# Halting Problem
# Part 2

Adam Wyner

CS3518, Spring 2017

University of Aberdeen

# The Halting Problem

- One of the most important theorems in CS theory
  - A specific problem algorithmically unsolvable
  - A proof that there isn't an algorithm for a problem!
- Computers are very powerful
  - Will every problem be solved with Pentium XI?
  - No: computing is limited in a fundamental way
- Is the following possible? Write a program to:
  1. read any program p plus any input i to p
  2. decide if p terminates for i

# The Russell Paradox

- The proof will use a technique invented by Bertrand Russell in 1901
- The original home of the paradox is axiomatic set theory:
  - a formally precise way of dealing with sets
  - aims to make set theory the foundation of maths
- Various axioms, e.g.
  - the union of two given sets is a set
  - the intersection of two sets is a set, etc.

# The Russell Paradox

- One key axiom. Given a predicate P, the following is also a set: all those elements x such that P(x) is true.
- The resulting theory turns out to be logically inconsistent
  - This means, there exist set theory propositions p such that both p and $\neg p$ follow logically from the axioms of the theory!
  - $\therefore$ The conjunction of the axioms is a contradiction
  - This theory is fundamentally flawed because any possible statement in it can be (very trivially) "proved" by contradiction

# This version of Set Theory is inconsistent

Russell's paradox:

- Consider the set that corresponds with the predicate $x \notin x$ :

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

- Now ask: is $S \in S$?

# Russell's paradox

Let $S = \{x \mid x \notin x\}$. Is $S \in S$?

- If $S \in S$, then S is one of those objects x for which $x \notin x$. In other words, $S \notin S$
- If $S \notin S$, then S is not one of those objects x for which $x \notin x$. In other words, $S \in S$
- We conclude that both $S \in S$ and $S \notin S$

Paradox!

# A playful version of Russell’s paradox

- x is a barber who cuts the hair of exactly those people who do not cut their own hair
- Does x cut his own hair?

# One example of ‘sophisticated’ set theory:

- To avoid inconsistency, set theory had to somehow change. One way is to replace the old axiom by a new one:

  Given a set S and a predicate P, construct a new set S’ consisting of those elements x of S such that P(x) is true.

- We will not worry about the possibility of logical inconsistency, but we’ll use Russell’s trick.

# The Halting problem

One of the first problems to have been shown uncomputable; many other results in computability have been proven as a corollary

# The Halting problem

- Halting is of intrinsic interest. Just like we want to know if a given program can sometimes come up with the wrong answer, we want to know if a program can sometimes come up with no answer.
- Recall the difference between recognising and deciding.
- We focus on a different problem first: the problem of whether a TM accepts a given string.
  - "Yes" implies that the TM halts on this string
  - Problem is similar to, but much harder than $A_{DFA}$ (start of this lecture)
  - $A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$

# The Halting Problem (Cont'd)

- $A_{TM}$ is known as the acceptance problem

  $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input string } w\}$

- Theorem we'll prove: $A_{TM}$ is undecidable
- First: $A_{TM}$ is Turing-recognisable:

> $U$ = On input $\langle M, w\rangle$ where $M$ is a TM and $w$ is a string:
>
> 1. Simulate $M$ on input $w$
> 2. If $M$ enters its accept state, accept;
>    If $M$ enters its reject state, reject

# Universal Turing Machine U

- U simulates another Turing Machine M using the same coding that we used when proving the set of TMs to be countable
- But that encoding "only" encoded
  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$
- A simulation of M has to encode also the way in which U "runs" this encoding of M
- Here is a sketch

# Simulation of a TM on the tape of a TM

- Put the input string w on the tape
- Encode the encoding of M on a tape as shown
- Mark the current state
- Mark the current position of the head.
- All of this is part of the input string to the TM (separate everything using suitable characters)

- So far, sounds familiar, but note the next move to add some aspects that are distinct to TMs.

# Simulation of a TM on the tape of a TM

- Add transition rules to make sure that, e.g.:
  In $q_0$, reading the first symbol w' of w,
  - Find on the tape the (encoded) transition rule that is applicable
    (i.e., the encoding for a rule $<q_0,w',q_1,w'',R>$ or $<q_0,w',q_1,w'',L>$)
  - Replace $q_0$ by $q_1$ as the current state
  - Update the position of the head (R or L, depending on the rule)
- If/when current state = $q_{acc}$ then accept w
- If/when current state = $q_{rej}$ then reject w

# Universal Turing Machine

- TM U is important in its own right
- U is a universal TM
  - Able to simulate any other TM (given its description)
- The first stored-program computer
  - Compare: No need to assemble a PC differently for each Java program we need to run!

# Universal Turing Machine

- TM U loops on ⟨M,w⟩ if M loops on w!
  - That's why it is not a decider

*U* = On input ⟨*M*,*w*⟩ where *M* is a TM and *w* is a string:

1. Simulate *M* on input *w*
2. If *M* enters its accept state, accept;
   If *M* enters its reject state, reject;
   If *M* loops, reject

But is there a TM to check this?

# $A_{TM}$ is undecidable

- Let's now prove the undecidability of

  $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input string } w\}$
- Proof by contradiction:
  - Assume $A_{TM}$ is decidable and obtain a contradiction
- Decidable ➔ there exists a decider H for $A_{TM}$

$$H(\langle M, w\rangle) = \begin{cases} \text{accept if } M \text{ accepts } w \\ \text{reject if } M \text{ does not accept } w \end{cases}$$

Think about what this means: w could be any string! For instance, w might encode a TM

# $A_{TM}$ is undecidable (Cont'd)

We build a new decider D with H as a "subroutine"

– D says: M does not accept $\langle M \rangle$

  $D(\langle M \rangle) = \text{not } H \langle M, \langle M \rangle \rangle$

D = On input $\langle M \rangle$ where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$
2. Output the opposite of what H outputs; that is if H accepts, reject and if H rejects, accept.

# $A_{TM}$ is undecidable (Cont'd)

D is like the barber who cuts the hair of exactly those people who do not cut their own hair!

- This is like running a program with itself as an input
- For instance, a Java compiler written in Java…

# $A_{TM}$ is undecidable (Cont'd)

- In summary

$$D(\langle M \rangle) = \begin{cases} \text{accept if } M \text{ does not accept } \langle M \rangle \\ \text{reject if } M \text{ accepts } \langle M \rangle \end{cases}$$

- What happens if we run D with its own description $\langle D \rangle$ as input? In this case we shall get:

$$D(\langle D \rangle) = \begin{cases} \text{accept if } D \text{ does not accept } \langle D \rangle \\ \text{reject if } D \text{ accepts } \langle D \rangle \end{cases}$$

- This is a contradiction!
  - D cannot exist; therefore H cannot exist

# $A_{TM}$ is undecidable (Cont'd)

- The steps of the proof are:
  - Assume that a TM H decides $A_{TM}$
  - Use H to build a TM D which accepts $\langle M \rangle$ iff M does not accept $\langle M \rangle$
  - Run D on itself
- Acceptance:
  - H accepts $\langle M, w \rangle$ iff M accepts w
  - D accepts $\langle M \rangle$ iff M rejects $\langle M \rangle$
  - D accepts $\langle D \rangle$ iff D rejects $\langle D \rangle$
  - Contradiction!

# $A_{TM}$ is undecidable (Cont'd)

- So far, we have "only" proven that $A_{TM}$ is not decidable....
- We return to the halting problem next week.