SHF:Medium:Title: Completely Automated Software Foundation and Its Applications in All People Programming Language(APPL) and Self-generated Software Cell Simulating Biological Cell

By Hugh Ching, September 30, 2011

Project Description

I. Introduction, objectives and expected significance

The goal of this proposed research project is to achieve complete automation for software. What is complete automation? Of all man-made objects, the computer and its software are the closest to complete automation. Yet, today all the software and hardware are partially automated in that they are basically different from living things, which, even before being fully understood, illustrate complete automation.

What is the basic difference between the living organism and man-made objects? The main difference is that the living system [Ref. 1, 2, 3] has been designed for permanent existence, and man-made objects, for temporary existence. Of all the design criteria, the requirement of permanence should be considered one of the most important because sooner or later objects not designed for permanent existence will become obsolete or useless. The requirement of permanence is primarily a value consideration in that an object of temporary existence will sooner or later become completely valueless or return to its salvage value. Thus, a desirable requirement for all man-made objects should be the achievement of permanent existence. To understand and achieve the requirement of permanence is a goal of this proposal, and all mankind. In fact, the feasibility of satisfying the requirement of permanence is demonstrated by the very existence of mankind, which can provide guidance for the technical understanding of the requirement of permanence.

The living system is characterized by both permanence and complete automation. All the processes of complete automation have been programmed into DNA [Ref. 3], which conceptually is in the form of software, with its basic machine code elements of A, C, T, and G. Technically, life science can be considered the documentation of the software of DNA. One of the first understandings of permanent existence might be that while life itself is finite, DNA, which propagates from life to life, has been designed for permanent existence. Life itself can become "permanent" through cloning the same DNA. The living system might be considered the ultimate form of complete automation, and a completely automated software foundation in this proposed research is just a starting point, which hopefully will eventually, in millions or even billions years, lead to the ultimate form represented by the living system. With the availability of the computer, the foundation for demonstrating permanent existence can start with the completely automated software and the silicon simulation of the DNA-protein system.

What is the relationship between permanence and complete automation? In this proposal, a completely automated software system will be designed with the three capabilities of self-generation, automatic update, and automatic documentation. In particular, using the three capabilities, the source code, once written, will never need to be manually updated to future versions of the software system. Thus, the source code has achieved permanent existence similar to DNA, which can self-generate and propagate permanently through sexual reproduction or cloning. But, unlike DNA, which is already fully developed and no longer needs update and documentation, the permanent source code will need an unlimited number of updates and unlimited amount of new documentations. DNA still has semi-conserved self-generation.

To advance the capabilities of the current software to those of DNA, almost an unlimited number of updates are needed. Manual update, as done with current temporary software, will be prohibitive; the update process must be completely automated. Similarly, the amount of documentation is also unlimited, for with each update in capability or format, the source code needs to be re-explained or re-documented.

Self-generation is necessary to make software system to be completely automated. Software is automated when it can be generated. Manually modification is only needed to change the capabilities of the software. A completely automated software system requires all the software in the system to be generated. Thus, the initial generator, which is written manually with a temporary programming language, must be able to generate itself or self-generate. The process of self-generation is simply using the first generator to write itself manually, statement by statement, and is possible because the first generator is just another program. All the other software is written with the initial generator.

Self-generation is not only needed in the modification of the user interface and in adding new capabilities, but is also needed in the creation of self-generated neural network of software cells, which are semi-conserved self-generating software systems, to simulate the network of biological cells, such as the brain.

The creation of a completely automated foundation for software should start with fundamental considerations. The optimal software design should treat the computer and the human as partners. In particular, the design should take into consideration the fundamental characteristics of both the computer and the human user. The computer is just a machine, and, as such, its characteristic in the design of a completely automated software foundation is that the computer should only be required to handle integers, which could be considered the native language of the machine, not to handle human native languages directly. In contrast, today's English-like source code is neither fully understandable to the user nor easily understandable to the computer. As recommended in this proposed research, computer source codes should be in the form of integers, which are easily understandable to the computer, such as today's virtual machine codes, and are separately explained to the user in human native languages through the human language user interface and completely automatically produced documentation programs.

The main characteristic of the human user is that nature has given the human the ability to access an unlimited amount of information. For example, when one goes to the library, one is able to access an unlimited amount of information without having to remember where the information is stored. This ability to access an unlimited amount of information is due to the fact that the human has been given Human Associative Memory (HAM) [Ref. 4], different from the memory of the computer. Admittedly, HAM is necessarily a very fuzzy system [Ref. 5, 6, 7]. Yet, it works. HAM is sufficiently accurate to operate in the tolerant environment of life science and exemplifies the qualitative nature of life science.

HAM is related to the basic concept of set theory, where similar objects are grouped together in a set, and which was originally invented by Cantor [Ref. 8, 9] to handle infinity, corresponding to permanence or the limitlessness of the information needed to be accessed to study the infinite reality. When applied in complete automation, HAM is primarily responsible for the complete elimination of all the technical barriers in computer usage. Retrospectively, the elimination of all the technical barriers, including English, which is a barrier to non-English speaking users, is a necessary condition of the requirement of permanence, where the slightest amount of barrier could be multiplied by infinity to become an infinite barrier. Technical barriers, when reaches a large enough amount, will prohibit complete automation.

Thus, the design of the completely automated software is based on three fundamental principles:

(1) Human Associative Memory must be included to allow the unlimited access of information.
(2) The computer reads the integer, which is the native language of the machine.
(3) Each record must have a permanent name, in the form of the integer for the computer to handle.

In conclusion, the main objective of this proposed research project is to introduce the concept of complete automation, which is directly related to the design requirement of permanence. The best example of a completely automated system is the living system. Although a life span is finite, the living system and, in particular, DNA have been designed for permanent existence through propagation to offspring. This proposed research project will address the question of how to satisfy the design requirement of permanence. Technically, permanence in software is achieved through complete automation, which characterizes life, and which, briefly, consists of the software capabilities of self-generation, auto-update, and auto-documentation. Completely automated software should be universal, as well as permanent, so that all existing and future software can be converted to the format of the completely automated software.

