A Model of Pulsation for Evolutive Formalizing Incomplete Intelligent Systems

Marta Franova, Yves Kodratoff
LRI, UMR8623 du CNRS & INRIA Saclay
Bât. 660, Orsay, France
e-mail: mf@lri.fr, yvkod@gmail.com

*Abstract***— The notion of pulsation concerns a possibility of a particular kind of intelligent controlled and secured evolution in dynamic real-world systems. It is related to fundamentals in intelligent systems and applications as well as to the topic of intelligence by design. In this paper we present a model of pulsation based on Ackermann's function. This brings more clarity to understanding Symbiotic Recursive Pulsative Systems that are important, for instance, for designing and implementing intelligent security systems or for automating robots' programming in incomplete domains and unknown environments. One particular application for these systems is our Constructive Matching Methodology for automating program synthesis from formal specifications in incomplete domains.**

Keywords-pulsation; Symbiotic Recursive Pulsative Systems; intelligent systems; intelligence by design; Ackermann's function; control; security; progress; practical completeness.

I. Introduction

For more than three decades now, we worked on automation of programs synthesis in incomplete domains via inductive theorem proving [2] [8]. Our approach differs from standard computer science approaches based on modularity of developed parts. This standard is called Newtonian Science in contrast to Cartesian Intuitionism [6] that provides a basis for Symbiotic Recursive Pulsative Systems (SRPS) roughly described in [6].

The notion of SRPS is very rich and complex. In our latest work [4] [5], we are trying to progressively disentangle this symbiotic complexity by presenting notions individually (as much as possible for symbiotic parts of a whole). Such a disentangling is important for perceiving the usefulness of working on particular SRPS for real-world applications where Newtonian Science has shown its limitations.

In this paper we focus on a model for the notion of pulsation. Such a systemic approach influences the overall perception, the guidelines for research and development and elaboration of details of SRPS. We will show in this paper that meta and fundamental levels in SPRS are symbiotic. In other words, their separation leads to a non-sense or an irrecoverable mutilation. This is important for understanding our work on Constructive Matching Methodology (*CMM*) for automating program synthesis from formal specifications in incomplete domains via inductive theorem proving.

The paper is organized as follows. In Section II, we recall the definition of symbiosis we work with and we present an example illustrating symbiosis of information that is present in recursive representations. In Section III, we present a way to construct Ackermann's function and to replace, for given two numbers *a* and *b*, its non-primitive recursive computation by a computation via an on-purpose generated sequence of primitive recursive functions that has to be used for *a* and *b*. In Section IV, we show that prevention and control can be modeled by Ackermann's function. Section V shows that even pulsative systems can be modeled by Ackermann's function. In Section VI, we speak about pulsative development of our Constructive Matching Methodology. Section VII presents an example of a technological vision for which the work presented in this paper is important.

II. Symbiosis of information in Recursion

As specified in [6], by symbiosis we understand a particular composition of two or several parts that make an indivisible whole. In other words, a separation of one sole part is a reason for extinction or for irrecoverable mutilation of the all other parts as well as the whole.

Let us point out that we speak here of symbiotic information and not of symbiotic computation.

Let us consider the following simple problem. On a sufficiently big table consider a stack of blocs a, b, c, d and e as shown in Figure 1.

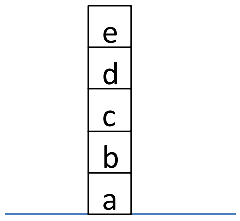

Figure 1. The stack of blocks before the intended action is taken

We say that a bloc m is clear if there is no other bloc on m. (In Figure 1, the bloc e is clear.) There can be at most one bloc on the top of the other. If n is on the top of m we say that n is top of m written as: n = top(m). Let us consider the primitive recursive procedure "put on table" as being hardware defined in the robot that will execute the following informal primitive recursive program makeclear:

```
makeclear(x) =
  if  x is clear then procedure ends
  else
    if    top(x) is clear
    then  put(top(x)) on table
    else  first makeclear(top(x)) and
          then    put(top(x)) on table
```

It can easily be checked that makeclear(b) results not only in clearing bloc b but also in the situation where blocs c, d and e are on the table. This means that the procedure makeclear contains in its description not only its direct effects (such as: the bloc b is cleared) but also the full description of all the secondary effects of any action performed. In Figure 2, these secondary effects are that the blocks c, d and e are on the table.

