Logic and Computers.
A Sketch of Their Symbiotic Relationships

Jordi Vallverdú
(Philosophy Department, Universitat Autònoma de Barcelona)
jordi.vallverdu@uab.cat

0. Introduction

It is a well established fact that Logic and Computer Sciences have a long and close relationship, which has benefited both sides of this symbiotic relationship. Therefore, this is a case of mutualist symbiosis, a relationship in which both fields benefit. Nevertheless, and according to the main studies about this topic, this relationship could looks like as a parasitist or commensalist one: that is, one field benefits but the other is harmed (in the first case), or one field benefits, while the other remains unaffected (in the second one).

Our departure hypothesis is: not only logic has influenced computer sciences development (a common topic on the historiographical approaches to computer sciences)[1], but computer sciences have also changed the logical thinking and practice. This is a mutualist symbiosis. This second idea, the impact of computer sciences into logical field, has been overlooked by most of the researchers who have studied the relationships between both fields. This symmetric symbiosis reaches from practical to very fundamental aspects of the relationship.

To be honest, we are not just talking about logic and computer sciences fields, but also of engineering and mathematics. From a conceptual and a practical point of view, there are crucial interconnections between all these

[1] Davis 1988, 2000.

disciplines. Logic is a parent discipline of computer science[2], as well as a basic constituent of computer science curricula. At the same time, logic has provided computer sciences with several tools, methods and theories. But more than a similarity there is also an equivalence relation between programs and logical formula[3]: a program is a sequence of symbols constructed according to formal syntactic rules and it has a meaning which is assigned by an interpretation of the elements of the language. These symbols are called *statements* or *commands* and the intended interpretation is an execution on a machine rather than an evaluation of a truth value. Its syntax is specified using formal systems but its semantics is usually informally specified. Then, a statement in a programming language can be considered a function that transforms the state of a computation. Going further[4], we can affirm that there is no line between software and mathematics. Translation problems between them are trivial, because they are equivalent. Therefore, by the Church-Turing Thesis we can establish an identity between *Programs, Logics and Mathematics*. There is a common nature between these domains, with different historical backgrounds and main objectives, but at the end they speak the same language. Ben-Ari affirms that

> a program is not very different from a logical formula. It is a sequence of symbols which can be parsed according to formal syntactical rules and it expresses a meaning, in this case a computation, according to the intended interpretation of the atomic elements and connectives of the language. In programming, these elements are called *statements* or *commands* because the intended interpretation is machine execution rather than just computation of a truth value[5].

According to the *Handbook of Logic in Computer Science*[6], there are several fields in which logic is present into computer science:

programming language *semantics*: to make sure that different implementations of a programming language yield the same results, programming languages need to have a formal semantics. Logic provides the tool to develop such semantics.

- type theory, linear logic, categorical theories
- Lambda calculus, Pi calculus

[2] Thomas, 2000.
[3] Ben-Ari, 1993.
[4] As Klemens, 2006, states.
[5] Ben-Ari, 1993, 244.
[6] Abramsky, Gabbay & Maibaum, 1993.

- specification logics (e.g. dynamic logic, Hoare logic, temporal logic…)
- Finite model theory, data base theory
- Term rewriting, unification, logic programming, functional programming
- Automated theorem proving

Program validation and verification (V&V): it uses temporal logic (finite state systems can naturally be represented as models for temporal logic; verifying if a system satisfies some desired property thus amounts to checking that the formula representing the specification is satisfied by the model representing the system).

- Process calculi and concurrency theory
- Modal logic, logics of knowledge.

Logic plays also an important role in very different areas of Computer Science, like computer architecture (logic gates), software engineering (specification and verification), programming languages (semantics, logic programming)[7], databases (relational algebra and SQL), artificial intelligence (automatic theorem proving), algorithms (complexity and expressiveness), and theory of computation (general notions of computability).

This is the reason for which specialists of these disciplines have worked together or have established working relationships. This is the reason of the present paper: to show the relationships between computer and logic research domains and how they have influenced each other.

1. From Logic to Computers

1.1. A little bit of history

In the second half of the nineteenth century a new branch of logic took shape: mathematical logic. Its aim was to link logic with the ideas of arithmetic and algebra, in order to make logic accessible to the algebraic techniques of formula manipulation[8]. Deductive reasoning could be then reduced to algebraic formulisms. The author who made this possible was George Boole (1815-1864). By reducing Logic to Algebra, Boole made possible to introduce

[7] At the same time, logic programming provides attractive models for inferential information flow (Benthem & Martínez, 2007: 54).
[8] Thomas, 2000.

mathematics into logical thinking through the Boolean Algebra. At a certain point, Boole realized Leibniz's conception of an *algebra of thought*. With only numbers 0 and 1, Boole built up an entire algebra, turning logic into a fully symbolic practice which could be computed easily. So easy that its implementation into a computational binary framework made Boolean circuits possible. This late idea came from Claude Shannon, whose 1938 master's thesis in electrical engineering showed how to apply Boole's algebra of logic into electronic switching circuits. Although some of the first computers were not working on a binary approach, like ENIAC or Harvard's Marc I (still decimal), very soon all computers used Boolean algebra implemented into Boolean circuits.

Between 1890 and 1905, Ernst Schröder wrote his *Vorlesungen über die Algebra der Logik*, where developed the ideas of Boole and De Morgan, including at the same time ideas of C.S. Peirce showing that the algebraization of logic was possible and with a great impact for the new logic studies.

By independent ways, the mathematic Charles Babbage (1791-1871) and his co-researcher Augusta Ada Byron, Countess of Lovelace (1816-52) and a mastered young mathematician too, developed the Difference Engine, first, and Analytical Engine, second. To be precise, Ada only collaborated in some aspects of the second machine: she expanded considerable effort on developing the first programs for the planned machine and on documenting its design and logic. In fact, Ada developed a method for calculating a sequence of Bernoulli numbers with the Analytical Engine, which would have run correctly whether the Analytical Engine had ever been built. Both machines were far beyond the capabilities of the technology available at the time. Nevertheless, as a theoretical concept, the idea of the Analytical Engine and its logical design are of enormous significance. This is the first realization that, by mechanical means, it might be possible to *program* complicated algorithms. The Analytic Engine had, in principle, all of the important components (Memory, Processor and Input/Output protocol) that are present in modern-day computer systems. For this reason Babbage has a strong claim to be the inventor (even if not the first builder) of the modern computer. Designed for military purposes (calculating firing maritime ballistic tables), the Difference Engine was intended to evaluate polynomial functions, using a mathematical technique called the *Method of Differences*, on which Babbage had carried out important work. With later digital computers was possible to

apply propositional calculus to the two voltage levels, arbitrarily assigning the symbols 0 and 1 to these voltage levels.

