Logic for Computer Science: The Engineering Challenge *

Wolfgang Thomas

RWTH Aachen, Lehrstuhl für Informatik VII,
52056 Aachen, Germany
thomas@informatik.rwth-aachen.de

Abstract. This essay is a reflection on the roles which logic played and can play in computer science. We recall the obvious merits of mathematical logic as a parent discipline of computer science, from which many fields in theoretical computer science emerged, but then address some unresolved issues in connection with the engineering tasks of computer science. We argue that logic has good perspectives here, following a tradition which is closer to Leibniz than to Hilbert and Gödel.

1 Introduction

Logic is a cornerstone of scientific methodology and thus belongs to the foundation of every scientific discipline. For computer science, logic plays a still more central role:

– Logic is a parent discipline of computer science; historically computer science emerged from problems and methods which were developed in mathematical logic.
– Logic is a basic constituent of the computer science curriculum; in fact, there is agreement that it is required in a stricter sense in computer science education than, for example, in mathematics.
– Logic has produced a large reservoir of methods and theories for computer science (which are often typical for this application area and no more to be counted to mathematical logic itself).

The present paper starts with an elaboration on these aspects.

But beyond these merits and contributions of "logic in computer science", there is also a deeper (and I think more problematic) level of the relation between logic and its computer science context. In the never-ending discussion of the role of "theory" in computer science, and why (for instance) logic should continue to be taught in the way it is, I find implicit criticisms and challenges which are

* Transcription of a lecture given at the Dagstuhl Anniversary Conference, Saarbrücken, August 2000, to appear in the proceedings volume Springer LNCS 2000 (Ed. R. Wilhelm)

rarely made explicit. Much of this discussion is due to the different viewpoints which scientists and engineers have. For classical mathematics (especially, for analysis), it is generally accepted that engineers have a legitimate special view and use of the subject. For the concepts and techniques of logic, which today are used (mostly in an implicit way, unfortunately) in the daily work of hundreds of thousands of software engineers, an engineering view has not yet emerged as natural and legitimate. Today, engineers usually have a rather distorted view of logic; many use the term "logic" just to mean a circuit, i.e. a realization of a Boolean formula. But logic has the potential to offer much more, namely to supply another basic "calculus" with a core of techniques which should be known and applied by every professional systems engineer. This would involve a certain move in the orientation of logic, from "logic *in* computer science" to what I would call "logic *for* computer science". In the second half of this paper I will try to explain these challenges in more detail, in which way they deviate from the focus of classical mathematical logic (and even of classical theoretical computer science), and why I find them to be promising tracks on which logic can contribute to progress in computer science.

2 Mathematical logic as an origin

There is not a single event which can be called the birth of computer science; indeed, this new discipline evolved by a complicated interaction between engineers, mathematicians, and also logicians. But there are eminent single contributions which surely were essential in forming this new scientific field, several of them from mathematical logic. Among them, Alan Turing's paper of 1936 *On computable numbers, with an application to the Entscheidungsproblem* is a prominent example. In this paper, one finds a proposal to capture in precise terms the most fundamental notion of computer science ("algorithm"), one finds the idea of a universal machine (anticipating the concept of programmable processor), and also first unsolvability results, showing principal limitations of the algorithmic method.

Turing's paper was a contribution to mathematical logic; it showed that the most famous problem of the subject at the time, "Hilbert's Entscheidungsproblem", is algorithmically unsolvable. Let us briefly recall these logical origins of computer science.

Mathematical logic is a relatively new branch of logic which took shape in the second half of the nineteenth century. At first, the aim was to join logic with the ideas of arithmetic and algebra, in order to make logic accessible to the powerful algebraic techniques of formula manipulation. In the works of Boole and Schröder, interesting parts of deductive reasoning were cast in algebraic formalisms (in "Boolean algebra" and Schröders "Algebra der Logik").

But these formalisms covered only small fragments of mathematical language and inference methods. In his pioneering monograph *Begriffsschrift*, Gottlob Frege overcame these deficits. He proposed a universal formal language (in particular, involving quantifiers), in which one could express all ordinary mathematics, and he developed a syntactic proof calculus which was strong enough to imitate mathematical proofs.

