LOGIC: Interdisciplinary Adventures in Mathematics, Philosophy, Computer Science, and Education

by
Harvey Friedman
University Distinguished Lecture
The Ohio State University
April 23, 2008

I want to thank President Gee for his (overly) kind remarks. I am very glad to see Gordon return to Ohio State this academic year.

I also want to thank the Mathematics, Philosophy, and Computer Science and Engineering Departments who thought of me for this honor. Thanks also to the Selection Committee and the Office of Academic Affairs for choosing me for this occasion.

In looking over the past University Distinguished Lectures, archived on the OSU website, the topics have generally had a clear and transparent relevance to our daily lives.

We have been delightfully treated to Climate History, Legal Aspects of Civil and Human Rights, Superhuman Capabilities of Animals, Psychological Aspects of Cancer, Energy, Food, and Water Scarcity, Americans with Disabilities Act, and The Federal Reserve on 9/11 – just to name a few.

Most recently, Kevin Boyle just gave a dramatic Lecture on the Dynamics and Implications of the Sacco/Venzetti Trials. And Marilyn Brewer gave a fascinating Lecture on the Psychology of Social Groups last Spring.

Now, I want to warn you that this talk will be relatively painful. For some of you, it's going to be about 50 minutes too long.

So I want to give a very short version of the talk.

LOGIC IS EVERYWHERE!

Let me make it even shorter:

LOGIC EVERYWHERE!

For those of you who have remained in the audience, here is the 50 minute version.

There will be a number of words and phrases that I won't have time to explain in detail. I have displayed these in yellow text, and I think you will find it informative to jot these down and search them on the Internet.

You can of course search for LOGIC WIKIPEDIA.

I am going to talk about LOGIC as the science of reasoning.

We all do reasoning. Some of it is entirely straightforward and second nature. Some of it is highly complex and subtle. And the quality of this reasoning that we all do ranges from obviously valid, to highly questionable – and worse!

But doing reasoning is one thing – having a deep understanding of what it is, how we do it, how we should do it – is quite another.

We do not have anything approaching a deep scientific understanding of reasoning in most contexts – not in common sense, not in the arts, and not in the sciences. These are wide open topics of research.

However, we do have a relatively deep scientific understanding of at least some key aspects of mathematical reasoning.

But first, I want to start with some very limited situations in common sense thinking, where we do have a deep understanding.

Let us consider the five logical connectives:

not ¬
and ∧
or ∨
if then →
if and only if ↔

The last four "connect" two sentences, creating a third. The first counts as "connecting one sentence", creating a second sentence.

Here is an example of valid reasoning involving just or, and if then.

Fred lives in Chicago, or Fred lives in New York.
If Fred lives in Chicago then Fred lives in a big city.
If Fred lives in New York then Fred lives in a big city.
THEREFORE, Fred lives in a big city.

In symbols, this falls under the valid inference

$p \vee q$
$p \rightarrow r$
$q \rightarrow r$
$\therefore r$

Here p stands for Fred lives in Chicago,
q for Fred lives in New York,
and r for Fred lives in a big city.

In this very limited context, we can mathematically define what we mean by a valid inference, and we can determine whether such an inference is valid by the method of truth tables.

So far, so good. But this is a severely limited context. It is nowhere near sufficient to reflect reasoning in even very elementary mathematics.

For this, we need to fast forward from the Greeks to the late 1800s and the great philosopher Gottlob Frege.

In addition to the sentential connectives, we must work with the two quantifiers

for all $\forall$
there exists $\exists$

You can't just say to somebody: for all. Or just say: there exists. There has to be some surrounding material.

There is a grammar for this, resulting in a formal language called Predicate Calculus (Gottlob Frege, 1879). Rather than present this grammar, let me just give a group of familiar examples.

1. Everybody loves everybody.
2. Somebody loves somebody.

3. Everybody loves somebody.
4. Somebody loves everybody.

5. Everybody is loved by everybody.
6. Somebody is loved by somebody.
7. Everybody is loved by somebody.
8. Somebody is loved by everybody.

In predicate calculus, this reads:

1. (∀x)(∀y)(L(x,y)).
2. (∃x)(∃y)(L(x,y)).
3. (∀x)(∃y)(L(x,y)).
4. (∃x)(∀y)(L(x,y)).

