On the Foundations of Computing Science

By Ulisses Ferreira

Abstract. In the present work, it is observed and demonstrated that the foundations of computing science, and even of science and logics, include forms of inference that are not regarded as valid, in neither logical nor scientific way. The present paper also shows two paradoxes of logics and scientific methods. Taking into consideration a certain rigor, the present paper argues that computing science is not mathematical logics, and that philosophy, psychology and other human sciences are in the foundations of that science.

1 Introduction - The Conceptual Diagram

This section and the following ones introduce one classification that can be used as a tool. An example of connection between this tool and the sample paradoxes that I will present below is the following: there exist two large classes of concepts, analytic and synthetic. Roughly, the former is the concept of *smallness* while the latter is the concept of *greatness*. Thus, the former can be seen as smaller than the latter and, hence, if in some observation a subject from the latter is seen as belonging to the former, there can be some contradiction, which can lead to a possible refutation.

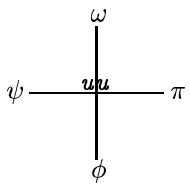
As one of the possible conclusions from my PhD thesis, I identify some of the usual concepts of computing science and place them in only four classes. Therefore, here I attempt to justify the present classification but knowing that it is a transcendental matter impossible to be either proven or refuted or both.

Most attempts to classify real world concepts necessarily lack precision, although attempts are often helpful. Because such classifications are empirical but takes a life time, here I present a classification from my standpoint, which illustrates the orthogonality of some paradigms and issues, skills, methods and approaches in the theoretical foundations. I illustrate how I identify the analytical and synthetic sides of computing science. Due to its empirical characteristic, our four-kind diagrams can be seen as simply a diagrammatic form of representation by some holistic view in computing science. Although fuzzy systems have been regarded as logics in their own right, I place these two notions as opponents in those diagrams, and this is because logics (which one can see as an analytical subject) is traditionally seen as the subject of the valid reasoning, hence, has some general meaning for all individuals. In contrast, fuzziness (which I see as a synthetic subject) typically requires individual truth threshold for each hypothesis, while fuzziness suggests, for every hypothesis, different truth threshold for different individuals. Furthermore, for I am placing logics and logic programming in the same analytical side in the referred to diagrams, and I am stating that

any hypothesis that involves a synthetic subject cannot be generally proven, i.e. with a universally-quantified claim, the thesis of the validity of these diagrams, including my synthetic and empirical classification, is not provable or refutable.

2 Synthesis in Programming

Most work in computing science and AI can be very briefly represented by the following diagram:



One of these hypotheses that I am presenting here is that the programming paradigms which I represent here in the Greek letters that were carefully chosen, π, ω, ϕ, ψ, for example, (functional-logic programming, equations, constraint programming, consistency can be placed in π), (imperative features including side-effects, states, communication between machines, interaction, small mobility can be placed in ω), (internet programming, strong mobility can be placed in ϕ), and (uncertainty, fuzzy systems, inconstancy can be placed in ψ), are orthogonal ways of seeing the world from the programming standpoint. Given this, PLAIN[11] programming language takes into consideration this orthogonality, that is, different points of view. Moreover, PLAIN provides uu (in the center of the diagram, above), which is a constant for representing the non-information. The original idea of uu in logics is back dated to Łukasiewicz, but I extended the same idea to the context of programming languages[12].

Programming is a relatively complex task and, because of this, every programming language has features relating to all kinds. However, one can analyze and perceive predominance to one specific kind in most programming languages. The language Smalltalk, for example, can be associated to the ω kind; while Haskell could be the best example of the π kind. Concerning the ϕ kind, the mobile-code languages for global computers nowadays are still very emphatic on code mobility. More specifically, they have not solved all practical problems with respect to security, which itself is only one problem. There are other issues, e.g. regarding differences of cultures, that have not been investigated. In fact, the ϕ kind is too recent from the programming perspective and one misses comparisons on scientific basis. PLAIN tries to achieve a suitable balance between these four kinds. So, in the next subsections, I shall discuss the four kinds and make comparisons between them.

2.1 Constructs and concepts of kind π

This kind of constructs is not easy to describe because it seems that all programming languages are in here. Formalism, discipline, precision and details are extremely important, and they are key motivations here. Most logic programming languages and functional programming languages are typically in this kind, and because strong typing goes together with discipline and methodology, these languages are not of the same point of view as untyped programming languages, which are in ω, for instance. Objects are concrete and well formed here. Although most of object-oriented programming languages are imperative, inheritance is another example of constructs of kind π. There are exceptions such as C++, which in turn has a strong influence from C. Here, sub-classing can be sub-typing in terms of OOP. The senses of order and pureness are important here. And because of the importance of order and clarity in programming, I regard the kind π as compulsory.

For this synthesis, because there has been much work on logic programming and functional programming, I have preferred not to concentrate on languages of this kind. Instead, in the following subsections, I use them only for comparison.

2.2 Constructs and concepts of kind ω

Here flexibility and expressiveness are keywords. Others are interaction, communication, states, imperative features, and efficiency of time, for instance.

It is interesting to note that, although C programs do not tend to be robust, C is among the most successful programming languages and its historical importance is undeniable. Java is one of its successors.

The most traditional languages used in knowledge-based systems are Lisp and Prolog, which are untyped programming languages, although based on functions and logics, respectively, which are in kind π. In both languages, *list* is a basic data structure that provides flexibility.

Although PLAIN is a strongly typed language, because its designer and other users require flexibility, PLAIN also permits heterogeneous lists in programs. The head of a list is accessed by using the name of the type as the head function call. It has been a nice experience to program with heterogeneous lists, and programmers should be pleasant to program. The following is an example interpreting [ ] as an empty list:

$$\begin{aligned} \mathit{public\ list\ reverse}([\,]) &= [\,], \\ \mathit{reverse}(\mathit{li}) &= \mathit{reverse}(\mathit{tail}(\mathit{li})) + \mathit{head}(\mathit{li}); \end{aligned}$$

In the above example, the *public* keyword states that the function *reverse* can be used by another agent. The + operator here concatenates two lists. The *head* function gives a list with only the first element of its argument. Therefore, the above function reverses the order of the elements of a list regardless of its type.

