are not impredicative. As a matter of fact, we have shown directly that they can be modelled quite simply in type-free λ-calculus (take G = λx.xx; then f = YG = G(YG) = ff = f(f)), a theory that predicatively lives in Martin-Löf Type Theory (Aczel, 1988). An alternative formalization of Rosen's diagram, as hinted by Chemero and Turvey (Chemero & Turvey, 2008), may be provided by the (x∈x) circularities, at the core of non-well-founded Set Theory, the Theory of Hyper-Sets, which turns out to be consistent, under restricted negation[xvi]. Note that, as proved in (Lindström, 1989), one can fully reconstruct the Theory of Hyper-Sets, thus the (x∈x) circularity, in predicative Type Theory and this in a predicative fashion. In conclusion, both the (xx) and the (x∈x) circularities are perfectly *predicative*, if treated in a rigorous mathematical framework.

The second point is that impredicative structures may be perfectly computable, as is the paradigmatic example, the Impredicative Second Order Type Theory, system F (see Girard et al., 1989). System F is the most relevant example of an impredicative logical frame. Far from being incompatible with computability, it has actually been a rigorous formal tool for characterizing a large class of *computable* functions, the recursive functions that are provably total in Second Order Peano Arithmetic (PA_2). This class, as defined in system F, provided an effective logical frame for the design of typed (polymorphic) programming languages (Cardelli & Longo, 1991). As further hinted in the note, system F is impredicative to the extent that some terms and types, as collections of terms, are defined by using a universal quantification over the very collection of types that is being defined. The terms in the impredicatively defined types,

in particular, are defined by using the type to which they belong and, even, the collection of all types[xvii]. In conclusion, the argument, "since this is impredicative, then it is not computable", is incorrect.

Nevertheless, the xx and (x∈x) circularities, which are at the core of type-free recursion and non-well-founded Set Theory, are very expressive; and in fact they do have a non-obvious mathematical connection with impredicativity. Within type-free λ-calculus and its "models", such as Scott's D_∞ models (see Barendregt, 1984), one can construct an Impredicative Theory of Types. This can be done in a relatively "simple" way, based on the "finitary projections" model in (Amadio et al, 1986). It can also be done in a much more complex way, which preserves the II order *logical* structure of Girard's impredicative system. That is, within a type-free model, one can "isolate" types (as partial equivalence relations), which forms an impredicative type-structure and satisfies Lawvere Topos-Theoretic understanding of quantification (Longo & Moggi 1991; Asperti & Longo, 1991)[xviii]. Similarly, models of the type-free λ-calculus (thus of xx) yield (approximated) models of non-well-founded set-theories, thus of (x∈x). Conversely, some approximated recursive domain equations, that is (approximated) models of (xx), may be given in Hyper Sets or Hyper Universes (see Forti et al., 1994; Longo, 2000 for a survey). This is why one can equivalently treat Rosen's circularities either by type-free λ–calculus (the (xx) circularity) or by Hyper-Universes (the (x∈x) circularity).

Finally, one should observe that all of this can be made fully effective. Scott's D_∞ models may be constructed as effective limits of recursively enumerable chains of recursive enumerable sets and the entire Type Theoretic construction can be fully effectivized (Giannini & Longo, 1984). This completes our second argument according to which even if closure to efficient causation did involve impredicativity (which is not the case, in Rosen's formalization), this would still not prove that it is not computable, since also Domain Theoretic solutions of recursive domain equations (typically, Scott's D_∞ models) are perfectly computable and, over them, one may construct models of (effective and) impredicative Type Theories. Their computer implementations are at the core of a large area of functional programming and its applications.

8. Towards a computer simulation of closure to efficient causation

The motivation for this article, as stated in the introduction, is to work towards a mathematical formulation of Rosen's concept of "closure to efficient causation" that would be intuitively understandable, biologically interpretable, and operationally generative. Rosen himself employed Category Theory, an extension of standard Set Theory in which mappings can themselves form objects. The type-free λ–calculus, at the syntactic level, and its categorical models, as mathematical/semantic interpretation, do indeed fulfil the key requirement that the *same* entity be able to occupy the three roles of *argument* (i.e. a set which is the input to an application or mapping), *result* (i.e. the set which is the end-point of the mapping) and *function* (i.e. the mapping itself). However, from the working theoretical biologist's point of view, this formalism has a severe limitation. It is, in a way, *too* general; once written down, either in category-theoretical terms or as one of Rosen's "relational diagrams", it just "sits there" and doesn't actually *do* anything. Rosen himself commented that the absence of explicit dynamics in these diagrams was no accident, because the diagrams represent the *organizational* features of living organisms that, as long as the organism stays alive, remain *invariant.*

