COMP90057 Advanced Theoretical Computer Science

Lectures 7-8-9-10-11: Undecidable Problems

Liz Sonenberg & Elena Kelareva

Drawing on material prepared by Harald Søndergaard & Tony Wirth

Lecture 7-8-9-10-11

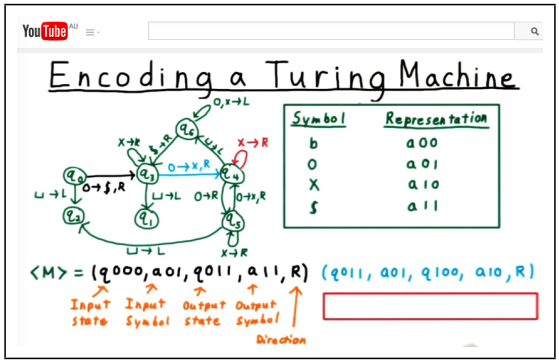
Second Semester, 2017

This block - moving from "what can machines do?" to "what can machines do?"

- Undecidable problems - with proofs
  - a proof that undecidable languages exist - using a counting argument that didn't involve showing a particular language was undecidable (Lecture 6)
  - proof that certain languages are undecidable (proof method - by contradiction)
  - proof that certain languages are undecidable (proof method - by reduction)
  - proof that certain languages are undecidable (proof method - by Rice's Theorem)
- Helpful techniques
  - Encoding machine descriptions as strings (for input to other machines)
  - Diagonalisation

Encoding machine descriptions as strings - i.e. programs as data

In the UTM discussion, Lecture 5, and in last week's Worksheet, you saw (different) ways to encode a TM as a string over a finite alphabet (so it could be an input to another TM):



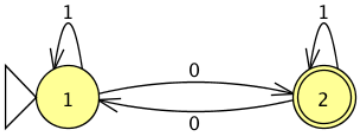
Now, let's do something similar for DFAs...

Encoding DFA descriptions as strings o

Assume w.l.o.g the DFA input alphabet is just 0s and 1s, the start state is labelled 1. We build an encoding using just 0s and 1s:

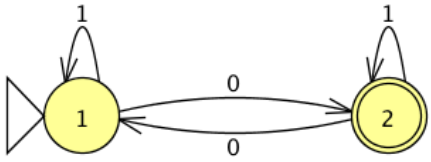
- start and end of encoding: 1111
- end of a category: 111
- end of a subcategory: 11
- change of subcategory: 1
- first substring of 0s represents the number of states
- next category of substrings of 0s say which state(s) are accept state(s)
- transitions are encoded by listing, for each state in turn, the next state on input 0 followed by the next state on input 1

Example: 1111001110011100101101001111 is the encoding of M_1 (denoted $\langle M_1 \rangle$)



Encoding DFA descriptions as strings over {0, 1}

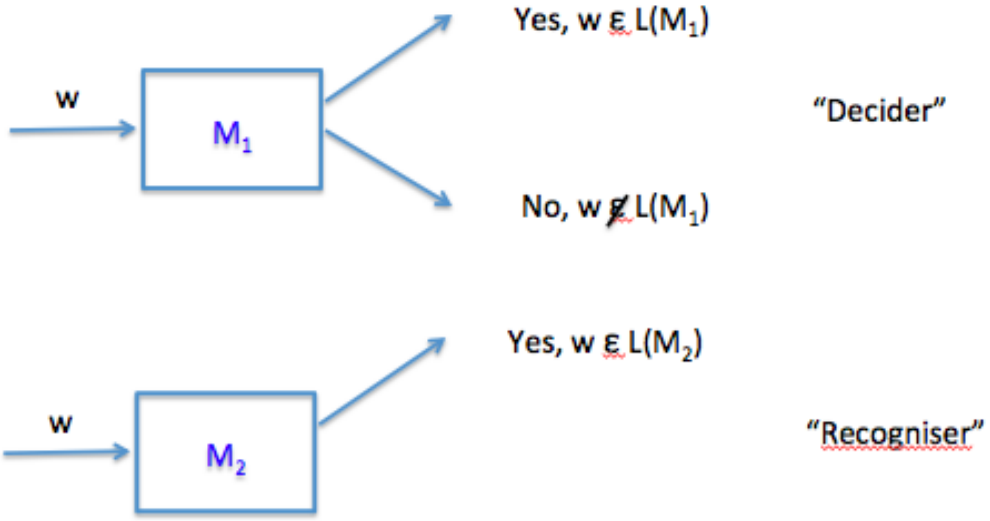
What language does M_1 accept?



What happens if M_1 is given input 111100?

What happens if M_1 is given input 1111001110011100101101001111 i.e. you run M_1 with input $\langle M_1 \rangle$?

