Limits of Computation

9 - More non-computable problems

Bernhard Reus

Last time

- we have seen that the *Halting Problem* for `WHILE`-programs cannot be decided by a `WHILE`-program, but ...
- ... can we say something at least?

More undecidable problems

- we show that *Halting Problem* is *semi-decidable* (and explain what that means)
- next, we generalise the *Halting Problem* and the proof of its undecidability
  - to prove *Rice's Theorem*
- Other undecidable problem:
  - *Tiling Problem*
- and non-computable function:
  - *Busy Beaver*

THIS TIME

`WHILE` Semi-Decidability

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

So the "membership test" may not terminate if the answer is "false", it only works reliably for "half" of the cases, hence the term "semi-decidability".

HALT is `WHILE` semi-decidable

Theorem *The Halting problem (for `WHILE`-programs) HALT is `WHILE`-semi-decidable.*

Proof The "semi-decision procedure" is as follows:

```
sd read PD {
  Res := <u> PD;    (* call  self-interpreter *)
  X := true         (* X is true *)
}
write X
```

u is universal WHILE-program (self-interpreter)

Decidable V Semi-decidable

Theorem *1. Any finite set $A \subseteq \mathbb{D}$ is `WHILE`-decidable.*

\ is "set difference"

2. If $A \subseteq \mathbb{D}$ is `WHILE`-decidable then so is $\mathbb{D} \setminus A$, its complement in $\mathbb{D}$.

read: "then so is its complement"

3. Any `WHILE` decidable set is `WHILE`-semi-decidable.

4. A problem (set) $A \subseteq \mathbb{D}$ is `WHILE`-decidable if, and only if, both A and its complement, $\mathbb{D} \setminus A$, are `WHILE`-semi-decidable.

Proofs as exercise!

Generalising the Halting Problem

- We have shown *(by contradiction)* that the *Halting Problem* cannot be decided by a `WHILE` program.
- The *Halting Problem* is a problem about a property of `WHILE`-programs, namely *whether they terminate (for specific input)*.
- We now *generalise* to all *("interesting")* properties (of a certain kind) of `WHILE`-programs.

"Interesting" Properties of Programs

informally

- "*interesting*" here means:

  non-trivial & *extensional* .

- A *non-trivial* property is one that not all programs have, but that at least one program has.
- An *extensional* program property is one that *depends exclusively* on the input-output behaviour of the program, i.e. its *semantics*.

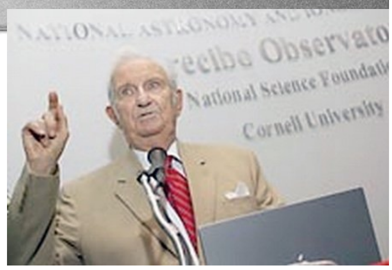
"Interesting" Properties of Programs

formally

Definition A *program property* A is a subset of $\texttt{WHILE}$-programs.
A program property A is *non-trivial* if $\{\} \neq A \neq \texttt{WHILE}$-programs.
A program property is *extensional* if for all $p, q \in \texttt{WHILE}$-programs such that $[\![p]\!]^{\texttt{WHILE}} = [\![q]\!]^{\texttt{WHILE}}$ it holds that $p \in A$ if and only if $q \in A$.

This says that:
if $\mathtt{p}$ has property A and its semantics is the same as that of $\mathtt{q}$, then also $\mathtt{q}$ must have property A.
If $\mathtt{p}$ does not have property A, and its semantics is the same as that of $\mathtt{q}$, then also $\mathtt{q}$ does not have property A.

Rice's Theorem



Henry Gordon Rice (1920–2003)

Rice's Theorem:

If A is an extensional and non-trivial program property then A is undecidable.

Proof by contradiction:

Assume A is decidable,
then show that the *Halting Problem* is decidable.

contradiction

Proof of Rice's Thm.

First we define two programs we need:

```
diverge read X {
  while true {
   }
  }
write Y
```

$[\![diverge]\!]^{\mathtt{WHILE}}(d) = \bot$ for any $d \in \mathbb{D}$.

Now assume A contains *diverge*

if not, swap role of A and its complement

$diverge \in A$

By non-triviality of A we know that there is a program that is not in A.
Let us call this program `comp`.