The expected significance of this research is to create permanent software based on complete automation. Permanent software can eliminate the updating cost associated with today's temporary software through auto-updating, rather than manual updating. Currently, updating or maintenance cost occupies the bulk of the software budget. Complete automation can also be employed to eliminate all the technical barriers in computer usage. In this research, an All People Programming Language (APPL) will be developed based on permanent software to allow all people over the age of six to program a computer by the elimination of all the technical barriers, including English, a technical barrier to non-English speaking population.

Lastly, this research will attempt to simulate the DNA-protein system of the living organism with permanent software. This simulation, if sufficiently successful, will transform our current non-living technology into a living technology based on complete automation. It was John von Neumann [Ref. 10, 11] in the later part of his life, who tried to make computer completely automated. This research will develop the algorithm and continue the unsolved problem of complete automation of von Neumann.

In the first year of the proposed three-year research project, a Universal Permanent Software (UPS) will be developed. As its name implies, UPS is permanent because it can be automatically update to all future versions of UPS, and UPS is universal because any software can be converted to UPS. Therefore, there is no excuse not to develop any temporary software not in the format of UPS. The second year of this research will be used mainly to develop All People Programming Language based on UPS. The third year will be devote to the attempt of connecting UPS to the DNA-protein system by developing initially a software cell, which can be used to simulate the biological cell. The software cell will be able to self-generate to create a self-generated neural network, which in the future might be able to simulate the brain.

Intellectually, an additional transformative consequence of this research could be the realization of the impossibility of empirical verification for permanent creations. Thus, the permanent UPS is accepted based on rigorous derivation, namely, logic. Empirical verification is the bulwark of science, but is NOT possible when a deterministic set of data can only be collected until infinity, which will never arrive.

Since the completely automated software is developed based on basic principles, the development of the software can simply converts existing software to the UPS format, fully or partially, and can start from any level, such as the machine, the language, and the application levels. Also, by eliminating technical barriers in computer usage, HAM helps the completely automated software to compete against the existing software in user-friendliness. UPS can immediately be implemented with these advantages.

II. Relation to the present state of knowledge in the field

For this proposed research, there are two types of present state of knowledge, one represented by the knowledge discovered by human beings and the other consisted of the knowledge existed in nature, which is not included in the state of knowledge by most people. The former knowledge is based mainly on a non-living technology originating from science. The later knowledge is the living technology consisted of the living system, which is characterized by complete automation. The requirement of permanence opens up a new area or era of knowledge dealing with the living system, which seems to have been conceptually designed for permanent existence. Permanent creations, such as Universal Permanent Software representing knowledge of human beings and DNA representing knowledge of nature, can no longer be subjected to empirical verification, which is the standard of science or the criteria for acceptance in science. Empirical verification requires a deterministic set of data, which cannot be collected when the phenomenon extends to temporal or spatial infinity, which by definition never arrives. Knowledge in this proposed research, therefore, are derived based on logic and can be described as post-science, knowledge beyond science, involving the analysis and the creation of permanent entities.

From the point of view of software design criteria, the proposed completely automated software can be distinguished from the current partially automated software in two ways. The first contrasting design criterion is permanence versus temporary existence. The second is constrained design versus free design.

Most software systems today are developed for temporary existence and are designed freely within the limit of scientific laws. The completely automated software is designed for permanent existence and is heavily constrained in terms of the user interface, the form of the source code, and the format of the data file, in addition to scientific laws. The requirement of permanence imposes the severe design constraint.

To achieve permanence, the completely automated software is constrained in its design. The advantage of being constrained is that the user and the developer of software know exactly the future formats of the user interface and the source code. However, these formats must be universally applicable, that is that the formats must be able to accommodate all the future possibilities; they must be able to replace all the existing and future user interfaces and the future forms of the source code. Or any existing and future software systems must be able to be converted to this universal system of completely automated software.

In the completely automated software, the function of the source code is solely for instructing the computer to perform some functions. The job of understanding the source code is assigned to separate auto-documentation programs and to the user interface, which is expressed in human understandable forms. In contrast, today's English-like source codes have been asked to perform the both functions of instructing the computer and helping human to understand the instructions. In trying to satisfy both the user and the computer, the English-like source codes might become hard to understand to both and ruin the possibility of complete automation and their permanent existence. Today's source codes will neither be readily understandable to non-English speaking people and to newly designed computer systems.

The proposed completely automated software requires all software languages, such as the proposed APPL, to use a Universal User Interface, a Universal Computer Source Code (UCSC), and a Universal Data File. The software achieves permanent existence through self-generation, auto-update, and auto-documentation. All the executable programs can be generated by feeding the Universal Computer Source Code into the generator. For all the programs to be generated, the initial program must be able to self-generate, resulting in potentially complete independence from past technical terminologies.

The Universal User Interface is similar to the familiar graphics windows user interface, if integers are inserted in front of the items; it is a tree-structured, numerical multiple-choice question format. The integer answers with other content-insensitive answers are recorded as the Universal Computer Source Code, which is similar to DNA and virtual machine codes. The integer answers will lead to an integer ADDRESS of the Universal Data File, which will either perform a function or return the ADDRESS of the function, after a prior instruction setting a flag to request the ADDRESS. The reason for returning the ADDRESS is to allow the generation of a statement containing the ADDRESS, which, thus, needs not be remembered by the user. In the completely automated software, all the technology has been replaced by the ADDRESSES, which are permanently fix and can be considered a part of Universal Permanent Numbers [Ref. 12, 13], which are simply the complete set of distinct integers from minus infinity to plus infinity and are a byproduct of UPS. The ADDRESSES are remembered by the software, not the user.

Self-generation allows the continual improvement of the self-generating generator, permanently. The old self-generating generators and all the source codes created with the old generator can be automatically updated to the new improved self-generating generators and new source codes, and vice versa. Auto-updating eliminates the updating cost, which currently occupies the bulk of the total software budget.

Auto-documentation allows the explanation of the Universal Computer Source Code to the full satisfaction of the user. As in the Universal User Interface, the documentation is presented in human native languages or even sound and graphics. While the computer user reads the human native language or a human understandable multimedia display, the computer reads the integers, which is the native language of the machine, and content-insensitive answers to the questions of the Universal User Interface, which is similar to the familiar menu system with numerical item numbers attached to items.

