From Hilbert's Program to a Logic Tool Box

J.A. Makowsky
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
janos@cs.technion.ac.il
www.cs.technion.ac.il/∼janos

Dedicated to Victor Marek on his 65th birthday

Abstract

In this paper I discuss what, according to my long experience, every computer scientists should know from logic. We concentrate on issues of modeling, interpretability and levels of abstraction. We discuss how the minimal toolbox of logic tools should look like for a computer scientist who is involved in designing and analyzing reliable systems. We shall conclude that many classical topics dear to logicians are less important than usually presented, and that less known ideas from logic may be more useful for the working computer scientist.

1 Teaching Logic

The following text is not a scientific paper. It is really a prose version of a set of slides in which I present my ideas on the subject. I have presented a first version of these slides at the LPAR'07 conference in Yerevan, Armenia, in October 2007. I do hope that I will finally turn these thoughts into a proper scholarly paper. In these sketchy notes I mostly give references to monographs, and not the original papers.

The Students I Have in Mind

I want to examine what we should teach from logic to our *non-specialized undergraduate students.* I mean, what does *every* graduate of Computer Science have to learn in/from logic? The current syllabus is often justified more by *the traditional narrative* than by *the practitioner's needs.* The practitioner's needs are determined by what he needs to understand his own activity in dealing with his computing environment. As a computer/computing engineer he should be aware of the inherent difference between *consumer products* and *life-critical hardware and software.* The occasional failure of consumer goods is beneficial to the functioning of the Fordistic consumer society in as it maintains the consumption cycles needed for its functioning. The failure of life-critical products is disastrous for all the parties involved. Life-critical products have to be properly *specified, verified, tested and certified* before they can be released. The practitioner therefore needs a basic understanding of what it means to properly specify,test, verify and possibly certify. a product.

Practically speaking,

- he should understand the meaning and implications of modeling his environment as precise mathematical objects and relations;
- he should understand and be able to distinguish intended properties of this modeling and side-effects;
- he should be able to discern different level of abstraction;
- he should master the (non-formalized) language of sets and second order logic which enables him to speak about the modeled objects;
- he should understand what it means to prove properties of modeled objects and relations;
- but he should also understand the *inherent limitations* of what can be achieved, and of his own activity.

2 Sets and the Logical Foundations of Mathematics

Whether we like or not depends on our philosophical position, if we at all have one, but it is a fact supported by a large social consensus, that the language of sets is the most used and most accepted way of modeling mathematical objects. A very convincing discussion, why sets are used that way, is given in (Blass & Gurevich 2008). We are used to model automata of all sorts including Turing machines as tuples of sets, functions and relations. We do the same when we discuss behavior of hardware and software, when we prove properties of modeled artefacts, and when we show that certain combination of properties of such artefacts cannot be achieved.

The emergence of the language of sets goes back to the work of G. Cantor and G. Frege, who both felt the need to put mathematics on new rigorous foundations upon which the growing edifice of real and complex analysis could be built. Cantor initiated the use

of sets for modeling natural, real and complex numbers and their functions, and Frege wanted to derive the rules of set formation from logic. Frege's program intended to derive the foundations of mathematics from logical principles. It derived set theory as the universal data structure for modeling mathematical objects from logic. The history of logic in the years between 1850 and 1950 is the history of successes and failures of Frege's program. This history forms the traditional narrative along which we are used to teach logic. I will argue that this narrative is misleading as far as the working mathematician or computer scientist or engineer is concerned.

Let us look at this traditional narrative the way I see it. We start by paraphrasing the history of Logicism from Frege to Gödel, and further to the re-evaluation of Frege's program.

Act I: Cantors Paradise

- First G. Cantor (1874 - 1884) created the Paradise of Sets.
- Then G. Frege (1879) created the modern Logical Formalisms, including the *correct binding rules for quantification*, and
- set out to lay the Foundations of Mathematics with his *Die Grundgesetze der Arithmetik*, Volume1 (1893), see (Burgess 2005)
- The book was not well received. Only G. Peano, author of *The principles of arithmetic, presented by a new method* (1889), (Kennedy 1973) wrote a positive review of it.

Act II: Paradise lost

- On 16 June 1902, Bertrand Russell pointed out, with great modesty, that the Russell paradox gives a *contradiction* in Frege's system of axioms.
- ... and with Russel's paradox started the *crisis* of the *Foundations of Mathematics*,
- G. Cantor had sensed this, when he noticed trouble with the "set of all sets" and his notion of cardinality. Let V be the set of all sets. Then its power set $P(V)$ is a subset of V. But Cantor proved that the cardinality of the power set $P(A)$ of a set A is always strictly bigger than the cardinality of A. On the other hand the cardinality of a subset of A is at most the cardinality of the set A, a contradiction.