This is of course extremely frustrating from the perspective of the traditional models of mathematical biology, which are almost entirely framed in terms of dynamical systems. Rosen (1973) did make an explicit attempt to articulate the relational theory of (M,R)-systems with traditional differential equations. Unfortunately, this proposal has not proved usable. Letelier et al. (2006) have made a most commendable attempt to put Rosen's category-theoretical formulation to work; but the actual results, so far, are not engaging. The key item in this formalism is what Rosen calls the "replication map", in his notation $b = \hat{}b^{-1}$. Letelier et al. (2006) only manage to provide an illustration of this for a simple arithmetical example. This does have the value of an "existence proof", showing that closure is both mathematically possible but non-trivial; but Letelier and colleagues recognize that this example is of little biological interest. These authors also make a most interesting attempt to make this formalism "work" in the case of a simple metabolic system; but unfortunately this attempt fails, on their own admission.

In this sort of situation, where there are strong qualitative intuitions but conventional dynamic systems theory is not able to express them, a most notable resource that has become increasingly available over the last 40 years is that of computer simulation. For example, the concept of autopoïesis (Maturana & Varela, 1980) – which, as we mentioned, has been compared to that of closure under efficient cause – has received several computational implementations (McMullin, 2004). The work of Fontana & Buss (1994), to which we have already referred[xix], is also explicitly inspired by the work on autopoïesis. To date, there have been virtually no attempts to provide an illustration of closure to efficient causation by means of a computer simulation. This is quite understandable, if we consider the position not only of Rosen but also of many other authors, who have been adamant in insisting that models of closure under efficient cause are intrinsically non-computable. If they were right, any attempt in this direction would indeed stand condemned in advance of having missed some essential feature. The thrust of the present article is to suggest that although it is quite understandable, this position may be mistaken; and indeed it may actually be misguided and counter-productive, since it has hindered attempts to develop simulations of closure under efficient cause.

If correct, the work presented in this article is not a final conclusion, the end of a road; on the contrary, it opens up a new perspective. Our formal demonstration that Rosen's equations do have a solution when expressed in terms of λ-calculus does not imply that this solution is biologically interesting; f = YG (where Y and G are the λ-calculus terms given above) is mathematically optimal (in a precise sense in view of its interpretation by least fixed points in Scott domains), yet it is not (necessarily) heuristically suggestive of "metabolism" as a biological phenomenon. The value of this solution is simply that of an existence-proof: computable solutions involving CTEC do exist, and we have exhibited one. It is essential to realize that this solution is absolutely not unique: there exist an unlimited number of more complex solutions, some of which may (and hopefully will) be susceptible of biological interpretation. The practical challenge now is to write computer programs in which the functions f, Φ and B are seriously interpretable as doing justice to their biological inspiration, i.e. "metabolism", "repair" and "replication".

As we have already indicated at greater length (Stewart & Mossio, 2007), we consider that a computational implementation of closure to efficient causation could best take the form of three computer programs, each of which writes the next one[xx]. The program "metabolism" – in barest outline, (fA) = B – takes a suitable input A: the material and energetic resources, which every living organism needs, as a thermodynamically open system functioning far from equilibrium. This function produces B – in the first instance, simply the whole network of biochemical reactions (but we will come back to the requirements on f). Working backwards, as it were, the second task is to write a program "repair" which, taking suitable input, will *write* the "metabolism" programme. Rosen writes (ΦB) = f, suggesting that B (in its role as an argument) may be a suitable input; but while this is not implausible we do not ourselves see a need to impose this as a constraint. We could, if necessary for plausible biological interpretation, write

(ΦX) = f (L2bis)

where X is any plausible resource (including A and B but possibly also further environmental resources). Relaxing the constraints in this way only multiplies the number of solutions to the three equations; but of course we now have to satisfy a new sort of constraint, i.e. biological plausibility.