All the capabilities of the completely automated software have been separately demonstrated by commercial products, which guarantee the successful development of APPL and have been demonstrated fully in the Self-generating Software System invented by PI [Ref. 14, 15]. In the existing commercial software, auto-updating is demonstrated by James Gosling with the auto-conversion of virtual machine codes [Ref. 16, 17, 18], auto-documentation by Jay Xiong [Ref. 19, 20], self-generation by Nelson Lin [Ref. 21], and the Universal User Interface by Ivan Sutherland with Graphics User Interface [Ref. 22]. Universal Computer Source Code is supported by DNA and work of Alan Turning [Ref. 23, 24, 25, 26].

Two contributions to the theoretical foundation of Universal Permanent Software are provided by T. L. Kunii and Lotfi Zadeh. Kunii has been working on computer graphics [Ref. 27, 28, 29, 30], and Zadeh has been proposing Computing with Words based on Computational Theory of Perception and fuzzy logic [Ref. 5, 6, 7]. The theoretical foundation of completely automated software could be guided by their vision of graphics and the human ability of perception of words, in addition to analysis of numbers. Zadeh's vision will become clear when software systems are being developed for the control of robot, which, like human, takes instructions in words. Robot software needs unlimited number of updates.

From the viewpoint of life science, Zadeh's work on fuzzy logic is particularly interesting. From the fact that infinite analyses and permanent entities are not subject to empirical verification, as in this proposal, the standards of rigor of social and life sciences need to be mathematics and logic, respectively, but the accuracy of their inputs and their outputs can be relaxed or the inputs and outputs are fuzzy or qualitative. In particular, Computing with Words could be directly related to Universal Computer Source Code, which can be considered "Computing with Integers." The Formula Expression of Cellular Data System [Ref. 30] of Kunii is being converted to Universal User Interface, in which the meaning of words are fuzzy and the integers are precise. The new Cellular Data System could be used as the data system for UPS. Fuzzy is a dirty word in science, but will be welcomed with open arms in post-science, where data are fuzzy.

Today's software is mainly used to do word procession, spreadsheet, database management, productivity, and scientific calculations. Computer graphics and robotics have to simulate the real world, in particular, the behavior of human beings. They present complexity orders of magnitude higher than the works done by the current software and computer. For example, robot functions, in order to imitate human motions, are nearly unlimited, and robot software, thus, needs the continual unceasing update by robot software developers. Each update, in essence, changes a software standard, which, however, is related to prior software standards through auto-updating. Today's software standards must be set by a software developer. They must also be manually updated usually by the standard setter or a permanent standard setting committee. What will be changed by Universal Permanent Software is that any programmer can change the software standard and still be connected to all the other software through auto-updating.

Since the conceptual design of a complete computing system always involves all its parts, the complete process of thinking still remains the same. The key point is the concept of discipline initially advocated by C. V. Ramamoorthy [Ref. 31, 32, 33, 34, 35] in software engineering. Chien Yi Lee [Ref. 36] and PI, guided by the concept of discipline believe that the computer is a set-theoretical machine, which speaks integers. Set-theoretical corresponds to Human Associative Memory, and integers correspond to Lee's Universal Computer Source Code, which can be auto-updated and auto-documented by the computer. In a historical sense, it could be said that software discipline starts with the vision of software engineering by people, such as Ramamoorthy, and works toward the goal of completely automation of this research.

III. Detailed technical explanations with discussions of social impacts:

Computer science up to now has no discipline of its own; a computer scientist can virtually design anything within the limits of laws of nature in science. This proposed research introduces the requirement of permanence as the first discipline of computer science. Permanence is achieved through complete automation and involves three fundamental principles, three innovations, and three capabilities, which are derived from the three fundamental principles and the three innovations. The principles, the innovations, and the capabilities, though same in number, do not necessarily have one-to-one correspondence.