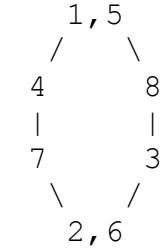
5. (∀x)(∀y)(L(y,x)).
6. (∃x)(∃y)(L(y,x)).
7. (∀x)(∃y)(L(y,x)).
8. (∃x)(∀y)(L(y,x)).

If you think hard about this, you will see that

1,5 are logically equivalent.
2,6 are logically equivalent.

1,2,3,4,7,8 are all logically inequivalent.

Here logically equivalent means: under any interpretation of the concepts involved, the statements are both true or both false.



The downward arrows indicate logical “implication”. In other words, we can infer 2,6 from 7. Also 1,5 each imply 8.

The intricacies of the predicate calculus are well understood, and it assumes a dominant role in modern logic [next slide for inference definition].

An inference in predicate calculus is considered valid if in any interpretation, if all of the premises are true, then the conclusion is true.

COMPLETENESS THEOREM (KURT GÖDEL, 1930). Every valid inference in predicate calculus can be backed up by a proof using only a fixed finite set of basic axioms and rules of inference.

However, it is known that there is no magic bullet like the method of truth tables to determine the validity of an inference in predicate calculus.

NO ALGORITHM (ALONZO CHURCH, 1936). There is no general method for determining whether an inference in the predicate calculus is valid.

Let's now see what happens when we focus predicate calculus on the basic structures of grade school and high school mathematics.

From grade school, we have the whole numbers, or integers

$$...,-2,-1,0,1,2,...$$

We can add (+), subtract (-), and compare ($<,\leq,>,\geq,=,\neq$).

EXAMPLES:

$(\forall \text{ integers } x)(\exists \text{ integer } y)(x < y+y)$.

$(\forall \text{ integers } x)(\exists \text{ integer } y)(x = y+y)$.

$(\forall \text{ integers } x,y)(x < y \rightarrow (\exists \text{ integer } z)(x < z \wedge z < y))$.

The first is true, whereas the second and third are false.

LINEAR ARITHMETIC OVER INTEGERS ALGORITHM (M. PRESBURGER, 1929). There is a general method for determining whether a sentence in the predicate calculus based on the integers, $+,-,<,\leq,>,\geq,=,\neq$, is true.

This result has been intensively studied in computer science, where the algorithm is reworked in order to run efficiently on real computers, for a variety of applications.

Also from grade school, we have the fractions, or rational numbers, n/m, where n,m are integers and m ≠ 0.

Again we can add (+), subtract (-), and compare (<,≤,>,≥,=,≠).

EXAMPLES:

(∀ rationals x)(∃ rational y)(x < y+y).

(∀ rationals x)(∃ rational y)(x = y+y).

(∀ rationals x,y)(x < y → (∃ rational z)(x < z ∧ z < y)).

This time, all three sentences are true.

LINEAR ARITHMETIC OVER RATIONALS ALGORITHM (modified PRESBURGER). There is a general method for determining whether a sentence in the predicate calculus based on the rationals, +,-,<,≤,>,≥,=,≠, is true.

Fast forward from grade school to high school. The real numbers "fill out" the rational numbers by incorporating such numbers as "the square root of 2" and π (the circumference of a circle of diameter 1).

We can add, subtract, and compare real numbers.

LINEAR ARITHMETIC OVER REALS. The true sentences of linear arithmetic over the reaLS are the same as the true sentences of linear arithmetic over the rationals.

In fact, we can incorporate integers, rationals, and reals, all mixed together:

LINEAR ARITHMETIC OVER INTEGERS, RATIONALS, REALS, TOGETHER. There is a general method for determining whether a sentence in the predicate calculus based on integers, rationals, reals, +,-,<,≤,>,≥,=,≠, is true.

Recall from grade school and high school that we can also multiply integers, multiply rationals, and multiply reals.

So what happens if we also use multiplication?

ALGORITHM with +,-,•,<,≤,>,≥,=,≠ over INTEGERS? NO!! Kurt Gödel 1931.

ALGORITHM with +,-,•,<,≤,>,≥,=,≠ over RATIONALS? NO!! Julia Robinson 1949.