With his *Begriffschrift* (1879), Gottlob Frege pioneered first attempts to put into symbols natural languages, which were used to make science. He tried to achieve the old Leibniz's dream: calculus as rational thinking (with the *characteristica universalis*), trying to develop a syntactic proof calculus that could be as good as mathematical proofs. Unfortunately, the system Frege eventually developed was shown to be inconsistent. It entails the existence of a concept R which holds of all and only those extensions that do not contain themselves. A contradiction known as "Russell's Paradox" follows.

Very often, Zermelo and Russell discovered the set theoretic paradoxes, which demolished the main logic proof building. Russell and Whitehead published between 1910 and 1913 their *Principia Mathematica*, in which they re-established the foundations of pure mathematics in logical terms[9]… something not so useful for practical purposes if we consider the fact that both authors required 379 pages to justify the truth of '1+1=2' (in the Volume I, §54.43 and completed in Volume II, §110.643).

The same bad news for idealistic approaches to mathematical entities were found to Hilbert's Program about the foundations of mathematics after Gödel's ideas on incompleteness. According to him, any logical system powerful enough to include natural numbers was also necessarily incomplete. Beyond this idea, Gödel also introduced the fundamental technique of arithmetization of syntax ('Gödel-numbering'), which led to the transformation of kinds of data useful for the purposes of computation[10]. At the same time, Gödel introduced the first rigorous characterization of computability, in his definition of the class of the primitive recursive functions.

Alfred Tarski created in 1929 the notion of independent semantics, a very important idea for the future of logics. The syntactic concepts comprising the notation for predicates and the rules governing inference forms needed to be supplemented by appropriate semantic ideas and Tarski was the guy who created the fundamentals of the semantics of First Order Logic (Robinson, 2000). With these conceptual tools, Tarski provided a rigorous mathematical concept of an interpretation.

With a deep insight for the work of the future, David Hilbert proposed stimulant problems which required from enthusiastic mathematicians

[9] Flach, 2005.
[10] Dawson, 2006.

generations for its solution. One of the problems, the *Entscheidungsproblem*, interested to a young and brilliant mathematician: Alan Turing. In 1936, Turing wrote a crucial paper, *On computable numbers, with an application to the Entscheidungsproblem,* in which he proposed the idea of an universal machine (the antecedent of the programmable processor). At the same time, Turing demonstrated that this problem was undecidable for first-order logic. The infallibility was off the agenda, because no fixed computation procedure by means of which every definite mathematical assertion could be decided (as being true or false). In the same year of 1936, Emil Post published a characterization of computability remarkably similar to that of Turing[11].

Nevertheless, four years before Turing's crucial paper, Alonzo Church introduced lambda calculus, as a new kind formulation of logic (basing the foundation of mathematics upon functions rather than sets) and a way to go away from the Russell paradox. Nevertheless, the Kleene-Rosser paradox showed that the lambda calculus was unable to avoid set-theoretic paradoxes. Today, lambda calculus is at the heart of functional programming and has had a big influence in compilations representation as well as in reasoning representation. The Lambda-definability made possible developing functional programming (like Lisp), also creating efficient compilers for functional languages. And related to the computer algebra, mathematical proofs by computers can be done more efficiently by lambda terms[12].

Also in 1936, Church realized that lambda calculus could be used to express *every* function that could be computed by a machine[13]. Perhaps Gödel had broken the gold dream of logical perfection but people like Church still worked on to developing more powerful logical systems. From 1936 to 1938, Turing had studied with Church at Princeton University. Some years later, the mathematician John von Neumann, developed his idea of a computer architecture after the ideas of Turing, leading to the so-called "von Neumann architecture"**,** that has been implemented in most of actual computers[14].

But not only Church established a strong relationship between logic and computability. In 1960 Curry first noticed what a bit later was called the *Curry-*

[11] van Benthem *et al* 2006.
[12] Barendregt, 1997.
[13] Wadler, 2000.
[14] There is also the Harvard architecture are several middle or combined options, like the Modified Harvard architecture. In fact, most modern computers instead implement a modified Harvard architecture.

Howard Isomorphism. The fundamental ideas of Curry-Howard are that proofs are programs, that formulas are types, that proof rules are type checking rules and that proof simplification is operational semantics. At the end, that the ideas and observations about logic were ideas and observations about programming languages.

Nevertheless, we must go back to the year 1954. In that date, Martin Davis carried out one of the first computational logical experiments: he programmed and runned the Presburger's Decision Procedure for the first order theory of integer addition[15]. He proved that the sum of two even numbers was itself an even number. A year later, in 1955, Evert Beth and Jaakko Hintikka (independently) described a version of the basic proof procedure, being a computationally powerful technique.

Dag Prawitz in 1960 made another important (re)discovery: the *resolution rule*, described something obscurely thirty years before by Herbrand in his Ph.D. Thesis and definitively coined in the actual form by John Alan Robinson in 1963. The process of its creation consisted on combining a known inference rule with unification. Ironically, the new mechanical efficiency brought also a cognitive opacity. According to Robinson, resolution was more machine-oriented than human-oriented[16]. Although resolution was to be a great contribution the paper in which it was described remained unpublished in a reviewer's desk for a year. Artificial intelligence and formal logic converged into a main research field, although the MIT view was being build (as we'll see in the next section).

This same critical process led to Larry Wos and George Robinson (at Argonne) to the development of a new rule, which they called *paramodulation*. This rule was implemented into several famous Argonne theorem provers, being perhaps one of the most famous William W. McCune's OTTER (see section 2.3). Theorem proving was one of the first field in which symbolic (as opposed to numeric) computation was automated.

All these results made possible a functional automated deduction technology, useful for several fields of academy as well as of industry.

[15] Robinson, 2000, 8.
[16] MacKenzie, 2004, 79.

1.2. The summer of the heuristic approach and other things

As Robinson (2000) remarks, two very important meetings for the history of computational logic took place in the 1950s:

Darmouth Conference (1956): this conference, championed by John McCarthy, gave birth to the field of AI. The crucial results of the common research of Herbert Simon (an economist) and Allen Newell (a mathematician) were presented there. They created a heuristic theorem-proving program, using the computer JOHNNIAC at RAND Corporation. With their computer program Logic Theorist, they solved automatically thirty-eight of the first fifty-two theorems in chapter 2 of the *Principia Mathematica*. Simon tell us that after informing Russell upon these results, he received an ironic answer: "if we'd told him this earlier, he and Whitehead could have saved ten years of their lives. He seemed amused and, I think, pleased"[17]. Logic Theorist can be considered like the first Expert System (ES).

Cornell Summer School in Logic (1957): Plenty of researchers attended this course; among them, Martin Davis, Hilary Putnam, Paul Gilmore and Herbert Gelernter. As Robinson (2000) recalls, Gelernter (a heuristic enthusiast from IBM) provoked Abraham Robinson to give a lecture using these methods and focusing on proof seeking. Gelernter's lecture influenced Gilmore, Davis and Putnam, researchers that would write their Herbrand-based proof procedure programs.