Working backwards again, the third step is to produce Φ by the « replication » function. Rosen writes: (Bf) = Φ, implying that the replication program takes f as input argument. This seems to us not only unnecessary, but also biologically quite implausible. To a first approximation we may consider that the metabolism function f includes enzymes (even if it is not reduced to that). Φ, as "replication" function, may be associated with nucleic acids. While the synthesis of nucleic acids clearly *requires* enzymes (as a part of the efficient cause of the process), it does not take enzymes (nor indeed proteins) as its *substrate.* We therefore prefer to write

(BZ) = Φ (L3bis)

where Z is again the sum of all plausible resources (including A, B, indeed f if that should be a good idea, and again possibly also further environmental resources). As before, relaxing the constraints formally multiplies possible solutions – but with the cost that now they should be biologically plausible.

We are not yet finished, however, because now that B is a program in its own right, the "metabolism" function (fA) = B has a much heavier task than initially envisaged. B, produced by f acting on A, must now represent not only the full network of biochemical reactions which occur in the cell; B must *also* be a *program* which, given the right input, will produce Φ. This is, of course, a new constraint on f; and in turn this becomes also a new constraint on Φ which must produced the new f. Thus, the set of three programs impose mutual constraints on each other, in a cyclical fashion. This does not, however, involve a vicious infinite regress, as we have shown; on the contrary, it is just a nice challenge for theoretical biologists with programming capabilities. The guidelines we propose are to follow the structure of the λ-calculus formulation, and to translate our terms into LISP programs (which are identical to the λ-calculus up to some added "syntactic sugar", as computer scientists say), or into any preferred (type-free) programming language.

9. Conclusions

As a final remark, we wish to note that although most workers in the field would probably agree that "closure to efficient causation" is a *necessary* condition for a living organism, as Rosen himself noted in his last work (Rosen, 2000) it may not be *sufficient*, for a fully satisfactory theoretical definition. And it may be, too, that such a full model would not be computable. The natural ecosystem of metabolic pathways is the turbulent cytoplasm of a living cell. Dynamical systems and dissipative structures of this sort are better understood in space-time continua, where (differentiable) dynamics are mostly analyzed by perturbative methods or geometric models. Dynamical notions such as sensitivity to initial conditions, topological transitivity and so on are essential. It has been noted by many that sufficiently chaotic dynamics are *not* approximated by discrete simulations; the latter yield informative, but *different* mathematical structures and evolutions (Longo & Paul, 2008; the "approximation", if any, goes the other way-round: continuous evolutions may approximate discrete dynamics, see the "Shadowing Lemma", see Pilyugin, 1999). Furthermore, there is the question of "local vs. global" causal entanglement which is proper to living systems in their many levels of organization, but which may also pose an obstacle to computability: the Theory of Criticality in Physics presents singularities which may be highly non-computable

(Bailly & Longo, 2006 & 2008). We believe in fact that criticalities and singularities are at the core of life phenomena, as well as circularities and "resonances" between different levels of organisations (Bailly & Longo, 2006 & 2008). These may be hardly expressed both in SDDS and Computational Models, even though there is no fully formal argument for such a "negative result".

To sum up: it may well be that a full model of "life itself" is not computable; but if so, the reason would *not* be the closure to efficient causation as expressed by Rosen. In fact, as we have shown, an equational presentation such as Rosen's *naturally* leads to λ-calculus terms, a paradigmatic functional frame over *discrete data types*. Biological invariance is turned into perfect computational iteration (this is at the core of discrete computation and λ-calculus in particular, under the form of recursive definitions). And to reiterate our conclusion, the fact that closure to efficient causation is computable, according to a standard mathematical definition of the term, in no way disqualifies it as a fundamental contribution to a theoretical definition of life.

Acknowledgements

The authors wish to thank Dr. Boris Saulnier, whose curiosity and scientific talent stimulated their investigations on the work of Rosen some years ago. Most of Longo's papers are downloadable from http://www.di.ens.fr/users/longo.

Appendix: On partial vs. total computable functions

A possible interpretation of Rosen's claim about non-computability of CTEC may concern the essential divergence of cycling computations. As a matter of fact, our proposed solution to the definitional circularity of CTEC, which Rosen claims to yield non-computable functions, actually gives computable, yet non-halting cycles. One may then argue that the apparent discrepancy between our result and Rosen's conclusions stems from the fact that he restricts his demonstration to total computable functions[xxi], and does not include the case of partial functions[xxii]. In this case, Rosen's demonstration would be formally correct in its own terms; but its validity would depend on a restricted definition of "computability", which actually excludes the class of partial computable mappings, in particular those that happen to compute *cycles*.