For this kind of list, the programmer can infer types[20] or, alternatively, instead of the word *head*, use the type identifier to get its first element. In this way, at the point one writes the type identifier in the code, the type checking is made at runtime. In this way, that typical function definition becomes here as follows:

$$\begin{aligned} \mathit{public\ int}\ \mathit{add}([\,]) &= 0, \\ \mathit{add}(\mathit{li}) &= \mathit{add}(\mathit{tail}(\mathit{li})) + \mathit{int}(\mathit{li}); \end{aligned}$$

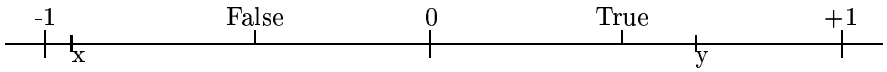
Thus, the above function computes the total value of the elements of a list. Such function definitions, together, illustrate that untyped lists and strong typing can coexist at the language level. A question is how to reconcile the flexibility of programming with the methodologies of pure languages, of kind π. Another question is whether type inference[30, 36] is better or worse than otherwise with respect to programming methodology. For instance, it can be seen that type inference makes *programmers* infer types. Programmers write a piece of code only once and, later, will look at it many times. Therefore, programmers normally infer types in languages like ML. This can be seen as a contrast between kinds ω and π, for the former is predominantly based on assertions and *facts* (type declarations) whereas the latter is predominantly based on *deduction* (type inference). Therefore, this pair of different points of view, ω and π, often indicates the issue of facts (e.g. predicate declaration) and deduction under either open- or closed-world assumption. The present author has proposed the former in [13].

2.3 Constructs and concepts of kind ψ

Here, uncertainty, inconsistency and relativity are some of the keywords for this kind of constructs, ψ. We can include here any constructs for knowledge representation that have vagueness. One approach is to avoid being impersonal and exact, in the programming paradigm, because of the consistency of this perspective. If so, surprisingly, probability theories may not enter this class, in spite of their undeniable importance in science, in general. In terms of programming and knowledge representation, exact probabilities are not easy to be found in the real world, let alone being represented using some formal language. It is known that probabilities are based on assumptions concerning whether events are related to each other. But one underlying subjective issue is "when can we consider that two events are independent from each other?". There seems to be no objective or precise answer for this question. In fact, depending on the philosophical point of view, I have totally different answers. For programming with probabilities, there is one problem of computational complexity[8] and another of representing the web of events, which is a difficult one. A similar web is represented using ad-hoc models, where the basic difference is that probability is traditionally mathematics while ad-hoc models are not so recognized. This means that probabilities require too precise values for representing imprecise knowledge about the real world.

Here, pictures, images, films, and diagrams can represent the imprecision and subjectivity of the real world in a more suitable way. My proposed programming

model for uncertainty extends MYCIN to predict the evaluation of hypotheses by introducing two variables, the least and the greatest factors of certainty, respectively denoted as x and y in the following diagram. As a result, PLAIN provides an intermediary state, called uu, since we have two thresholds for $false$ and for $true$ as follows:



Since x never decreases and y never increases, reasoning is monotonic in this setting. However, if I force $x = y$ constantly in the above setting, it is as non-monotonic as almost all expert system shells with attached certainty factors (most of them are based on production rules). In this way I have two alternative options for all hypotheses. Monotonic reasoning suggests that the truth is only one. By providing two variables, computations also have the ability to predict, at least in some sense, and that is *essential*: the value $y - x$ represents the variables in the program that have not been explored and can contribute to prove or refute the hypothesis. The investigation of the veracity of these variables can be very costly. The system therefore anticipates the evaluation of the hypotheses in accordance with the representation, above. This is similar to *lazy evaluation* in the sense that *variables in the premise list are evaluated at most once.* In this model, inconsistency can be obtained in the form of $False > True \vee x > y$, if the language and its implementation allow this situation.

If one user wants an agent to represent him or her, he or she wants to personalize the behavior of his or her agent if the task is not simple. Due to the huge number of different certainty factors, the community needs suitable languages to program agents.

2.4 Constructs and concepts of kind ϕ

In ϕ, among other notions, as humans, we learn by induction and analogy, and also use metaphors to communicate. We humans need to make comparisons and need broad and abstract views. And since we have had a broad or general perspective, we are able to start solving hard problems in an easy way, and then to investigate them deeply, top down. Aims are necessary, but they should be set only after having such a view and before solving the problem in question.

We humans cannot investigate some complex subject deeply without an initial broad view. Goal-driven programming, i.e. where programmers set goals and the underlying system does the job, is motivated by ϕ. In this sense, Prolog and relational languages have this feature in ϕ. As another example, inductive logic programming is a combination between π and ϕ because, while logical and formal propositions are there, induction and learning by experience are in this class.

Because of its synthetic nature, this class of notions is difficult to implement on a computer, but I am advancing in this direction. Mobile agents, for instance, are applications whose motivations fit in this kind ϕ since mobile agents are flexible and free enough to gain experience abroad.

As regards "mobility", the difference between constructs of kind ω and ϕ is that, in the latter, differences of cultural and religious backgrounds are probably involved, while in the former, mobility essentially is related to changes of states, concurrence, unexpected effects, higher-order functions and hardware circuits: everything happens in the same machine in ω. While ω is driven by issues such as flexibility, communication and curiosity, here truth is a key motivation. In comparison to π, ϕ is not mainly concerned with syntax and details but instead broader concepts, such as paradigms, and also semantics.

