UNDERGRADUATE TEXTS IN COMPUTER SCIENCE

Beidler, Data Structures and Algorithms

Jalote, An Integrated Approach to Software Engineering, Second Edition

Kozen, Automata and Computability

Merritt and Stix, Migrating from Pascal to C++

Zeigler, Objects and Systems

Dexter C. Kozen

Automata and Computability

Springer

Lecture 38

Gödel's Incompleteness Theorem

In 1931 Kurt Gödel [50, 51] proved a momentous theorem with far-reaching philosophical consequences: he showed that *no* reasonable formal proof system for number theory can prove all true sentences. This result set the logic community on its ear and left Hilbert's formalist program in shambles. This result is widely regarded as one of the greatest intellectual achievements of twentieth-century mathematics.

With our understanding of reductions and r.e. sets, we are in a position to understand this theorem and give a complete proof. It is thus a fitting note on which to end the course.

The Language of Number Theory

The first-order language of number theory L is a formal language for expressing properties of the natural numbers

$$\mathbb{N} = \{0, 1, 2, \ldots\}.$$

The language is built from the following symbols:

- variables $x, y, z, \ldots$ ranging over $\mathbb{N}$;
- operator symbols $+$ (addition) and $\cdot$ (multiplication);

- constant symbols 0 (additive identity) and 1 (multiplicative identity);
- relation symbol $=$ (other relation symbols $<$, $\leq$, $>$, and $\geq$ are definable);
- quantifiers $\forall$ (for all) and $\exists$ (there exists);
- propositional operators $\vee$ (or), $\wedge$ (and), $\neg$ (not), $\rightarrow$ (if-then), and $\leftrightarrow$ (if and only if); and
- parentheses.

Rather than give a formal definition of the well-formed formulas of this language (which we could easily do with a CFG), let's give some examples of formulas and their interpretations.

We can define other comparison relations besides $=$; for example,

$$x \leq y \stackrel{\text{def}}{=} \exists z\ x + z = y,$$
$$x < y \stackrel{\text{def}}{=} \exists z\ x + z = y \wedge \neg(z = 0).$$

Many useful number-theoretic concepts can be formalized in this language. For example:

- "q is the quotient and r the remainder obtained when dividing x by y using integer division":

$$\text{INTDIV}(x, y, q, r) \stackrel{\text{def}}{=} x = qy + r \ \wedge\ r < y$$

- "y divides x":

$$\text{DIV}(y, x) \stackrel{\text{def}}{=} \exists q\ \text{INTDIV}(x, y, q, 0)$$

- "x is even":

$$\text{EVEN}(x) \stackrel{\text{def}}{=} \text{DIV}(2, x)$$

  Here 2 is an abbreviation for 1+1.

- "x is odd":

$$\text{ODD}(x) \stackrel{\text{def}}{=} \neg\text{EVEN}(x)$$

- "x is prime":

$$\text{PRIME}(x) \stackrel{\text{def}}{=} x \geq 2 \ \wedge\ \forall y\ (\text{DIV}(y, x) \rightarrow (y = 1 \vee y = x))$$

- "x is a power of two":

$$\text{POWER}_2(x) \stackrel{\text{def}}{=} \forall y\ (\text{DIV}(y, x) \wedge \text{PRIME}(y)) \rightarrow y = 2$$

- "y is a power of two, say 2^k, and the kth bit of the binary representation of x is 1":

$$\text{BIT}(x,y) \stackrel{\text{def}}{=} \text{POWER}_2(y) \ \wedge \ \forall q \ \forall r \ (\text{INTDIV}(x,y,q,r) \rightarrow \text{ODD}(q))$$

Here is an explanation of the formula $\text{BIT}(x,y)$. Suppose x and y are numbers satisfying $\text{BIT}(x,y)$. Since y is a power of two, its binary representation consists of a 1 followed by a string of zeros. The formula $\text{BIT}(x,y)$ is true precisely when x's bit in the same position as the 1 in y is 1. We get hold of this bit in x by dividing x by y using integer division; the quotient q and remainder r are the binary numbers illustrated. The bit we are interested in is 1 iff q is odd.

$$\begin{array}{rcl} y & = & 1000000000000 \\ x & = & \underbrace{1101100}_{q}\underbrace{1010001011011}_{r} \end{array}$$

This formula is useful for treating numbers as bit strings and indexing into them with other numbers to extract bits. We will use this power below to write formulas that talk about valid computation histories of Turing machines.