In our view, understanding the computability of CTEC (and the long-lasting discussion this issue has engendered) requires recalling the distinction between *diverging computations* and *non-computability*. Whereas the former refers to computations which do not satisfy the halting condition imposed by Rosen, the latter is related to the *undecidability of the halting problem* (i.e. the inexistence of an algorithm uniformly and effectively deciding for any machine and input whether the machine stops on it) formulated in the famous paper by Turing in 1936. As a matter of fact, Turing's halting theorem shows two facts. First, the mathematically well-defined, total function deciding the general halting problem is not computable. Indeed, Classical Mathematics is full of well-defined, total, yet non-computable functions. Second (and as a consequence), classical (sequential) Theory of Computability essentially deals with *partial* computations, i.e. with computable functions, which diverge (do not halt) on some or *all* inputs. Of course, computations occur in discrete time and, thus, *if* they halt, they halt in finite time (Rogers, 1967, p. 5). Nevertheless, divergence is essential for computability, since not every partial computable function may be extended to a total computable function (a key result, see Rogers, 1967).

There is one more fundamental reason for Computability Theory to deal with *partial* recursive functions in an essential way (Rogers, 1967; Barendregt, 1984). The reason is that the class of partial functions is effectively enumerable, whereas the class of total computable functions is not. Moreover, and this is crucial, the enumeration of the class of partial functions gives the Universal Turing Machine, which enumerates and computes all of them. More precisely, partiality is essential to obtain a set of indexes, which is effectively *recursively enumerable* (r.e.). By contrast, any non-empty sub-set of the total computable functions has a *non* r.e. set of indexes, as shown by the Rice-Shapiro Theorem (Rogers, 1967, p. 324). In fact, even the constant or the primitive recursive functions (which contain no universal primitive recursive function) have non-r.e. sets of all r.e. indexes. In contrast, there is no way to develop a Theory restricted to *total* and *computable* functions which would contain their universal function, a key theoretical and practical property of computability[xxiii].

To sum up, there is no expressive Theory of Computability restricted to total functions, nor even of any significant subclass of total functions: the impossibility of an effective enumeration of all programs, and the inexistence of internal universal functions, forbid

developing such a theory.

References

1. Aczel, P., 1988. Non-wellfounded sets. CSLI Lecture-Notes 014.
2. Amadio, R., Curien, P.L., 1998. Domains and lambda-calculi. Birkhuaser.
3. Amadio, R., Bruce, K. Longo, G. 1986. The finitary projection model for second order lambda calculus and the solutions to higher order domain equations. First IEEE Conference on Logic in Computer Science (LICS 86). Boston, 122–130.
4. Asperti, A., Longo, G., 1991. Categories, Types and Structures. MIT Press, Boston.
5. Bailly, F., Longo, G., 2008. Extended Critical Situations. Journal of Biological Systems, Vol. 16, No. 2, pp. 309-336.
6. Bailly, F. Longo, G., 2006. Mathématiques et sciences de la nature. La singularité physique du vivant. Hermann, Paris.
7. Barendregt, H., 1984. The Lambda Calculus: its syntax and semantics, Revised and expanded edition. North Holland, Amsterdam.
8. Barendregt, H., Longo, G., 1980. Equality of lambda terms in the model T-omega. In: Seldin, J., Hindley, R. (Eds.), To H.B Curry: Essays in Combinatory Logic, Lambda-calculus and Formalism. Academic Press, London, pp. 303-337.
9. Cardelli, L., Longo, G., 1991. A semantic basis for Quest. Journal of Functional Programming, 1 (4), 417-458.
10. Chemero, A., Turvey, M.T., 2008. Autonomy and hypersets. Biosystems, 91 (2), 320-330.
11. Chemero, A., Turvey, M.T., 2007. Hypersets, complexity and ecological approach to perception-action. Biological Theory, 2 (1), 23–36.
12. Chemero, A., Turvey, M.T., 2006. Complexity and “Closure to Efficient Cause”, Proceedings of the 10th International Conference on Artificial Life, Bradford Books, 13-19.