ALGORITHM with +,-,•,<,≤,>,≥,=,≠ over REALS? YES!! Alfred Tarski 1931, 1948.

This algorithm over the reals continues to be reworked intensively by computer scientists, so that it runs efficiently enough on real computers for many applications.

We now move to a profound foundational problem about mathematics. How do we get a precise handle on what we mean by "correct" or "correctly done" mathematics? In other words, what are the standards for a mathematical proof?

Mathematical proofs are based on axioms and rules of inference for Predicate Calculus. But Predicate Calculus reasoning can only be part of the answer.

Mathematics relies on the ability of mathematicians to "construct" mathematical objects. In other words, we have to construct the integers, the rationals, the reals, and we also have to construct basic operations on them such as addition, subtraction, and multiplication.

We also have to build functions on the reals as is needed for Calculus and other parts of mathematics.

The standard way of unifying the various modes of constructing mathematical objects is through set theory.

Sets arise when multiple objects are conceived of as a single unit.

Sets have elements, and the sole determining feature of a set is just what its elements are.

Thus we have the first axiom of set theory.
The remaining six axioms below are rather powerful set existence (construction) axioms.

ZERMELO SET THEORY (1908)

Extensionality: If two sets have the same elements, then they are equal.

Pairing: Given two sets x,y, there is a set z whose elements are exactly x,y.
Union: Given a set x, there is a set y whose elements are exactly the elements of the elements of x.
Power Set: Given a set x, there is a set y whose elements are exactly the subsets of x.
Separation: Given a set x and a description of a property of sets, there is a set y whose elements are exactly those elements of x which obey that property.
Choice: Given a set x whose elements are each nonempty and have no elements in common with each other, there is a set y which has exactly one element in common with these elements of x.

These easily suffice to develop the main basic infrastructure of contemporary mathematics. In other words, natural numbers, integers, rationals, reals, complexes, addition, subtraction, multiplication, division, order, ordered pairs, functions, metric spaces, topological spaces, measure spaces, continuity, differentiability, analyticity, etcetera.

Once this infrastructure is built, the mathematician is free to sit entirely on top of it, ignoring how this infrastructure was built.

There are two additional axioms needed for the gold standard of the foundations of mathematics – the ZFC axioms. ZFC = Zermelo Fraenkel with the Axiom of Choice.

The first is practically never used:

FOUNDATION. Given a nonempty set x, there is an element y of x which has no elements in common with x.

The second is almost never used, and the known uses of Replacement are particularly interesting:

REPLACEMENT. Given a set x, and a description of a unique assignment of a set to all elements of x, there is a set whose elements are exactly these assignments.

Why do we believe in these fundamental axioms for mathematics? No definitive answer has been given.

One longstanding observation is that all of these axioms,

with the exception of INFINITY, are easily seen to hold in the universe of finite sets – by merely using induction.

This leads to the conjecture that our axioms of set theory arise from transferring the "simple" principles that "obviously hold in the finite sets" to infinite set theory, and then merely adding the axiom of Infinity.

I can report some ongoing major progress on this conjecture in recent weeks. But I will kindly spare you the details of what I am doing with this.

Are these powerful ZFC axioms enough to prove or refute all statements in mathematics?

FIRST INCOMPLETENESS THEOREM (KURT GÖDEL 1931, B. ROSSER 1936). ZFC is not sufficiently strong to prove or refute all statements in integer arithmetic (+,•) – UNLESS(!) the axioms of ZFC are contradictory.

More generally, for any system of finitely many axioms sufficient to develop basic integer arithmetic, there remains statements in integer arithmetic which cannot be proved or refuted – UNLESS(!) the axioms of the system are contradictory.

But are the axioms of ZFC contradictory? We believe that they are not contradictory, but

SECOND INCOMPLETENESS THEOREM (KURT GÖDEL 1931). ZFC is not sufficiently strong to prove that the axioms of ZFC are not contradictory – UNLESS(!) the axioms of ZFC are contradictory.

More generally, any system of finitely many axioms sufficient to develop basic integer arithmetic, is not sufficiently strong to prove that its axioms are not contradictory – UNLESS(!) the axioms of that system are contradictory.

We now change our focus from mathematics and philosophy to computer science and computer assisted education.

Large computer programs are notoriously expensive to create and are notoriously prone to bugs. There are many reasons for this.