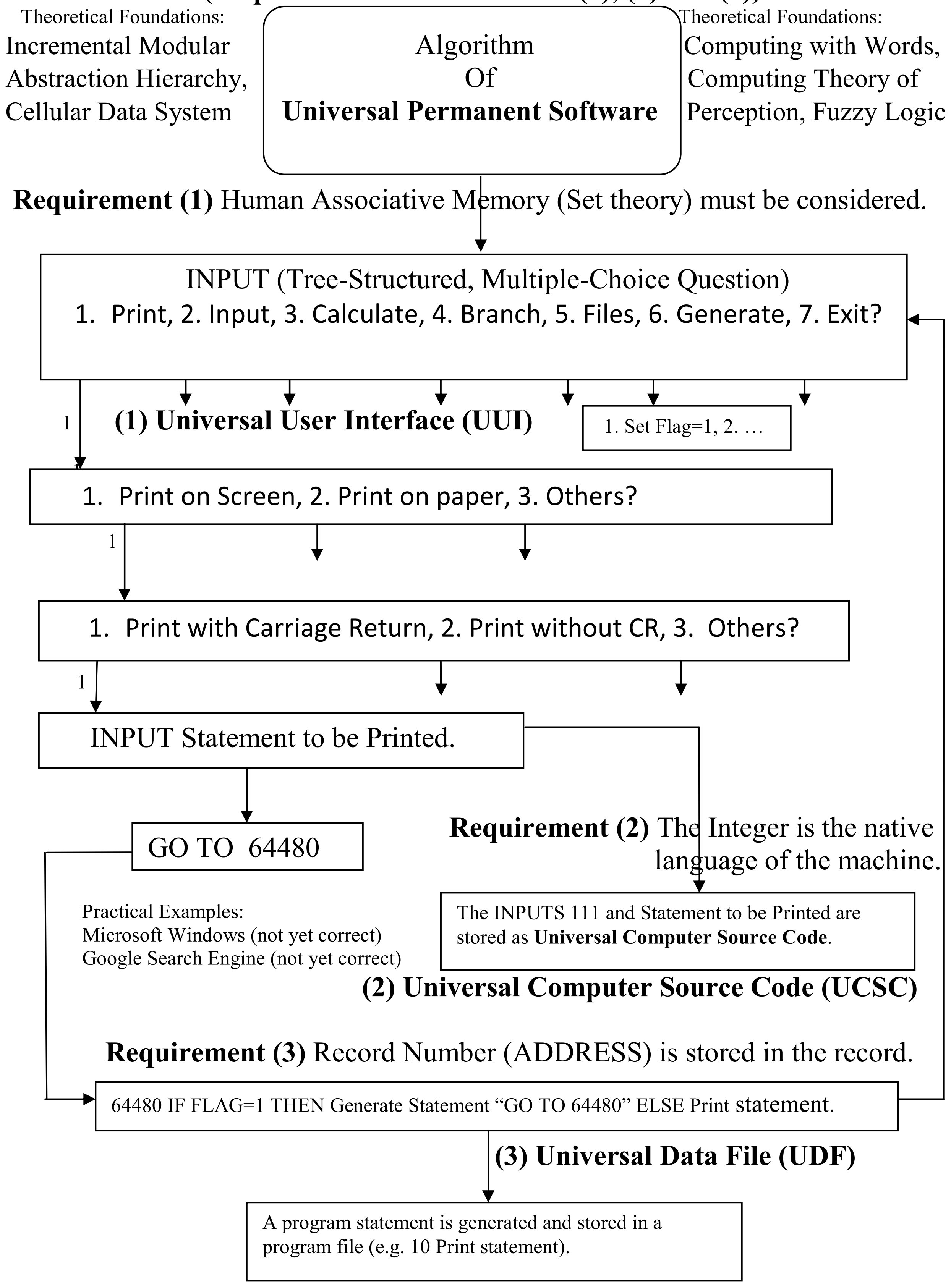
Correct computer software should start with fundamental considerations, such as the three below:
(1) Human Associate Memory must be designed into software in order for human users to access unlimited amount of information,
(2) Humans read their native languages, and the computer reads the integer, which is the native language of the machine, and
(3) Each record must have a permanent distinct name, which is stored in the record so that the name is remembered and can be automatically retrieved by the computer, and the name, such as an ADDRESS, must be an ordered distinct integer, which is necessary to avoid redundancy, as the amount of the integer names can be unlimited. The name is also in the form of the integer, the native language of the computer.

The goals of completely automated software are to achieve technology independence, non-obsolescence, automatic updating, automatic documentation, self-generation, etc. **Figure 1** shows the algorithm for the development of Universal Permanent Software satisfying the requirement of permanence or complete automation. Three criteria and three innovations are inserted near the flow chart areas where they apply. To satisfy the requirement of permanence, UPS boils down to three technical design criteria: (1) Human Associate Memory must be designed into software in order for human users to access unlimited amount of information (2) Humans speak their native languages, and the computer speaks integers, and (3) The record number needs to be stored in the record. Software design should be based on these criteria.

In Universal Permanent Software, as shown in **Figure 1**, the numerical choices of tree-structured, numerical multiple-choice question, as exemplified by the question, 1. Print, 2. Input, 3. Calculate, 4. Branch, 5. Files, 6. Generate, 7. Exit? guide the user to a call statement, such as call 64480 or GOSUB 64480. The LABEL or ADDRESS 64480 is where the operation is to be performed. It is difficult for human users to remember a large number of these LABELs of ADDRESSES. But, if the user has set a flag before being guided to the call statement, the program generator will, instead of performing the operation, return the LABEL or ADDRESS 64480, or even generate the statement GOSUB 64480. Thus, (3) the record number 64480 must also be stored in the record. There can be an unlimited number of operations or ADDRESSES, such as 64480, as long as they can be retrieved, using the Human Associative Memory, which should always be included in the design of completely automated software.

The program generator with the above multiple-choice user interface is written by an initial program generator with a user interface based on a computer language, such as BASIC in the form of (1) ABS-HEX$; (2) IF…THEN-LPRINT; (3) LSET-RIGHT$; (4) RMDIR-WRITE#; (5) Generate; (6) Others; (7) Exit? The user interface of this initial program generate is arranged alphabetically so that all the BASIC statements will be included, but is only understandable to a programmer who knows BASIC. The initial program generator is used to write a generator with a user interface in human native language. The initial generator will write itself so that it can self-generate. The integer answers of the initial program generator

Figure 1 Sample Algorithm of Universal Permanent Software (Requirement of Permanence: (1), (2) and (3))


Theoretical Foundations:
Incremental Modular Abstraction Hierarchy, Cellular Data System
Algorithm Of **Universal Permanent Software**
Theoretical Foundations:
Computing with Words, Computing Theory of Perception, Fuzzy Logic
Requirement (1) Human Associative Memory (Set theory) must be considered.
INPUT (Tree-Structured, Multiple-Choice Question)
1. Print, 2. Input, 3. Calculate, 4. Branch, 5. Files, 6. Generate, 7. Exit?
1
(1) Universal User Interface (UUI)
1. Set Flag=1, 2. …
1. Print on Screen, 2. Print on paper, 3. Others?
1
1. Print with Carriage Return, 2. Print without CR, 3. Others?
1
INPUT Statement to be Printed.
GO TO 64480
Requirement (2) The Integer is the native language of the machine.
Practical Examples:
Microsoft Windows (not yet correct)
Google Search Engine (not yet correct)
The INPUTS 111 and Statement to be Printed are stored as **Universal Computer Source Code**.
(2) Universal Computer Source Code (UCSC)
Requirement (3) Record Number (ADDRESS) is stored in the record.
64480 IF FLAG=1 THEN Generate Statement "GO TO 64480" ELSE Print statement.
(3) Universal Data File (UDF)
A program statement is generated and stored in a program file (e.g. 10 Print statement).

for writing the generator with the user interface shown in **Figure 1** can be auto-updated to the Universal Computer Source Code for the newly written generator; there is no need for the new generator to write itself for self-generation! The auto-update or conversion program is completely automatically created.