13. Chu, D., Ho, W.K., 2007a. Computational realizations of living systems. Artificial Life, 13 (4), 369–381.

14. Chu, D., Ho, W.K., 2007b. The Localization Hypothesis and Machines. Artificial Life, 13 (3), 299-302.

15. Chu, D., Ho, W.K., 2006. A category theoretical argument against the possibility of artificial life: Robert Rosen's central proof revisited. Artificial Life, 12 (1), 117-134.

16. Church, A., 1940. A formalization of the simple theory of types. J. Symb. Logic 5, 56-58.

17. Church, A., 1932/33. A set of postulates for the Foundation of Logic. I and II Annals of Mathematics, XXXIII, 348-349, and XXXIV, 839-864.

18. Fontana, W., Buss, L.W., 1994. The arrival of the fittest: toward a theory of biological organization. Bull. Math. Biol., 56 (1), 1-64.

19. Forti, M., Honsell, F. 1983. Set theory with free construction principles. Ann. Scuola Norm. Sup. Pisa, Cl. Sci., 4 (10), 493-522.

20. Forti, M., Honsell, F., Lenisa M., 1994. Processes and Hyperuniverses. In: Privara et al. (Eds.), MFCS'94 Conference Proceedings, Springer LNCS, 841, 352-363.

21. Fox-Keller, E., 2001. The Century of the Gene. Harvard University Press, Cambridge, MA.

22. Giannini, P., Longo, G., 1984. Effectively given domains and lambda calculus semantics. Information and Control, 62 (1), 36-63.

23. Girard, J.Y., 1986. The system F of variable types, fifteen years later. Theor. Comp. Sci., 45, 159-192.

24. Girard, J.Y., Lafont, Y., Taylor, P., 1989. Proofs and Types, Cambridge U. Press, Cambridge, MA.

25. Hindley, R., Seldin J., 1986. Introduction to Combinators and Lambda-Calculus. London Mathematical Society.

26. Jacob, F., 1970. La logique du vivant. Gallimard, Paris.

27. Kercel, S., 2007. Entailment of ambiguity. Chemistry & Biodiversity, 4, 2369-2385.

28. Landauer, C., Bellman, K.L., 2002. Theoretical Biology: Organisms and Mechanisms. AIP Conference Proceedings, 627 (1), 59-70.

29. Letelier, J.C., Soto-Andrade, J., Abarzua, F.G., Cornish-Bowden, A., Cardenas, M.L., 2006. Organizational invariance and metabolic closure: Analysis in terms of (M,R) systems. Journal of Theoretical Biology, 238 (4), 949-961.

30. Letelier, J., Marín, G., Mpodozis, J., 2003. Autopoietic and (M,R) systems. Journal of Theoretical Biology, 222, 261–72.

31. Lewontin, R. C, Rose, S., & Kamin, L., 1984. Not in Our Genes: Biology, Ideology, and Human Nature, Pantheon, New York.

32. Lindström, I., 1989. A construction of non-well-founded sets within Martin-Löf's type theory. J. Symbolic Logic, 54 (1), 57-64.

33. Longo, G., 2008. Critique of Computational Reason in the Natural Sciences. In*:* Gelenbe, E., Kahane, J.-P. (Eds.). Fundamental Concepts in Computer Science, Imperial College Press/World Scientific, London.

34. Longo, G., 2000. Cercles vicieux, Mathématiques et formalisations logiques. Mathématiques, Informatique et Sciences Humaines, 152, 5-26.

35. Longo, G., 1983. Set-Theoretical Models of Lambda-Calculus: Theories, Expansions, Isomorphisms. Annals of Pure and Applied Logic, 24 (2), 153-188.

36. Longo, G., Moggi, E., 1991. Constructive Natural Deduction and its omega-Set Interpretation. Mathematical Structures in Computer Science, 1 (2), 215-254.

37. Longo, G., Moggi, E., 1990. A category-theoretic characterization of functional completeness. Theoretical Computer Science, 70 (2), 193-211.

38. Longo G., Paul T. 2008. The Mathematics of Computing between Logic and Physics. In: Cooper, S.B., Lowe, B., Sorbi, A. (Eds.). Computability in Context: Computation and Logic in the Real World, Imperial College Press/World Scientific, London.

39. Louie, A.H., Kercel, S., 2007. Topology and life redux: Robert Rosen's relational diagram of living systems. Axiomathes, 17 (2), 109-136.