If there are no free (unquantified) variables, then the formula is called a *sentence*. Every sentence has a well-defined truth value under its natural interpretation in N. Examples are

$\forall x \ \exists y \ y = x+1$	"Every number has a successor."
$\forall x \ \exists y \ x = y+1$	"Every number has a predecessor."

Of these two sentences, the first is true and the second is false (0 has no predecessor in N).

The set of true sentences in this language is called *(first-order) number theory* and is denoted Th(N). The *decision problem* for number theory is to decide whether a given sentence is true; that is, whether a given sentence is in Th(N).

Peano Arithmetic

The most popular proof system for number theory is called *Peano arithmetic* (PA). This system consists of some basic assumptions called *axioms*, which are asserted to be true, and some *rules of inference*, which can be applied in a mechanical way to derive further theorems from the axioms.

Among the axioms of PA, there are axioms that apply to first-order logic in general and are not particular to number theory, such as axioms for manipulating

- propositional formulas, such as $(\varphi \wedge \psi) \rightarrow \varphi$;
- quantifiers, such as $(\forall x \ \varphi(x)) \rightarrow \varphi(17)$; and
- equality, such as $\forall x \ \forall y \ \forall z \ (x = y \wedge y = z \rightarrow x = z)$.

In addition, PA has the following axioms particular to number theory:

$\forall x \ \neg(0 = x+1)$	0 is not a successor
$\forall x \ \forall y \ (x+1 = y+1 \rightarrow x = y)$	successor is one-to-one
$\forall x \ x+0 = x$	0 is an identity for $+$
$\forall x \ \forall y \ x+(y+1) = (x+y)+1$	$+$ is associative
$\forall x \ x \cdot 0 = 0$	0 is an annihilator for $\cdot$
$\forall x \ \forall y \ x \cdot (y+1) = (x \cdot y)+x$	$\cdot$ distributes over $+$
$(\varphi(0) \wedge \forall x \ (\varphi(x) \rightarrow \varphi(x+1))) \rightarrow \forall x \ \varphi(x)$	induction axiom

where $\varphi(x)$ denotes any formula with one free variable x. The last axiom is called the *induction axiom*. It is actually an axiom *scheme* because it represents infinitely many axioms, one for each $\varphi(x)$. It is really the induction principle on N as you know it: in words,

- if φ is true of 0 (basis), and
- if for any x, from the assumption that φ is true of x, it follows that φ is true of $x+1$ (induction step),

then we can conclude that φ is true of all x.

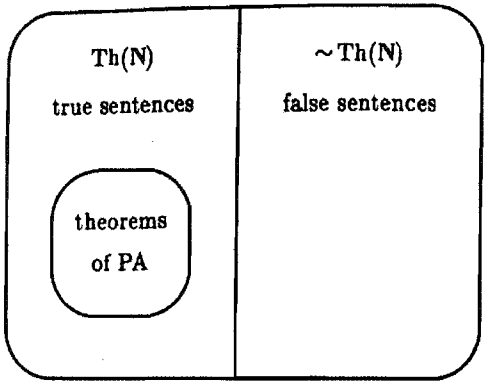
There are also two *rules of inference* for deriving new theorems from old:

$$\frac{\varphi \quad \varphi \rightarrow \psi}{\psi}, \qquad \frac{\varphi}{\forall x \ \varphi}.$$

These two rules are called *modus ponens* and *generalization*, respectively.

A *proof* of φ_n is a sequence $\varphi_0, \varphi_1, \varphi_2, \ldots, \varphi_n$ of formulas such that each φ_i either is an axiom or follows from formulas occurring earlier in the list by a rule of inference. A sentence of the language is a *theorem* of the system if it has a proof.

A proof system is said to be *sound* if all theorems are true; that is, if it is not possible to prove a false sentence. This is a basic requirement of all reasonable proof systems; a proof system wouldn't be much good if its theorems were false. The system PA is sound, as one can show by induction on the length of proofs: all the axioms are true, and any conclusion derived by a rule of inference from true premises is true. Soundness means that the following set inclusions hold:



A proof system is said to be ***complete*** if all true statements are theorems of the system; that is, if the set of theorems coincides with Th(N).