To further illuminating the simplicity of Universal Permanent Software, the two lessons on a demonstration system of All People Programming Language is given below:
Lesson 1: The Complete Learning Curve of All People Programming Language (APPL)
Question: Using Universal User Interface, how to program the instruction: PRINT "Hello"?
Answer: Type: "111Hello".
Question: What is Universal User Interface?
Answer: It can have infinite number of forms. For example, the first-level question here can be:
Choose one: (1) Print, (2) Input, (3) Calculate, (4) Branching, (5) Files, (6) Generate, (7) Exit?
After typing "1" the second-level question appears:
Choose one: (1) Print on Screen, (2) Print on paper, (3) Others?
After typing "1" the third-level question appears:
Choose one: (1) Print without a carriage return, (2) Print with a carriage return?
After typing "1" the content-insensitive question appears:
What do you want to print?
Here the user types "Hello".
Question: How is the instruction to print on screen generated?
Answer: This is not the job of the user; it is the job of the original "hardware" (generator) manufacturer.

The above is the complete learning curve for the All People Programming Language.

Lesson 2 How to Generate an Instruction Generating Instruction
In a self-generating generator, the generator must be able to generate an instruction generating instruction, which, for example, is in the form GOSUB 64480, where the ADDRESS 64480 is unknown to the user.
Question: How to write GOSUB 64480, without knowing 64480?
Answer: Type "6" for the question: Choose one: (1) Print, (2) Input, (3) Calculate, (4) Branching, (5) Files, (6) Generate, (7) Exit? After typing "6" the second-level multiple-choice question appears:
Choose one: (1) Write instruction generating instructions, set flags for auto-update, (2) Others?
After typing "1" the following third-level question appears:
Choose one: (1) Write an instruction generation instruction, (2) Others?
After typing "1" the program returns to the first-level question:
Choose one: (1) Print, (2) Input, (3) Calculate, (4) Branching, (5) Files, (6) Generate, (7) Exit?
Now to generate GOSUB 64480, the user goes through exactly the same process as above in generating the instruction to print "Hello", as follows:
After typing "1" the second-level question appears:
Choose one: (1) Print on Screen, (2) Print on paper, (3) Others?
After typing "1" the third-level question appears:
Choose one: (1) Print without a carriage return, (2) Print with a carriage return?
After typing "1" the content-insensitive question appears:
What do you want to print?
<Enter> [NOTE: Here the input is irrelevant, and, thus, typing <Enter> is sufficient.]

The above is the complete learning curve for APPL. In fact, **Lesson 2** on how to generate an instruction generating instruction is just repeating the same process of **Lesson 1** on how to generate an instruction and is not necessary, as all other operations, if explained sufficiently in the Universal User Interface.

From **Lesson 1**, the Universal Computer Source Code for program the instruction, PRINT "Hello" is 111Hello. But, in the initial program generator, the Universal Computer Source Code is 371Hello. The automatically produced auto-update program simply converts 371 to 111. The completely automatic conversion from one number to another and back can be used in an encryption algorithm, where 371 can be the plaintext and 111 can be the ciphertext. In addition to being explained in the Universal User Interface, 371 and 111 will be explained fully in an automatically produced auto-documentation program.

Technically, a Universal Permanent Software will be developed based on Javascript, Javascript Data Base (JSDB), and HTML. The development will continue for three years. Three programmers will assist in the software development. The developed software will be posted on the Internet for world-wide use and comments. Attempt will be made to convert several commercial software systems to test the Software.

The following technical explanations are included to give the reader of this proposal several different perspectives of the design of the completely automated software, namely, from the points of views of **(1) Conceptual Design of Universal User Interface, (2) Universal Computer Source Code and Universal Data File,** and **(3) Conceptual Design of All People Programming Language (APPL).**

(1) Conceptual Design of Universal User Interface: A Perceptive for Understanding the Software

The entire computer system design can be view from the point of view of the Universal User Interface. If the Universal User Interface is done correctly, the rest of the completely automated software system will fall into place naturally. Since the complete automated software is an interconnect system, the design of one parts will affect all other parts of the system. The Universal User Interface provides an efficient way not just to start the design of the whole software system, but also to explain the whole software system.

This research proposes that the design of a computer system starts from the user interface with the criteria that the user interface be designed for universal usage, that is that it can be used to accomplish any known software task or to replace any existing user interface in functionality. The conceptual design of a Universal User Interface is based on the completely automated software system, which can self-generate, auto-update, and auto-documentation, and employs a tree-structured, numerical multiple-choices user interface. In comparison to the existing common Graphics Windows User Interface, this Universal User Interface has been designed to perform two tasks, which the Windows User Interface cannot, (1) Auto-update from one user interface to another structurally similar user interface and (2) Generate instruction-generating instructions, a needed feature in the completely automated software system.

The two basic considerations in the design of a user interface for any machine are user instructions and machine instructions. A designer of a user interface can make the user instructions very clear to the user, but almost completely ignore the requirement of the machine. For example, the computer basically is an integer manipulating machine or its “native language” is the integer, and as the user interface becomes easier for human to understand, it might become more and more difficult for the machine to process.

However, there is an even more fundamental design consideration than the satisfaction of the user and the machine. This fundamental consideration must be based on value. The value consideration for the design of a computer system involves design factors, for example, speed, memory resource, user friendliness, updating cost, etc. Of all these factors, one of the most important factors should be permanence, which involves, for example, permanent user friendliness, permanent elimination of updating cost, etc. Permanence is an important factor because a permanent entity will sooner or later becomes infinitely more valuable than a temporary entity of the same functionality. Two facts supporting this conclusion are (1) all living things have been created for permanent existence and (2) the bulk of the software budget is spent on updating of software from one temporary version to another temporary version. It should be quite obvious by now that the updating cost will be infinite, if the updating has to be extended to infinity in time. One might even go as far as questioning whether the value of today’s temporary software is really positive or negative, when it imposes a permanent burden of manual updating on the user.

