Limits of Computation

8 - Our first non-computable problem

Bernhard Reus

Last time

- We have defined a self-interpreter in `WHILE` (universal program for `WHILE`).
- It takes as input a `WHILE`-program (as data object = binary tree) and an input value, interprets this program with the given input and writes the result as output.

A non-computable problem

THIS TIME

- we define formally what computability and decidability means (for `WHILE`)
- we consider a decision problem:
  the *Halting Problem*,
  and prove *it is* `WHILE`–undecidable!

Deep Thought

"What is the Ultimate Answer to Life, the Universe, and Everything?"

Problems Revisited

Remember

- we restricted to problems of the form:
  - can we compute a given function of type L-data$\rightarrow L$-data$_\perp$?
  - can we decide membership in a set (i.e. can we compute whether a given element is in a given set – yes or no?)
- we now narrow this down to our chosen notion of computability:

`WHILE` Computability

Definition A partial function $f : \mathbb{D} \to \mathbb{D}_\perp$ is `WHILE`-*computable* if there is a `WHILE`-program p such that $f = [\![p]\!]^{\texttt{WHILE}}$, in other words if f is equal to the semantics of p (we can also say "if p implements f").

Slogan: a `WHILE`-computable function on trees is one that can be implemented in `WHILE`.

partial function $f : \mathbb{D} \to \mathbb{D}_\perp$ so

Notation

$f(\mathtt{d}) = \perp$ means that f is undefined at $\mathtt{d}$

$f(a)\downarrow$ means that f is defined at a

`WHILE` decidability (formally)

Definition A set $A \subseteq \mathbb{D}$ is `WHILE`-decidable if, and only if, there is a `WHILE`-program p such that $[\![p]\!]^{\texttt{WHILE}}(d)\downarrow$ (meaning $[\![p]\!]^{\texttt{WHILE}}(d)$ is defined) for all d in $\mathbb{D}$, and, moreover, $d \in A$ if, and only if, $[\![p]\!]^{\texttt{WHILE}}(d) = \text{true}$.

Slogan: a `WHILE`-decidable set or problem on trees is one for which the membership test can be implemented in `WHILE`.

Our first Non-computable Problem

A decision problem:

Definition 8.3 The *Halting problem*—as set HALT $\subseteq \mathbb{D}$—is defined as follows:

$$\mathrm{HALT} = \{\, [p, d] \in \mathbb{D} \mid \llbracket p \rrbracket^{\mathtt{WHILE}}(d) \downarrow \,\}$$

`WHILE`-program as data

`WHILE`-data

the list (and thus the program) are encoded but we drop the encoding brackets

Big Question:

is HALT `WHILE`-decidable?

About the Halting Problem

- Solving the Halting Problem would be most useful, e.g. a compiler could check for termination of function calls and warn about non-termination like a type checker warns about incompatible types.
- Note that simply interpreting the program on its input does not work: if the interpreter does not terminate, one cannot return the answer ‘no’.

IDEA

Proof of the Undecidability of the Halting Problem

- Assume a `WHILE`-program `h` exists that *DOES* solve the Halting Problem.
- With `h`'s help write a new `WHILE`-program `r`
- establish a contradiction
- so that the assumption that `h` exists must be wrong.

Establishing a contradiction to destroy a robot or computer is a very popular SciFi plot line (a.k.a. Logic Bombs).

The Barber of Seville Paradox

strictly speaking not a "paradox" as the contradiction can be resolved.

http://www.wno.org.uk/4299

The Barber of Seville says:

"In my town Seville, I shave all men who do ***not*** *shave themselves. Those who actually shave themselves, I do not shave."*

This is a version of Bertrand Russell's paradox (Famous Welsh logician, 1872-1970)

The Barber of Seville Paradox

The Barber of Seville says:

"In my town Seville, I shave all men who do ***not*** *shave themselves. Those who actually shave themselves I do not shave."*

http://www.wno.org.uk/4299

Does the barber shave himself?

(note that he lives in Seville and is a man):

No implies he shaves himself

Yes implies he does not shave himself

contradiction

"paradox" can be resolved by saying such a Barber does not exist :-). Does not contradict any laws of nature.

The Barber of Seville Paradox as Diagonalisation

diagonal

man of Seville no

man of Seville no.

shaves	1	2	3	4	5	6	…	3466	3467	…
1	yes	yes	no	yes	no	no		no	no	
2	no	no	yes	no	no	yes		no	yes	
3	no	yes	no	no	no	no		no	yes	
4	yes	yes	no	no	yes	yes		yes	yes	
5	no	no	no	no	no	no		no	no	
6	yes	yes	no	no	no	yes		yes	yes	
…	…	…	…	…	…	…		…	…	
3466	yes	yes	no	no	yes	yes		yes	yes	
3467	no	yes	yes	no	yes	yes		yes	no	
…										
Barber's row	no	yes	yes	yes	yes	no	…	no	yes	…

Barber's row is the negated diagonal. It thus can't be one of the rows of the table, so Barber can't be a male from Seville, but he is!