In comparison to functional languages in π that adopt type inference, for instance, from the ϕ standpoint, I observe that type inference makes programmers infer types. To answer this kind of question there ought to be experimental and empirical research, not proofs. PLAIN is experimental since it was not conceived for commercial purposes. This is validated by twenty years of experience in programming and compilers. The Unix system and C are two successful examples of what one or two professionals can do, in contrast with PL/I, which in turn was designed by some greater number of people. In this way, it can be better to rely upon one's own experience than to make experimental research among non-experienced programmers, while the result of one's work may happen months or years later.

The fewer the number of programmers the more independence a designer has for trying different constructs. The right moment to release a language to others is relevant, and I am nearly at this point. Yet, by using PLAIN, the author has experienced important insights concerning languages and paradigms. One of the key ideas in PLAIN is its hybrid and well-balanced paradigm, i.e., it tries to combine well-balanced features from different points of view, while accepting that divergences among people and, hence, researchers are natural. Although PLAIN is a large language in comparison to functional languages, it is meant to be concise and relatively smaller, as it provides common constructs. Here rests the difference between a multiple and a hybrid paradigm, at least in comparison to a naïve approach. I have programmed and experimented with different kinds of constructs. The implementation should be sophisticated enough to hide complexity. On the other hand, a good hybrid paradigm is not hard to learn, since learning can be incremental, and makes programming much easier, since I have taken into consideration different kinds of motivations. Thus, if a person likes functional programming, it is easy to program in PLAIN, and if a person likes object-oriented programming, it should be equally easy to program in the same language. However, although the PLAIN philosophy is to be a hybrid language, the concept of "pure function", for instance, (or simply function. It is the opposite to imperative function) is present in the language, they are declarative, never making use of global objects, and this is guaranteed at compilation time. However, functions can be applied from any code of any paradigm. In this sense, PLAIN is different from PL/I where the key motivation was to bring together features for both engineering applications and commercial applications in *the same* paradigm. I believe[15] that, after some time, programmers naturally find that applications suggest the paradigm to use, and that they can benefit from a

hybrid language after some time using a particular paradigm. However, at the present point it is an open issue that deserves future validation and perhaps even further work.

At the practical level, issues such as robustness and security are keywords in the mobile agents field of research. Even in programming, in society, laws have to be established, which stresses the relevance of philosophical issues in programming. There are other keywords, such as dynamic linking (which I did not adopt but may be necessary for efficiency), naming, and global-scope identifiers, i.e. "global" not in the old sense of global variables. The subjective aspects of such issues suggest new or open problems.

3 Synthesis in Knowledge Representation and Reasoning

For AI researchers, although the present classification is not complete (for instance, vision is important for AI and is a synthetic notion related to ψ up to some extent), the four-kind diagram can be seen from the knowledge/belief representation and reasoning/inference standpoint as follows:

- π: perception, precision, specialization, functional programming (traditionally LISP programs), logic programming (traditionally Prolog) and inheritance in class- or frame-based systems. Deductive rules, consistency, deductive reasoning and search. Analysis, complexity and efficiency. Closed-world assumption and negation as failure. Learning by deduction.
- ω: reasoning, in particular non-monotonic, natural negation, ambiguity and redundancy, knowledge, interaction, diversity, curiosity and learning by acquiring facts.
- ψ: feeling, fuzzy logics, uncertainty, partial information, incompleteness, subjectivity, inconsistency and belief.
- ϕ: intuition, perspective, monotonicity, absolute truth, speculation, axiomatization. Semantic web, inductive reasoning, generalization, analogy and metaphors. Objectivity as opposed to subjectivity. Open-world assumption, neural networks, inductive logic programming, broad view, synthesis and common sense. Learning by induction.

In terms of AI, this four classes can represent four types of intelligence. I observe that although the above concepts are interesting for AI, the classification is still the same as for programming languages and for computing science, which will be presented in the next section. While ω and π are analytical kinds, ϕ and ψ are synthetic ones. For instance, a country could well use many deductive rules of logic programming to decide whether a person may be regarded as a citizen of that country or not. However, deductive logics is not very appropriate for, in the airport, deciding whether a person from abroad may enter the home country or whether the person will go back to the place from where he or she came, because the number of rules from the real world is practically impossible to count. Therefore, as programs need synthetic thinking, programming languages should also provide synthetic tools, in particular for mobile agents. In [14], I

presented constructs with uncertainty for filling in this gap. Continuing, while ω and ψ are closely related to internal judgments (by reasoning and feeling, respectively), π and ϕ work like input devices (by five-sense perception and intuition, respectively). This classification has strong influence from Carl Jung psychological types[22], but the observation with respect to computing science is almost entirely based on my own experience in both areas as well as my empirical observation during my PhD studies. Perception above, in the π kind, corresponds to a refinement of what Jung called *sensation*. According to him, the four psychological functions are: thinking, feeling, sensation and intuition. More generally, he called the first two *rational functions* while he called the last two *irrational functions*. These four functions are somewhat similar to what I called ω, ψ, π, ϕ, respectively. Philosophically, Jung's work itself had some influence from Immanuel Kant.

In terms of humanity, the essence of ψ is seen in the romanticism. An atheistic view of ψ is exposed in [34].

Regarding mobility, I can classify its scopes as (individual,ω), (social,π), (geographic,ϕ) and (universal,ψ). With respect to languages, natural and artificial, I can still observe differences among the main concepts: (vocabulary, ω), (grammar and syntax, π), (semantics, ϕ) and (pragmatics, ψ), for instance.

4 Synthesis in Foundations

Mobile agents and the Internet have brought new ideas to theoretical computing science in the last few years.

As an example, I recently had a conceptual insight over computation: "...at the moment that I conceive the idea of moving computation from one place to another, I also observe that a general notion of computation transcends pure mathematics and meets the physical world". This itself requires new, informal and philosophical discussions.