Act III: Hilbert's Program

D. Hilbert around 1920 designs a program to provide a secure foundations for all mathematics. In particular this should include[1]:

- *Formalization* of all mathematics: all mathematical statements should be written in a precise formal language, and manipulated according to well defined rules.
- *Completeness*: a proof that all true mathematical statements can be proved in the formalism.
- *Consistency*: a proof that no contradiction can be obtained in the formalism of mathematics. This consistency proof should preferably use only "finitistic" reasoning about finite mathematical objects.
- *Conservation*: a proof that any result about "real objects" obtained using reasoning about "ideal objects" (such as uncountable sets) can be proved without using ideal objects.
- *Decidability*: there should be an algorithm for deciding the truth or falsity of any mathematical statement.

[1]This subsection is a quote from Wikipedia. Its author is unknown.

Hilbert's Logic Lectures

In 1928, D. Hilbert and W. Ackermann publish *Grundzüge der theoretischen Logik*, (Hilbert & Ackermann 1928; 1949; 1950). Here are the points of interest to us:

- The Logic in question is *Second Order Logic*.
- What we call *First Order Logic*, is called there the *restricted calculus*.
- They *prove* soundness of the calculus, and ask the question of *completeness*.

The book is soon translated into English, French and Russian and remains the most widely used reference for more than thirty years. K. Gödel, as a graduate student, reads the book in 1928. The original book contains several technical mistakes which are fixed in subsequent editions. The first English edition (Hilbert & Ackermann 1950) gives credit to A. Church and W. Quine for pointing out some mistakes in the second German edition.

The book states as the main problem of Logic its axiomatization and proofs of the

- Independence of the axioms
- Consistency of the axioms
- Completeness of the axioms
- The decision problem of the consequence relation

Act IV: Rise and Fall of Hilbert's Program

Initial successes:

- Leopold Löwenheim (1915), Thoralf Skolem (1920), Mojżesz Pressburger (1929), Alfred Tarski (1930), Frank Plumpton Ramsey (1930), László Kalmár (1939) and many others prove partial *decidability* results for fragments of Logic, and for Arithmetic, Algebra, Geometry.
- In 1929 Kurt Gödel proves the *completeness* of the Hilbert-Ackermann axiomatization of the the *restricted* (first order) calculus.

Final blows:

- 1931 K. Gödel proves that every recursive theory which contains arithmetic is *incomplete*.
- 1931 K. Gödel proves that every recursive consistent theory which contains arithmetic cannot prove its own consistency.
- 1936 Alonzo Church and Alain Turing show that already for the *restricted* calculus with free relation variables the set of tautologies is not computable (but is semi-computable). Hence, they gave a negative solution to the Decision Problem.

The most comprehensive account of solvable and unsolvable cases of the Decision Problem can be found in (Börger, Grädel, & Gurevich 1997).

Act V: Clarifications and Repairs

Out of the ashes rise the four classical sub-disciplines of mathematical logic:

Set Theory arises from work by E. Zermelo, D. Mirimanoff, J. von Neumann, A. Fränkel, K. Gödel and P. Bernays. Alternative approaches were developed by, among others, W. Quine, W. Ackermann, and J.L. Kelley and A.P. Morse, and more recently, by P. Aczel.

Set theory, in contrast to using sets in mathematical practice, is mostly concerned with settling questions around the axiom of choice and cardinal arithmetic, or in formulating alternatives, such as the axiom of determinacy, and in clarifying their impact on questions in topology and analysis. Today, set theory is a highly specialized branch of technical mathematics with little impact on computer science.

Proof Theory arises from work by W. Ackermann, G. Gentzen, J. Herbrand, D. Hilbert and P. Bernays.

It has developed into a full-fledged theory of proofs, comprising the analysis of (transfinite) consistency proofs in terms of ordinals and fast-growing functions, program extraction from proofs, and resource analysis related to provability. It also plays an important role in all aspects of automated reasoning, an important branch of Artificial Intelligence.

Recursion Theory arises from work by E. Post, J. Herbrand, K. Gödel, A. Church, A. Turing, H. Curry. Recursion theory developed at one side into degree theory, classifying non-computable functions according to their different levels of complexity, at the other side it developed into Computability Theory and has become one of the pillars of Computer Science education in its own right.

Model theory arises from work by T. Skolem, A. Tarski, A. Robinson, R. Fraïssé and A. Mal'cev.

Two main directions evolve, classification theory, and a more algebraic and geometric theory, linking model theory with algebraic geometry and number theory. Although finite model theory has its early origin, it was through Automata Theory, Database Theory and Complexity Theory that it evolved into its own discipline with a legitimate place in advanced Computer Science education.

... and for long this remained the classical divide of **Mathematical Logic**.

J. Shoenfield's monograph (Shoenfield 1967) is possibly the only monograph covering all aspects of Mathematical Logic up to the boundaries of research of his time. Since then the four classical disciplines pursue their own paths, and among the younger generations of researchers it cannot be taken for granted that they have studied the four disciplines in depth.