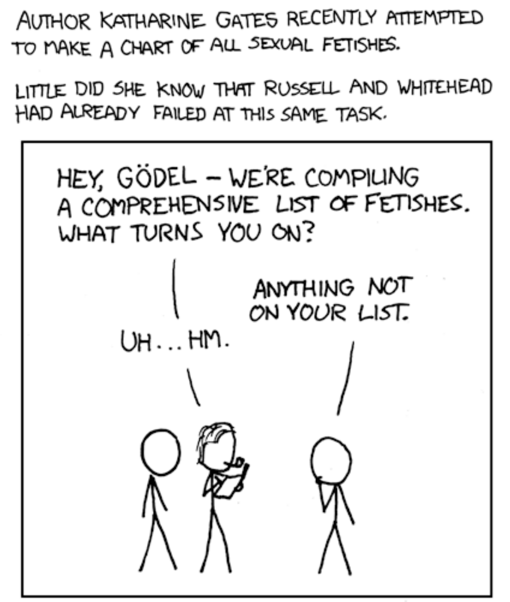
contradiction

What is the table entry (x,y) ?

At row x and column y we put yes if man # x shaves man # y and false otherwise.

More on that theme

The problem is the implicit self-reference.



http://xkcd.com/468/

Proof of HALT's Undecidability

Assume a program deciding the Halting Problem existed:

`h read A {B} write C`

then define `r` as:

```
r read X {
A:= [ X, X ];
B;
Y:= C;
while Y { Y:=Y }
}
write Y
```

r

Does $[\![\mathtt{r}]\!](\mathtt{r})\downarrow$ hold?

in other words: does program r terminate when run with r as input?

$$Y = C = [\![h]\!]^{\mathtt{WHILE}}\,[\,r, r\,]$$

Y = true means *h* says termination but *r* behaves otherwise.
Y = false means *h* says non-termination but *r* behaves otherwise.

```
if ⟦r⟧(r)↓then
r doesn't terminate
else r terminates
```

$\mathtt{if}\ [\![\mathtt{r}]\!](\mathtt{r})\downarrow \mathtt{then}\ [\![\mathtt{r}]\!](\mathtt{r}) = \perp$
$\mathtt{if}\ [\![\mathtt{r}]\!](\mathtt{r}) = \perp \mathtt{then} [\![\mathtt{r}]\!](\mathtt{r})\downarrow$

and derive a **contradiction**.
Then `h` cannot exist.

Proof of HALT's Undecidability

- The proof was using the Barber paradox technique.
- Can we also understand (reformulate) this as a proof by ***diagonalisation***?
- In order to do that, first note that we can enumerate all `WHILE`-programs (like we could enumerate all men in Seville). Why is that?

The Halting Problem as Diagonalisation

- How does `r` behave for arbitrary input programs `X`:
- If `X` run on input `X` terminates, `r` does not terminate.
- If `X` run on input `X` does not terminate, `r` does terminate.
- So `r` behaves a bit like the Barber.

```
r read X {
A:= [X,X];
B;
Y:= C;
while Y {
    Y:=Y
   }
}
write Y
```

`read A {B} write C`
assumed to decide the *Halting Problem*

The Halting Problem as Diagonalisation

`WHILE` program B no

WHILE program A no

does program A terminate when input is program B?	1	2	3	4	5	6	…	3466	3467	…
1	yes	yes	no	yes	no	no		no	no	
2	no	no	yes	no	no	yes		no	yes	
3	no	yes	no	no	no	no		no	yes	
4	yes	yes	no	no	yes	yes		yes	yes	
5	no	no	no	no	no	no		no	no	
6	yes	yes	no	no	no	yes		yes	yes	
…	…	…	…	…	…	…		…	…	
3466	yes	yes	no	no	yes	yes		yes	yes	
3467	no	yes	yes	no	yes	yes		yes	no	
…										
r's row	no	yes	yes	yes	yes	no	…	no	yes	…

diagonal

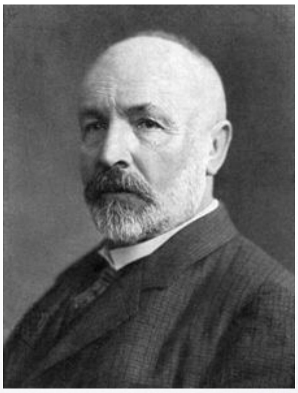
`r`'s row is the negated diagonal. It thus can't be one of the rows of the table, so `r` can't be a `WHILE` program, but it is!

contradiction

"`r` terminates if its argument (program) does not terminate when it is given itself as input; otherwise `r` does not terminate."

Diagonalisation Idea

- ... is very clever
- ... needs items of interest to be enumerable
- ... was discovered by Cantor in 1891 to show the existence of sets that are larger than the set of natural numbers.



Georg Cantor
1845-1918

END

Next time:
More on *semi*-decidability and more undecidable problems