This success suffered from a drawback and some irritation which originated in Cantorian set theory by the set theoretic paradoxes, for example, by the paradox of the set of sets which are not an element of itself (discovered independently by Zermelo and Russell). Cantor himself had been aware of the subtleties which had to be observed when dealing with infinite sets (and he had spoken of consistent and inconsistent sets). But for a formal reconstruction of the foundations of mathematics, as designed by Frege, the Zermelo-Russell paradox came as a surprise and a shock. Hilbert, who felt like Cantor, was concerned about the perspective that mathematics might be put into doubt. He proposed what is called "Hilbert's Program": to get rid of the worries about the foundations of mathematics in two steps:

- by simulating ordinary mathematics in a sufficiently strong formal system (with a syntactic proof calculus),
- by showing with elementary means ("finitist methods", which were not subject to doubt) that in this formal system a contradiction like $0 = 1$ could not be derived.

Part of the second item was "Hilbert's Entscheidungsproblem": it asked for a procedure by which it could be decided whether a given formula (like $0 = 1$) is or is not derivable in the proof calculus.

In pursuing this program, mathematical logicians clarified a concept which proved to be central in the subsequent formation of computer science, namely the concept of a formal system, with a clear separation of syntax and semantics, with the notion of a formal proof calculus (defining "computational steps"), and its properties of soundness, completeness, and consistency. The master example of such a formal system was first-order logic (or predicate logic). Later, in computer science, formal systems were created in hundreds of different versions, for example in the definition of specification languages, process calculi, and programming languages. But in the original context of first-order logic, the breakthrough results of mathematical logic were established:

- Gödel's completeness theorem, showing that a first-order formula is valid (true in every model) iff it can be derived in the proof calculus,
- Gödel's incompleteness theorem, which states that the sentences which are true in the fixed model of arithmetic cannot be generated completely by an axiom system (like the axioms of first-order Peano arithmetic),
- Church's and Turing's clarification of the notion of algorithm and the proof that Hilbert's Entscheidungsproblem is undecidable for first-order logic.

The last two results meant that the second part of Hilbert's Program could not be carried out in the form as originally envisaged. On the other hand, the admirable and tedious work of Frege, Russell, Whitehead, and many others had produced the astonishing fact that the first part of Hilbert's program was indeed realizable, first in systems of higher-order logic, and finally, with the development of set theory, even in first-order logic (based on the first-order axiom system ZFC, "Zermelo-Fraenkel set theory with the axiom of choice").

The idea of coding a significant part of science in such a formal manner was not new: Two hundred years earlier, Gottfried Wilhelm Leibniz had formulated the far-reaching vision of a *characteristica universalis*, a universal language in which knowledge could be expressed and manipulated in a computational fashion:

> *It should be possible to set up a kind of alphabet of human thoughts, and to invent and to decide everything by a combination of its letters and by the analysis of the words composed from them.*

Leibniz had overoptimistic views about the realizability of his project (maybe typical for scientists who have to raise funds):

> *It would cost no more work than what is already now invested in many treatises and encyclopedias. I think that some selected persons can do the job within five years, but that after two years they are already able to master by an unfallible calculus the disciplines which are required most for life, i.e., moral and metaphysics.* [1]

At first sight, Leibniz's vision looks much too ambitious to be feasible, even when restricted to the domain of mathematics; indeed, I do not know of any mathematician or philosopher who agreed to Leibniz in that his program might be worth trying. Leibniz himself could provide only very small technical steps towards his goal (among them the sketch of a fragment of Boolean algebra). Nevertheless, only two centuries later the program was realized for the domain of mathematics.

However, an important difference has to be noted: The aim of mathematical logic was to clarify a very general methodological question, that of consistency of mathematical assumptions and reasoning; so it was sufficient to code mathematics *in principle*, without any claims on a practical use of the formalization. On the other hand, Leibniz took the approach of a knowledge engineer who wanted to set up a *practical calculus of information processing*. Only in the continuation of logic within computer science, this practical aspect began to play a role again, when logic programming and automated theorem proving were developed. These two views of logic, that of a foundational discipline as perceived by Hilbert and Gödel and that of a framework for practical computation as suggested by Leibniz, point precisely to the question which profile logic should have today in the context of computer science.

The great success of mathematical logic was first seen in the fact that a number of new mathematical subjects came into existence, among them recursion theory, model theory, set theory, and proof theory. The *Handbook of Mathematical Logic* [2] gives a first impression of their beauty and strength. These new mathematical subjects were created in the very short time of only two or three generations, and they helped to establish new connections between

[1] Quotations from [5] (my translation from Latin)

logic and other mathematical subjects (for example, algebra). On the elementary level, a core theory emerged which is now part of the undergraduate curriculum: Propositional logic, syntax and semantics of first-order logic, a proof calculus, its soundness and completeness, basic undecidability and incompleteness results, and expressiveness results (like the compactness theorem or separation results on the expressive power of logics).