This is not going to improve until there is a major overhaul in the programming environments. And it is clear that logic will play a central role in this process.

One crucial missing ingredient is what are called “formal specifications”. These are completely precise specifications of how the program is required to behave.

We don’t have to specify the behavior of the program completely. We need only specify the behavior in situations that we care about.

In any case, formal specifications need to be written in versions of Predicate Calculus tailor made to the intended applications.

Formal specifications are needed in order

i. Support reusability of software.
ii. Support verification.

By verification of software, we mean a mathematical proof that the software obeys the formal specification.

Verification of software goes far beyond the usual software testing in common practice in industry.

We want a programming environment where a team of software developers not only writes code, but also is responsible for verification of the code.

It has long been realized that some serious heavy duty tools are needed in order to enable programmers to feasibly verify the code that they write – even if existing programming languages are overhauled.

Without appropriate tools, it will be considerably harder to verify code than to write it. This is generally not acceptable.

Major components of these badly needed tools are logic based algorithms, or decision procedures. Many of these were first developed by logicians, before they were refined and optimized by computer scientists. For example, algorithms discussed earlier for linear arithmetic over the integers, rationals, and reals, and algorithms with multiplication added, over just the reals.

Let me give a very basic example of this approach.

We want to write a program that, given any nonempty finite sequence of integers, returns the greatest term.

In other words, we want to program a function f:FSQ → INT such that

$(\forall\alpha)(\text{length}(\alpha) > 0 \rightarrow f(\alpha) = \max(\alpha))$.

This is the formal spec. Obviously we need to have some math in the background – in particular, the concept of $\max(\alpha)$. More generally, $\max(\alpha;n,m)$ = maximum value of α from position n through position m, provided $n \leq m$.

Now we write the code.

The programmer sees that they need to code an auxiliary function $g:FSQ \times N \rightarrow Z$ with this formal spec:

$(\forall\alpha,n)(1 \leq n \leq \text{length}(\alpha) \rightarrow g(\alpha,n) = \max(\alpha;1,n))$

The programmer writes code for g in this recursive functional style:

$g(\alpha,1) = \text{Val}(\alpha,1)$.
$g(\alpha,n+1) = \max(g(\alpha,n),\text{Val}(\alpha,n+1))$.

Implementation of Val is at the base of the system.

Code must be written for max of two integers. Probably already written much earlier than this code:

$\max(x,y) = \text{if } x \geq y \text{ then } x \text{ else } y$.

To finish the code, the programmer writes

$f(\alpha) = g(\alpha,\text{length}(\alpha))$.

To recap, the entire program looks like this:

CODE

$\max(x,y) = \text{if } x \geq y \text{ then } x \text{ else } y$.
$g(\alpha,1) = \text{Val}(\alpha,1)$.
$g(\alpha,n+1) = \max(g(\alpha,n),\text{Val}(\alpha,n+1))$.
$f(\alpha) = g(\alpha,\text{length}(\alpha))$.

SPECS

length(α) > 0 → f(α) = max(α).
1 ≤ n ≤ length(α) → g(α,n) = max(α;1,n).

The programmer needs to verify the program. Programmer writes

"prove 1 ≤ n ≤ length(α) → g(α,n) = max(α;1,n) by induction on n ≥ 1".

In this very basic example, note that the above logical formula is just the second spec statement, so that we can expect the system to try this induction on its own, without help from the programmer!

The system comes back with DONE. This is because the system has basic logical manipulations, together with the ability to prove the induction step

g(α,n) = max(α;1,n) → g(α,n+1) = max(α;1,n+1)

using the line of code

g(α,n+1) = max(g(α,n),Val(α,n+1))

and the mathematical fact

max(α;1,n+1) = max(max(α;1,n),Val(α,n+1)).

The verification is completed by the system using the mathematical fact

max(α) = max(α;1,length(α)).

(I left out some inequalities for readability).

This work is still highly exploratory, and it is far from clear just how to design a working system for cost effectively creating verified code for real world programming situations.

We now come to potential applications of logic in education.

The general idea is that through the creative use of logic and logical analysis, we can develop systems that automatically generate multi step homework and exam problems subject to Instructor controlled parameters.