Act VI: 100 years later - Fixing Frege

If only G. Frege had not been so scared by B. Russel's letter. C. Wright, P. Geach and H. Hodes suggested, and G. Boolos proved (1987) that a modified Frege program actually is feasible, (Boolos 1998a; 1998c; 1998b). They noted that the famous contradiction stemmed from the axiom which states, roughly, that the extension of any concept is a set. However, this axiom is only used to derive an abstraction principle, called Hume's principle, which states, again roughly, that two extensions have the same cardinality if and only there is a bijection.

So we have for the modified Frege system:

Frege: The Peano Postulates can be deduced in dyadic second order logic from Hume's principle and suitable definitions of the natural numbers (Frege's Arithmetic).

Boolos: Frege's arithmetic is interpretable in second order Peano Arithmetic.

Some (the Neo-Logicists) argue that this justifies a revival of Logicism. But it also creates new problems. A thorough discussion of the pros and cons of Neo-Logicism can be found in J. Burgess (Burgess 2005). A thorough discussion of *abstraction principles* similar to Hume's Principle can be found in K. Fine (Fine 2002).

That much for the "big crisis".

At least the set theory needed for the foundations of Computer Science can be derived from logical principles.

3 The Foundations of Mathematics and Computer Science

What Frege and Russel and Whitehead had in mind, viz. to build the foundations of mathematics from scratch, was done in a more intelligible (but still not too user friendly) way by E. Landau in his *Foundations of Analysis* first published in German in 1930, and in English in 1951, with many reprints, the latest in 1999 with a German-English vocabulary by the American Mathematical Society, (Landau 1999). In this book

he explicitely constructs the real and complex numbers from the standard model of Peano arithmetic. Landau's style is very dry and concise, and the text was written for mature mathematicians. A more pedagogical version of the same constructions can be found in S. Feferman's (Feferman 1964), which I also would love to see reprinted. One can view such a foundation of Analysis as a *pragmatic* version of Frege's program. Roughly, one proceeds as follows:

- One starts with a cumulative hierarchy of sets, based on the empty set alone (or with urelements) and natural set construction principles which allow to construct also infinite sets.
- Then one defines inductively the natural numbers with a successor function, and the sets of finite words over a (not necessarily) finite alphabet (a set), with an append operation for each element of the alphabet.
- One then proceeds with defining the number systems $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and their arithmetic operations *inductively* and using *quotient structures*.
- Then one construct the reals $\mathbb{R}$, using Dedekind cuts.
- In similar ways one constructs other structures, say, groups, fields, topological spaces, Banach spaces, Lie algebras, which are specified *axiomatically*.
- *Existence* of axiomatically defined objects had to be established by an *explicit sequence of set construction steps within the cumulative hierarchy*.
- Clearly, one can apply the same methods to model objects of the computing world, such as automata, formal languages, programs, data structures, etc.

One *should* adapt Landau's way for modeling the basic data structures of Computer Science. I have attempted to do this in the course *Sets and Logic for Computer science*, which we teach in the third semester in our Computer Science undergraduate program.

Set Theory in Computer Science

Besides using set theory for modeling purposes, the computer scientist uses only few ingredients of set theory:

1. The Cantor-Bernstein Theorem to prove equicardinality,
2. The fact that a countable union of countable sets is countable,
3. The fact that the cardinality of the power set of a set A is always bigger than that of A,
4. The relationship between the termination of processes and well-orderings.
5. The Recursion Theorem.
6. Some Fixed-Point Theorems.

The first three of these go ultimately back to Cantor's original work and are very basic. The Recursion Theorem and Fixed Point Theorems should be taught in a more advanced course.

Recursion Theory vs Computation Theory

Recursion Theory got its name for a good reason: The computable functions over the natural numbers were defined recursively, and early Recursion Theory consisted in studying the strength of various proposed recursion schemes. Recursion schemes can be replaced by register machines. Computability Theory studies usually the computable relations and functions over sets of words. The three approaches are inter-translatable, but they are not the same. It is a pity that teaching all these complementary notions of computability is not always part of the Computer Science curriculum. Be that as it may, Computation Theory has emancipated itself from Logic and in Computer Science the two are often taught independently. A good exception is the monograph (Papadimitriou 1994).

Proof Theory

Proof theory evolved around the question what type of consistency proofs are at all possible. As a spin-off the field of deduction-based automated reasoning and automated theorem proving came into being. Proof theory is also used in the foundations of programming languages, where it generated a rich literature of deep insights into the nature of programming. Some basic principles of automated reasoning do belong into a beginner's course on Artificial Intelligence, and some basic facts about functional programming do belong into a basic course on Programming Languages. However, only little of this rich material is suitable for the undergraduate student I have in mind. A comprehensive survey of Proof Theory is (Buss 1998).