Without developing a long history about AI debates, I consider necessary to expose, at least briefly, some historical notes on its basic schools. There were two basic AI approaches[18]:

a) The MIT View: it considered AI as a heuristic, procedural, associative way of producing artificial-generated knowledge. Marvin Minsky and Seymour Papert were members of this approach. For these authors, formal logic was inadequate for the representation of knowledge required by any general approach to AI. They considered such a view as too static and rigid, preferring a procedural approach.

b) The Edinburgh-Stanford View: on the other hand, we have the logic view, championed by John McCarthy, who considered that AI knowledge could be mechanized because it could be axiomatized declaratively using

[17] Stewart and Simon, 1994.
[18] Robinson, 2000.

First Order Logic. They considered computational logic as the only way to achieve an Artificial Intelligence.

To be honest, both approaches were highly symbolic had more things in common than differences. In the middle of an AI's civil war, they were also called *neats* (logicists) and *scruffies* (proceduralists). It was only later that two real AI confronted approaches appeared, which can be summarized as *top down* and *bottom up* approaches:

i. *Top Down:* symbol system hypothesis (Douglas Lenat, Herbert Simon). The *top down* approach constitutes the classical model. It works with symbol systems, which represent entities in the world. Following to Brooks (1990, 4): "The symbol system hypothesis, states that intelligence operates on a system of symbols. The implicit idea is that perception and motor interfaces are sets of symbols on which the central intelligence system operates. Thus, the central system, or reasoning engine, operates in a domain independent way on the symbols". SHRDLU (Winograd), Cyc (Douglas Lenat) or expert systems are examples of it.

ii. *Bottom Up*: physical grounding hypothesis (situated activity, situated embodiment, connexionism). On the other side, the bottom up approach (championed by Rodney Brooks), is based on the physical grounding hypothesis. Here, the system is connected to the world via a set of sensors and the engine extracts all its knowledge from these physical sensors. Brooks talks about "intelligence without representation": complex intelligent systems will emerge as a result of complex interactive and independent machines.

In this sense, the MIT View and the Edinburgh-Stanford View both belonged to the *top down* approach. But let me continue with the conceptual analysis of computational logic.

1.3. About programming languages and several logics

During the 20th century logicians, mathematicians, philosophers and computer scientists studied a wide range of alternative formalisms designed for specific applications which do not appear to be easily handled by classical logic[19]: intuitionist, temporal, etc. Temporal logic, for example, is widely used

[19] Galton, 1992.

for automated verification of semiconductor designs, as a specification language for design[20].

And as Wadler (2000) remarks, in 1969, W. A. Howard put together the results of Curry and Prawitz, and wrote down the correspondence between natural deduction and lambda calculus Moreover, Wadler also states that the "Curry-Howard correspondence led logicians and computer scientists to develop a cornucopia of new logics based on the correspondence between proofs and programs." This implied the common work between logicians and computer scientists.

Not all philosophical efforts towards better reasoning were developed under deductive modes of reasoning: the work of Mill or Peirce in non-deductive modes of reasoning, like induction or abduction, influenced also the field of Artificial Intelligence[21]. Traditional logic was not only flawed, it lacked soundness and completeness: *non-monotonic logics* appeared as the way to develop new powerful logics. But as Turner (1985) notes, while non-standard logics were investigated by philosophers and logicians since the 1930's at least, it was only in the late 1960's that McCarthy and others began to exploit those resources, in AI. And as Ben-Ari (1993) notes, the special demands of computer science generated interest in non-standard logical systems (like modal, temporal, many-valued, epistemic, intuitionistic, and other recent systems), leading to some highly applications of logic to software: the specification and verification of programs. For example, logic for action requires from certain previous, contextual and operational knowledge that must be used in order to achieve that action (Mishra, Aloimonos, Fermuller, 2009).

Although several authors worked on the relationships between the logics logics of knowledge and belief (as did Rudolf Carnap, Jerzy Los, Arthur Prior, Nicholas Rescher), it was G. H. von Wright who recognized that our discourse concerning knowledge and belief exhibits systematic features that admit of an axiomatic-deductive treatment[22]. He wrote a seminal work on epistemic logic (1951, *An Essay on Modal Logic*), whose ideas were extended by Jaakko Hintikka in *Knowledge and Belief: An Introduction to the Logic of the Two Notions* (1962). Hintikka's book was a first attempt to combine efficiently knowledge and action (McCarthy & Hayes, 1969). This enabled the existence

[20] Halpern et al., 2001.
[21] van Benthem, 2000.
[22] Hendricks & Symons, 2009.

of a logic based on possible worlds (formalized by Kripke structures). Multi-agent systems can define a system as a non-static entity that changes over time. According to van Benthem & Martínez[23], Hintikka opened the possibility of working formally with a crucial idea: *information*. Information could then be considered as an information flow from states information to agents information (who have beliefs, knowledge or other informational situations). Perhaps it was not the perfect knight for logic, but better married than alone. Independently of Hintikka and belonging to a different research field, economy, Robert Aumann developed in the 1970's similar ideas. Aumann gave a mathematical formulation in a set-theoretical framework to the idea of common knowledge.

Later, in the 1980s and 1990s, epistemic logicians focused on the logical properties of systems containing groups of knowers and later still on the epistemic features of the so-called "multi-modal" contexts[24]. It was in the 1980's that the notion of knowledge from epistemic logic deeply influenced computer science. TARK conferences (and later LOFT conferences) were an example of this new interdisciplinary approach. After considering the notions of inference, observation, introspection and self-correction agent (interactive) powers, logic was closer to the idea of behavioral equilibrium in groups of agents. This enabled a broad range of theories about it: logical dynamics (van Benthem), information dynamics in computer science (Abramsky), interactive epistemology or mathematical learning theory (Kelly)[25]. Since then logical thinking may be considered as an interaction of its dynamical and social properties. As van Benthem (2007: 25) notes: "information is a pervasive aspect of reality, prior to cognitive action", that is, information is more than (formal) knowledge. *Substructural logics* arose (as nonclassical logics) in response to problems in foundations of mathematics and logic, theoretical computer science, mathematical linguistics, and category theory (Dosenn and Schroder-Heister, 1993). Here are some of those logics: (1) lambek calculus (1950's), (2) linear logic (Girar 1980's), (3) fuzzy and multivariate logics, (4) relevance logics (Lukasiewickz), and (5) BCK logic.

[23] Benthem & Martínez, 2007.
[24] Hendricks & Symons, 2009.
[25] See van Ditmarsch, van der Hoek & Kooi, 2008 for an analysis on Dynamic Epistemic Logic.