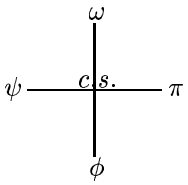
Regarding the two-axis diagram, there probably is the same correspondence in computing science at a very high level. Respectively, psychology (ψ), sciences related to the machine, engineering and physical characteristics in general (ω), mathematics (π) and philosophy (ϕ) form a four-leg table that can support computing science. Typical questions in science can also be placed under this classification e.g. *what*, *where*, *when* (factual and flat information in general, ω); *how* (π); *why* (ϕ); *for what* and *for whom* (ψ). *Who* can be either in ω or ϕ. These four legs are not sharp, too clear, exact or mathematical classification, e.g. in philosophy, the Platonist view can be placed in ϕ while the constructivistic view can be placed in π. On the other hand, in logics, theories of truth, interpretations and model theory can be placed in ϕ while proof theory can be placed in π. These four kinds are somewhat fuzzy[26], relative, incomplete and personal. However, the hybrid semantics of that four-kind diagram contrasts with solely fuzzy views of the universe, e.g. [26].

I have presented two different philosophical views in computing science, that can be summarized in the following way, depending on the adopted meaning for "executable code".

- Traditional view: computation has purely mathematical semantics. There are modal notions, such as mobile computation. Executable code is local to the machine, and a machine is only a physical notion. Correspondence between semantics[31]. Domain theory [1, 3, 19], denotational semantics[2, 38, 39]. Curry-Howard isomorphism[18], rewriting systems[4, 25], category theory[27, 28, 41], functions, logics in computing science and so forth. Functional programming, input and output operations as monads.
- An alternative philosophical view: computation by machines is a unique physical notion. Executable code is mobile, and a machine can be an interpreter. The equivalence between operational and denotational semantics does not necessarily hold from this point of view. In particular, although denotational semantics is still useful, the operational semantics is the most suitable semantics to capture this view of computation if the notions of space and time are part of the semantics. In the real world, uncertainty- or probability-based computation is performed. Computation can be incomplete. Space-time logical calculi. Input and output operations as physical interaction (by side-effect) in the real world.

Notice that these views are not necessarily holistic, although I use the second view to illustrate "computation in the real world" in the present paper.

I can refer to computation in the latter philosophical view as not only computation carried out by machines but also *computation by humans*. For example, interaction for a machine is roughly equivalent to human five-sense perception. However, as will become clearer in this section, while there is some (rough) equivalence between human thinking and machine computation, whether one can reproduce or only simulate human computation in a machine is one more philosophical issue in the foundations of computing science. Continuing the discussion with a diagram:



There is the ϕ kind since we all also learn by induction and analogy, as well as we make use of metaphors to communicate, mainly when the concepts are abstract, and we have only words. Possibly, that is why prophets and visionaries often use metaphors in their speech. And because we humans make use of induction, analogies and also use metaphors in our speech[7], a number of good

examples are both didactically relevant and essential scientific method, as computing science, in the broadest sense, is not totally mathematical. While in ω I talk about knowledge representation, in ψ, one regards with *belief* representation. Although the forms of representation can be the same, these two concepts are different. In ϕ, broad view and intuition are key words that normally lead to prediction[24].

In logics, while some of the main motivation in π would be proof and syntax, some of the main motivation in ϕ would be (perhaps informal) models and probably non-mathematical semantics. Since Gödel incompleteness theorem, it is known that proof (π), truth (ϕ) and incompleteness or inconsistency (ψ) are not equivalent notions in both the mathematical world and the real world. While, in π, *true* and *false* would be mere symbols that are manipulated according to some well-formed syntax, here in ϕ models would be closely connected to the real world. While in π a key motivation is to find differences between apparently similar objects, here in ϕ, a key motivation consists in finding similarities between objects apparently very different from each other. And while π can correspond to Aristotle and deductive reasoning, ϕ can correspond to Socrates and inductive reasoning[35]. Thus, we all need a sufficiently broad view of the world to make good analogies: the broader the view, the better the analogy. Metaphorically speaking, if I place elements of a set in order and I want to connect elements that share a property different from the chosen order, I have to see distant objects, perhaps at once, to make comparisons. Following this reasoning, the notion of perspective and, therefore, distances and mobility, can be obtained. At a somewhat refined level, induction is an orthogonal notion in the sense that it depends on experience and, hence, time.

While in π the related quantifier is existential, here in ϕ the related quantifier is universal. In terms of games, while π may summarize the skills played by Eloisa ($\exists$), ϕ can represent the skills played by Abelardo ($\forall$).

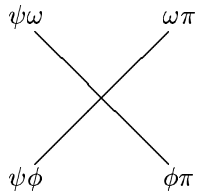
Many aspects of computing science have always had interesting philosophical components. As part of future work, one may want to define those bases explicitly, in a way that they can be identified by undergraduate students over the world. As well as philosophy, there are other human sciences which are important for the foundations of computing science.

With respect to the ψ kind, for being a transcendental notion, while π is concerned with granularity and what can be perceived and countable, ψ, like a set of water molecules, although a countable set, suggests what is uncountable and is beyond one's five senses. Like in modern physics, ψ suggests hypotheses and observable models, but not what can be directly perceived. Thus, belief is an important issue here. ψ, as I lexically suggest, is a place where psychology has something to contribute to computing science. By comparing and stating differences between humans and machines, we humans also learn the limits of what can be computed by machines, an approach complementary to complexity. As an example, if someone wants to build some application for identifying e-mail messages that are important or interesting, he or she has to have a very personal agent computation, probably based on some analogical measure, not

only deductive systems such as natural deduction and Prolog, in which, for each consequent all pieces of its (conjunctive) premise must necessarily be tt. Thus, the notion of "importance" is not only personal but it also naturally requires some analogical interval.