Furthermore, these problems can be automatically graded. The student enters their multi step solution in a special format, and obtains feedback after each step – from the computer.

The entire history of all student responses on all previous problems is stored and this can be used to adjust the future problems generated by the system, subject to control by the Instructor.

Furthermore, all of this data can be used to fine tune the Instructor's Lectures, since the Instructor knows how well the students are doing with the homework and exam problems, down to the last detail, every day.

Ideas like this have of course been around for quite some time. The real challenge is to design such a system, course by course, where the automatically generated multi step problems truly reflect the goals of the course, and the interaction engages the students appropriately.

This requires an array of sophisticated problem templates, reflecting various degrees of difficulty. Logic is of great use here since the problems must have formal structure to support real time computer grading.

A much bigger challenge is to create the appropriate structured modes through which students enter multi step solutions, and receive feedback from the system in real time.

I got engaged with these ideas in connection with a practical need at OSU. All computer science majors are required to take MATH 366, which is a course in basic discrete mathematics and basic proof structure.

We know that the clear majority of students do not leave the course with enough working understanding of proof structure to meet the needs of their majors.

The common view is that there simply is not enough time and resources in a one quarter course, under normal teaching methods, for the students to have nearly enough of the needed quality interactive time to absorb these difficult and subtle skills.

I have time to present a couple of the templates from the Syrus project.

TEMPLATE 1. (∀ integers n,m,r)(x α y ∧ z β w → u γ v).
TRUE OR FALSE? IF TRUE, PROVE. IF FALSE, GIVE COUNTEREXAMPLE.

TEMPLATE 2. (∀ integers n,m)(∃ integer r)(x α y ∧ z β w).
TRUE OR FALSE? IF TRUE, PROVE. IF FALSE, GIVE COUNTEREXAMPLE.

Here x,y,z,w,u,v are among the letters n,m,r, and α,β,γ are among the symbols <,>,≤,≥,=,≠.

TRUE/FALSE is checked in real time. Proofs are entered interactively, with many steps, and real time feedback.

Obviously, it is no surprise that logic would be highly useful in teaching what amounts to basic logic in the context of elementary discrete mathematics.

But we believe that these ideas are strongly applicable to elementary mathematics courses, including K–12.

Some future targets are:

∈-δ proofs.
Courses in the mathematical sciences.
And beyond...

EXAMPLE 1 from SYRUS.

(∀ integers n,m,r)(n < m ∧ r < m → n ≤ r).
True or False? If true, prove. If false, give counterexample.

False.

Counterexample:

n = 1 m = 2, r = 0.
Want ¬(1 < 2 ∧ 0 < 2 → 1 ≤ 0).
More interaction optional.
DONE.

EXAMPLE 2 from SYRUS.

(∀ integers n,m,r)(n < m ∧ ¬(m ≥ r) → n < r).
True or false? If true, prove. If false, give counterexample.

True. Two methods available.

Direct:
Let n,m,r be integers.
Assume n < m ∧ ¬(m ≥ r).
Want n < r.
Have n < m, ¬(m ≥ r). Have m < r. Have n < r.
DONE.

Indirect:
Let n,m,r be integers.
Assume ¬(n < m ∧ ¬(m ≥ r) → n < r). Want contradiction.
Using truth tables, convert to:

Assume n < m, ¬(m ≥ r), ¬(n < r). Derive contradiction.
Have m < r. Have n < r.
DONE.

EXAMPLE 3 from SYRUS.

(∀ integers n,m)(∃ integer r)(r > n ∧ m < r).
True or false? If true, prove. If false, give counterexample.

True.

Let n,m be integers.
Set r = max(n,m)+1.
Want max(n,m)+1 > n ∧ m < max(n,m)+1.
More interaction optional.
DONE.

EXAMPLE 4 from SYRUS.

(∀ integers n,m)(∃ integer r)(n < r ∧ r ≤ m).
True or false? If true, prove. If false, give counterexample.

False.

Counterexample:
n = 0, m = 0.
Assume (∃ integer r)(0 < r ∧ r ≤ 0).

Want contradiction.
Have $0 < r \wedge r \leq 0$.
Have $0 < r$, $r \leq 0$.
DONE.

I hope you've found these Interdisciplinary Adventures of some interest.

Thank you very much.