Model Theory

The main tools of classical model theory almost all derive from the compactness theorem and variations of the model existence theorem. Using these tools one proves preservation theorems, the omitting types theorem and develops a general understanding of the possible structure of models of first order theories. I have described how to use these tools in the Computer Science context, in database theory, the foundations of Logic Programming, and the specification of data types, (Makowsky 1984; 1992). It turned out, however, that the Compactness Theorem is mostly suitable when dealing with infinite structures, and the most prominent application of the Compactness Theorem in Computing is Herbrand's Theorem with its ramifications in Automated Reasoning and Logic Programming. The most important tools from Model Theory in algorithmic applications are the Ehrenfeucht-Fraïssé games and the Feferman-Vaught Theorem and its variations. The former is omnipresent in Finite Model Theory, cf. (Ebbinghaus & Flum 1995; Libkin 2004) and the a survey of the uses of the latter can be found in (Makowsky 2004). For other failures of classical theorems of First Order Logic when restricted to the finite case, cf. (Gurevich 1988).

One should add here that the combination of the Compactness Theorem with the Ehrenfeucht-Fraïssé

games leads to Lindström's characterization of First Order Logic. The attempts to develop an *Abstract Model Theory* are documented in the monumental (Barwise & Feferman 1985). This line of reasoning had a considerable impact on Finite Model Theory and Descriptive Complexity in providing techniques for defining logics which capture complexity classes. For the advanced student applications of Finite Model Theory to Computer Science are surveyed in (Grädel *et al.* 2007).

The classical textbooks in Logic

The available undergraduate texts of Logic for Computer Science follow too often the narrative of the em Rise and Fall of Hilbert's Program. They emphasize the classical Hilbertian topics.

- Logic is needed to resolve the paradoxes of set theory.
- First Order Logic is THE LOGIC due to its completeness theorem.
- The main theorems of logic are the *Completeness Theorem* and the *Compactness Theorem*
- The tautologies of First Order Logic are not recursive.
- Arithmetic Truth is not recursive enumerable.
- One cannot prove CONSISTENCY within rich enough systems.

This is NOT what a Practitioner of Computing Sciences NEEDS !

Other texts are often written with a very special agenda reflecting the author's research interest or his particular tastes. Finally, there are texts which are really written with the undergraduate Computer Science student in mind. But then they are often either written specifically with programmers in mind, and do not deal with the data modeling issues[2]. Logic is first of all language in which we express ourselves before we prove statements. We first have to formulate a *specification*, a *database query*, an *intermediate assertion*, a *loop invariant*, before we prove them to hold, to be valid, or for two of them to be equivalent or not. Logic deals with **definability** issues as much as with **provability** issues, something which in the Hilbertian tradition is easily forgotten. An introductory text in Logic for Computer Science should choose its topics in way, that the student meets the topics taught again in later courses. When we teach Linear Algebra in the first year of a mathematics curriculum, most of the topics reappear in vector analysis, differential equations, physic, statistics etc. We have to build our syllabus of logic keeping this in mind.

[2]The recent book by R. Bornat(Bornat 2005) is a lovely introduction to Logic for programmers.

4 So what Does a Practitioner of Computing Sciences Need?

We distinguish between knowledge of *theoretical orientation* and *practical* knowledge, which consists of *tools* and *skills*. In our case this means:

Theoretical orientation:

- *awareness* that our domain of discourse is an *idealized world of artefacts* which models fairly accurately the artefacts which allow us to run and interact with computing machinery.
- *awareness* of the different levels of abstractions.
- *awareness* that in this world of artefacts there are *a priori limitations*. Not everything is realizable, computable, etc.

Practical knowledge:

- *tools* which allow us to *model new artefacts*, whenever they arise;
- *tools* which allow us to *prove properties* of the modeled artefacts.

The student needs a carefully adapted blend of the *practical Frege* program, with the knowledge of its *limitations*. He needs both *proficiency* and *performance* in his practical knowledge.

5 Lessons from 150 years of History

I have spent so much space reviewing what I consider noteworthy in the evolution of the Logicists program because I do want to draw some lessons from it which are *not foundational* but *practical*. I would like the B.Sc. graduates of Computer Science to be familiar with the following:

Lesson 1. Modeling the world

Our scientific language: Natural Language enhanced by precise use of boolean operations, quantification and the use of naive language of sets.

Our universal data structure: A cumulative world of sets.

Modeling the world: We model *all* artefacts of our computing world by constructed objects in the world of sets.

Modeling involves side effects:
Modeled artefact have properties not intended.

Digression: the ordered pair: Ordered pairs could be introduced via an abstraction principle:
$(x, y) = (x', y')$ if and only if $x = x'$ and $y = y'$.
But usually the ordered pair is modeled directly:
N. Wiener: $(x, y)_W := \{\{\{x\}, \emptyset\}, \{\{y\}\}\}$,
K. Kuratowski: $(x, y)_K = \{\{x\}, \{x, y\}\}$,
Simplified : $(x, y)_S = \{x, \{x, y\}\}$.
Now one has to verify that $(x, y) = (x', y')$ if and only if $x = x'$ and $y = y'$. All the proposed versions do satisfy this, but the proofs differ. The simplified version requires the axiom of foundation.