3 Logic in computer science

Apart from the new subjects created within mathematical logic, many areas in theoretical computer science developed as offsprings of logic. For example, the above-mentioned logic subjects of recursion theory, model theory, and proof theory all gave rise to new disciplines in theoretical computer science with a new specific orientation: From recursion theory, the area of complexity theory emerged, addressing the quantitative refinement of computability, with many new concepts and methods. Similarly, model theory took a specific shift in response to "the challenge of computer science" (see Gurevich's paper [3]), by focussing on finite models and establishing the new field of descriptive complexity theory. Finally, the subject of proof theory had many continuations in computer science, notably type theory, which itself plays a central role e.g. in programming language semantics.

Today it seems impossible to give a complete list all fields in computer science which are rooted in logic. Here is an excerpt (and the reader may consult the *Handbook of Logic in Computer Science* [1] to get a more detailed picture):

- programming language semantics,
- type theory, linear logic, categorical theories,
- λ-calculus, π-calculus,
- specification logics, e.g., dynamic logic, Hoare logic, temporal logic, systems like VDM, Z,
- finite model theory, data base theory,
- term rewriting, unification, logic programming, functional programming,
- automated theorem proving,
- program verification,
- process calculi and concurrency theory,
- modal logic, logics of knowledge

This list is to be complemented by families of concrete software systems which were designed as direct outgrowths of theories of logic. Among these "practical successes" of logic, there are the following:

- Systems for circuit design,
- Relational data base systems,
- Expert systems,
- Model checkers and theorem provers.

Despite this rich landscape of theoretical subjects and concrete systems, the status of logic in computer science is under dispute (e.g., regarding its role in the curriculum), and logic faces criticism of practitioners as being too formal and too remote from the world of software (or systems) development practice. When leafing through the proceedings of logic conferences in computer science, one gets the feeling that this nice and deep research is not terribly influential in mainstream computer science. A standard reply to this is that the practice of computer science is not yet scientific and that some time in the future the relevance of the precise methods will be appreciated. I think that this kind of reply makes things too easy and avoids facing some challenges which in fact can prove very fruitful for logic.

4 Some challenges in the context of engineering

A characteristic feature of mathematical logic is its concentration on formal systems as a whole. Usually, a logical framework is a formal system, and the statements and claims made are concerned with global properties, like consistency or completeness, expressiveness in comparison with other formal systems, or questions of decidability and complexity of algorithmic problems about these systems. Often, this involves the reduction of the phenomena under consideration to the "atomic level", on which the technical work is then performed. This applies not only to classical mathematical logic but also to most of the above-mentioned logic-oriented areas in theoretical computer science.

Some well-known examples might illustrate this. In Turing's analysis of the notion of algorithm, one finds a reduction of the conceivable computational processes to the most elementary units, the Turing machine moves, and these units are argued to be "complete" for discrete computation. Similarly, in the conception of a first-order proof calculus, some very elementary proof steps are isolated and formulated as proof rules, and the calculus as a whole is shown to be sound and complete. Similar statements can be made about other calculi, like the λ-calculus or the π-calculus, and many more formalisms (see, for example, the concluding section of Milner's Turing Award Lecture [4]). The maturity and experience which logic has gained in setting up, analyzing, and comparing formal systems allows today such studies of high subtlety and scholastic refinement; it is fun to play on this stage. (In other branches of theoretical computer science, like the theory of formal languages, the same tendency is to be seen, only different types of formal systems are studied.)

The study of formal systems and their global properties corresponds to the situation in the natural sciences where one also tries to reduce existing phenomena to elementary units (facts and laws) such that the observed phenomena can be explained from them. This scientific analysis is useful and essential also in computer science for a deeper understanding of the "natural laws" of information processing, but it is somewhat opposite to the interests of an engineer. He is less concerned with the extreme reduction of processes or objects, but more with the synthesis of systems from "usable" components which very rarely are "atomic", and he needs a clear terminological framework which supports this synthesis. This explains why the computer science professional usually handles units which are of a quite different nature than the structures which he sees in his undergraduate courses, say in logic or theoretical computer science. The (software or systems) engineer would appreciate from logic *concepts and techniques as thinking tools* [2]*, which are clean, adequate, and convenient, to support him (or her) in describing, reasoning about, and constructing complex software and hardware systems.*

This is different from the conception of uniform general theories; it emphasizes construction rather than reduction. In the present landscape of logic, such constructive and useful tools exists. Let us mention some of them:

– Propositional logic and ordered binary decision diagrams,
– temporal logic and model-checking,
– Horn clause logic and logic programming,
– the relational data model.

