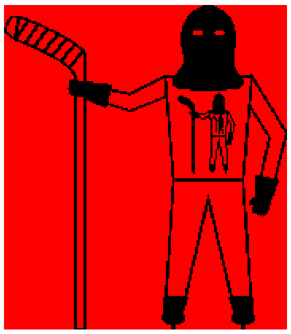
Lecture 25: Gödel and Computability



Halting Problems Hockey Team

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Menu

- Review and finish Gödel's Proof from Monday
- Discuss Quiz
- Computability

Proof – General Idea

- Theorem: In any interesting axiomatic system, there are statements that cannot be proven either true or false.
- Proof: Find such a statement

Gödel's Statement

G: This statement does not have any proof in the system.

Possibilities:

1. *G* is **true** $\Rightarrow$ *G* **has no** proof
   System is *incomplete*
2. *G* is **false** $\Rightarrow$ *G* **has a** proof
   System is *inconsistent*

Finishing The Proof

- Turn *G* into a statement in the *Principia Mathematica* system
- Is *PM* powerful enough to express "This statement does not have any proof in the *PM* system."?

How to express "does not have any proof in the system of *PM*"

- What does "have a proof of *S* in PM" mean?
  - There is a sequence of steps that follow the inference rules that starts with the initial axioms and ends with *S*
- What does it mean to "**not** have **any** proof of *S* in PM"?
  - There is **no** sequence of steps that follow the inference rules that starts with the initial axioms and ends with *S*

Can PM express unprovability?

- There is **no** sequence of steps that follows the inference rules that starts with the initial axioms and ends with S
- Sequence of steps:

  $T_0, T_1, T_2, \ldots, T_N$

  T_0 must be the axioms
  T_N must include S
  Every step must follow from the previous using an inference rule

Can we express "This statement"?

- Yes!
  - That's the point of the TNT Chapter in GEB
- We can write turn every statement into a number, so we can turn "This statement does not have any proof in the system" into a number

Gödel's Proof

G: This statement does not have any proof in the system of *PM*.

If *G* is provable, PM would be inconsistent.
If *G* is unprovable, PM would be incomplete.
PM can express *G*.
Thus, **PM cannot be complete and consistent!**

Generalization

All logical systems of any complexity are incomplete:

there are statements that are *true* that cannot be proven within the system.

Practical Implications

- Mathematicians will ***never*** be completely replaced by computers
  - There are mathematical truths that cannot be determined mechanically
  - We can build a computer that will prove only true theorems about number theory, but if it cannot prove something we do not know that that is not a true theorem.

What does it mean for an axiomatic system to be complete and consistent?

Derives **all** true statements, and **no** false statements starting from a finite number of axioms and following mechanical inference rules.

What does it mean for an axiomatic system to be complete and consistent?

It means the axiomatic system is weak.

Indeed, it is *so* weak, it cannot express: "This statement has no proof."

Pick one:

incomplete

Derives **some, but not all** true statements, and **no** false statements starting from a finite number of axioms and following mechanical inference rules.

Incomplete Axiomatic System

some false statements

Derives **all** true statements, and **some** false statements starting from a finite number of axioms and following mechanical inference rules.

Inconsistent Axiomatic System

Inconsistent Axiomatic System

Derives **all** true statements, and **some** false statements starting from a finite number of axioms and following mechanical inference rules.

some false statements

Once you can prove one false statement, everything can be proven! false ⇒ anything

Quiz Answers

1. b, e (read Tyson's essay)
2. SS0 = "successor of the successor of 0" = 2
3. MU is not a theorem (read Chapter IX)

Results on these questions were quite poor!
Only 6 people got >= 4 points on quiz

Doing these reading may not have a great direct impact on your grade, but they are very interesting and worthwhile.

If that isn't enough motivation, we'll have another quiz some day next week (on the same material).

Surprise Quiz?

Can this be a true statement:

Q: You will have a surprise quiz some day next week.

If the quiz is Friday, it is not a surprise. Q is false.
Since the quiz can't be Friday, if the quiz is not on Monday, it isn't a surprise if it is on Wednesday. Q is false.
Since the quiz can't be Wednesday, if is not a surprise quiz if it is on Monday. Q is false.