Kuratowski's version is the accepted definition today. But all definitions have **side effects**, e.g., x is an element of $\{x, \{x, y\}\}$ but not of $\{\{x\}, \{x, y\}\}$. Proving properties of objects which depend on the use of ordered pairs should not use these side effects, but only the defining property.

The distinction between specified properties and side effects should be taught early on!

Fixing levels of abstraction: Introducing structures, and fixing which sets are not further to be analyzed.

A graph is a pair $< V, E >$.
A finite automaton is a tuple $< S, \Sigma, R, I, T >$.

Like in the foundations of Analysis, as practiced by R. Dedekind, E. Landau and N. Bourbaki, we need the *precise language mix* of *normalized natural language* augmented by the *language of sets* to *model* the *idealized artefacts* of computer science. To model the artefacts we also need basic tools.

Artefacts:
strings, concatenation, natural numbers,
graphs, relational structures stacks, arrays;
circuits, Turing machines, register machines;
specification and programming languages,

Tools: Inductive definitions, proofs by induction;
enumerations,
proving countability and uncountability;
well-orderings (for termination)

Is this not "too denotational" ? .... our friends may ask.

Yes, this approach *does* map everything into *sets*. But "truth" does not necessarily *presuppose* a world of sets. Truth in the sense of Frege's world is defined by the laws (introduction and elimination rules) of logic and of the Fregean constructs. It does leave your *foundational options* open ...

Lesson 2. Modeling Computability and its limitations (when modeled)

We have already said that computability is usually taught in a separate course, be it together with formal languages or with an introduction to basic complexity theory. Nevertheless, our student should understand that computability is modeled over different domains, computing operations, resource restrictions.

Natural numbers and recursion:
The original definition of the set of *recursive functions*.

Natural numbers and register machines:
Close to early programming languages.

Turing machines and words: Close to assembly languages.

Other models: Logic programs, Lambda calculus, cellular automata, quantum computing

Showing their equivalence involves modeling also

- translation between the domains;
- translations between programs (interpreters and compilers).

Here I want to stress The different basic structures involved, and their bi-interpretability. In terms of knowledge of orientation and practical knowledge we have:

Orientation:
Not everything is computable.
Not everything is feasibly computable.

Tools:
Using the non-solvability of the Halting Problem to prove non-computability.
Using different types of reducibilities (and simulations).

I have observed that even my colleagues are sometimes imprecise: The Church Turing Hypothesis is often carelessly invoked. There is also a trend to say *computable* when actually one means *feasibly computable*, where *feasibly computable* may mean computable in *deterministic* polynomial time, sometimes computable in polynomial time with *randomized algorithms*.

It is also important to *distinguish* between complexity classes defined as *equivalence classes of problems* under certain reductions, and sometimes defined as classes of decision (counting, approximation) problems solvable in a specific computational model.

Using polynomial time Turing reductions, the class $[SAT]_T$ of problems reducible to SAT is of the first type, NP is of the second type, and $NP = [SAT]_T$ is a theorem. In the case of counting problems $[\#SAT]_T$ is of the first type, $\#P$ is of the second type, and $\#P = [\#SAT]_T$ is not true. Using reductions in First Order Logic we still have $[SAT]_{FOL} = NP$, but not every problem X, which is NP-complete with respect to polynomial time Turing reductions satisfies $[X]_{FOL} = [SAT]_{FOL}$.

It is important to insist that slogans are replaced by precise definitions.

Lesson 3. Modeling Syntax and Semantics

We look at Propositional, First Order, Second Order Logic, or any other logic of *assertions*. Again we model them in our framework of sets.

Syntax:

The syntax is an inductively defined set of words, the well formed expressions.

Semantics: *Structures* are interpretations of the basic non-logical symbols. *Assignments* are interpretations of the variables. The *meaning function* associates with structures, assignments, and formulas a truth value.

> What is the meaning of an assertion ?

Without free variables: The meaning of an assertion is a *truth value*.

> But this is misleading!

With free variables: The meaning of an assertion is the set of interpretations of its free variables. In the case of first order variables only it is *a relation*.

As in Classical Geometry one speaks of the *geometrical lieu* of all points satisfying an equation, we can speak of the *logical lieu* defined by a formula. In modern data base parlance this is called a *query*.

We define usually *logical validity* via truth values. It would be preferable to define *validity* and *logical consequence* directly for formulas *with free variables*.

Our student is more likely to meet in the sequel of courses formulas with free variables that just formulas without.

Do we need the Completeness Theorem?

For the **practical knowledge** we need:

- The semantic notion of *logical consequence*.
- Enough basic logical equivalences to to prove the *Prenex Normal Form Theorem (PNF)*.
- Introduction and elimination rules for quantifiers (via constants).
- A *game theoretic* interpretation of formulas in PNF.