But these concepts and techniques are just mosaic pieces of a more comprehensive "discrete system theory" which an engineer could use. Much has to be done to complete this mosaic. To give some more detailed perspective, I list five general challenges, the first four being more of methodological nature, the last giving a kind of research direction.

4.1 Pragmatics is important

The clients of classical mathematical logic were mathematicians with an interest in the foundations of mathematics. This is a small, excellently qualified audience. In computer science, logic is (or should be) applied by hundreds of thousands of average software engineers. It is obvious that the two communities need rather different presentations of logic. Moreover, the impact of the software engineers' logic education is (via the quality of their software products) by far greater than the impact which logic has ever reached in foundational studies. Logic should respond to this challenge, and it would gain a much higher significance by a tighter connection to engineering. The pragmatics of logic formalisms, i.e., their suitability for everyday use, is here more important than classical criteria like completeness.

Let me illustrate this with a very small example. Propositional temporal logic of linear time is known to be expressively equivalent to the first-order language over labelled orderings of order type ω. For a logician or a mathematician it is trivial to use first-order formulas rather than temporal formulas. But in practice, it makes a difference whether one has to write down explicitly the variables for time points (as is necessary in first-order logic) or whether one may use temporal operators which spare this. Experience shows that engineers prefer very much the variable-free framework over first-order logic. Such aspects are irrelevant in classical logic but have to be addressed if a widespread use of a formalism is important.

[2] This term is due to C. Jones; see his contribution to this volume.

4.2 Building a new model theory

In classical model theory, one considers first-order structures and relations and operations like extension, elementary extension, the formation of products, etc. Usually, one considers one model at a time. An average software engineer, modelling some application say in the object-oriented UML-framework ("Unified Modelling Language"), may handle hundreds of structures at the same time, of different sorts, and with much more complicated interactions like instantiation, multiple references, inheritance, etc. Neither is there (up to now) a well-defined semantics for the full range of the UML language, nor is there a clear and unambiguous terminology which would guarantee a consistent use of the object-oriented framework. To supply a clean and clear way of handling this chaotic world of models is both a very practical and theoretically demanding task. A student who compares the models of his logic course to the complexities of the models which he has to treat in his software engineering project work may come to the conclusion that theory is not very useful for him.

4.3 Merging the languages of formulas and diagrams

There are two basic approaches to the specification of systems and their behaviour: Formula based frameworks (like temporal logic, VDM, Z) and diagram based formalisms (like SDL, UML, Statecharts). Both have their typical advantages. By their conception, formula based frameworks are "compositional"; their formulas or terms are constructed inductively, and the definition of the semantics usually follows this inductive structure. On the other hand, diagrams and graph-like objects are usually more flexible in use, and also algorithmic problems like satisfiability or simplification ("minimization") are often solved more easily here than over formulas. Classical results giving a precise connection between the two approaches are, for instance, the equivalence between Boolean formulas and ordered binary decision diagrams, and the equivalence between regular expressions (or monadic second-order logic over words) and finite automata. The large-scale use of specifications by diagrams seems to be typical for computer science (and probably is another aspect of pragmatics). Theories which support merging diagram-based languages with term- or formula-based ones would help in designing better specification languages.

4.4 Taking hierarchy seriously

The description of large (software or hardware) systems is only possible by referring to their hierarchical structure, often reflecting different levels of abstraction. A "specification" is often more a kind of book than a kind of formula. The basic models of logic (and of theoretical computer science), like first-order structures or automata, are flat, and their measure of complexity is often simply their size (number of elements or states). This is highly inadequate in the study of non-trivial systems; "hierarchy level" should be a first-class parameter. There are promising theoretical models supporting hierarchical descriptions and constructions, like communicating or hierarchical state-machines, statecharts, or Gurevich's abstract state machines. But the theory of their behaviour is not yet well developed, and more work has to be invested to make it accessible to engineers.

The systems of computer science cover such a wide range of levels of hierarchy today that it even seems doubtful to try to cover them by just one methodology. In natural science, it is agreed that different levels of organization require different concepts and laws, as seen in the division of science into fields like physics, chemistry, and biology. The hierarchical world of information processing systems has reached a richness where the same question arises. An example may illustrate this aspect: It is clear that in the memory cells of a processor a single bit matters. But on the level of the world-wide web this is no more true; there, it usually does not even matter whether a whole server is down. So, in teaching "foundations of computer science", it is probably no more appropriate to map everything (in principle) to the flat world of finite automata or Turing machines. This is like trying to explain chemical or biological phenomena just with the concepts and laws of physics.