Lecture 39

Proof of the Incompleteness Theorem

Gödel proved the incompleteness theorem by constructing, for any reasonable proof system, a sentence of number theory φ that asserts its own unprovability in that system:

$$\varphi \text{ is true} \iff \varphi \text{ is not provable.} \tag{39.1}$$

Any reasonable proof system, including PA, is sound; this means that for any sentence ψ,

$$\psi \text{ is provable} \Rightarrow \psi \text{ is true} \tag{39.2}$$

(a proof system would not be worth much if some of its theorems were false). Then φ must be true, because otherwise

$$\begin{aligned} \varphi \text{ is false} &\Rightarrow \varphi \text{ is provable} && \text{by (39.1)} \\ &\Rightarrow \varphi \text{ is true} && \text{by (39.2),} \end{aligned}$$

a contradiction. Since φ is true, by (39.1) φ is not provable.

The construction of φ is quite interesting by itself, since it captures in no uncertain terms the notion of self-reference. The power that one needs to construct such a self-referential sentence is present in Turing machines and all modern programming languages. For example, the following is a C program that prints itself:

```
char *s="char *s=%c%s%c;%cmain(){printf(s,34,s,34,10,10);}%c";
main(){printf(s,34,s,34,10,10);}
```

Here 34 and 10 are the ASCII codes for double quote (") and newline, respectively. Although it's a mind-bender to try to figure out what this program does, it's worth the attempt, because once you understand this you have understood the main idea behind Gödel's construction.

We'll construct Gödel's self-referential sentence in Supplementary Lecture K. For now we take a simpler approach that still retains the most important consequences. We will argue that in PA or any other reasonable proof system for number theory,

(i) the set of theorems (provable sentences) is r.e., but

(ii) the set Th(N) of true sentences is not,

therefore the two sets cannot be equal, and the proof system cannot be complete. This approach is due to Turing [120].

The set of theorems of PA is certainly r.e.: one can enumerate the theorems by enumerating all the axioms and systematically applying the rules of inference in all possible ways, emitting every sentence that is ever derived. This is true for any reasonable proof system.

The crux then is to show:

Lemma 39.1 *Th*(N) *is not r.e.*

Proof. We prove this by a reduction $\sim\text{HP} \leq_m \text{Th(N)}$. The result will then follow from Theorem 33.3(i) and the fact that $\sim$HP is not r.e. Recall that

$$\text{HP} = \{M\#x \mid M \text{ halts on input } x\}.$$

Given $M\#x$, we show how to produce a sentence γ in the language of number theory such that

$$M\#x \in \sim\text{HP} \iff \gamma \in \text{Th(N)};$$

that is,

$$M \text{ does not halt on } x \iff \gamma \text{ is true.}$$

In other words, given M and x, we want to construct a sentence γ in the language of number theory that says, "M does not halt on x." This will be possible because the language of number theory is strong enough to talk about Turing machines and whether or not they halt.

Recall the formula $\text{BIT}(y,x)$ constructed in Lecture 38, which allows us to think of numbers as bit strings and extract bits from them. Using this as a starting point, we will be able to construct a series of formulas culminating in a formula $\text{VALCOMP}_{M,x}(y)$ that says that y represents a valid computation history of M on input x; that is, y represents a sequence of configurations $\alpha_0, \alpha_1, \ldots, \alpha_N$ of M, encoded over some alphabet Δ, such that

(i) α_0 is the start configuration of M on x,

(ii) α_{i+1} follows from α_i according to the transition function δ of M, and

(iii) α_N is a halt configuration.

These are the same valid computation histories we saw in Lecture 34. Once we have the formula $\text{VALCOMP}_{M,x}(y)$, we can say that M does not halt on x by saying that there does not exist a valid computation history:

$$\gamma \stackrel{\text{def}}{=} \neg\exists y \ \text{VALCOMP}_{M,x}(y).$$

This constitutes a reduction from $\sim$HP to Th(N).

It remains only to provide the gory details of the construction of γ from M and x. Here they are. Assume that configurations of M are encoded over a finite alphabet Δ of size p, where p is prime. Every number has a unique p-ary representation. We use this representation instead of the binary representation for convenience.

Let the symbols of the start configuration of M on $x = a_1 a_2 \cdots a_n$ be encoded by the p-ary digits $k_0, \ldots, k_n$ as shown:

$$\begin{array}{cccccccc} \vdash & a_1 & a_2 & a_3 & a_4 & \cdots & a_n \\ s & - & - & - & - & \cdots & - \\ k_0 & k_1 & k_2 & k_3 & k_4 & \cdots & k_n \end{array}$$

Let the blank symbol $\sqcup$ be encoded by the p-ary digit k.