40. Louie, A.H., 2007. A Living System Must Have Noncomputable Models, Artificial Life, 13 (3), 293-297.

41. Louie, A.H., 2006. Any material realization of the (M,R)-systems must have noncomputable models. Journal of Integrative Neurosciences, 4 (4), 1-14.

42. Martin-Löf, P., 1975. An intuitionistic theory of types. In: Shepherdson, R. (Ed.), Logic Colloqium 73, North-Holland, 73-118.

43. Maturana, H., Varela, F.J., 1980. Autopoïesis and cognition: the realization of the living. Reidel, Boston.

44. McMullin, B., 2004. Thirty years of computational autopoïesis. Artificial Life, 10, 277-295.

45. Mikulecky, D., 2001. Robert Rosen (1934–1998): a snapshot of biology's Newton. Comput. Chem., 25, 317–327.

46. Nomura, T., 2007. Category Theoretical Distinction between Autopoiesis and (M,R) Systems. Proc. 9th European Conference on Artificial Life (ECAL 2007), Lisbona.

47. Oyama, S., 1985. The ontogeny of information: developmental systems and evolution. Cambridge University Press, Cambridge.

48. Pilyugin, S.Y., 1999. Shadowing in dynamical systems. Springer, Berlin.

49. Poincaré, H., 1906. Les mathématiques et la logique. Revue de métaphysique et de morale, 14, 294-317.

50. Rogers, H., 1967. Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York.

51. Rosen, R., 2000. Essays on Life Itself. Columbia University Press, New York.

52. Rosen, R., 1991. Life itself: a comprehensive enquiry into the nature, origin and fabrication of life. Columbia University Press, New York.

53. Rosen, R., 1973. On the dynamical realization of (M,R)-systems. Bulletin of Mathematical Biology 35, 1-9.

54. Scott, D., 1972. Continuous lattices. In: Lawvere, F.W. (Ed.). Toposes, algebraic Geometry and Logic. SLNM 274, Springer-Verlag, pp. 97-136.

55. Stewart, J., 2004. La vie existe-t-elle? Réconcilier la génétique et la biologie. Vuibert, Paris.

56. Stewart, J. Mossio, M. 2007. Is "Life" Computable? On the Simulation of Closure under Efficient Causation. Proceedings of the 4th International Conference on Enactive Interfaces, Grenoble, France, pp. 271-276. Available at: http://acroe.imag.fr/enactive07/proceedings.php.

57. Stoy, J., 1977. Denotational Semantics, M.I.T. Press.

58. Weyl H., 1918 Das Kontinuum, Berlin.

59. Wolkenhauer, O., Hofmeyr, J.H.S., 2007. An abstract cell model that describes the self-organization of cell function in living systems, Journal of Theoretical Biology, 246 (3), 461-473.

60. Zaretzky, A., Letelier, J., 2002. Metabolic Networks from (M,R) Systems and Autopoiesis Perspective. Journal of Theoretical Biology, 10 (3), 265-280.

---

[i] Corresponding author. Telephone: +33.1.43.54.60.36. Fax: +33.1.43.54.60.36

[ii] A reflexive object is an object A whose object of endomorphisms, $A \to A$ is isomorphic to – or is a retraction of – A itself.

[iii] The relation between CCC and typed and type-free λ-calculus has been proposed by D.S. Scott since 1970. For a full account see Barendregt (1984), Asperti & Longo (1991), Amadio & Curien (1998), Longo & Moggi (1990).

[iv] A topos is more than a CCC, since it contains also the representation of all sub-objects, a relevant logical property.

[v] I.e. dimension is preserved by topological isomorphisms.

[vi] For recent reflections on this point, see Longo & Paul (2008), where the difficult relation between modelling of physical processes and Computability Theory is more closely analyzed.

[vii] In case the reader prefers to have a term independent of A or "model" a process that, at the beginning, uses only one A, one can also "abstract" with respect to A and set $H = \lambda y.\ \lambda x.((xy)x)(xy)$, which yields $HA = G$.

[viii] For a general set-theoretic semantics, see (Longo, 1983); for a general Category Theoretic one, see (Longo & Moggi, 1990), quoted above.

[ix] The reasons for this failure in (physics and) mathematics itself are discussed in Longo & Paul (2008). Its epistemological sense, for "human thought", is critically examined in Longo (2008).

[x] See also Tim Gwinn's analysis on this point: http://www.panmere.com/?p=79.

[xi] In the Appendix, we discuss an alternative explanation of the divergence between our results and Rosen's ones.