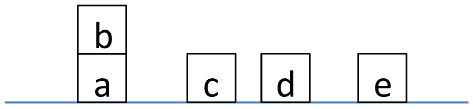

Figure 2. The stack of blocks after the action 'makeclear' is taken

For some other primitive recursive procedures the secondary effects do not modify the environment, but this should not be a barrier for general perception of primitive recursive procedures to be seen as invisible procedural 'seeds' containing symbiotically related the effects (i.e., the results of the computations) and the secondary effects (i.e., the consequences of the computation of a particular value). Therefore, implementing recursive procedures is interesting in all the environments where the control over the secondary effects is important.

The above procedure makeclear is an example of primitive recursion. A recursion that is not primitive goes even further in representing symbiotically information that concerns control, rigor and reproducibility. Ackermann's function is a suitable representative for explaining how non-primitive recursion modelizes a particular kind of pulsation in SRPS.

In the following sections, we shall give a formalized presentation of the pulsation starting by a presentation of a construction procedure that results in Ackermann's function. It will become clear how this construction and the notion of pulsation are linked together. Then, we shall present a practical application of this notion.

III. ACKERMANN'S FUNCTION

The idea to model pulsation by Ackermann's function comes from the understanding how this function can be constructed. The practical use of this function becomes then exploitable by a particular 'simplifying' the computation of its values.

A. A Construction

Let ack be Ackermann's function defined by its standard definition, i.e.,

$$ack(0,n) = n+1$$
$$ack(m+1,0) = ack(m,1)$$
$$ack(m+1,n+1) = ack(m,ack(m+1,n)).$$

We shall show here how this function can be constructed.

By definition, each primitive recursive function f is a composition of a finite number of primitive recursive functions and of f itself.

Since ack is a non-primitive recursive function (see a proof in [12]), by definition of non-primitive recursion, it is a particular composition of an infinite sequence of primitive recursive functions. We shall build a function ack' as a particular composition of an infinite sequence of primitive recursive functions built so that the definitions for ack and for ack' (defined below) are identical.

Let us construct such an infinite sequence of primitive recursive functions f_0, f_1, f_2, …, f_n, f_{n+1}, …. respecting the following relationships

$$f_0(n) = n+1$$
$$f_{i+1}(n+1) = f_i(f_{i+1}(n))$$

for each i from 0, 1, 2 …. We are thus able to define a new function ack' as follows:

$$ack'(0,n) = f_0(n) \text{ and}$$

$$ack'(m+1,n+1) = f_{m+1}(n+1).$$

This definition is still incomplete since the value for ack'(m+1,0) is not yet known.

Since we want ack' to be a non-primitive recursive function, we need to guarantee that it cannot be reduced to any of f_i. In order to do so, we shall simply perform a progressive diagonalization on this infinite sequence of functions by defining the value of $f_{i+1}(0)$ as being the value of f_i in 0+1, i.e.,

$$f_{i+1}(0) = f_i(1).$$

In other words, we define

$$ack'(m+1,0) = f_m(1).$$

By this construction we see that f_{i+1} is more complex than f_i for each i. It is obvious that

$$ack'(m,n) = ack(m, n) = f_m(n).$$

This construction is at the same time a guarantee that ack is not primitive recursive, since it is indeed a composition of an infinite sequence of primitive recursive functions each of them more complex than those before it and ack cannot be reduced to any one of them. As a by-product, we have thus simplified also the standard presentation of the non-primitive character of ack, which is usually done by a proof by a projection of Ackermann's function ack into a sequence of primitive recursive functions $a_m(n) = ack(m,n)$ and showing that ack grows more rapidly than any of these primitive recursive function (see [12]). The difference thus lies in our use of an indirect construction (instead of a projection) and relying on a particular diagonalization.

To our best knowledge, this construction with a use of progressive diagonalization was not presented so far. Note that the notion of pulsation that refers to this construction of Ackermann's function has no relation to measures of the computation complexity of a function, such as Ritchie's hierarchy [11].

