MAU22C00 Lecture 13

John Stalker

Trinity College Dublin

Axioms for arithmetic

- $\{\forall x.[\neg(x' = 0)]\}$
- $\{\forall x.[(x + 0) = x]\}$
- $(\forall x.\{\forall y.[(x + y') = (x + y)']\})$
- $\{\forall x.[(x \cdot 0) = 0]\}$
- $[\forall x.(\forall y.\{(x \cdot y') = [(x \cdot y) + x]\})]$

The first just says that 0 is not the successor of any natural number.

You can think of the second and third as an inductive (recursive) definition of addition.

You can think of the last two as an inductive definition of multiplication.

Rules of inference for arithmetic

- From a statement of the form $(X = Y)$ we can derive $(Y = X)$.
- From statements of the form $(X = Y)$ and $(Y = Z)$ we can derive $(X = Z)$.
- From a statement of the form $(X = Y)$ we can derive $(X' = Y')$.
- From a statement of the form $(X' = Y')$ we can derive $(X = Y)$.
- Suppose Q is the Boolean expression P with all free occurrences of v replaced by 0 and R is P with all free occurrences of v replaced by v'. From Q and $[\forall v.(P \supset R)]$ we can derive $(\forall v.P)$. The same holds with any other variable in place of v.

The first three are basic facts about equality. Some people include them in first order logic.

The last rule of inference is the principle of mathematical induction.

Formal proof that 2 times 2 equals 4

1. $\{\forall x.[(x \cdot 0) = 0]\}$
2. $[(0'' \cdot 0) = 0]$
3. $[(0'' \cdot 0)' = 0']$
4. $[(0'' \cdot 0)'' = 0'']$
5. $[\forall x.(\forall y.\{(x \cdot y') = [(x \cdot y) + x]\})]$
6. $(\forall y.\{(0'' \cdot y') = [(0'' \cdot y) + 0'']\})$
7. $\{(0'' \cdot 0') = [(0'' \cdot 0) + 0'']\}$
8. $(\forall x.\{\forall y.[(x + y') = (x + y)']\})$
9. $(\forall y.\{[(0'' \cdot 0) + y'] = [(0'' \cdot 0) + y]'\})$
10. $\{[(0'' \cdot 0) + 0''] = [(0'' \cdot 0) + 0']'\})$
11. $\{(0'' \cdot 0') = [(0'' \cdot 0) + 0']'\}$
12. $\{[(0'' \cdot 0) + 0'] = [(0'' \cdot 0) + 0]'\}$
13. $(\forall x.\{[x + 0] = x\})$
14. $\{[(0'' \cdot 0) + 0] = (0'' \cdot 0)\}$
15. $\{[(0'' \cdot 0) + 0] = 0\}$
16. $\{[(0'' \cdot 0) + 0]' = 0'\}$
17. $\{[(0'' \cdot 0) + 0'] = 0'\}$
18. $\{[(0'' \cdot 0) + 0']' = 0''\}$
19. $\{[(0'' \cdot 0) + 0''] = 0''\}$
20. $[(0'' \cdot 0') = 0'']$
21. $\{(0'' \cdot 0'') = [(0'' \cdot 0') + 0'']\}$
22. $(\forall y.\{[(0'' \cdot 0') + y'] = [(0'' \cdot 0') + y]'\})$
23. $\{[(0'' \cdot 0') + 0''] = [(0'' \cdot 0') + 0']'\}$
24. $\{(0'' \cdot 0'') = [(0'' \cdot 0') + 0']'\}$
25. $\{[(0'' \cdot 0') + 0'] = [(0'' \cdot 0') + 0]'\})$
26. $\{[(0'' \cdot 0') + 0] = (0'' \cdot 0')\}$
27. $\{[(0'' \cdot 0') + 0] = 0''\}$
28. $\{[(0'' \cdot 0') + 0]' = 0'''\}$
29. $\{[(0'' \cdot 0') + 0'] = 0'''\}$
30. $\{[(0'' \cdot 0') + 0']' = 0''''\}$
31. $[(0'' \cdot 0'') = 0'''']$

Comments

This system isn't really optimised for proving theorems *within* the system, more for proving theorems *about* the the system.