As a resume of the explained data, we can conclude that:

Author	Development	Formalization of
Gentzen	Natural deduction	Proofs
Church	Lambda-calculus	Programs

As noted by Wadler (2000), lambda calculus, both typed and untyped, inspired several programming languages such as: LISP, 1960: J. McCarthy; Iswim, 1966: Peter Landin; Scheme (Lisp dialect), 1975: Guy Steele (one of the three future programmers of JAVA) and Gerald Sussman; ML ('metalenguage'), 1979: Milner, Gordon, Wadsworth, later 'Standard ML', the base of primordial LCF theorem prover, wich later also inspired HOL and Isabelle theorem provers; Miranda, 1986: David Turner; Haskell, 1987: Hudak, Peyton Jones, Wadler et al, and, finally, O'Caml, 1996: Xavier Leroy.

Keith Clark, Alain Colmerauer, Pat Hayes, Robert Kowalski, Alan Robinson, Philippe Roussel, etc. deserve a lot of credit for promoting the concept of logic programming and helping to build the logic programming community. Currently, most programming is not done in logic programming but in imperative programming language (like Pascal, C, etc.). Other languages, like Prolog, for *Programming in Logic*, were developed from a syntactic-arithmetic perspective, rather than semantic, in order to program not with a set of instructions but directly with formal arithmetic logic as a total solution[26]. According to Ben-Ari[27] Prolog was the first logic programming language and extensive implementation efforts transformed this language into a practical tool for software development. The history of Prolog is, perhaps, the history of the birth of logic programming. All started when Alain Colmerauer was working in 1963 on parsing and syntactic analysis. As Wadler (2000) remarks, at the end of this decade, and working on automatic translation, he tried to make automatic deduction from texts instead of just parsing them. Colmerauer studied the resolution principle and contacted Bob Kowalski, who worked at Edinbourgh. Kowalski and D. Kuehner had recently devised a refinement of resolution (SL-resolution), which permitted, as Wadler (2000) also states, linear deductions with great efficiency and made possible the Edinburg Structure-Sharing Linear Resolution Theorem Prover. The meeting and sharing of ideas between Colmerauer and Kowalski created the

[26] Kowalski, 1982.
[27] Ben-Ari, 181.

language Prolog. Then the Marseille-Edinburg Prolog implementation started, offering an immediately applicable, available and attractive way to do logic programming. Prolog is expressive enough to execute non-procedural programs, and yet also contains enough 'compromises' with the real world to allow the execution of many programs efficiently[28].

In 1970 Robinson (2000) suggested that the new field should be called *computational logic*, and in December 1971 Bernard Meltzer convinced his university to allow him to rename his research department as "Department of Computational Logic". The Horn clause version of resolution, as Robinson (2000: 12) declares, "could be used to represent knowledge declaratively in a form that (…) could then be run procedurally on the computer. Knowledge was both declarative and procedural at once." This led to sequential logical programming, an approach that had also LISP as the crucial reference. By its competence and elegance, PROLOG was used as the first Kernel Language (KL0) of the Japanese Fifth Generation Project (FGP) of Computing Systems (1979-1981). The FGP dominated the mainstream in the 1980's on computational logic (Robinson, 2000). With the increase of complex tasks assigned to a single CPU (for example in multiprogramming operating systems and real-time systems), was developed the concurrent computation and the concurrent logic programming. In that case, the multiple tasks can be executed sharing a single CPU rather than being executed in true parallelism on multiple CPUs. But the problem with parallel and concurrent computation is the difficulty of constructing and verifying algorithms that can benefit from parallelism, and of expressing these algorithms conveniently in a programming language. Nevertheless, logic programs have a natural interpretation as concurrent computations because do not express algorithms procedurally but instead express them declaratively where the formulas has no inherent ordering. Concurrent logic programming languages specify concurrent procedural interpretations for Horn clause programs, just as Prolog specifics a sequential procedural interpretation for the clauses. As Ben-Ari, 1993 remarks, an example of this case of concurrent logic programming language is GHC (Guarded Horn Clauses). A GHC program consists of a set of *guarded clauses*. A guarded clause is like an ordinary clause except that the literals in the body may be preceded by a sequence of literals called *guards*. They are separated from the body literals by a vertical bar rather than by a comma:

[28] Ben Ari, 1993, 8.

A :- G1,…,Gk | B1,…, Bn.

There is another kind of logic, the Constraint logic programming (CLP) which combines the flexibility and ease of declarative programming in logic with the power of search techniques that were developed during research on AI. According to van Roy & Seif, there are almost 30 useful programming paradigms[29]. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem. One programming language can support different paradigms (defined as a 'set of programming concepts'). Perhaps we can classify them under two main programming paradigms, declarative and imperative, as I resume it in this box:

	Declarative		**Imperative**
Programming Paradigms	*Functional*	*Logical*	*Imperative/Procedural*
	Evaluation of mathematical functions avoiding state and mutable data	Use of mathematical logic for computer programming	Statements that change a program state.
Languages	APL, FP, Lisp, Erlang, Haskell, ML, F#, Scheme	PROLOG	FORTRAN, ALGOL, COBOL, Java, C, C++

As it is usual in scientific and technical opposite controversies, both approaches have interesting benefits and annoying side problems. One of the most famous approaches was exemplified by Edsger Dijkstra in this paper "Go To Statement Considered Harmful"[30]. Dijkstra argued against the abusive use of imperative constructs and considered the command 'GO TO' as the archetypical kind of such an abuse IN his own words: "For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of *go to* statements in the programs they produce. More recently I discovered why the use of the *go to* statement has such disastrous effects, and I became convinced that the *go to* statement

[29] Van Roy & Seif, 2004. Van Roy also offers a very useful visual chart of programming paradigms; see here http://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng.pdf

[30] Letter to *Communications of the ACM* (CACM), vol. 11 no. 3, March 1968,147-148.

should be abolished from all "higher level" programming languages (…) The unbridled use of the *go to* statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. (…) The *go to* statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program". Structured programming, according to Dijkstra, could not solve these problematic statements. At the same time, there was an attempt to develop a new managerial approach to production programming[31]. Perhaps imperative languages are not the perfect choice but it is undoubtable that they are dominating several computing fields. As a middle point position, Bob Kowalski proposed the equation ALGORITHM = LOGIC + CONTROL. A new paradigm was in the agenda, with a basic necessary characteristic: to maintain a strict separation of the declarative from the imperative aspects. From this perspective, algorithms consist of a problem description (logic) along with a strategy to carry out computations of these descriptions (control).

First-order logic has also been used at the heart of a lot of database systems, like its syntactic variants 'structured query language' (SQL) or 'query-by-example' (QBE). First-order logic has also been implemented into database using *relational algebra*, and at the same time making possible to scalate correctly the large databases[32].