B. A 'Simplification' of the Computation

The above construction of the Ackermann's function shows immediately that the computation of its values for given m and n using non-primitive recursive definition can be 'simplified' - or, rather, replaced - by a definition of m primitive recursive functions obtained by a suitable macro-procedure.

Our recursive macro-procedure will simply compute, step by step, each of the values $f_{i+1}(0)$ (for $i < m$) in advance and will define the whole f_{i+1} with this already computed value. This may not lead to a fast computation but we are not concerned now with computational efficiency of this way of proceeding, only by its practical feasibility and reproductiveness. In no way is our presentation an attempt to optimize Ackermann's function. On the other hand, computing in advance some values is a known technique, we have just adapted it here for our macro.

Note that there exists efficient algorithms that go further with the computation of the values of Ackermann's function than our macro-procedure, but these known algorithms are based on a relation of Ackermann's function with a kind of usual exponentiation function. Our way of proceeding is thus useful for practical applications that will be based on use of SRPS. Indeed, not many practical applications (such as security information system or robots programming themselves in unknown environments, for instance) can be modeled by exponential functions. Therefore, our macro, even though less efficient, aims at general use of different systemic non-primitive recursive functions in the framework of SRPS.

We define a macro-procedure, ack_macro, that uses a standard program of LISP which adds a text at the end of the file that will contain the programs generated by ack_macro. We thus create an auxiliary file F that stores the functions f_i generated by ack_macro. Our ack_macro uses thus the LISP procedures add_to_file and load_file. The procedure add_to_file(*text*,*F*) adds the *text* at the end of the file *F*. The procedure load_file(*F*) loads the file *F* in order to make computable the functions written in the file. Our macro-procedure ack_macro(m,n) uses the infinite sequence of functions defined above as being representative of Ackermann's function.

```
Step 1:
  text:= { $f_0(n) = n+1$ }
Step 2:
  Create the file F (empty at start) and
  add_to_file(text,F)
  load_file(F)
Step 3:
  i:=0
  aux:= compute the value of $f_i(1)$
Step 4:
  text := {  $f_{i+1}(0)$= aux and
             $f_{i+1}(n+1) = f_i(f_{i+1}(n))$}
  add_to_file(text,F)
  load_file(F)
  aux := compute the value of $f_{i+1}(1)$
  i:= i+1
  if i < m
  then Go to step 4
  else stop
```

Figure 3. A macro-procedure for computing particular values of ack

ack_macro(m,n) is now completed and file F collects the definitions of m primitive recursive functions. We are now able to compute $ack(m,n) = f_m(n)$.

In the next section, we shall explain how prevention and control are modeled by Ackermann's function.

IV. PREVENTION AND CONTROL IN RECURSION

We have seen above in the example of the program makeclear that primitive recursion captures the effects (computation) and the secondary effects (consequences of the computation). We have also seen that the non-primitive recursive Ackermann's function is obtained using a diagonalization procedure. This diagonalization brings forward complementary information about the process of this symbiotic information in the recursion. Since diagonalization is a meta-level procedure, we understand this complementary information as a kind of meta-level prevention. In particular, we interpret it as a prevention factor simply because diagonalization prevents ack to be reduced to computation and consequences of computations of functions from which it is constructed.

It is interesting to note that some scientists may intuitively 'feel' that Ackermann's function provides a model of human thinking of 'everything' for a particular situation. The makeclear program mentioned shows that this intuition can be presented in terms of symbiosis of the information included in a particular situation. Note that the above macro-procedure (Figure 3) simplifies only the computation of thinking of 'everything'. In order to illustrate this particular 'simplification' of the computation we may mention that, as it can be checked, the trace of the computation of the value for ack(3,2) using the standard definition shows (see [3]) that the value ack(1,1) is computed twenty-two times for obtaining the result of ack(3,2). This is not the case for the computation of simplified $f_3(2)$. However, it is necessary to understand that the overall complexity of this situation remains the same since, in order to be able to 'simplify' (i.e., to define the above macro-procedure), we already need to have available Ackermann's function equivalent to the constructed sequence of f_i. In other words, the principle and effectiveness of 'thinking of everything' remain on the global level. The simplification concerns only focusing on one particular local level defined by the two values *a* and *b* instantiating Ackerman's variables. Of course, the macro-procedure is general, but for *a* and *b* given, it generates only the finite sequence of primitive functions $f_0, f_1, \dots, f_a$.