It would have been nice, for example, to have had the following additional axioms:

- $\{\forall x.[(0 + x) = x]\}$
- $(\forall x.\{\forall y.[(x' + y) = (x + y)']\})$
- $\{\forall x.[(0 \cdot x) = 0]\}$
- $[\forall x.(\forall y.\{(x' \cdot y) = [(x \cdot y) + y]\})]$

Even more useful would have been rules of inference "The expressions $(X + Y)$ and $(Y + X)$ are freely interchangeable, where X and Y are any numerical expressions" and the same with $\cdot$ in place of $+$.

Gödel

Kurt Gödel had the daring idea that this system, which can barely prove $2 \cdot 2 = 4$, might be strong enough to prove it's own inconsistency.

He was nearly right!

We can encode statements about arithmetic in arithmetic. Gödel was (I think) the first person to do this.

Gödel's idea was to craft a statement which asserts that it is false.

We need two ingredients: self reference and a Boolean expression to identify (encodings of) true statements.

The first part is tricky, but can be done.

The ideas are similar to the ones used to produce quines, although Gödel's work predates quines by decades.

Quines

A quine is program which reads no input and produces its own source code as output.

Examples in various languages:

- Javascript:

```
function a() {
  document.write(a, "a()");
}
a()
```

- Bourne shell:

```
#! /bin/sh
q="'" qq='echo \#! /bin/sh;echo q=\"$q\" qq=${q}$qq$q;echo eval \$qq'
eval $qq
```

More quines

- Scheme:

  ((lambda (p) (write (list p (list (quote quote) p))))

  `(quote (lambda (p) (write (list p (list (quote quote) p))))))`

You can find these and hundreds of other examples, in many languages, at http://www.nyx.net/~gthompso/quine.htm

There are people who do these things for fun. Tanaka Tomoyuki (not the creator of Godzilla) even wrote a palindromic quine!

The trick

Here's a description from David Madore:

It is impossible (in most programming languages) for a program to manipulate itself (i.e. its textual representation — or a representation from which its textual representation can be easily derived) directly.

So to make this possible anyway, we write the build the program from two parts, one which call the code and one which we call the data. The data represents (the textual form of) the code, and it is derived in an algorithmic way from it (mostly, by putting quotation marks around it, but sometimes in a slightly more complicated way). The code uses the data to print the code (which is easy because the data represents the code); then it uses the data to print the data (which is possible because the data is obtained by an algorithmic transformation from the code).

We can do something like this in arithmetic with an expression and its encoding.

Arithmeticising truth and provability

To complete Gödel's programme we need a Boolean expression which identifies encodings of true statements.

Unfortunately (?) there's a theorem of Tarski which says the set of encodings of *true* statements is *not arithmetic*.

This is hard. I won't even sketch a proof.

Gödel showed that the set of *provable* statements *is* arithmetic though.

So we can't construct a statement saying "This statement is not true" but we can construct one saying "This statement can't be proved".

The choice to keep the formal system minimal is made to make the proof of Gödel's theorem easier.

The precise statement depends on the details of our formal system but he showed that such a sentence exists in *any* formal system for arithmetic.

An improved version by Rosser says "If x is the encoding of a proof of this statement then there is y such that $y < x$ and y is an encoding of the negation of this statement."

Conclusions

Gödel showed that the set of encodings of provable statements is arithmetic and Tarski showed that the set of encodings of true statements is not.

So there's a true statement which can't be proved or a provable statement which isn't true.

So no formal system for elementary arithmetic is both sound and complete.

This doesn't really depend on the details of the encoding or formal system.

Rosser's argument gives the stronger result that no formal system for elementary arithmetic is both consistent and complete.

Again, this doesn't really depend on the details of the encoding or formal system.

Alternative formulations

These results are usually expressed in terms arithmetic sets.

It's also possible to express them in terms of language categories. The language of provable statements is recursively enumerable. The language of true statements is not. So they can't be the same set.

This formulation eliminates–or really just hides–the encoding.

It's also possible to express them in terms of idealised machines. There is a (non-deterministic) algorithm for proving theorems in a formal system. We can implement this with a Turing machine. It's possible to show that there is no Turing machine which prints all true statements without every printing a false statement.