Let C be the set of all sextuples (a,b,c,d,e,f) of p-ary digits such that if the three elements of Δ represented by a, b, and c occur consecutively in a configuration α_i, and if d, e, and f occur in the corresponding locations in α_{i+1}, then this would be consistent with the transition function δ. For example, if

$$\delta(q,a) = (p,b,R),$$

then the sextuple

$$\begin{array}{ccc} a & a & b \\ - & q & - \end{array} \qquad \begin{array}{ccc} a & b & b \\ - & - & p \end{array}$$

would be in C.

Now it's time to define some formulas.

- "The number y is a power of p." Here p is a fixed prime that depends on M.

$$\text{POWER}_p(y) \stackrel{\text{def}}{=} \forall z\ (\text{DIV}(z,y) \wedge \text{PRIME}(z) \rightarrow z = p)$$

- "The number d is a power of p and specifies the length of v as a string over Δ."

$$\text{LENGTH}(v,d) \stackrel{\text{def}}{=} \text{POWER}_p(d)\ \wedge\ v < d$$

- "The p-ary digit of v at position y is b" (assuming y is a power of p).

$$\text{DIGIT}(v,y,b) \stackrel{\text{def}}{=} \exists u\ \exists a\ (v = a + by + upy\ \wedge\ a < y\ \wedge\ b < p)$$

- "The three p-ary digits of v at position y are b, c, and d" (assuming y is a power of p).

$$\begin{aligned}\text{3DIGIT}(v,y,b,c,d) \stackrel{\text{def}}{=} \exists u\ \exists a\ (v = a + by + cpy + dppy + uppppy \\ \wedge\ a < y\ \wedge\ b < p\ \wedge\ c < p\ \wedge\ d < p)\end{aligned}$$

- "The three p-ary digits of v at position y match the three p-ary digits of v at z according to δ" (assuming y and z are powers of p).

$$\begin{aligned}&\text{MATCH}(v,y,z)\\ &\stackrel{\text{def}}{=} \bigvee_{(a,b,c,d,e,f)\in C} \text{3DIGIT}(v,y,a,b,c)\ \wedge\ \text{3DIGIT}(v,z,d,e,f)\end{aligned}$$

- "The string v represents a string of successive configurations of M of length c up to d" = "All pairs of three-digit sequences exactly c apart in v match according to δ" (assuming c and d are powers of p).

$$\text{MOVE}(v,c,d) \stackrel{\text{def}}{=} \forall y\ (\text{POWER}_p(y)\ \wedge\ yppc < d) \rightarrow \text{MATCH}(v,y,yc)$$

- "The string v starts with the start configuration of M on input $x = a_1a_2\ldots a_n$ padded with blanks out to length c" (assuming c is a power of p; n and p^i, $0 \le i \le n$, are fixed constants depending only on M).

$$\begin{aligned}\text{START}(v,c) \stackrel{\text{def}}{=} &\bigwedge_{i=0}^{n} \text{DIGIT}(v,p^i,k_i)\ \wedge\ p^n < c\\ &\wedge\ \forall y\ (\text{POWER}_p(y)\ \wedge\ p^n < y < c \rightarrow \text{DIGIT}(v,y,k))\end{aligned}$$

- "The string v has a halt state in it somewhere."

$$\text{HALT}(v,d) \stackrel{\text{def}}{=} \exists y\ (\text{POWER}_p(y)\ \wedge\ y < d\ \wedge \bigvee_{a\in H} \text{DIGIT}(v,y,a))$$

Here H is the set of all p-ary digits corresponding to symbols of Δ containing halt states.

- "The string v is a valid computation history of M on x."

$$\begin{aligned}\text{VALCOMP}_{M,x}(v) \stackrel{\text{def}}{=} \exists c\ \exists d\ (&\text{POWER}_p(c)\ \wedge\ c < d\ \wedge\ \text{LENGTH}(v,d)\\ &\wedge\ \text{START}(v,c)\ \wedge\ \text{MOVE}(v,c,d)\ \wedge\ \text{HALT}(v,d))\end{aligned}$$

- "The machine M does not halt on x."

$$\neg\exists v\ \text{VALCOMP}_{M,x}(v)$$

This concludes the proof of the incompleteness theorem. □