For the **knowledge of orientation** we might state (but not prove) the Completeness Theorem for our redundant set of manipulation rules.

Here are the arguments for and against proving the Completeness Theorem in the first course of Logic.

The classical argument pro:

- Completeness and its corollary, *Compactness* is at the heart of logic.

My arguments against:

- None of these are part of the practical knowledge we aim at.
- The proof of the Completeness Theorem is a waste of time at the expense of teaching more the important skills of understanding the manipulation and meaning of formulas.
- First Order Logic is not privileged in our context. We deal very often with finite structures, where the Completeness Theorem is not true.
- *Second Order Logic* anyhow is the natural logic we work in, and not taking that seriously confuses the student.

> We should instead concentrate on understanding quantification

As **tools** we need to

- Read, write and *understand the meaning* of First Order **and** Second Order formulas.
- Understand the relationship between *projection of relations* and first order quantification.
- Understand that *Relational Calculus* and First Order Logic are really the same (i.e., bi-interpretable).
- Introduce immediately after the proof of the Prenex Normal Form Theorem the *Ehrenfeucht-Fraïssé Game*, and proceed to show the easy direction of the *Ehrenfeucht-Fraïssé Theorem*, i.e., if a formula (say in Prenex Normal Form) of quantifier rank k is true in one but not in another structure, then we can derive from the formula a winning strategy for player I (the spoiler) for the game with k moves.
- Play with the *game interpretation* of quantifiers to analyze the *amount of quantification* needed to express, say "there exists at least n elements x such that $\phi(x)$".
- One can even point out that (easy direction) of *Ehrenfeucht-Fraïssé Theorem* holds also for Second Order Logic.

Lesson 4. Limitations of formalisms: Definability

Before we find time to prove the Completeness Theorem, I would like the students to understand the difference between First Order (FO) and Second Order (SO) Logic.

- Look at the statement

  "There are an equal number of x
  with $P(x)$ and with $Q(x)$"

  where P, Q are unary predicate symbols.

  This is expressible in SO but not in FO, and we can even show the proof having the Ehrenfeucht-Fraïssé Games available.
- We can even be daring, and show that connectedness on finite graphs is SO-definable, but not FO-definable.
- In the natural numbers $\mathbb{N}$, *multiplication* is SO-definable, but not FO-definable, using addition only.

However, multiplications is FO-definable using addition and squaring.

The negative result we cannot prove in an undergraduate course, as we need the decidability of FO Pressburger Arithmetic. But we can explain it.

Lesson 5. Interpretability and Reducibility

Again before we use our time prove the Completeness Theorem I would like the students to understand what it means that a structure is FO-interpretable in another structure. Let look at the case of the natural numbers $\mathfrak{N}$ and the integers $\mathfrak{Z}$ with their arithmetic operations.

The integers $\mathfrak{Z}$ with their arithmetic are *FO-interpretable* inside the natural numbers $\mathfrak{N}$ with their arithmetic.

To get the interpretation we define a new structure from $\mathfrak{N}$, called a *transduction* $T(\mathfrak{N})$, and which will be isomorphic to $\mathfrak{Z}$, as follows.

- The new universe consists of equivalence classes of pairs of natural numbers such that $(x, y) \sim (x', y')$ iff $x + y' = x' + y$.
- The new equality is this equivalence.
- The new addition is the old addition on representatives.
- Same for multiplication.

T is a semantic map. Its syntactic counterpart is the *interpretation* $S : Formulas \to Formulas$, defined as follows:

For any SO-formula ϕ we let $S(\phi)$ be the result of substituting the *new* definitions of addition and multiplication and equality for the corresponding symbols. In the exact definition one has to be careful with the renaming of free variables.

S and T are intimately related:

$$\mathfrak{Z} = T(\mathfrak{N}) \models \phi \text{ iff } \mathfrak{N} \models S(\phi)$$

which is the *Fundamental Property* of *Transductions* and *Interpretations*.

In the same way we can see that

- The Cartesian product is interpretable in the disjoint union.
- Many graph transformations are given as transductions.
- All implementations of one data structure in another are of this form.
- Transductions and interpretations are everywhere

6 The Fundamental Properties of SO and FO

In teaching our students to think and speak Second Order Logic, we should teach

- that isomorphic structures satisfy the same SO sentences;
- the Fundamental Property of Transductions and Interpretations;
- the Prenex Normal Form Theorem and its visualization as a two person game.

and we should practice thinking in SO as the natural language of specifying properties of modeled artefacts.

The Fundamental Properties of FO

Besides the properties of SO we have

The *Ehrenfeucht-Fraïssé Theorem*:

Two structures *can be distinguished* by a sentences of quantifier depth k iff Player I (the spoiler) can force a win in the EF-game of length k.

or, equivalently

Two structures *cannot be distinguished* by a sentences of quantifier depth k iff Player II (the duplicator) can force a win in the EF-game of length k.