This makes explicit that 'thinking of everything' keeps its theoretical order of complexity after presented simplification. It is only the computational complexity that is simplified. Systems requiring a simultaneous handling the prevention and control factors such as information flow security systems [7] [10] are practical examples of a problem requesting to think of 'everything'.

V. Pulsative Systems

The above sections will help us explaining how Ackermann's function enables us to formally specify the notion of the **pulsation**. This is interesting not only from the point of view of building particular formal theories for unknown domains, but also for understanding the difference between revolution, innovation and evolutive improvement in this building process.

In the context of program synthesis, we have defined the notion of oscillation in [5] and [6]. Since the notion of oscillation provides an informal background for the notion of pulsation, we shall recall this notion here.

In scientific fields, the obvious basic paradigm is, for a given problem, to find an idea leading to a solution. For instance, in program synthesis, for a given problem one tries to find a *heuristic* that solves the problem. This can, in general, be expressed by the formula

$$\forall \text{ Problem } \exists \text{ Idea Leads\_to\_a\_solution(Idea,Problem).}$$

We shall call this formulation: "first paradigm."

However, another and rather unusual (except in Physics) paradigm is to find an idea that provides a solution for all problems. We shall show how Ackermann's function provides a model for this last paradigm. First, however, let us express this second paradigm by the formula

$$\exists \text{ Idea } \forall \text{ Problem Leads\_to\_a\_solution(Idea,Problem).}$$

We shall call this formulation: "second paradigm."

The difference between these two formulas lies in the fact that in this second case the 'Idea' obtained is unique, while in the first formula each problem can use its own Idea.

We have explained in [6] that the goal of *CMM* is to build a program synthesis system (Idea) that solves the problem of program construction in incomplete theories (e.g., unknown environments in space). We thus globally work with the second paradigm. However, in our everyday research (which means to acquire fruitful experiences enabling to build relevant knowledge), we work locally with the first paradigm while keeping in mind the second paradigm. This means that we *mentally oscillate* between two paradigms. The second paradigm presents a global vision and the direction of the solution we seek and, to make this goal achievable, we perform our everyday work in the framework of the first paradigm following nevertheless the direction imposed by the second paradigm. We call **oscillation** this approach of *symbiotic* switching between the two above paradigms. We speak here about symbiotic switching, since both paradigms are in reality considered simultaneously and cannot be separated.

Let us consider now a potentially infinitely incomplete theory. In unknown environments that may be seen as a framework for potentially infinitely incomplete theories, building a formal theory becomes then a process of *suitable completions* of a particular initial theory T_0. We shall say that this theory T_0 is **practically complete** when it formalizes solutions for the problems met so far. Since the theory is potentially incomplete, sooner or later we shall meet a problem that cannot be solved in the framework of T_0. In the vocabulary of scientific discoveries we may say that we need a conceptual switch (a new axiom or a set of axioms) that *completes* T_0. Note that we speak here about *completion* and

- not about a *revolution* - which would mean in a sense rejecting T_0
- not about a *innovation* - which would simply mean a particular reformulating T_0.

Thus, in fact this completion T_1 contains T_0 and it is coherent with T_0. However, since a new conceptual switch guarantees that T_1 is more powerful than T_0, we consider this particular kind of completion as a suitable model for one step of *improvement*, or pulsation, in our search for suitable completions. Since we consider here a potentially infinitely incomplete theory, we can then see the **pulsation** (particular improvement) as an infinite sequence of theories $T_0, T_1, \dots, T_n, \dots$. In this sequence, T_{i+1} completes and thus is coherent with T_i for all $i = 0, 1, 2, \dots$

We have seen that, in the infinite sequence from which Ackermann's function is built, the function f_1 relies on (is coherent with) f_0, and f_{i+1} relies on f_i for each i. It means that Ackermann's function really provides a model for evolutive improvement (or progress in Bacon's sense [1]). We understand it different from revolution and innovation.