Returning to the π-ψ axis, I can state that ψ suggests computation in the set of complex numbers, while the π kind, although $\pi = 3.141592...$ in arithmetics, is mainly concerned with the set of integers. As a synthesis, Gödel incompleteness theorem, at another level of abstraction, is a piece of work which involves five different notions that I have identified in the above conceptual diagram: (π: proofs and consistency), (ϕ: truth and validity), (ψ: inconsistency), and finally (uu: incompleteness).

I can divide the same diagram in two diagonals as follows



forming two divisions, each of which one can be referred to as *side*. I could refer to ϕ and ψ as "synthetic side of the foundations of computing science" while ω and π would be "analytical side of the foundations of computing science". Therefore, this representation of those foundations of computing science would include the whole picture. The $\phi\pi$ side can be referred to as inductive and deductive reasonings, as well as knowledge/belief acquisition, whereas the $\psi\omega$ side can be referred to as knowledge/belief representation.

With respect to synthesis and analysis, which divide the diagram in two halves, $\psi\phi$ represents the purely subjective side of the foundations of computing science, while $\omega\pi$ represents the objective side of the same foundations. I simplify the language and refer to them as synthetic and analytical sides, respectively, having in mind that they are *not* independent from each other. The former is predominantly inductive while the latter is essentially deductive. In computing science, the analytical and concrete side has been well developed while the importance and even the presence of the synthetic side has been little perceived. This contrast is probably not simply due to the fact that deductions are very easy to implement in a machine. Indeed, while formal proofs are appropriate as a scientific method for the analytical side, *concerning the synthetic side, formal proofs of general claims do not work.* Frege[40] tried to prove that arithmetic was analytic[21]. As an example, tableau calculi are also called the method of *analytic tableaux.* From [5]: *Any calculus that starts with the formula to be proven, reducing it until some termination criterion is satisfied, is called* analytic. *In contrast, a* synthetic *calculus derives the formula to be proven from axioms.* In contrast, I am referring to *analysis* and *synthesis* as opposite mental processes. In fact, not only deductive logics, in its pure sense, is normally

analytical at a higher level of abstraction, for Babbage himself called his work *analytical engine*, and so forth. These pieces of evidence are too informal and synthetic for being used as proof of the validity of this classification. In contrast with logics, there are branches of mathematics that seem to be synthetic.

Because of this contrast between synthesis and analysis, for some logicians, inference based on uncertainty might not be considered as part of logics, at least from the point of view of deduction and the universality of the classical logic. In the present work, I let them be opposites to form the same axis, as, in a sense, they complement each other because their natures are essentially inductive and deductive; synthetic and analytical; on open and closed word; subjective, and exact and with valid values for all; fuzzy and precise etc, respectively. Interestingly enough, logics may be seen as a branch of philosophy, which I classify as synthetic. It is true that nowadays, as there are many logics, to propose a logic one has to have a broader view than he or she would need to simply use the same logic.

I can also see the above diagram from the standpoint of the other diagonal and refer to $\phi\pi$ division as *perception* and *learning* while $\psi\omega$ division may be referred to as *reasoning* and *thought*. Perception here means not only by using five senses (programmers and logicians have to pay attention to forms and details) but also intuition (philosophers and researchers need to make use of their own insights and see the world abstractly, from a broad perspective).

An interesting explanation, extracted from [10], as regards intuitionistic logic: *"What distinguishes the intuitionists is the degree to which they claim the precedence of intuition over logical systems and the extent to which they feel their notions have been misunderstood by classically trained mathematicians and logicians"*. The idea of precedence of intuition over logical systems[32] is in accordance with the idea of trying to view the whole picture without details before starting concentrating on the latter, in a top-down fashion, for those who like software engineering. As known, intuitionism is a philosophical view[9].

As well as perception and intuition[29], in the above diagram, by completing the symmetry, thoughts are not only based on reason (to deduce hypothesis) but also based subjectively on feeling. Feeling is very personal. Even if the researchers decide not to implement such subjects as part of limitations of what computers can do, those subjects are still essential to the foundations of computing science. Nevertheless, it is easy to understand why this ψ kind has not been exploited in computing science, and an answer is that this is also a natural consequence that computers have become increasingly complex. Deep Blue has already beaten the Grandmaster Kasparov in chess, but computing scientists took some time until the machine was able to beat him, a challenge which could be regarded as relatively simple, besides its computational complexity.

So far, there has not been any implementation running in the computer that could be regarded as representing feeling, at least in a universal way. Nonetheless, the notion of agent was introduced to represent people in their transactions in the daily life. The diagonals of the diagram also represent the idea that, among many other skills, human beings learn by communication and facts, perception

and deduction, intuition and induction, feeling and belief. The diagram is not complete with respect to this either, for instance, motivation and pain are outside the present classification.

The computing science and AI communities have been discussing the differences between humans and machines in terms of the meaning of thoughts. For instance, in [5], the same applies to discovering proofs and deduction. Even with a set of wffs, deduction might be difficult for a machine due to the possibility of combinatorial explosion. Here one has two philosophical views supporting the answers for the question about whether we humans are machines, or whether we humans are much more than this. Not only here but also as part of my PhD thesis dissertation, I have presented, as examples, some skills that form the synthetic side in the diagram. Although specific answers for such questions are outside the scope of the present piece of work, one of the main results from my observations is that, *if one wants to adopt the view which equates any person with any machine, first it is a good idea to study intuition, feeling, analogy, induction, belief and other subjects which are parts of the human beings, and thus, philosophy and other human sciences.*

Finally, technology introduces useful insights into theoretical computing science. The introduction of mobile agent technologies has led the community to want agents to be autonomous and flexible to represent users. For simple tasks, there is relatively little to add to what we humans already know. But users want to use complex application programs.