Mathematically, Human Associative Memory (HAM) corresponds to the set theory. Thus, the computer should be a “set theoretical” machine. The proposed Universal User Interface for human and machine communication is, thus, the familiar tree-structured, numerical multiple-choice question format, where the computer follows the integer answers to the numerical multiple-choice questions, and the user reads the explanations of the numerical items in human native language, graphics, sound, or any human understandable multi-media representations. In terms of the existing technology, the Universal User Interface employs the same concepts as those used by the computerized library search system or the menu system. Additionally, the integer answers are separately explained by auto-documentation.

The above discussion is reversing the entire process of thinking based on technology, after having solved the problem of completely automated software. The advantage of the above thinking process is to narrow down the whole problem of computing into a clear-cut simple problem of the design of the user interface and also allow other to challenge just the design of the Universal User Interface.

The user does not have to care about what is going on in the background, but should accept and contribute, with user’s innate ability, primarily, the Human Associative Memory, to the design of the Universal User Interface to satisfy the requirement of complete automation. This discussion contents that the requirement of complete automation can be reduced to the condition of the simple conceptual design of the user interface. Zadeh’s Computing with Words supports the design of Universal User Interface, whose foundation of Human Associative Memory is generally fuzzy and qualitative, but adequate.

(2) Universal Computer Source Code and Universal Data File

2.1 Introduction

Universal Computer Source Code is the universal solution sought by UNCOL - “UNiversal Computer Oriented Language, a universal intermediate language, discussed but never implemented.” UNCOL is considered one of the never-attained "Holy Grails" of computing, an intermediary universal assembly language that should allow running any computer language on any computer hardware. By definition, “the Universal Computer Source Code consists of a set of integer Temporary Numbers whose transformation to other distinct Universal Computer Source Codes are made permanently flexible and completely automated by the Universal Permanent Software.” Temporary Numbers are in direct contrast to Universal Permanent Numbers, which are distinct integers from minus infinity to plus infinity. Both of these Numbers play essential roles in Universal Permanent Software. The definition of the Universal Computer Source Code is defined by Chien Yi Lee above based on Universal Permanent Software, which has been demonstrated as a completely automated software system.

Physical science deals with non-violable laws of nature governing material objects. There might also be non-violable laws of nature in social science, governing human behavior. Computer science is closely related to life science, but, thus far, has no disciplines or requirements of its own. It should satisfy the requirement of complete automation. Complete automation is the solution to unlimited complexity, which characterizes life. From the point of view of value, permanent entities, such as life and Universal Permanent Software, are infinitely more valuable than temporary entities, which characterize all the man-made objects. Complete automation is needed in the creation of permanent entities. Permanence, as a design specification, could represent the first, and currently the only, discipline for computer science.

Universal Permanent Numbers and Universal Computer Source Code are remembered and managed by the completely automated Universal Permanent Software. The Universal Computer Source Code represents a natural, rather than an artificial, standard, because they satisfy the requirement, or the design specification, of permanence. The goal of computer science should be complete, not partial, automation, and the globalization of standards should be based on natural, not the current artificial, standards.

Examples of natural standards are Temporary and Permanent Numbers. Temporary Numbers are used as Universal Computer Source Code. Universal Permanent Numbers can be used as ADDRESSES, permanent book and published article numbers, URLs, ID numbers of products for completely automated global customer service, all the land parcel numbers on earth, all the heavenly bodies in the universe, etc.

It is important to note that the Universal Computer Source Code is but one of three major innovations in the Universal Permanent Software; the other two are Universal User Interface and the Universal Data Files. With these innovations, Universal Permanent Software achieves complete automation through self-generation, auto-update, and auto-documentation. All People Programming Language should and will be release as free software because auto-update operates best in a free environment, an environment without economic barriers. Thus, Universal Permanent Software of this research will provide software with both technical and economic freedoms, in addition to political freedom, or freedom from legal restrictions.

Universal Computer Source Code consists of a set of integers which can be easily updated to future version whose form is already known. For example, the set of source code can be from 1111 to 9999, of DNA can be from 111 to 444, or of virtual machines from 00 to FF. Because it is freely portable or can be easily updated to future versions, it will never become obsolete as computer technology changes.

On the other hand, the current source code is not designed for permanence, because the future format is not known. It should not be in human language which is difficult for machine to understand and is unpredictable due to the unlimited variety of human languages. In particular, the current source code should not be in English, which sets up a barrier for non-English users. There is no reason that every programmer or user has to learn English before she or he can learn the source code or use the software. In addition to being the native language of the machine, the integer is an international human language.

2.2 Complete Elimination of Technical Barriers

Science and technology are the two pillars of our current technological society. Science represents knowledge, and technology, the application of knowledge. But, there is a dark side to technology. In practice, current technology also represents the technical barrier in the communication between the human and the machine. Today, to communicate with a machine or a computer, the user has to overcome the technical barriers, which, in practice, is identified with technology. But, there is one curious fact that has never been taken seriously by the users of machines. The fact is that the computerized library search system or a search engine based on a hypertext or a software menu system can retrieve an unlimited amount of information without any technical barriers, except for some very simple explanation.

How is it possible for humans to retrieve an unlimited amount of information? Or the question should be "What is it that allows humans to access an unlimited amount of information?" The answer is Human Associative Memory. Machines or computers have basically direct access memory. The Human Associative Memory naturally groups items to be remembered into categories. Correspondingly, the set theory in mathematics groups associated items into classes or subclasses in the study of infinity.

However, there is a major difficulty in the computerized library search system. The difficulty involves the updating an old system into a new system, when the arrangement of the multiple-choice items needs to be changed. For example, new areas of study will need new categories, and growth of new areas of study will continue to infinity in time. The completely automated software can be automatically updated from one computerized library system to another, if the systems satisfy the requirement of complete automation. Thus, all the future computerized library system should be designed according to the condition of permanence specified by completely automated software.

Universal Permanent Software has also completely eliminated all the technical barriers in computer usage by employing Universal User Interface, which is similar to the computerized library search system.

Universal User Interface is simply the tree-structured, numerical multiple-choice question format. The currently popular graphic windows user interface is similar to Universal User Interface, but lacks the integers in the multiple choices. While the human user is interested in reading the items explained in human native language, the computer is interested in reading the integers. The windows user interface is extra friendly to the human user. Universal User Interface is friendly to both the user and the computer. In the communication between humans and machines, it is only logical that the user interface treats both parties equally. Universal User Interface introduced in Universal Permanent Software is intended to be the only permanent user interface for human-machine to communicate. Universal Permanent Software will help create a full-employment economy, because people can always get jobs writing the Software.