We say that two structures are k-isomorphic if Player II can force a win in the EF-game of length k.

Furthermore:

k-isomorphism is preserved under the formation of disjoint unions of structures.

Modified versions also hold for Monadic Second Order Logic, but *not* for SO.

Combining EF-Games and Interpretations

Combining games and interpretations gives a very powerful tool to compute the meaning function of a FO formula in a complex structure by reducing this computation to simpler structures.

If G is obtained from graphs H_1, H_2 by applying disjoint unions, Cartesian products, and first order definable transductions T_1, T_2, say

$$G = T_1(H_1 \times T_2(H_2))$$

then the truth of the formulas of quantifier rank k in G is uniquely and effectively determined by the the truth of the formulas of quantifier rank k which hold in H_1 and H_2.

This is the *Feferman-Vaught Theorem*. It allows us to compute the meaning function for FO-formulas (or MSO-formulas) of composite structures by reducing its computation to the meaning functions of the formulas on the components. I have surveyed how to use the Feferman-Vaught Theorem in Computer Science in my paper in (Makowsky 2004).

7 My Logic Tool Box

So we finally come to the description of the **Logic Tool Box** I would like to give to our students. Tools to do what, you will ask. Tools to think *rigorously* in order to

approach the disciplines of programming and information processing, tools to model accurately new artefacts, as they occur, tools to grasp the scope of abstraction and modularity. Our students are not logicians. Logic per se is not their main interest. Logic is for Computer Science what Hygienics is to Medicine. They should learn **rigorous informal reasoning before** they learn to model this kind of reasoning as *formalized proof sequences*. Needless to say that each tool comes with a **required skill** how to use it.

My Logic Tool Box contains:

Modeling tools:

- Basic set construction principles;
- Inductive definitions;
- Proofs by induction;
- Basic cardinality arguments.

Logic tools:

- Propositional Logic and its axiomatization.
- Second Order Logic as the main formalism to express properties of the modeled artefacts.
- The semantic notion of logical consequence and validity.
- Validity over finite structures.
- Quantifier manipulation rules.
- Skolem functions.
- The Fundamental Property of Transductions and Interpretations.
- First Order Logic as an amenable fragment of Second Order Logic.
- The *Ehrenfeucht-Fraïssé Theorem* and its refinements.
- The Feferman-Vaught Theorem and its variations.

We said before that the Completeness Theorem for First Order Logic holds only, if we define validity over all First Order Structures. For Second Order Logic one would have to explain the difference between Henkin's notion of validity and standard second order quantification. Just stating the Completeness Theorem for First Order Logic misleads the student, and explaining its true subtleties may be beyond the undergraduate level.

Where these tools work

I have chosen the Logic Tool Box with a view on the courses our student has to take during his undergraduate studies. Ideally, the course I have in mind, **Sets and Logic for Computer Science**, should be taught in the second or third semester. The student should have studied already **Discrete Mathematics** and **Algorithms and Data Structures**, so the teacher can rely on the intuition of the students, and the examples developed in these courses. The course should play the same role as the course Number Systems used to play when it was still customary to teach it, cf. (Feferman 1964),

The modeling skills taught in our course should help him in the following (usually compulsory) courses:

- Automata and Formal Languages
- Introduction to Computability
- Database Systems
- Graph Algorithms
- Principles of Programming Languages
- Computer Architecture
- Introduction to Artificial Intelligence

The more specialized topics of Logic should be taught in advanced courses: A course **Advanced Topics in Logic** could have three parts, covering the Completeness and Incompleteness Theorem, Ordinals and Termination, and Temporal and Modal Logic. Other topics belong there where they are rally used, in the courses on **Verification**, **Automated Theorem Proving**, **Principles of Logic Programming**, **Database Theory**, **Functional Programming** and so forth.

8 What was omitted?

I have not included in my discussion what is called in the standard classification Non-classical Logics. These logics can also be modeled using set-theoretic tools, and indeed they are. When they find applications to Computer Science, as Temporal Logic (Manna & Pnueli 1995) or the Logic of Knowledge, (Fagin *et al.* 1995), they also find there way into more advanced courses.

I have not included in my discussion two classical concerns of the debate around the Foundations of Mathematics and Computer Science: Epistemology and degrees of constructivism. A delightful and insightful presentation and discussion of these matters from a contemporary point of view can be found in (Shapiro 2000). Although I tend to be a Platonist, viewing mathematical concepts as *real*, I am aware of the difficulties inherent in this position, cf. (Maddy 1990; 1998). I am also aware of the social and cultural mechanisms at work which strongly influence how science evolves, cf. (Wilder 1981). However, I strongly object to the arguments which take the social and cultural mechanisms at work as a justification for the erroneous claim that scientific truth is purely social and cultural.