Let us now come back to our notion of pulsation. We have seen that, in the informally specified notion of oscillation, we switch coherently between two paradigms. In our interpretation, the second paradigm, i.e.,

$$\exists \text{ Idea } \forall \text{ Problem Leads\_to\_a\_solution(Idea,Problem)}$$

represents the idea of Ackermann's function and the first paradigm, i.e.,

$$\forall \text{ Problem } \exists \text{ Idea Leads\_to\_a\_solution(Idea,Problem).}$$

represents particular primitive recursive functions from which Ackermann's function is constructed. In the definition of Ackermann's function we have seen that

$$f_{i+1}(0) = f_i(1).$$

Analogously, we shall state that the sequence of completing theories can be written as:

$$T_{i+1} = T_i + A_{i+1},$$

where A_{i+1} is an axiom (or a set of axioms) representing the conceptual switch that enables solving the problem unsolvable in T_i. Let us stress the fact that by pulsation we understand an infinite sequence of theories T_0, T_1, …, T_n, $T_{n+1,}$ … with the just mentioned property and not only one particular step in this sequence. This means that pulsative systems are systems that are formalized progressively and potentially indefinitely.

We have seen above that Ackermann's function is also a model for symbiotic consideration of prevention and control. We could see that f_0 must be defined in a way that guarantees the non-primitive recursion of the constructed infinite sequence. We could see that, with respect to our requirement, f_0 must be defined in a way that guarantees the non-primitive recursion of the constructed infinite sequence. Indeed, if f_0 were a constant, for instance 3 (which would mean that f $f_0(n) = 3$ for all n), the resulting infinite composition would also be the constant 3. This means that, even though f_0 is the first function of this infinite construction, since it must be defined as a symbiotic part of the final composition, the prevention and control factors must be taken into account in this function.

So, we can see that Ackermann's function provides in fact a model for the ***improvement*** that guarantees symbiotic handling prevention and control already from the start.

VI. ON PULSATIVE DEVELOPMENT OF *CMM*

Roughly speaking, *CMM* is developed as a methodology for automation of program synthesis in incomplete domains via inductive theorem proving (ITP). It represents an experimental work that illustrates this paper. For understanding this section it is not necessary to present a formalization of this particular application (it can be found in [5]). However, it is useful that we describe what we understand by a methodology.

Given a non-trivial goal, its methodology is a full formalized description of all the problems that arise in achieving this goal and, of course, of the complete solutions for these problems. In other words, a methodology is a full 'know-how' of a successful achieving the given goal.

Automating program synthesis in incomplete domains via ITP is far from a simple problem. This is because a unified know-how is not available even for by-hand construction of inductive proofs that are necessary for program synthesis. This means that a unified know-how must first be found. This is the goal of our *CMM*.

It is important to note that we are still at the level 0 of pulsative development of *CMM*. In other words, we work on defining a powerful primitive recursive f_0 with respect to the overall goal of resulting non-primitive recursive SRPS for *CMM*. This means that already level 0 has required several decades of research and many useful results not known in automation of ITP were obtained so far. A full bibliography of these results can be found in [9]. We have described above the process of building f_0 by oscillating between two above mentioned paradigms. However, we still need to work on transmission of the technical details of this oscillation. We have explained in [6] that Cartesian Intuitionism, and thus *CMM* as well, cannot use tools developed by Newtonian approaches.

Understanding the process of oscillation between the two paradigms described above is very important for the development of SRPS (namely the systems on level 0) in various domains. However, a detailed illustration in the framework of program synthesis would be too much complex for readers that are not expert in this particular topic. We intend to present a compact but detailed illustration on an example that concerns a 'safe' transmission of relevant scientific knowledge. This problem was already pointed out by Francis Bacon. By a 'safe' transmission of knowledge we understand a transmission that guarantees that no mutilation is possible during such a transmission and that all the creative potential of the knowledge and know-how to be transferred is preserved. Our book [3] is an example of such a safe transmission. We shall tackle this topic also in one of our future papers.

VII. A PULSATIVE TECHNOLOGICAL VISION

It is interesting to be focused on the topic of SRPS in general and of *CMM* in particular because, in long term consideration, this seems to be the only way how robots will be able to

- formalize recursively unknown domains (e.g., in space research) handling perfectly control, rigor and evolutive improvement;
- perform experiments necessary for finding such suitable formalizations;
- program themselves autonomously with the help of the formalizations found.