What is more important than the elimination of all the technical barriers in computer usage is the elimination of all the technical barriers in the world. The overall elimination of all the technical barriers in the world will turn the concept of technology on its head. Technology when being pushed to a certain level can be used to eliminate itself! Such a phenomenon has already been demonstrated in the living system. The Universal Permanent Software is just the beginning of the end of technology. And the key to the total elimination of technical barriers is the Universal Permanent Numbers, which can be used to represent and replace all the technical barriers and are remembered by the computer, not the user.

How to achieve the overall elimination of technology, which, in practice, stands for technical barriers? Initially, all technology should be automated by converting it to the form of software. Then, partially automated software needs to be converted into completely automated Universal Permanent Software. All the Universal Permanent Software can be updated automatically to a form, which has eliminated all technical barriers. Thus, all the technical barriers in the world can be eliminated by transforming all knowledge to software and convert all software to Universal Permanent Software. As knowledge grows and human civilization advances, complexity will become our major problem, as well as our greatest benefit. Complete automation is the only way to solve the problem of unlimited complexity. Technical barriers are a primary part of the problem of complexity. Unlimited complexity will produce overwhelming technical barriers, which, therefore, must be eliminated using complete automation.

Universal Permanent Software has identified the DNA-protein system as an example of Universal Permanent Software and, thus, connected man-made technology to the existing living technology. Once connected, man-made technology should start to use the living technology as a guide, whenever possible.

In conclusion, Universal Permanent Software will allow us to eliminate technology and leave us with just knowledge. But, where have the technical barriers gone? They are transformed into Universal Permanent Numbers and stored as the ADDRESSES in Universal Data Files, the hardwired part of Universal Permanent Software. Universal Permanent Numbers, as the ADDRESSES, are unseen to the user.

2.3Universal Data File:ADDRESSES as Universal Permanent Numbers Remembered by Computer
By following a flow of tree structured multiple-choice questions, the user will eventually be led to an address of the Universal Data File. This is the third innovation in the completely automated software. With this innovation, program generator can write itself. What is the Universal Data File? It is made up of records containing the ADDRESSES of the records. How exactly does a program generator generate itself? Normally, in program generator, a program statement is generated at the end of the tree structure, numerical multiple-choice question, such as "PRINT", "SAVE AS", "CALCULATE", where, for example, the "PRINT" ADDRESS is 64480 (the computer remembers, the user needs not to remember this number 64480). By setting a flag, the program generator can generate, in addition to a statement, a statement generating statement. For example, If flag = 0 (Flag is not set), then the generator actually outputs the statement (for example, print out whatever you want to print out). If flag =1 (Flag is set), then the generator generates GOSUB (64480) (here GOSUB (n) is in BASIC; c is call label or n; Assembly uses JMP n where n = ADDRESS). The generator remembers the ADDRESS 64480, while the user uses

Human Associative Memory to set the flag and then to generate the statement. Thus, the statement generating statement, GOSUB (64480) is generated without the user supplying the number 64480.

The ADDRESSES in Universal Date File can be considered as Universal Permanent Numbers, which are distinct integers from minus infinity to plus infinity. The reason that integers instead of English words are used as the ADDRESS or the label is that integers can be arranged in an efficient order by the computer. Universal Permanent Numbers are remembered and managed by Universal Permanent Software. All the current number systems, such as ISBN, land parcel numbers, product numbers, and Unicodes, should use Universal Permanent Numbers, so that they can be searched globally and be managed by Universal Permanent Software. In contrast, Universal Computer Source Code and all the current source code are temporary numbers, which should be, but not, completely flexible for updating.

(3) Conceptual Design of All People Programming Language (APPL)

The vision of APPL is to allow all people over the age of six to use a computer and to write a computer program under the following scenario:

After a user turns on the computer, the user sees on the screen a numerical multiple-choice question, which is the first level of several levels of the tree-structured numerical multiple-choice question. The user will communicate with the computer through this tree-structured numerical multiple-choice question format and other questions, which require content-insensitive answers.

The above consists of the full extent of the learning curve for a computer user, just to answer the user fully understandable tree-structured multiple-choice questions and other content-insensitive questions. Thus, anybody over the age of six should be able to use or program a computer, if the questions are explained in the human native language or in multimedia illustrations understandable to a child.

Technically, the integer answers to the numerical multiple-choice questions are the instructions to the computer. These integer answers can be easily auto-documented into full documents by separate completely automatically generated documentation programs and can be auto-updated to other tree-structured numerical multiple-choice question formats by separate automatic update programs.

The answers to the tree-structured numerical multiple choice questions and content-insensitive questions are recorded in a file. These answers will have permanent existence because they can be auto-update to all future versions of the tree-structured numerical multiple-choice question formats, as life is designed for permanent existence through the propagation of DNA from one generation to the next. All other forms of user interface should be convertible to the Universal User Interface. Being permanent and universal, the completely automated software is, thus, called Universal Permanent Software (UPS).

IV. Outline of the general plan of work and budget allocation (3years x $368,880 = $1,106,640)

The initial phase of the project will be the research and the development of completely automated software, from which the design of All People Programming Language will be planned. In the second year, All People Programming Languages will be developed to replace some English-like source code languages and will be developed based on the design specification of the completely automated software. APPL will be self-generated into various major human native languages or even into a multimedia user interface for all people over the age of six to use. There will be a total of about two or three versions of the All People Programming Languages, occupying the bulk of the proposed budget. It is expected that a part of the work will be devoted to the development of the auto-documentation program, which is available commercially for English-like source code languages. Auto-documentation programs will be developed to document the Universal Computer Source Codes. The sponsoring technology-oriented university and its associated universities will provide the natural setting for the development of APPL.