4.5 The challenge of the web

In the past ten years, the development of the world-wide web has caused a revolution in the world of information processing. The framework for the publication and exchange of scientific results is changing deeply and rapidly. Today, a large part of scientific knowledge is avalaible not only in symbolic form (i.e., in texts), but also in a format which supports machine-based search, analysis, and composition. This gives a completeley new perspective to Leibniz's project of a universal framework for the management of knowledge. It is rather clear that new kinds of "inference" and "composition of propositions" have to be developed to handle the potentials of the web adequately. Leibniz would probably be enthusiastic about this wonderful new arena for logic. But in academic logic, these practical Leibnizian tasks do not attract much interest. Instead, computer scientists, in particular data base researchers, are addressing these questions. [3] Sometimes I have the impression that we are living in a golden age of logic but that logic does not know it.

5 Conclusion

In the sections above, I argued that computer science gives to logic new challenges and perspectives, in particular, to develop clean, adequate, and convenient methods for modelling and constructing discrete systems (software and hardware systems). For achieving this, logic would no more stay just a foundational science, but also function as an engineering-oriented (however theoretical!) discipline. It should give to system engineers mathematical tools which they require in any constructions which are done "according to professional standards".

[3] see the paper of G. Weikum in this volume

For other parts of mathematics, it is agreed which methods and tools belong to such a standard: For example, every engineer has to know how to use linear differential equations and the Laplace transform. This is part of what is called "mathematical modelling". The restriction of this term to continuous models is no more adequate today, because highly nontrivial discrete systems, especially software systems, occur in the daily practice of virtually every engineer (not only in software engineering). In an evolving discipline of "discrete modelling" and "discrete system theory", logic has a significant part (together with other fields like data structures, automata theory, algorithmics). In the long run, this discrete system theory should provide a "calculus" which is to be applied in any professionally performed construction of software systems.

When this challenge is taken seriously, the focus of logic will be shifted beyond the scope of classical mathematical logic, even more as it already did during the formation of theoretical computer science. Some mathematical logicians will say that these tasks should be carried out by computer scientists, and some logicians in computer science may say that data base theorists, programming language researchers, or software specialists will do the job. Anyway, if logicians of any flavour would agree that challenges like the ones mentioned above are interesting and not just an outgrowth of a fashion, then their expertise would contribute to a much faster progress. Moreover, there would be less discussion whether logic institutes be closed or logic professorships cancelled. The best response to the challenges raised above is an intensive cooperation between logicians, computer scientists, and engineers.

A word of caution seems to be in order. The idea to develop a new way of teaching logic to engineers does not mean to throw all the treasures away which logic has given us. Especially for basic courses it is important to present coherent and lucid theory, as it was developed in logic by a long process of scientific effort. At the present time, it seems that a comprehensive and polished treatment of "logic for engineers" does not yet exist. Many more steps are needed to arrive at it, especially to separate the lasting principles from the hot but sometines not so deep topics.

The task of shaping clean, adequate, and convenient theoretical frameworks which can be taught to and are usable by engineers, is hard and requires the study of engineering practice. It will not be funded much, will for a long time not share the glory of industrial partnerships (as many "applied" projects do), and it will be progressing slowly. But the long-term impact will be high, and I am certain that over the coming decades the demand for this kind of research will grow, in the same way as the demand for reliable and manageable software systems will grow.

6 Acknowledgment

I thank several colleagues for their helpful comments on a draft of this paper, in particular Heinz-Dieter Ebbinghaus, Jörg Flum, Erich Grädel, and Reinhard Wilhelm.

References

1. S. Abramsky, D. M. Gabbay, T.S.E. Maibaum (eds.) *Handbook of Logic in Computer Science*, Vols. I - IV, Clarendon Press, Oxford 1992-1995.
2. J. Barwise (ed.), *Handbook of Mathematical Logic*, North-Holland, Amsterdam 1977.
3. Y. Gurevich, Logic and the challenge of computer science, in: *Current Trends in Theoretical Computer Science* (E. Börger, ed.), Computer Science Press, 1988, pp. 1-57.
4. R. Milner, Elements of interaction - Turing Award Lecture, *Comm. ACM* 36(1), 1993, pp. 78-89.
5. G.W. Leibniz, Anfangsgründe einer allgemeinen Charakteristik (Latin original untitled), in: *Die philosophischen Schriften von Gottfried Wilhelm Leibniz* (C. I. Gerhardt, ed.), Vol. VII, Berlin 1890, p. 185 ff.