To truly represent people in our complex society, agents *have to* simulate subjective thought too, we humans want them to behave based on our personal tastes and personal opinions, for instance. Therefore, knowledge or belief representation can also be seen as a programming paradigm, not only as a subject in AI. The aim of introducing subjectivity in programming languages is difficult and ambitious. The author used to work on an expert system for diagnosis of heart diseases and was intrigued when observing that the users, experts, did not want to attach real numbers (in fact, floating-point numbers) to our rules as part of the certainty measure. Instead, I made use of words, carefully chosen to represent those real numbers. From that experience, the rôle of subjectivity in programming could be felt: "What do such subjective words mean?". There seems to be some common agreement which, in those cases, is more important and easier to understand in our every-day life than numbers. More recently, intelligent agents have been conceived to represent users and, for some reason, some of us also want them to be mobile. This scenario leads the present paper to conclude that other more subjective, less exact sciences are also essential in the foundations of computing science.

The intuitive part of this work is also based on a symmetry in pairs of four key concepts, namely, knowledge(ω)-intuition(ϕ), deduction(π)-belief(ψ). A difference between knowledge and belief is that belief is a kind of weak knowledge. In this sense, there exists some threshold between the two, and which determines what is knowledge and what is belief. Furthermore, this threshold is itself of course synthetic, and somewhat subjective. In short, a matter of belief. As

an example, although it may be safe to state that most men acknowledge that Marilyn Monroe was beautiful, it is reasonable to think that beauty is subjective.

5 Sciences and Deductive Logics

Based on observation, we can classify concepts related to computing science, such as methods, forms of inference, mental abilities and subjects of research, dividing them in two classes, i.e. synthetic and analytical. This classification, for being original, differs, for example, from the classification of Immanuel Kant[23]. Here, there is no formal nor precise definition of *synthetic* and *analytical*. Roughly speaking, some analytical idea is motivated by exactness, whereas some synthetic thing is motivated by generality. In this way, this classification itself is synthetic. The analytical class is divided in two subclasses, namely, ω and π, whilst the synthetic class is divided in two other subclasses, namely, ϕ and ψ. Deductive logics is a key concept that belongs to the analytical class, whereas induction and analogy are two of the key concepts that belong to the synthetic class.

Logics and exact sciences, in turn, as it is known, are partially based on deductions as well as observable and deduced facts, while such facts can be used as examples, which in turn can be used as proofs for existential propositions. On the other hand, propositions based on a finite number of cases are traditionally regarded as less relevant. As an example, it is known that belief, induction and analogy have not been accepted as valid methods of sciences[33], in particular, mathematics and exact sciences.

Nonetheless, synthetic concepts are significant in computing science and, among them, there are those empirical concepts, together with belief, induction, analogy and so on. A few forms of inference, such as the ability to weigh up possibilities, can be deeply studied in computing science, while the ability to weigh up possibilities is not traditionally regarded as a mathematical method.

Here it can be observed that computing science not only profits directly from logics and mathematics but that that science has a direct connection with the real world, while it belongs to the same world. Furthermore, the typical place where computing science has many notions of the synthetic side is in the interaction with the real world, including the interaction with our senses at work, and in the *applications* of the analytical notions. An example of this is in section 5.1. I see computing science as the organized knowledge about the world. In fact, while the logical, scientific and mathematical kind π in the foundations of computing science has been well studied since Babbage, no significant contribution was done in *philosophy of computing science*. Today, this is a new side of the foundations of computing science.

In general, these synthetic aspects are subjects inside philosophy and human studies. Citing Immanuel Kant[23], "it is absolutely necessary that humanity believe in God". Science and religion have been traditionally very separate from each other. Nonetheless, like religion, science seeks the truth, although its methods are constrained.

In this Kantunian sense, what is most useful to science is not only to acquire knowledge on whether Jesus of Nazareth really existed, or what he did, in small details etc. The concept of one or more entities morally superior to humans, and who have the properties of omnipotence, omnipresence and omniscience, certainly has relevant impact on sciences. As a clear example, if an individual is atheist and their proposal is very ambitious (for example, Sigmund Freud, for there is some evidence[6]), selfish or urges for power at any cost, he or she may manipulate results or forge input data in the absence of other scientists, or may steal someone else's ideas or the credits of someone else's work, as what he or she knows is that they are not going to be punished, whereas, typically, religious scientists do not *normally* do those for they know that, later, they would be somehow punished for the fault in question. Therefore, whether we humans believe or not, God's eyes play some relevant rôle in science.

In the short history of computing science, there has already been the public case of the suicide of Alan Turing, with the additional observation here that, as it is known, such an action can be seen as very anti-ethical, and the reason may be as follows: Given that all deaths, in particular suicides, should be known to the scientific community, if the whole humanity had been influenced by him and had done the same sort of thing, science would be extinct together with the humanity. Therefore, any suicide is an action against science, at least it should be seen as such, and this makes verifications of all the individual's work appropriate, if he or she had had international or historic reputation but made such a moral mistake, while scientists personal lives should almost never be of great interest for the public. As well as his or her work, the scientist's ethic-religious system is what is important for others. In the case of Dr. Turing, he may be a hero for his work during the II world war, but his suicide was a public action somehow against the reputation of his own computing science.

Likewise, striking from behind by gossips, slanders and non-authorized (personal or professional) references are also very anti-ethical as, like a chemical or biological weapon, they irradiate in the academic and scientific community without giving any opportunity to the victim to defend himself or herself, at least in time to avoid losses, if the victim is not intelligent and wise enough, which is not the general situation. And because sciences seek the truth, other researchers are also victims of those lies in question. Therefore, given that the material of computing science is information, ethics is essential indeed, and should be taught as part of the formal course.

In the following sections, a couple of examples of paradoxes in science and logics are presented. Each of them proves that philosophy and psychology are in the foundations of computing science.

5.1 A Paradox Example - Analogy

This section proves by contradiction that mathematical logics is not the only foundation of computing science.

An example of this is in the content of deductive proofs. Regardless of how mathematical and formal the problem be, what makes some given proof support

some problem **X** and not some different problem **Y**, is the *analogy* which is made between a given problem and the representation of its relevant context, whether this representation is formal or not. Applied proofs are regarded as correct only if

1. The deduction is logically valid, i.e. in accordance with the deductive rules of the used logic.
2. The representation of the problem is correct and, in particular, complete where the variable is universally quantified.