Tentatively, the existing English-like source code languages to be converted are BASIC, Java Script, JSDB, and HTML. The programming language will be converted to Universal Permanent Software and into human understandable All People Programming Languages, such as (1) In English, (2) A symbolic language, and (3) A Children's Programming Language for educational purpose. On the average, $368,880.00 will be spent every year. A demonstration desktop Children's Programming Language, which can be learned in 15 minutes by 12 years old, has been tested. It is a simple version of a full-fledged web-based APPL, which will be able to fully replace English-source code languages.

A Children's Programming Language based on BASIC for teaching English-speaking children and adults is available for demonstrating two Universal User Interfaces, auto-update, and self-generation and will be the prototype for future development. The first year of the research will be for the development of UPS based on the Children's Programming Language. The second year will be for the development of All People Programming Languages. The third year will start the exploratory investigation of the feasibility of the development of a self-generated neural network of software cells, which will be a software simulation of the network of biological cells. This last part of the research will be speculative, but should be done to ascertain the correct direction of the development of the completely automated software and for the understanding of the biological system. The feasibility to start this third year research is supported by the similarity of the DNA system and the completely automated software in the following example:
Technically, the most basic Universal User Interface for quart-decimal radix system of DNA is:
1. <u>adenine</u> (abbreviated A), 2. <u>cytosine</u> (C), 3. <u>guanine</u> (G), and 4. <u>thymine</u> (T) 5. Other functions.
If (1) is picked, the next question would be:
*1. AA, 2. AC, 3. AG, 4. AT, 5. Others
If (1) is picked, the third level question would be (codons):
**1. AAA, 2. AAC, 3. AAG, 4. AAT, 5.Others
where 5. Others can expand the questions to include an unlimited number of operations or functions. All biological study of DNA would start with these questions, which correspond to the initial program generator. Research on the living system, such as that of Miller [Ref. 1], will be very helpful here.

Technically, there are two parts to a generator of UPS. The top part stores and generates Universal Computer Source Code from Universal User Interface, and the bottom part generates the program with Universal Data File. This construction of a self-generating software system or UPS is similar to that of ribosome, where RNA moves through two parts of ribosome to make protein. The biological locks and keys are similar to the ADDRESSES, which identifies the locations of the instructions to be generated.

Other Universal User Interfaces can be constructed from the base-64 radix systems of codons and other higher structures, as in Dr. Kunii's Incrementally Modular Abstraction Hierarchy (IMAH). The main purpose of the research is to document DNA with UPS, which currently simulates just ribosome. Life science could be defined as the documentation of DNA. If DNA can be understood, most other things could become readily understandable because everything starts from and ends with DNA, just as all completely automated software starts and ends with Universal Computer Source Code, which, as DNA, exists permanently. The physical manipulation of DNA has been pioneered by Ned Zeeman [Ref. 37]. In sum, once a connection has been made between software and DNA, the living system should guide the development of the software, for the living system represents the most advanced existing technology.

UPS can self-generate itself (with different names using an external file) to an unlimited number of UPS e.g. S1, S2, S3, S4… until the disk is full. The multi-cellular design of the living organism is the ultimate fault-tolerance scheme. The cells can differentiate themselves either with an external file or by cell-to-cell communication. Software self-generation mirrors the possibility of mankind self-creation [Ref. 38].

The sponsoring university will supply the facility and will back up this research project with its full faculty and student body. The University sits right in the heart of Silicon Valley and teaches the most advance and current technical computer science using the most up-to-date instructors from the neighboring high-tech companies. The President of the University will back the research project personally by advising the technical management and the educational aspects of the project. This will be a research project of university-wide interest.

V. Relation to longer-term goals of the Principal Investigator's project

The completely automated software, seen from a long-term perspective, will be developed mainly for robot software, which needs constant updating and unlimited programming with constant changes in robot functions. Complete automation, for both software and hardware, can start from a Self-Manufactured General Purpose Robot with the ability of touch. The problem of touch or collision without bouncing off has been studied by Hugh Ching and Ta-You Wu with the new physics concept of jumpulse [Ref. 39, 40, 41], which is a sudden change of force, as Newton's impulse is a sudden change of momentum. The Robot will be able to replace all human physical labors and will be programmed by the completely automated software, which has the unlimited ability of completely automatic update.

All rational decisions should be based on value [Ref. 42, 43]. For example, the decision to perform and to fund this research on the completely automated software should be based on the value consideration that over 90% of software budget is occupied by maintenance or update from old to new versions of temporary software. The decision on what Robot to construct will also be analyzed based on value. Ultimately, the robot might be improved to a point to realize that the human is the most advanced robot, where computer hardware and software are merged into one single unit, namely, the DNA-protein system.

VI. Broader impacts resulting from the proposed activities to enhance scientific and technological understanding and its potential benefits of the proposed activity to society at large.

The broad impacts of UPS can be expressed in terms of freedom. UPS is free from technical barriers. Open source is a freedom by choice as described by Richard Stallman [Ref. 44], but is a necessary freedom for UPS to exchange freely. To eliminate software cost barrier, UPS will be freely distributed. But, does the technical design of UPS free? The answer is No; the design of UPS is severely constrained. Thus, UPS contributes to the fundamental societal debate on freedom. In science, material objects are constrained by non-violable laws of nature in science. In social science, the mathematical relationship, say, between the price and the rate of return constrains investment behaviors; financial crises result from this constraint; Free Market exists within this constraint. Is software design constrained? Currently, it is almost completely free. But, UPS, with the requirement of permanence, introduces new severely constrained software design criteria, which, however, are fully compensated by the universality of UPS. UPS could have a transformative impact on the fundamental belief of the teaching of freedom.

A major intellectual impact of the proposal could be the realization that the acceptance of permanent creations no longer has the luxury of empirical verification. For example, since DNA propagates to time infinity, drug tests can only be conclusive until infinity, which never arrives. The same applies to UPS.

In conclusion, this proposal attempts to achieve complete automation for software, yet the meaning and the purpose of complete automation are barely understood. However, the mere speculation on the very foundation of life and its mechanical counterpart could impact the basic beliefs in the origin and the destiny of the living system. This proposal can only be considered a humble beginning of a grand expedition to achieve the goal of complete automation.