The last of our covered topics in this section this section is information. On the early 1980's, Jon Barwise and John Perry took seriously the idea of study the concept of 'information' from a logical perspective. They set up 'situation semantics' and founded, in 1983 at Stanford University, the Center for the Study of Language and Information (CSLI). It was a place of great interaction among philosophers, mathematicians and computer scientists. On the other hand, not only the American researchers were involved into this new analysis, European colleagues were also working on natural language semantics (and the informational turn), as Peter van Emde did, who was a pioneer on the study of the parallels between natural and programming languages. By 1986 it was created the Institute for Language, Logic & Information (ILLI, but originally known in Dutch as ITLI, the Instituut voor Taal, Logica en Informatie), later renamed as ILLC (1991, Institute for Logic, Language and Computation). In 1990, the European Association for Logic, Language and Information (FoLLI) created the annual ESSLLI summer schools, with a great

[31] MacKenzie, 2001: 39.
[32] Halpern et al, 2001.

orientation toward mixed research fields. Alexandru Baltag, focusing on information aspects, and a researcher at Oxford University, explains in his website[33] that Logic is currently evolving into a new interdisciplinary field, devoted to studying 'qualitative informational interactions'; i.e. its sources, flows, gatherings, as well as processing, combining and upgrading (or more generally the 'dynamics') of qualitative Information. In sum, logic naturally underlies most of theoretical computer science, but it also has the potential of acting as a unifying force for many academic disciplines around the concept of Qualitative Informational Interaction.

1.4. New Journals and the Institutionalization of the Field

In 1984 the first issue of The Journal of Logic Programming appeared (Robinson, 2000). Several publications on the field appeared across the next years, like: Journal of Logic and Computation (1990)[34], Journal of Functional and Logic Programming (1995)[35], Journal of Multiple-Valued Logic and Soft Computing (1996)[36], or Transactions on Computational Logic (TOCL, 2000)[37]. The increasingly growth of journals in the field denotes a strengthening of this research area and the existence, for the first time, of a critical mass of specialists on the field. A huge literature was also being written. For example, Michael Poe *et al* compiled in 1984 a comprehensive bibliography on Prolog programming with more than 700 entries. In a few short years, the concepts, methods, and topics of computer scientists changed and were shaped to attend the requirements of professionalization and institutionalization.

1.5. Programming robots

Computational logic has also provided solutions to engineering developments on robotics. Flux high-level programming language, for example, enables robots with an internal model of their environment. With an implementation of fluent calculus, this language provides a solution to the

[33] http://www.comlab.ox.ac.uk/people/Alexandru.Baltag/ accessed in March 4rth, 2014.
[34] http://logcom.oxfordjournals.org/, accessed in March 4rth, 2014.
[35] http://scholar.lib.vt.edu/ejournals/JFLP/, accessed in March 4rth, 2014.
[36] http://www.oldcitypublishing.com/MVLSC/MVLSC.html, accessed in March 4rth, 2014.
[37] http://tocl.acm.org/, accessed in March 4rth, 2014.

classic *frame problem*, using the concept of state update axioms This allow us to address several aspects of reasoning about actions like ramifications, qualifications, nondeterministic actions, concurrent actions, continuous change or noise inputs.

2. From Computers to Logic

In Thomason it is explained[38] that the AI authors have made real contributions to philosophical logic, and that they also have draw heavily on past work in logic and philosophy. Computer science is not a passive receiver of logic, on the contrary: it is a direct arrow onto logic development. John McCarthy challenged philosophers to contribute to AI by surveying a number of problems of interest to both fields. This was not only a possible way of improving AI, but it was also an opportunity to obtain a new and significant philosophical advancement. From the perspective of AI, there were, said McCarthy, several philosophical problems, like the importance of common sense, the roles of logic, non-monotonic reasoning, free will, knowledge and belief, reasoning about context, and some more general epistemological concerns.

Van Benthem[39] states that computer sciences are changing academia and that at the same time they have a cultural role, providing a steady stream of new notions and information to the academia itself (brain sciences, logic, philosophy, etc.). For example, Turing machines, by allowing for the computability of mathematical notation, lead to deep results as the undecidability of natural questions like the halting problem. On the other hand, computer sciences provided a great number of ideas that were later decoupled from its initial practical setting, such as: automata theory, complexity theory, semantics of programs, type theory and linear logic, process theory, database theory, AI, and so on.

Once computers were made, they proved to be useful for practical purposes like the mechanization of theorem proving. Considering proofs as programs, Simon and Newell created[40] (and called their program) the Logic Theory Machine, which was also programmed by J.C. Shaw, a scientist at

[38] Thomason, 1989.
[39] Floridi, 2008.
[40] MacKenzie, 2001: 66.

RAND Corporation and Carnegie Technical Institute. To Newell, Simon and Shaw's surprise, Logic Theorist (the other name for Simon and Newell's program) produced a shorter, more elegant proof of theorem 2.85 of Whitehead and Russell's Principia Mathematica – indeed, the axioms and theorems to be proven were taken from the Principia. Nevertheless, The Journal of Symbolic Logic would not publish the description of a proof co-authored by a computer program. Logic Theorist represented a cutting-edge use of computers because of its reliance on heuristics, and because it manipulated not numbers but information, represented in symbolic form.

Soon it was clear that, as MacKenzie remarked (2001: 72), the automation of mathematical proof would require not just propositional logic, but also predicate logic. In this sense, the initial contributions of logicians to automated theorem proving focused on automating 'decision procedures', that is, algorithms.

In summer of 1958, MacKenzie continues, Hao Wang was able to design programs which had proven all the propositional logic theorems from the *Principia*, using less than 30 seconds of CPU. This drove Wang to defend the superiority of the algorithmic approach over the heuristic one. Wang was skeptic about Simon & Newell's heuristic approach, considering their research as an amateur and unprofessional work. Hilary Putnam and Martin Davis spent the summer of 1958 studying algorithms for propositional calculus and developed the so-called Davis-Putnam procedure for checking the satisfiability of formulae in propositional calculus.

In early July 1959, Robert S. Ledley and Lee B. Lusted wrote for *Science* a paper entitled "Reasoning Foundations of Medical Diagnosis"[41]. At the conclusions section they wrote: "The mathematical techniques that we have discussed and the associated use of computers are intended to be an aid to the physician. This method in no way implies that a computer can take over the physician's duties. Quite the reverse; it implies that the physician's task may become more complicated. The physician may have to learn more; in addition to the knowledge he presently needs, he may also have to know the methods and considerations under consideration in this Today you can change 'physician' for 'logician' and the value of their words is still meaningful. Computer science would not exist without logic, but once it was created, it changed the ways of doing logic. This is a reflexive process. And there are

[41] *Science*, vol. 130, num. 3366, 9-21.

several forms of analyzing such symbiotic relationship; one of those ways is to think about the influence of computers upon logic.