[xii] A paradigmatic example is the tree of the recursion operator Y above, a *finite* term, of course, whose Böhm-tree is infinite. Böhm-trees thus express the infinite regress mentioned by Rosen in a perfectly computational fashion (apply f = YG that generates B which writes Φ which generates f..., the intended metabolic *cycle*). For example, fractionability, in some cases, boils down to the effective transformation from x to *(xx)*, where the first and the second *x* are distinct and play different roles in the computation (see sect. 5 on non-commutativity).

[xiii] This variant facilitates the demonstration of infinite regress; but it is not necessary, either for CTEC or for infinite regress. It might have been preferable for Rosen to demonstrate his theorem on the diagram representing canonical (M,R) systems; but in the event, nothing hangs on this.

[xiv] In their most recent paper on the subject, Chemero and Turvey claim to have learned that "there is no necessary connection between impredicative definitions and non-Turing-computability" (Chemero & Turvey, 2008, p. 327).

[xv] As a matter of fact, in 1902, there was some confusion between *impredicative* definitions and the predicate (x∈x), that is x belongs to x, which belongs to x... a *circularity* in Frege's formalization of Cantor's Set Theory which lead to Russell's paradox (or, better, inconsistency) *in type-free theories with unrestricted negation*. The issue of impredicativity was later clarified by Poincaré (1906) and H. Weyl (1918). Since Russell's work in the Theory of Types, in the early 1900's, after his paradox, the mathematical setting where impredicativity has been rigorously analyzed is Type Theory, in particular in the modern sense of Church (1940). The work by Martin-Löf and Girard, since the '70s, represented the branching of Type Theory into a predicative frame (Martin-Löf, 1975) and an impredicative one, (Girard, 1986). It resulted that "non-well-foundedness" of Frege-Cantor Set Theory is *not* an impredicativity as it corresponds to (xx) of type-free l-calculus, x applied to x, applied to x....

[xvi] Hypersets were invented, under different names, by Finsler in the '30s and, later by D.S. Scott in the '60s. They were rigorously treated first in (Forti & Honsell, 1983), to which (Aczel, 1988) extensively refers.

[xvii] The type-theoretic notion of impredicativity is fully general, that is, this definition yields also the set-theoretic one, by changing, roughly, types into sets (and t:T, that is t has type T, into t∈T) . Second order types are defined by a universal quantification (for all X, that is ∀X) referring to the very collection of types, Type, that is being defined (formally: (∀X:Type.A) is in Type). Moreover the terms in these types, also use, in their definition, a universal quantification over the collection of all types. The relative consistency of this theory was first assured by a difficult consistency (normalization) theorem, see (Girard et al., 1989).

[xviii] The delicate logical issue, here, is that the step of "isolating" an impredicative fragment within a type-free model (which may live in predicative Theory of Types) is a highly impredicative conceptual construction.

[xix] We are not able to do justice here to the relation between the present article, and the work of Fontana & Buss which is also centred on l-calculus; this relation would require an entire article in itself, as they claim that the key circularities of life phenomena are suitably representable in l-calculus (we do not go so far and just analyse some equations derived from Rosen's approach).

[xx] We retained here the l-calculus formulation because it is perfectly concise and precise for identifying argument, result and function.

[xxi] Here, Rosen makes a sound distinction. In section 4D, and in particular equation [4D.2], he defines what he calls *recursive* functions. In the standard terminology (see Rogers, 1967), these are called *primitive* recursive functions. As Rosen writes, they satisfy the condition: " *f*(n) entails *f*(n+1) for every n". These functions are all total functions, and they form a (proper) subset of the class of total computable functions. Later, Rosen observes (p. 192): "it is perfectly possible to define mappings *f* in terms of algorithms, which do not satisfy this condition". Thus, there exist total algorithmic functions that are not (primitive) recursive, as it is well-known.

[xxii] A function is total if it *always* associates to an (finite) input a (finite) output; it is partial otherwise (it may diverge on some or all inputs).

[xxiii] Quite pragmatically, we may observe that operating systems (OS) and compilers are just programs, yet they act on programs (they are components of a Universal Turing Machine!); the key point being that in sequential computers which are never turned off, the OS and compilers are computable functions which run indefinitely. The same may be said of most network processes, which are not studied as input-output halting functions: they are ongoing computational processes, not input-output halting relations.