Undecidability

Informatics 2A: Lecture 30

Alex Simpson

School of Informatics
University of Edinburgh
als@inf.ed.ac.uk

27 November, 2012

Prelude: Russell's paradox (1901)

Define R to be the set of all sets that don't contain themselves:

$$R = \{S \mid S \notin S\}$$

Does R contain itself, i.e. is $R \in R$?

Russell's analogy: The village barber shaves exactly those men in the village who don't shave themselves. Does the barber shave himself, or not?

Turing machines: summary

- Assuming $|\Sigma| \geq 2$, any kind of 'finite data' can (in principle) be coded up as a string in Σ^*, which can then be written onto a Turing machine tape. (E.g. natural numbers could be written in binary, or in decimal if Σ contains the digits $0, \ldots, 9$.)
- According to the Church-Turing thesis, any 'mechanical computation' that can be performed on finite data can be performed in principle by a Turing machine.
- Any decent programming language has the same 'computational power in principle' as a Turing machine. (E.g. Micro-Haskell is Turing complete.)

Universal Turing machines

Think about Turing machines with input alphabet Σ.

Such a machine T is itself specified by a finite amount of information, so can in principle be 'coded up' by a string $\overline{T} \in \Sigma^*$. (Details don't matter).

So one can imagine a universal Turing machine U which:

- Takes as its input a coded description $\overline{T}$ of some TM T, along with an input string s, separated by a blank symbol.
- Simulates the behaviour of T on the input string s. (N.B. a single step of T may require many steps of U!)
  - If T ever halts (i.e. enters final state), U will halt.
  - If T runs forever, U will run forever.

If we believe CTT, such a U must exist — but in any case, it's possible to construct one explicitly.

The concept of a general-purpose computer

Alan Turing's discovery of the existence of a universal Turing machine (1936) was in some sense the fundamental insight that gave us the general-purpose (programmable) computer!

In most areas of life, we have different machines for different jobs. So isn't it remarkable that a single physical machine can be persuaded to perform as many different tasks as a computer can . . . just by feeding it with a cunning sequence of 0's and 1's!

The halting problem

The universal machine U in effect serves as a recognizer for the set

$$\{\overline{T}\_s \mid T \text{ halts on input } s\}$$

But is there also a machine V that recognizes the set

$$\{\overline{T}\_s \mid T \text{ doesn't halt on input } s\} \ ?$$

If there were, then given any T and s, we could run U and V in parallel, and we'd eventually get an answer to the question "does T halt on input s?"

Conversely, if there were a machine that answered this question, we could construct a machine V with the above property.

Theorem: There is no such Turing machine V!
So the halting problem is undecidable.

Proof of undecidability

Why is the halting problem undecidable?

Suppose V existed. Then we could easily make a Turing machine W that recognized the set

$$\{s \in \Sigma^* \mid \text{ the TM coded by } s \text{ runs forever on the input } s\}$$

(W could just write two copies of its input string s, separated by a blank, and thereafter behave as V.)

Now encode W itself as a string $w \in \Sigma^*$. What does W do when given the input w?

- If W accepts w, that means W runs forever on w!
- But if W runs forever on w, then W will accept w!

Contradiction!!! So V can't exist after all!

Decidable vs. semidecidable sets

In general, a set S (e.g. $\subseteq \Sigma^*$) is called decidable if there's a mechanical procedure which, given $s \in \Sigma^*$, will always return a yes/no answer to the question "Is $s \in S$?".
E.g. the set $\{s \mid s \text{ represents a prime number}\}$ is decidable.

We say S is semidecidable if there's a mechanical procedure which will return 'yes' precisely when $s \in S$ (it isn't obliged to return anything if $s \notin S$).

Semidecidable sets coincide with recursively enumerable sets, i.e. those that can be 'listed' by a mechanical procedure left to run forever. Also with recursively enumerable (i.e., Type 0) languages as defined in lectures 28–9

The halting set $\{\overline{T}\,\_\,s \mid T$ halts on input $s\}$ is an example a semidecidable set that isn't decidable. So there exist Type 0 languages for which membership is undecidable.

Undecidable problems in mathematics

The existence of 'mechanically unsolvable' mathematical problems was in itself a major breakthrough in mathematical logic: until about 1930, some people (the influential mathematician David Hilbert, in particular) hoped there might be a single killer algorithm that could solve 'all' mathematical problems!

Once we have one example of an unsolvable problem (the halting problem), we can use it to obtain others — typically by showing "the halting problem can be reduced to problem X."
(If we had a mechanical procedure for solving X, we could use it to solve the halting problem.)

Example: Provability of theorems

Let M be some reasonable (consistent) formal logical system for proving mathematical theorems (something like Peano arithmetic or Zermelo-Fraenkel set theory).

Theorem: The set of theorems provable in M is semidecidable (and hence is a Type 0 language), but not decidable.

Proof: Any reasonable system M will be able to prove all true statements of the form "T halts on input s". So if we could decide M-provability, we could solve the halting problem.

Corollary (Gödel): However strong M is, there are mathematical statements P such that neither P nor $\neg P$ is provable in M.

Proof: Otherwise, given any P we could search through all possible M-proofs until either a proof of P or of $\neg P$ showed up. This would give us an algorithm for deciding M-provability.

Example: Diophantine equations

Suppose we're given a set of simultaneous equations involving polynomials in several variables with integer coefficients. E.g.

$$\begin{aligned} 3xy + 4z + 5wx^2 &= 27 \\ x^2 + y^3 - 9z &= 4 \\ w^5 - z^4 &= 31 \\ x^2 + y^2 + z^2 + w^2 &= 2536427 \end{aligned}$$

Hilbert's 10th Problem (1900): Is there a mechanical procedure for determining whether a set of polynomial equations has an integer solution?

Matiyasevich' Theorem (1970): it is undecidable, whether a set of polynomial equations has an integer solution.

(By contrast, it's decidable whether there's a solution in real numbers!)

Examples from Language Processing itself

(The snake bites its own tail . . .)

- Pretty much all natural problems involving regular languages / DFAs / NFAs are decidable. E.g. "do two DFAs define the same language?": apply the minimization algorithm and see if they're isomorphic.
- This isn't true for context-free languages. E.g. it's even undecidable, given a context-free grammar G with terminals Σ, whether or not $\mathcal{L}(G)$ is the whole of Σ^*.
- It's also undecidable, given CFGs G_1 and G_2, whether $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ is a context-free language.

So undecidability does crop up 'naturally' in many areas of mathematics.

Clicker questions

What is the status of determining whether $\mathcal{L}(G)$ is nonempty ...

Q1: ... when G is a regular grammar?

1. Decidable
2. Semidecidable
3. Not even semidecidable
4. Don't know

Clicker questions

What is the status of determining whether $\mathcal{L}(G)$ is nonempty ...

Q1: ... when G is a regular grammar?

Q2: ... when G is a context-free grammar?

1. Decidable
2. Semidecidable
3. Not even semidecidable
4. Don't know

Clicker questions

What is the status of determining whether $\mathcal{L}(G)$ is nonempty ...

Q1: ... when G is a regular grammar?

Q2: ... when G is a context-free grammar?

Q3: ... when G is a context-sensitive grammar?

1. Decidable
2. Semidecidable
3. Not even semidecidable
4. Don't know

That's all folks!

That concludes the official course syllabus.

On Thursday, John and I will present a joint revision lecture, in which we shall discuss:

- the exam structure
- examinable material
- pointers to UG3 (and upwards) Informatics courses that continue from this one