The second item describes a basic analogy, which in its turn is essentially an informal concept, somewhat personal. On the other hand, this concept is fundamental in the context of applied logics and, hence, of the computability theory, and so on. As a metaphor (analogy), this resembles the incompleteness theorem, because the fact that analogy is not part of the mathematical methods implies that those methods are not sufficient in computing science. That is, analogy supports mathematics while the former is not supported by the latter. Therefore, mathematics is not sufficient in this rigorous sense, while rigor is a kind of ideal in mathematics. As a consequence of this, philosophy, psychology and other human studies which deal with analogy[7], are new components in the foundations of computing science, whereas philosophy and psychology support mathematics and computing science. In this way, mathematics in computing science is a fundamental tool for letting the involved concepts be precise and clear.

5.2 Another Paradox Example - Induction

The previous example applies to the content of proofs. In contrast, the present example applies to the human inference forming generalizations, which one refers to as induction[16, 17]. As it is well known, both deductive logics and sciences, in a rigorous sense, reject not only analogy but also induction. By the way, both analogy and induction are referred to by philosophy as *irrational*. The fact is that, in rigorous way, both science and deductive logics reject the inductive method, in particular because its validity is not universally acceptable. Nonetheless, science itself works in the presence of panels (i.e. committees) in Master or PhD courses/research, in selections for professorship, and considerations for publications. In the end, however mathematical the particular subject is, it depends on the present induction. For the same level of reputation, even taking it as the minimum, we humans tend to *believe* that the more the number of examiners, the more accurate the result is. However, this exemplifies that logics and science work by using some method rejected by them, which constitutes a paradox at the practical level. Establishments can rely upon the intellectual attributes of the examiners, but *belief*[37], as a synthetic concept, cannot be supported by logics, mathematics nor sciences in some rigorous sense.

Moreover, between us, the above proof (as well as the previous one) not only applies to computing science... It shows how logics itself is still part of philosophy!

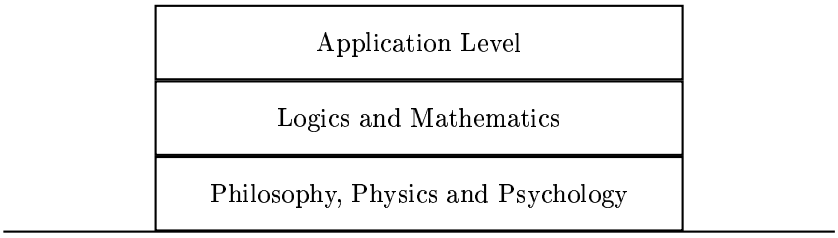
This is the most general meaning of the above deduction, whereas one of the consequences is that philosophy is in computer science, in the foundations.

The present author observes from both examples, above, that although the synthetic and human aspects are not entirely supported by logics, the latter is always supported by human beings in a synthetic way. Note that logics is traditionally philosophy. This shows the hierarchy of the large subjects inside the foundations of computing science, for philosophy and psychology can support the others. From this, different *theories* of computing science with synthetic notions can exist instead of a unique, mathematical and analytical theory. The novel area is called *philosophy of computing science.*

6 Conclusions

This paper is a synthetic discussion, on philosophy of computing science, presenting some of the links between concepts, trying to describe a semantic network. Because of this, given its purpose, it was not meant to be like almost all scientific papers in computing science, where there exists a precise focus on a particular and analytical subject.

Logics, in a rigorous sense, is not the only foundation for philosophy, psychology, computing science etc. More than this, it was shown that there can exist the following hierarchy of subjects in computing science:



Computing science has the bizarre characteristic of being both exact and philosophy for being related to the reality. For these two facts, methods rejected by science that are applied in the daily life should be considered. In the above example of committee, in order to make the applied method be consistently scientific, one should consider not only the object, the criteria and grades, but also other variables of the real world, such as the names of the examiners, which in turn ought to be public. In this case, the scientific knowledge would necessarily be referred to as something broader.

It is known that programming is one of the key subjects in computing science. However, if one observes the somewhat empirical characteristics in programming, the ideas contained here will become clearer. Yet, there seems to be nothing wrong in the way that programmers still work, and will probably continue doing.

Furthermore, although one can prove that a given program is correct, there can be proofs of proofs of program correctness etc. Programs are either correct or not with respect to some representation of a relevant model from the real world. Therefore, although there are programming techniques including those suggested and imposed by programming languages, programming is a complex task that requires some very basic synthetic skills.

My classification is itself synthetic and, as such, can be neither proved nor refuted. However, exceptions exist. Other synthetic subjects can be existentially proved, and some such subjects can be equally refuted. The classification here is essentially based on intuition, analogy and induction (not only because of any possible contribution to three different areas of research, but also because of its inductive nature, I had to present sample applications to programming languages, knowledge representation, and foundations of computing science), as well as many observations on the real world. On the other hand, such a classification is not scientific by its nature, only philosophic, but philosophy and science go together. The term *science* has a sense weaker than *computer science* has had since Gödel and Turing.

Finally, the two-axis diagram reflects only some particular philosophical view. Because of this, I do not expect that it can be used as a universal tool, nor accepted by the whole community as valid. However, sections 5.1 and 5.2, in a sense, show that the diagram somehow can be useful, by taking two concepts classified in ϕ, and because, according to the corresponding classification, logics belongs to the class π. Naturally, the classification can be used by others who like it.

Acknowledgement

I want to thank the people from The Trinity College, University of Dublin, for having approved my research towards the Doctor in Philosophy.