In 1962, John Alan Robinson found a new and, as MacKenzie (2001: 77) wrote, "powerful rule of deductive inference which became paradigmatic in automatic proving techniques: resolution". Resolution is the deductive system most often used in mechanizations of logic. And in the 1960's, Dijkstra and Hoare worked on the effects of structural sequential programs running on single computers, which afterwards lead to dynamic logic (developed by Salwick, Pratt, and others). Morevoer, as van Benthem (2007) remarks, "the study or never-ending computation over infinite data streams, which cannot be constructed, only observed", opened a new fundamental theory: co-algebra (Aczel and Mendler, 1989), a field that already changed mathematical proof methods in analysis and set theory.

In the 1980's the analysis of distributed systems and parallel computation on many computers open the way for Milner's process algebra, which became an excellent general theory of communication processes in physics and biology, among others.

Despite engineers' preference for variable-free frameworks over first order logic, such inclination is quite silent: computer sciences still have a pragmatic influence over logics. For example, computer scientists offered logicians new challenges and perspectives for modeling and constructing discrete systems (Thomas, 2000). The interest in the specification of complex systems has allowed the merge of formula based frameworks (like temporal logic, VDM, Z) with diagram based formalisms (like SDL, UML, Statecharts).

2.1. Educational purposes

Among a broad list of computer programs for learning of (formal) logic, several applications like Tarski's World, Hyperproof, Logic Daemon, Jape 3.2., CPT I or MyC have been developed. The first steps towards a visual approach to logic through computer tools was made by J. Barwise and J. Etchemendy. The later is involved nowadays in The Openproof Project[42] at Stanford's Center for the Study of Language and Information. The center is mainly concerned with the application of software to problems in logic, and since the early 1980's they have been developing innovative and

[42] See: http://ggweb.stanford.edu/openproof/ accessed on March, 3rd 2014.

commercially successful applications in logic education. *Hyperproof*, one of the leading programs in this new era of teaching logic is a *heterogeneous* logic program; the term *heterogeneous* arises from the formal integration of the diagrammatic and sentential representations. Barwise and Etchemendy designed Hyperproof to support reasoning in which information from differing modalities (sentential and graphical) is combined or transferred from one modality to another. Oberlander, Stenning and Cox[43] argued that the virtue of Hyperproof, lays not in visualization, but in its multimodality. I disagree completely: the choice of *seeing* in completely 3D visual environment allows for a different kind of cognition, better adapted to human cognitive capacities[44]. Hyperproof has a better cognitive fitness and introduces students into the new research paradigm of e-Science (Vallverdú, 2009). Visual thinking has been rehabilitated in epistemology of mathematics and logic due to computer sciences (Mancosu, Jørgensen & Andur, 2005).

The educational logic platforms have reached even the cellular phones commercial domain. The program Logic 101 is designed for constructing derivations in introductory logic courses. It checks several parameters, such as the syntax of symbolic sentences or whether each line can be derived from previous lines, and also indicates derivable lines by colour, and displays derivation rules for each derivable line. The completed derivations can even be sent by e-mail. Logic 101 runs under both iPhone and iPod.There are also two other logic apps there: Syllogism and Logic 100.

We can also find plenty of Java applets (Tilomino 2, xLogicCircuits, Truth Table Constructor, DC Proof, and so on) freely hosted and distributed among important education institutions all around the world. These technological machines and artifacts make possible not only to study logics with contemporary tools but also to think differently about logics. That the Internet has changed deeply the ways by which we make science and establish human relationships is something very clearly known. Computational logic has made significant contributions to this, especially to the very-large-scale integration (VLSI). VLSI is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip[45]. Testing

[43] J. Oberlander, K. Stenning and R. Cox (1996), The Multimodal Moral, online document. See: ftp://ftp.cogsci.ed.ac.uk/pub/graphics/itallc96.ps, accessed on April 14th 2014.
[44] Vallverdú, 2010.
[45] See: http://en.wikipedia.org/wiki/Very-large-scale_integration, accessed on April 14th 2014.

and verifying modern hardware and software are really complex tasks and computational logic has contributed to make this process smoother. At the same time, the Internet has modified the social ways of communicating and debating computational logic. Websites, middleware, multimedia tools have provided a new thinking space for the logic community.

2.2. From theory of computing to logics

One of the seminal influences between computer science and logic was the Curry-Howard correspondence, which is the simple observation that two at-the-time-seemingly-unrelated families of formalisms, the proof systems on one side and the models of computation on the other were in fact structurally the same kind of objects[46]. If it is true that the works of Curry[47] and Howard[48].belonged to the field of logic, its consequences were analyzed by experts belonging to the computer science domain. N. G. de Bruijn (used the lambda notation for representing proofs of the theorem checker *Automath*, and represented propositions as "categories" of their proofs), who was very likely unaware of Howard's work stated the correspondence independently, but also Stephen Kleene, Joachim Lambek, among others, made significant contributions to this debate.

Another field in which an initial philosophical development has led to a computer science innovation is that of *game semantics*. This is an approach to formal semantics that grounds the concepts of truth or validity on game-theoretic concepts. And as it is well known, Paul Lorenzen was the first to introduce, in the late 1950's, a game semantics for logic. By mid-1990's, game semantics had changed several related fields, such as theoretical computer sciences, computational linguistics, AI or the way we understand programming languages, especially through the efforts from van Benthem who took a deep interest on the relation between logic and games. For van Benthem[49], game logics describe general games through powers of players for forcing outcomes. Specifically, they encode an algebra of sequential game operations such as choice, dual and composition. In this sense, logic games

[46] See: http://en.wikipedia.org/wiki/Curry-Howard_correspondence, accessed on April 14th 2014
[47] Curry, 1934, 1958.
[48] Howard, 1969.
[49] van Benthem, 2003.

are special games for specific purposes such as proof or semantic evaluation for first-order or modal languages. This author has made some crucial to computer logic as well as some critical thinking about the future of logic, arguing that there is something like a triangle of theory, reality, and new design, and also an interaction between all these different perspectives. Phenomena like reasoning or information flow suggest a natural Triangle of perspectives: (somewhat normative) *theory*, *empirical reality*, but also *virtual reality*, the construction of *new systems and new forms of behavior* by the interplay of the former two. Accordingly, theoretical logic, empirical psychology, and constructionist computer science form a natural Triangle of disciplines, each approaching the topics on his agenda with a different thrust. There is thus a fusion of fields more than a movement of influences from one field to another [50].