$comp \notin A$

Proof of Rice's Thm (II)

We wish to decide whether $[\![p]\!]^{\mathtt{WHILE}}(e) \neq \bot$.

e as tree literal

```
q read X {
  Y := <p> "e";      (* run p on value e *)
  Res := <comp> X    (* run comp on input X *)
}
write Res
```

We now consider the behaviour of q and whether it is in A or not. If $[\![p]\!]^{\mathtt{WHILE}}(e) = \bot$ then clearly $[\![q]\!]^{\mathtt{WHILE}}(d) = \bot$ for *all* $d \in \mathbb{D}$. On the other hand, if $[\![p]\!]^{\mathtt{WHILE}}(e)\downarrow$ then $[\![q]\!]^{\mathtt{WHILE}}(d) = [\![comp]\!]^{\mathtt{WHILE}}(d)$ for *all* $d \in \mathbb{D}$. Therefore we get that

$$[\![q]\!]^{\mathtt{WHILE}} = \begin{cases} [\![diverge]\!]^{\mathtt{WHILE}} & \text{if } [\![p]\!]^{\mathtt{WHILE}}(e) = \bot \\ [\![comp]\!]^{\mathtt{WHILE}} & \text{if } [\![p]\!]^{\mathtt{WHILE}}(e) \neq \bot \end{cases}$$

Proof of Rice's Thm (III)

$$[\![q]\!]^{\texttt{WHILE}} = \begin{cases} [\![diverge]\!]^{\texttt{WHILE}} & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) = \bot \\ [\![comp]\!]^{\texttt{WHILE}} & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) \neq \bot \end{cases}$$ implies

$$\begin{array}{rccl} diverge \in A & \text{iff} & q \in A & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) = \bot \\ comp \notin A & \text{iff} & q \notin A & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) \neq \bot \end{array}$$

because A is extensional

Proof of Rice's Thm (III)

$$[\![q]\!]^{\texttt{WHILE}} = \begin{cases} [\![diverge]\!]^{\texttt{WHILE}} & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) = \bot \\ [\![comp]\!]^{\texttt{WHILE}} & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) \neq \bot \end{cases}$$ implies

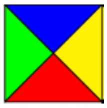
$$\begin{array}{cl} q \in A & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) = \bot \\ q \notin A & \text{if } [\![p]\!]^{\texttt{WHILE}}(e) \neq \bot \end{array}$$

because of our assumptions

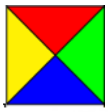
contradiction

So if we can decide A, we can decide HALT

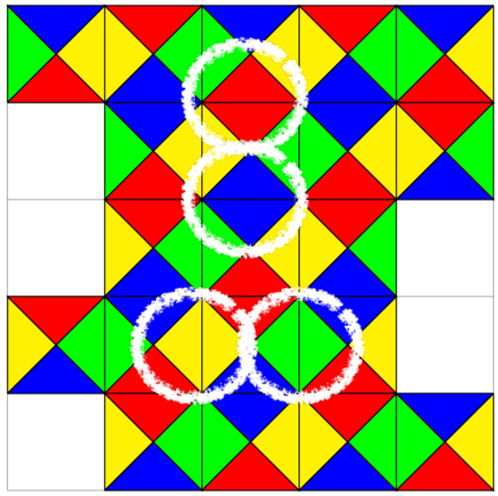
Tiling Problem



1



2

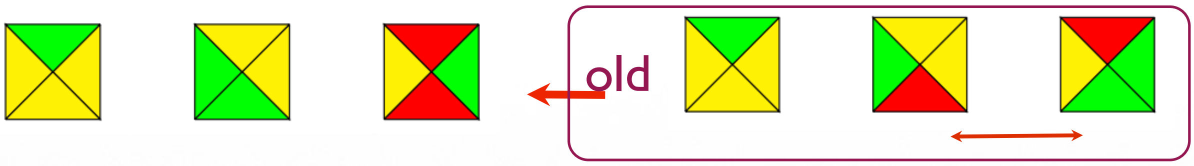


Given an infinite supply of tiles that are of a finite type (here:1,2), can we tile any ***arbitrary large*** quadratic floor (i.e. the plane) so that the patterns of all tiles ***match*** (in all 4 directions)?

you are not allowed to rotate the tiles, rotation requires a new tile type

DECISION PROBLEM

Tiling Problem (cont'd)



- Note that swapping South-facing colours of tile type 2 and 3 as above will mean we cannot even tile a 3x3 square. *(Try it out yourself! Exercises.)*
- The tiling problem for arbitrary finite sets of tile types is ***undecidable***.

Tiling Problem is Hard

- ***1961**: **H. Wang*** presents an algorithm that decides whether any given finite set of tile types can tile the plane. In his proof he assumed that any set that could tile the plane would be able to do so periodically (ie with a repeating pattern like a wallpaper)

- ***1966**: **Robert Berger*** proved ***Wang's conjecture wrong***. He presented a case where the tiles would only tile the plane without repeating pattern, allegedly using 20,426 distinct tile shapes! *"Undecidability of the domino problem"*, Memoirs of the AMS in 1966.

current record is using only 13 tiles

Undecidability of Tiling the Plane

- tiling problem for a *fixed size floor*, $n \times n$, is clearly decidable (check all tilings);
- infinity of the plane alone is not the reason why tiling of the plane is undecidable;
- How to prove undecidability?
  if the *Tiling Problem* was decidable so would be the *Halting Problem* for *Turing Machines* (encode sequence of *Turing Machine* states by tiles which match if they are successor states in a step).