References

1. S. Abramsky. *Handbook of Logic in Computer Science*, volume 3: Semantic Structures, chapter Domain Theory, pages 1–168. Oxford University Press, 1994.
2. L. Allison. *A Practical Introduction to Denotational Semantics.* Number 23 in Cambridge Computer Science Texts. Cambridge University Press, 1986. Reprinted 1995.
3. R. Amadio and P.-L. Curien. *Domains and Lambda-Calculi.* Number 46 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
5. W. Bibel and E. Eder. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1: Logical Foundations, chapter Methods and Calculi for Deduction, pages 67–182. Oxford University Press, 1993.
6. S. Blackburn. *The Oxford Dictionary of Philosophy.* Oxford University Press, 1994.

7. D. Burrell. *Analogy and Philosophical Language*. Yale University Press, 1973.
8. A. Chagrov and M. Zakharyaschev. *Modal Logic*, volume 35, chapter Complexity Problems. Oxford University Press, 1997.
9. M. Dummett. *The Philosophy of Mathematics*, chapter The Philosophical Basis of Intuitionistic Logic. Oxford Readings in Philosophy. Oxford University Press, 1996.
10. R. L. Epstein. *The Semantics Foundations of Logic*, chapter Intuitionism, page 277. Oxford University Press, 1995.
11. J. U. Ferreira. The plain www page. *URLs http://www.ufba.br/~plain or http://www.cs.tcd.ie/~ferreirj*, 1996–.
12. J. U. Ferreira. *uu* for programming languages. *ACM SIGPLAN Notices*, 35(8):20–30, August 2000.
13. J. U. Ferreira. *Computation in the Real World: foundations and programming languages concepts*, chapter 9 **uu** in Globallog. PhD thesis, The Trinity College, University of Dublin, 2001.
14. J. U. Ferreira. *Computation in the Real World: foundations and programming languages concepts*. PhD thesis, The Trinity College, University of Dublin, 2001.
15. P. Forrest. *The Dynamics of Belief: A Normative Logic*. Philosophical Theory. Basil Blackwell, 1996.
16. D. Gillies. *Artificial Intelligence and Scientific Method*, chapter 1 The Inductivist Controversy, or Bacon versus Popper, pages 1–16. Oxford University Press, 1996.
17. D. Gillies. *Artificial Intelligence and Scientific Method*, chapter 5 Can there be an inductive logic?, pages 98–112. Oxford University Press, 1996.
18. J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1993.
19. C. Gunter and D. Scott. *Handbook of Theoretical Computer Science*, volume B Formal Models and Semantics, chapter 12 Semantic Domains, pages 633–674. The MIT Press/Elsevier, 1990.
20. C. A. Gunter. *Semantics of Programming Languages: structures and techniques*. Foundations of Computing Series. The MIT Press, 1992.
21. I. Hacking. *What Is a Logical System?*, chapter What Is Logic?, pages 1–33. Number 4 in Studies in Logic and Computation. Claredon Press, Oxford University, 1994.
22. C. G. Jung and A. Storr. *Jung: Selected Writings*, chapter Psychological Typology (1936), pages 133–146. Fontana Pocket Readers. Fontana Paperbacks, second edition, 1986. Selected and Introduced by Anthony Storr.
23. I. Kant and translation by Norman K. Smith. *Immanuel Kant's Critique of Pure Reason*. Macmillan Press Ltd, 1787, 1929.
24. F. N. Kerlinger and H. B. Lee. *Foundations of Behavioral Research*. Harcourt College Publishers, fourth edition, 2000.
25. J. W. Klop. *Handbook of Logic in Computer Science*, volume 2 Background: Computational Structures, chapter Term Rewriting Systems, pages 1–116. Oxford University Press, 1992.
26. B. Kosko. *Fuzzy Thinking: The New Science of Fuzzy Logic*. HarperCollinsPublishers, Flamingo, 1994.
27. S. M. Lane. *Categories for the Working Mathematician*. Graduate texts in mathematics. Springer, second edition, 1998. Previous edition: 1971.
28. F. W. Lawvere and S. H. Schanuel. *Conceptual Mathematics: a first introduction to categories*. Cambridge University Press, 1997.
29. P. Maddy. *The Philosophy of Mathematics*, chapter Perception and Mathematical Intuition. Oxford Readings in Philosophy. Oxford University Press, 1996.

30. J. C. Mitchell. *Foundations for Programming Languages.* Foundations of Computing. The MIT Press, 1996.
31. C.-H. L. Ong. *Handbook of Logic in Computer Science*, volume 4: Semantic Modelling, chapter Correspondence Beteen Operational and Denotational Semantics: the full abstraction problem for PCF, pages 269–356. Oxford University Press, 1995.
32. C. Parsons. *The Philosophy of Mathematics*, chapter Mathematical Intuition. Oxford Readings in Philosophy. Oxford University Press, 1996.
33. K. Popper. *The Logic of Scientific Discovery.* Karl Popper, 1972.
34. B. Russell. *History of Western Philosophy*, volume III, part 2, chapter 18, The Romantic Movement, pages 651–659. Routledge, 1946. Edition published in 2000.
35. B. Russell. *History of Western Philosophy*, volume I, part 2, pages 101–226. Routledge, 1946. Edition published in 2000.
36. D. A. Schmidt. *The Structure of Typed Programming Languages.* Foundations of Computing Series. The MIT Press, 1994.
37. M. Swain, editor. *Induction, Acceptance, and Rational Belief.* D. Reidel Publishing Company, 1970.
38. R. D. Tennent. *Semantics of Programming Languages.* PHI series in computer science. Prentice Hall, Inc., 1991.
39. R. D. Tennent. *Handbook of Logic in Computer Science*, volume 3: Semantic Structures, chapter Denotational Semantics, pages 170–322. Oxford University Press, 1994.
40. J. van Heijenoort. *From Frege to Gödel.* Harvard University Press, 1967.
41. R. F. C. Walters. *Categories and Computer Science.* Cambridge Computer Science Texts. Cambridge University Press, 1991.