The P = NP problem, a relationship between the complexity classes P and NP, as Ivancevic & Ivancevic (2007: 124) described it, "is an unsolved question in theoretical computer science"; it is considered "to be the most important problem in the field". Logic has played a crucial role in the development of the theory of NP-completeness by formalizing the concept of combinatorial explosion. At the same time, this intensive research led to a new set of relativizing proofs, natural proofs and algebrizing proofs[51].

2.3. Automation of sound reasoning

Beyond educational uses of computers into logical arena, we can find fields in which the introduction of computers has changed the way of doing and understanding logic. One of them is *proof automation*. Despite the fact that most of computer programs related to proofs have been designed for proof verification (Coq, HOL, etc.), some of them are automated reasoning programs. OTTER, as Raynor (1999:217) wrote "is a fourth-generation Argonne National Laboratory deduction system whose ancestors (dating from the early 1960s) include the TP series, NIUTP, AURA, and ITP". Otter must have a heuristic or strategy in order to avoid the many millions of possible deducible conclusions, but is doesn't imply that it must be 'person-oriented

[50] van Benthem, 2006.
[51] Garey & Johnson, 1979.

reasoning'[52]. This automatic canonicalization and simplification (by demodulation or subsumption) can led us to a neat proof that would not be so obvious to a human investigator[53]. Does it imply that this automation creates black boxes or unintuitive proofs? No, as Robinson said (2000:15), the Argonne group undertook "a large-scale pursuit of computing real mathematical proofs". Machines can do mathematics as well as humans. The Argonne program found a proof of the Robbins conjecture. Indeed, as Robinson continues, in approximately 20 hours of search on a SPARC 2 workstation, the program EQP (designed by McCune), found a proof in which it could be demonstrated that every Robbins algebra was a Boolean algebra.

2.4. Verification systems

Computer sciences also need logic. The work on verification and validation (V&V) of huge and complex systems requires logical tools in order to make things easier and more effective. As soon as these tools appeared, its use on proof verification allowed for a complete view on automatic reasoning and automated V&VThis led to the actual IEEE Std 610.12-1990 V&V standard. Proofs are now faster, more reliable and cheaper. But this came with a price as one can see by a short list of the historical V&V mistakes: Ariane 5 rocket, London Ambulance Dispatching System, Therac-25 or the Nike disaster. Debugging was considered a necessary task in the beginnings of systematic computer programming. Computers models and their results should be reliable. This process could be done with a statistical approach, although, as Dijkstra said in 1969: "program testing can be used to show the presence of bugs, but never to show their absence!"[54]. On that same year, Tony Hoare wrote the article "An Axiomatic Basis for Computer Programming", which led to the consolidation of the formal verification process. At that point, two thirds of the cost of software projects were dedicated to locating and remove programming bugs.

[52] Wos & Fitelson, 2005.
[53] Dana Scott, quoted by Wos & Fitelson, 2005, 717.
[54] McKenzie 2001, 45.

2.5. From logic and computer sciences to other fields

Finally, we have the social software. It is a new (since 1996) and, as Pacuit (2005: 10) remarks, interdisciplinary research program that borrows mathematical tools and techniques from game theory and computer science in order to analyze and design social procedures. Rohit Jivanlal Parikh (Pacuit's PhD advisor), who was influential in the development of the field, is leading these researches. The goals of research in this field are modelling social situations, developing theories of correctness, and designing social procedures[55].

3. Other related issues: Patents

I would like to finish my analysis with a brief overview about one key problem for any researcher dedicated to computer sciences: patents. Klemens (2006) has made an extended study of the main problems of patenting mathematics or programs (for him, equivalent domains). I am not against (computer logic) patents, but we should consider its limits. Can we patent numbers or mathematical operations? If not (and obviously we cannot), why do we allow patented algorithms?

Let me consider though an example shown to me by Thomas Hales[56], the Ståalmarck's method[57]. Ståalmarck's method is a proof search algorithm for finding proofs of propositional tautologies in a proof system called *the dilemma proof system*. The search procedure is based on a notion of *proof depth* corresponding to the degree of nestings of assumptions in proofs. Ståalmarck's algorithm has been successfully used in industrial applications since 1989, for example in verification of railway interlockings and of aircraft control systems. It has a Swedish Patent (467 076), where Ståalmarck's

[55] For further readings on social software, see Parikh, 2002. See also: http://en.wikipedia.org/wiki/Social_software_(social_procedure) , accessed on april 14th 2014.
[56] Private electronic correspondence January, 28th 2009.
[57] G. Ståalmarck, "A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula". Swedish Patent No. 467 076 (approved 1992) U.S. Patent No 5 276 897 (approved 1994), European Patent No 0403 454 (approved 1995).

algorithm is a patented technique for tautology-checking which has been used successfully for industrial-scale problems, implemented as a HOL derived rule.

Conclusions

From several evidences we must conclude that the relation between logic and computer sciences is mutual and symbiotic. Both fields have influenced each other and as a result of this symbiotic process, their ideas and performances were improved. Moreover, such mixture of techniques and ideas has produced a spontaneous and continuous interchange between both communities of researchers. In sum, this opened up new thinking spaces through which boundaries between are blurred under a deep layer of common interests, tools and ideas.

Acknowledgements

Financial support for this research was received from the Spanish Government's DGICYT research project: FFI2011-23238, "Innovation in scientific practice: cognitive approaches and their philosophical consequences". I thank to Amnon Eden, Thomas Hales and Johan van Benthem for their support, suggestions, answers or commentaries to my queries in some step of my long and slow research. I thank to Peter Skuce for his patience and linguistic support. Finally, I really appreciate the deep, hard and fine task of two patient anonymous reviewers, who suggested very good ideas to improve the quality and coherence of this paper. After this process, I need to clarify that all errors, mistakes, etc. in the article are exclusively my own. This is the true way of making science.

References

Abramsky, S., Gabbay, D.M., and Maibaum T.S.E., 1993, *Handbook of Logic in Computer Science*, Vol 2, London, Oxford University Press.

Aczel, P. and Mendler N.P., 1989, A Final Coalgebra Theorem. *Category Theory and Computer Science,* 357-365.

Barendregt, H., 1997, The Impact of the Lambda Calculus in Logic and Computer Science. *The Bulletin of Symbolic Logic*, 3 (2), 181-215.

Ben-Ari, M., 1993, *Mathematical Logic for Computer Science*, UK, Prentice Hall.

Bringsjord, S. and Ferrucci, D.A., 1998), Logic and Artificial Intelligence: Divorced, Still Married, Separated ...?, *Minds and Machines*, 8 (2), 273-30.

Brooks, R., 1990, Elephants Don't Play Chess. *Robotics and Autonomous Systems*, 6 (1-2): 3-15.

Curry, H., 1934, Functionality in Combinatory Logic. In: *Proceedings of the National Academy of Sciences*, 20, 584-590.

Curry, H. B. and Feys, R., 1958, *Combinatory Logic Vol. I*, Amsterdam, North-Holland.