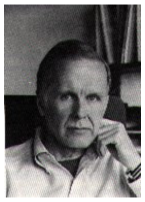
Your quiz score is (max last-quiz next-quiz)

Question 5: Computer Scientists

Alan Turing	16	Computability (rest of today), cryptography (exam 1, ps4), models of computing (later)
Ada Byron	14	First programmer
David Evans	13	Teaches this class
Bill Gates	10	Pancake sorting
Yourself	9	Awesome PS9 project
Grace Hopper	5	First Compiler, COBOL

Others receiving votes: Bach, Euclid, Godel, Escher, Doug Hofstadter, Aaron Bloomfield, Bjarne Stroustrup, Charles Babbage, John Backus (2), Linus Torvalds, Steve Jobs, Steve Wozniak, Alonzo Church, Emil Post, Frances Allen, Gordon Moore, Herman Hollerith, James Cohoon, John Lach, John McCarthy, John von Neumann, Kaspersky, Kinga Dobolyi, Neil de Grasse Tyson, Noam Chomsky, Paul Reynolds, Peter Naur, Richard Stallman, Sid Mayer, Will Wright

John Backus (1924-2007)



- Chemistry major at UVA (entered 1943), flunked out first year
- Joined IBM as programmer in 1950
- Developed Fortran, first commercially successful programming language and compiler
- Invented BNF (replacement grammars)

I flunked out every year. I never studied. I hated studying. I was just goofing around. It had the delightful consequence that every year I went to summer school in New Hampshire where I spent the summer sailing and having a nice time.

John Backus on Simplicity

"Because it takes pages and pages of gobbledygook to describe how a programming language works, it's hard to prove that a given program actually does what it's supposed to. Therefore, programmers must learn not only this enormously complicated language but, to prove their programs will work, they must also learn a highly technical logical system in which to reason about them.

Now, in the kinds of systems I'm trying to build, you can write a program as essentially an equation, like equations in high school algebra, and the solution of that equation will be the program you want... The entire language can be described in one page. But there's a catch: They're what I call applicative (functional) language, which means there's no concept of a stored memory at all. ..."

FL Programming

```
def f ≡ iszero → ~0
                ; + o [f o dec, id]
```

e1 → e2 ; e3	(if e1 e2 e3)
~0	(lambda (x) 0)
o	composition

No need to name parameters! All expressions are applied to whatever the function is applied to.

Question 4

- What is computer science?
- Answer for this course: study of information processes
  - How to describe them precisely (procedures)
  - How to predict their properties (analysis, so far mostly running time)
  - What **problems** can be solved by different types of procedures (algorithms, polynomial)

↑ finish eventually (now) ↑ finish in a reasonable amount of time (later)

Computer Science

- Another common definition (Knuth's):

"The study of algorithms"

Algorithms

- What's an algorithm?

  A procedure that always terminates.
- What's a procedure?

  A precise (mechanizable) description of a process.

Computability

- Is there an algorithm that solves a problem?
- Computable (decidable) problems:
  - There is an algorithm that solves the problem.
  - Make a photomosaic, sorting, drug discovery, winning chess (it doesn't mean we know the algorithm, but there is one)
- Uncomputable (undecidable) problems:
  - There is no algorithm that solves the problem.

    There might be a procedure, but it doesn't always terminate.

Are there any uncomputable problems?

The Hapting Problem

Input: a specification of a procedure *P*

Output: If evaluating an application of *P* halts, output true. Otherwise, output false.

Alan Turing (1912-1954)

- Codebreaker at Bletchley Park
  - Broke Enigma Cipher
  - Perhaps more important than Lorenz
- Published *On Computable Numbers …* (1936)
  - Introduced the Halting Problem
  - Formal model of computation

    (now known as "Turing Machine")
- After the war: convicted of homosexuality (then a crime in Britain), committed suicide eating cyanide apple

5 years after Gödel's proof!

Halting Problem

Define a procedure halts? that takes a procedure specification and evaluates to #t if evaluating an application of the procedure would terminate, and to #f if evaluating an application of the would not terminate.

```
(define (halts? proc) … )
```

Examples

```
> (halts? '(lambda () (+ 3 3)))
#t
> (halts? '(lambda ()
              (define (f) (f))
               (f)))
#f
```

Halting Examples

```
> (halts? `(lambda ()
              (define (fact n)
                 (if (= n 1) 1 (* n (fact (- n 1)))))
              (fact 7)))
#t
> (halts? `(lambda () (fact 0)))
#f
> (halts? `(lambda ()
             (define (fibo n)
                (if (or (= n 1) (- n 2))) 1
                   (+ (fibo (- n 1)) (fibo (- n 2)))))
             (fibo 100))
#t
```

Can we define halts? ?

- We could try for a really long time, get something to work for simple examples, but could we solve the problem – make it work for all possible inputs?

Informal Proof

```
(define (paradox)
  (if (halts? paradox)
      (loop-forever)
      #t))
```

If paradox halts, the if test is true and it evaluates to (loop-forever) - it doesn't halt!

If paradox doesn't halt, the if test if false, and it evaluates to #t. It halts!

Charge

- Friday: Other uncomputable problems: now we have one uncomputable problem, how do we decide if a new problem is uncomputable
  - Why virus scanners will never work perfectly
- PS6: will be accepted Monday without penalty or extension required