From a more pragmatic point of view I tend to be a Formalist, viewing the observable part of the mathematical and logical enterprise as happening on (virtual) paper written with (virtual) pencils. Concerning the degrees of constructivism I subscribe to, I only want to remark that often *non-constructive* is confused with *lack of detail*. The axiom of choice is an example of *lack of detail*. I assume that the choice function exist and I want to proceed from there, filling in the details (implementation) later, or leaving them to others. Software engineering always proceeds like this and is not considered non-constructive even by the most extreme constructivists. Using a cardinality argument to prove the existence of, say, transcendental numbers, expander graphs or other combinatorial objects, is consid-

ered non-constructive, but can be explained in the same way.

Our students, however, should rather follow the advise of the Rabbinic Sages, who admonish us not to study Kabbala (Jewish Mysticism) before the mature age of forty years and before serious exposure to the more down-to-earth matters of Talmud and Torah. Our students should view the Philosophy of Mathematics and of Computer Science as something to be left for later. Children do not question linguistic principles before they learn their first language. Scientists should not question Science before they master the craft.

Acknowledgments

I would like to thank N. Francez, D. Giorgetta, S. Halevy and D. Hay for stimulating discussions and suggestions about how to teach Logic to Computer Science students.

I would like to thank the Trade Union of University Professors (Irgun HaSegel) of Israel for giving me time to prepare this paper. At the moment of completion we were in the forth week of our teaching strike.

References

Barwise, J., and Feferman, S., eds. 1985. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer Verlag.

Blass, A., and Gurevich, Y. 2008. Why sets? In Avron, A.; Dershowitz, N.; and Rabinowich, A., eds., *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*. Springer. In press.

Boolos, G. 1998a. The consistency of Frege's "foundations of arithmetic". In *Logic, Logic, Logic*. Harvard University Press. 182–201.

Boolos, G. 1998b. *Logic, Logic, Logic*. Harvard University Press.

Boolos, G. 1998c. On the proof of Frege's theorem. In *Logic, Logic, Logic*. Harvard University Press. 275–90.

Börger, E.; Grädel, E.; and Gurevich, Y. 1997. *The Classical Decision Problem*. Springer-Verlag.

Bornat, R. 2005. *Proof and Disproof in Formal Logic*. Number 2 in Oxford Texts in Logic. Oxford University Press.

Burgess, J. 2005. *Fixing Frege*. Princeton University Press.

Buss, S., ed. 1998. *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishers.

Ebbinghaus, H., and Flum, J. 1995. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning About Knowledge*. MIT Press.

Feferman, S. 1964. *The number systems: foundations of algebra and analysis*. Addison-Wesley.

Fine, K. 2002. *The Limits of Abstraction*. Oxford University Press.

Grädel, E.; Kolaitis, P.; Libkin, L.; Marx, M.; Spencer, J.; Vardi, M.; Venema, Y.; and Weinstein, S. 2007. *Finite Model Theory and its Applications*. Springer.

Gurevich, Y. 1988. Logic and the challenge of computer science. In Börger, E., ed., *Trends in Theoretical Computer Science*, Principles of Computer Science Series. Computer Science Press. chapter 1.

Hilbert, D., and Ackermann, W. 1928. *Grundzuge der theoretischen Logik*. Springer.

Hilbert, D., and Ackermann, W. 1949. *Grundzuge der theoretischen Logik, 3rd edition*. Springer.

Hilbert, D., and Ackermann, W. 1950. *Principles of Mathematical Logic*. Chelsea Publishing Company.

Kennedy, H. 1973. What Russel learned from Peano. *Notre Dame Journal of Formal Logic* 14.3:367–372.

Landau, E. 1999. *Die Grundlagen der Analysis*. American Mathematical Society.

Libkin, L. 2004. *Elements of Finite Model Theory*. Springer.

Maddy, P. 1990. *Realisms in Mathematics*. Oxford University Press.

Maddy, P. 1998. *Naturalisms in Mathematics*. Oxford University Press.

Makowsky, J. 1984. Model theoretic issues in theoretical computer science, part I: Relational databases and abstract data types. In Lolli, G., and al., eds., *Logic Colloquium '82*, Studies in Logic, 303–343. North Holland.

Makowsky, J. 1992. Model theory and computer science: An appetizer. In Abramsky, S.; Gabbay, D.; and Maibaum, T., eds., *Handbook of Logic in Computer Science*, volume 1. Oxford University Press. chapter I.6.

Makowsky, J. 2004. Algorithmic uses of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic* 126:1–3.

Manna, Z., and Pnueli, A. 1995. *Temporal verification of reactive systems*. Springer.

Papadimitriou, C. 1994. *Computational Complexity*. Addison Wesley.

Shapiro, S. 2000. *Thinking about Mathematics*. Oxford University Press.

Shoenfield, J. 1967. *Mathematical Logic*. Addison-Wesley Series in Logic. Addison-Wesley.

Wilder, R. 1981. *Mathematics as a Cultural System*. Pergamon Press.