Davis, M., 1988, Influences of mathematical logic on computer science. In: *The Universal Turing Machine: A Half-Century Survey*, USA: OUP.
-- 2000, *The Universal Computer: The Road from Leibniz to Turing*, USA: W. W. Norton and Company.

Dawson, J.W., 2006, Gödel and the Origins of Computer Science. In: *CiE 2006*, LNCS 3988, 133-136.

Ditmarsch van, H., van der Hoek & W, Kooi, 2008, *Dynamic Epistemic Logic*, London, Springer.

Dosen, K. and Schroder-Heister. P., 1993, *Substructural Logics*, USA, Clarendon Press.

Floridi, L., 2008, *Philosophy of Computing and Information: 5 Questions,* UK: Automatic Press/VIP.

Flach, P.A., 2005, Modern Logic and Its Role in the Study of Knowledge. In: *A Companion to Philosophical Logic*, London, Blackwell, 680-693.

Galton, A., 1992, Logic as a Formal Method, *The Computer Journal,* 35 (5), 431-440.

Garey, M.R. and Johnson, D.S., 1979, *Computers and Intractability: A Guide to the NP-Completeness,* New York: W.H. Freeman.

Gentzen, G., 1934, Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift,* 39, 176-210, 405-31.

Halpern, J.Y. et al., 2001, On the Inusual Effectiveness of Logic in Computer Science. *The Bulletin of Synthetic Logic*, 7 (2), 213-236.

Harrison, J., 2009, *Handbook of Practical Logic and Automated Reasoning,* UK, CUP.

Hendricks, V. and Symons, J., 2009, Epistemic Logic. In: *The Stanford Encyclopedia of Philosophy (Spring 2009 Edition)*, Edward N. Zalta (ed.). Accessed on March 3rd, 2014: http://plato.stanford.edu/archives/spr2009/entries/logic-epistemic/

Howard, W., 1969, The formulae-as-types notion of construction. In: Seldin, J. et al., *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 479-490.

Ivancevic, V.G. and Ivancevic, T.T, 2007, *Computational Mind: A Complex Dynamics Perspective*, Berlin, Springer.

Klemens, B., 2006, *MA+H You Can't Use. Patents, Copyright and Software*, Washington, D.C., Brookings Institution Press.

Kowalski, R., 1982, Logic as a computer language. In: *Logic programming,* London, Academic Press, 3-18.

Mackenzie, D. 2004, *Mechanizing Proof: Computing, Risk, and Trust*, Cambridge (MA), The MIT Press.

Mancosu, P., Jørgensen, K.F. and Andur, S., 2005, *Visualization, Explanation and Reasoning Styles in Mathematics,* USA, Springer.

McCarthy, J. and Hayes, P.J., 1969, Some Philosophical Problems from the standpoint of the Artificial Intelligence. Accessed on March 4th, 2014: http://www-formal.stanford.edu/jmc

Mishra, A., Yiannis Aloimonos, Y, & Fermuller, C. (2009), Active Segmentation for Robotics, *IROS 2009.* Downloadable at: http://www.umiacs.umd.edu/~mishraka/downloads/iros2009_activeSeg.pdf, Accessed on April 12th 2014.

Pacuit, E., 2005, *Topics in social software. Information in strategic situations*. PhD thesis, New York.

Parikh, R., 2002, Social Software. *Synthese*, 132, 187-211.

Poe, M.D. , Nasr, R., Potter, J. and Slinn, J., 1984, A KWIC (Key Word in Context). Bibliography on Prolog and Logic Programming. *Journal of Logic Programming*, 1 (1), 81-142.

Post, E., 1936, Finite combinatory proceses. Formulation I. *Journal of Symbolic Logic* 1, 103-105.

Raynor, J., 1999, *The International Dictionary of Artificial Intelligence*, Chicago, Glenlake Publishing Company, Ltd.

Robinson, J.A., 2000, Computational Logic: Memories of the Past and Challenges for the Future. In: *CL 2000, LNAI 1861*, Berlin, Springer Verlag, 1-24.

Stewart, D, 1994, Herbert A. Simon: Thinking Machines, *OMNI Magazine,* Interview June 1994. Accessed on March 4th, 2014: http://www.astralgia.com/webportfolio/omnimoment/archives/interviews/simon.html

Thomas, W., 2000, Logic for Computer Science: The Engineering Challenge. Lecture given at the Dagstuhl Anniversary Conference, Saarbrücken.

Thomason, R., 1988, Philosophical logic and artificial intelligence. *Journal of Philosophical Logic*, 17 (4), 321-327.
-- 1989, *Philosophical Logic and Artificial Intelligence*, Dordrecht, Kluwer Academic Publishers.

Turner, R., 1985, *Logics for AI*, Chichester, Ellis Horwood.

Vallverdú, J., 2009, Computational Epistemology and e-Science. A New Way of Thinking. *Minds and Machines*, 19 (4), 557-567.
-- 2010, Seeing for Knowing. The Thomas Effect and Computational Science. In: *Thinking Machines and the Philosophy of Computer Science: Concepts and Principles,* USA, IGI Global Group, 140-160.

van Benthem, J. 2000, Reasoning in reverse. In: *Abduction and Induction: Essays on their Relation and Integration*, Dordrecht, Kluwer Academic, ix-xi.
-- 2003, Logic Games are Complete for Game Logics. *Studia Logica,* 75, 183-203.
-- 2006, Where is Logic Going, and Should It?.*Topoi,* 25 (1-2),117-122.

van Benthem, J., et al, 2006, *The Age of Alternative Logics. Assessing Philosophy of Logic and Mathematics Today,* Series: Logic, Epistemology, and the Unity of Science , Dordrecht, Springer.

van Benthem, J., 2007, An Interview on the Philosophy of Information, accessed on March 4th 2014 http: // www.illc.uva.nl/ Research / Publications / Reports/PP-2008-01.text.pdf

van Benthem, J. and Martínez, M., 2007, The stories of logic and information. In: *Philosophy of information*, Amsterdam, North Holland.

van Roy, P and Seif, H., 2004, *Concepts, Techniques and Models of Computer Programming*, Cambridge (MA), MIT Press.

Wadler, P., 2000, Proofs are Programs: 19th Century Logic and 21st Century Computing, *Dr. Dobb's Journal* (published as "Old ideas form the basis of advancements in functional programming"), downloadable from: http://homepages.inf.ed.ac.uk/wadler/papers/frege/frege.pdf, accessed on April 12th 2014. See also: http://homepages.inf.ed.ac.uk/wadler/topics/history.html, ibid.

Wos L. and Fitelson, B., 2005, The Automation of Sound Reasoning and Successful Proof Finding. In: *Companion to Philosophical Logic*, USA, Blackwell, 709-723.