By formalizing an unknown domain we mean its progressive exploration and acquiring experiences – through experiments – that lead to facts enabling a progressive formalization of this domain.

Of course, a successful achievement of this technological vision will require not only *CMM* but also the tools developed in Machine Learning, Big Data, Computational Creativity and some other maybe not yet known scientific fields that will become known as soon as scientific community overcomes artificial human factors that are a barrier for seriously investigating this technological vision.

Let us recall once again that each unknown environment is potentially infinitely incomplete and thus the notion of pulsation really has an enormous importance for Science.

VIII. CONCLUSION

There are technological visions that need to be solved in the framework of Symbiotic Recursive Pulsative Systems and thus, they need to be tackled by Cartesian Intuitionism. This means that all the notions of SRPS and their algorithmic elaborations should become widely known so that really symbiotic long-term collaborations become possible. This need for symbiotic collaborations requires also a replacement of Newtonian management strategies by the management strategies that are proper to Cartesian Intuitionism. This paper extends thus our preliminary work on transmission of fundamental notions of Cartesian Intuitionism and SRPS by presenting the origin and the motivation for the model of pulsation inherent to SRPS. By its practical applications and already existing use mentioned in the paper this notion shows its importance for Science already now and not only for future technological visions. Indeed, this notion allows to consider progress as different from innovation for which a control of negative secondary effects appearing in future is not handled systematically. The Ackermann's function as a model for pulsation allows to provide such a control since the control of the secondary effects is built in SRPS themselves and already from the start of their design. This paper shows that Ackermann's function should not be considered as a simple abstract mathematical curiosity but as a legacy with a rich scientific potential.

ACKNOWLEDGMENT

We thank to Michèle Sebag and Yannis Manoussakis for a moral support. Anonymous referees of the submitted version provided a very useful feedback.

REFERENCES

[1] F. Bacon, The Advancement of Learning; Rarebooksclub, 2013.

[2] M. Franova, "CM-strategy: A Methodology for Inductive Theorem Proving or Constructive Well-Generalized Proofs"; in, A. K. Joshi, (ed), Proc. of the Ninth International Joint Conference on Artificial Intelligence, pp. 1214-1220, 1985.

[3] M. Franova, Formal Creativity: Method and Use – Conception of Complex "Informatics" Systems and Epistemological Patent (Créativité Formelle : Méthode et Pratique - Conception des systèmes « informatiques » complexes et Brevet Épistémologique); Publibook, 2008.

[4] M. Franova, "Cartesian Intuitionism for Program Synthesis"; in. S. Shimizu, T. Bosomaier (eds.) , Cognitive 2013, The Fifth International Conference on Advanced Cognitive Technologies and Applications ; pp. 102-107, 2013.

[5] M. Franova, "A Cartesian Methodology for an Autonomous Program Synthesis System"; in M.Jäntti, G. Weckman (eds)., proc. of ICONS 2014, The Ninth International Conference on Systems; ISBN, 978-1-61208-319-3, pp. 22-27, 2014.

[6] M. Franova, "Cartesian versus Newtonian Paradigms for Recursive Program Synthesis"; International Journal on Advances in Systems and Measurements, vol. 7, no 3&4, pp. 209-222, 2014.

[7] M. Franova, D. Hutter, and Y. Kodratoff, Algorithmic Conceptualization of Tools for Proving by Induction «Unwinding» Theorems – A Case Study; Rap. de Rech. N° 1587, L.R.I., Université de Paris-Sud, France, Mai 2016.

[8] M. Franova and Y. Kodratoff, "Cartesian Handling Informal Specifications in Incomplete Frameworks"; Proc. INTELLI 2016, The Fifth International Conference on Intelligent Systems and Applications, pp. 100-107, 2016.

[9] M. Franova, List of publications (retrieved 2017/05/17) https://sites.google.com/site/martafranovacnrs/publications

[10] H. Mantel, A Uniform Framework for the Formal Specification and Verification of Information Flow Security; PhD thesis, Universitty of Saarlandes, 2003.

[11] R. W. Ritchie, "Classes of predictably computable functions", Trans. Amer. Math. Soc. 106, pp. 139-173, 1963.

[12] A. Yasuhara, Recursive Function Theory and Logic; Academic Press, New York, 1971.