Reduction

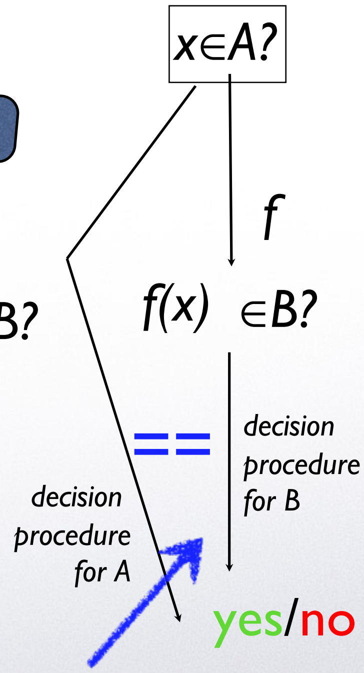
- already several times this principle was used:
  to show that a problem is undecidable (non-computable), argue that if it was decidable (computable) then we could decide (compute) the Halting Problem.
- *Halting Problem* is *"at most as hard"* than these other problems.
- This is called a (computable) problem *reduction*.
- We have seen that the *Halting Problem* can be reduced to any *non-trivial* & *extensional* program *property* or the Tiling problem. Thus those can't be decidable either.

informally

Reduction

from A to B

- Assume $A \subseteq X$ and $B \subseteq Y$.
- The decision problem $x \in A?$ can be reduced to the decision problem $y \in B?$
  …
- …if we find a computable (total) function
  $f\colon\ X \to Y$ *such that*



$x \in A$ "holds exactly if" $f(x) \in B$

iff

Reduction

formally

Definition **(Reduction).** Suppose one is given $A \subseteq X$ and $B \subseteq Y$. Define A to be *effectively reducible* to B if there is a total computable function $f : X \to Y$ such that for all $x \in X$, we have $x \in A$ if, and only if, $f(x) \in B$.
Symbolically we write this relation as $A \leq_{\text{rec}} B$ ("A is effectively reducible to B").

B is at least as hard as A

Theorem *If $A \leq_{\text{rec}} B$ and B is decidable then A is also decidable. Contrapositively, if $A \leq_{\text{rec}} B$ and A is undecidable then B is also undecidable.*

Proof in Exercises.

Other undecidable Problems

- Decision Problem: do languages accepted by two given *context free grammars (CFGs)* overlap?
- Decision Problem: is a CFG ambiguous?
- Decision Problem: does a CFG generate *all* words over a given alphabet?
- Rewriting problem (does one string rewrite into another one using a set of given rewrite rules)
- Type Checking/Inference for functional languages with polymorphic types, where the *type of all types* is *a type itself*, is undecidable.

Haskell avoids this "type of all types"

Joe Wells: "Typability and Type-checking in System F are equivalent and undecidable", Annals of Pure and Applied Logic, 1999.

Other (famous) undecidable Problems in Mathematics

- Given a system of Diophantine (i.e. polynomial) equations with integer coefficients, does it have an integer solution?

proved undecidable by Matiyasevic in 1977

- word problem for groups
- Hilbert's *Entscheidungsproblem* (is a given formula valid in arithmetic?)

There are many others...

Dealing with Undecidability

- use approximation of problem (if appropriate)
- give up on uniformity (restricted input)
- give up on (full) automation
- be content solving a simpler problem

a non-computable function on numbers

Busy Beaver

- *“Can we compute the function BB that for every input number n returns the greatest number that can be outputted by any `WHILE`-program that is (as syntactic string) at most n characters long (when run with input 0)?”*
- T. Radó in his 1962 paper "*On Non-Computable Functions*" (using Turing-Machines where *n* is the number of states)

Tibor Radó
(1895 -1965)

Busy Beaver Research

- In theoretical computer science the *Busy Beaver Problem* for *Turing Machines* has been a challenge some researchers could not keep away from.
- Marxen and Bundtrock *“Attacking the Busy Beaver 5”, Bulletin of the EATCS, No. 40, 1990, pp. 247-251,* used a significant amount of resources to compute *BB(5)* for TMs:

BB(5) Attack

- They used a brute-force simulation technique going through 99.7% of all 88 million 5-state TMs but spending much thought on how to speed up simulation (using “macro-machines” that can simulate several steps in one).
- C program of about 8,000 lines
- it took 10 days to run on a 33MHz CPU
- Result: *BB(5)* **≥ 4098**

END

Next time:
Is there a program that can return “itself”?