Informal short-hand descriptions

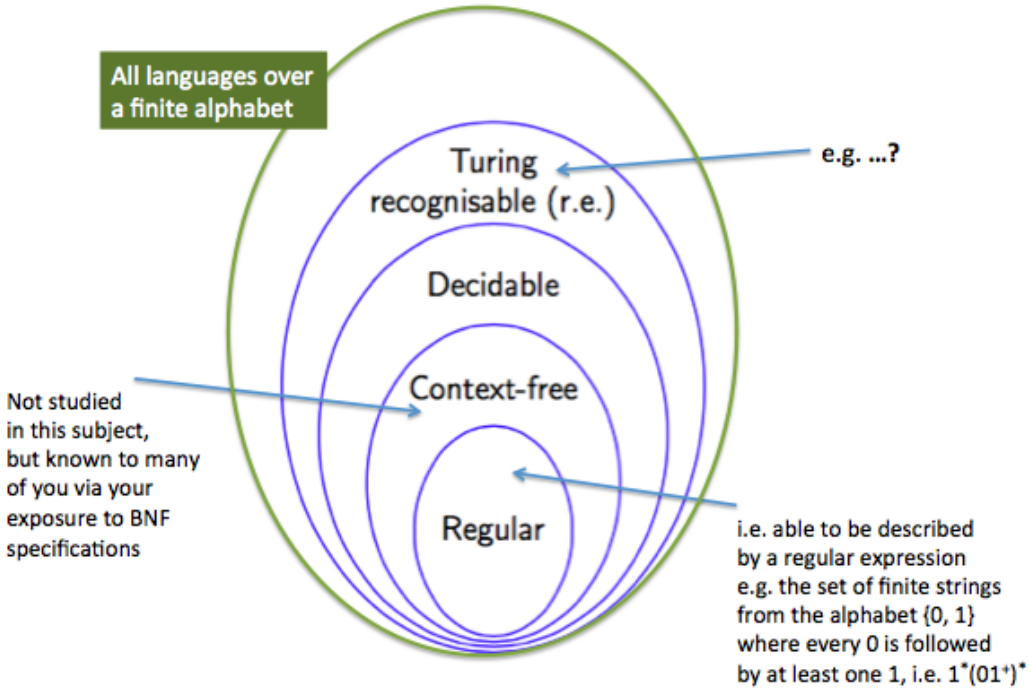


Note: M_2 may correctly say "No" on some inputs, but is not guaranteed to halt on all inputs.

Decision Problems and Languages

- a decision problem is a problem that has a yes/no answer, e.g.
  - Is the language $L(M)$ accepted by Turing machine M, empty?
- representing a decision problem as a language recognition problem?
  - encode the problem P as a string $\langle P \rangle$ in some language Σ and consider whether there is a TM M_P such that $\langle P \rangle \in L(M_P)$ iff P has answer yes
  - e.g.is there a TM M_P such that for any TM M, $\langle M \rangle \in L(M_P)$ iff $L(M)$ is empty
- showing a decision problem is (un)decidable?
  - To show a language is decidable, or to show it is Turing-recognisable, you need to show there is an appropriate TM (respectively a *decider* or a *recogniser*).
  - To show a language is not Turing-recognisable, you have to show that none of the countably infinite TMs are recognisers for that language. *Notice that brute force can't succeed because...*

A Hierarchy of Categories of Languages



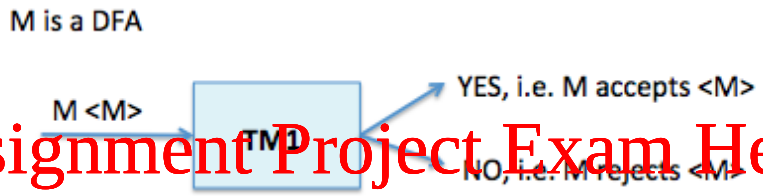
Where on this picture are the languages that are not Turing recognisable?

Decision prob. SA_{DFA}: Given a DFA M, will it accept $\langle M \rangle$?

This is the Deterministic Finite Automaton Self Acceptance decision problem.

Describe an algorithm, that given a DFA will decide (yes/no) whether the DFA accepts its own encoding....

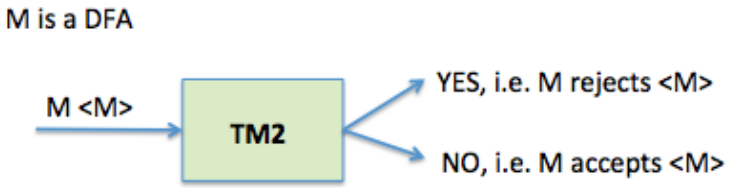
So, by the Church-Turing thesis there is a Turing machine, $TM1$, that is a decider for this problem. i.e. SA_{DFA} is decidable.



Decision prob. SR_{DFA}: Given a DFA ... ct $\langle M \rangle$?

Describe an algorithm, that given a DFA will decide (yes/no) whether the DFA rejects its own encoding....

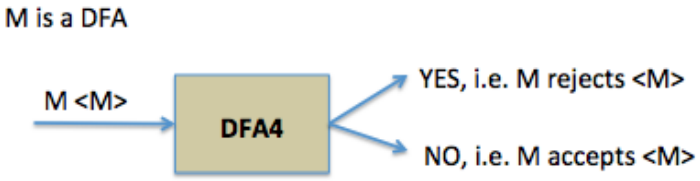
So, by the Church-Turing thesis there is a Turing machine, $TM2$, that is a decider for this problem. i.e. SR_{DFA} is decidable.



Decision prob. SR_{DFA}: Given a DFA M, will it reject $\langle M \rangle$?

We know there is a Turing machine that is a decider for this problem.

Is there a DFA, say *DFA*4, that is a decider for this problem?



Proof there is no such DFA

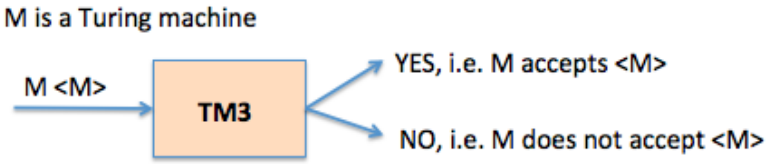
Proof (by contradiction):

- Suppose there is such a DFA, *DFA*4
- Consider what happens when *DFA*4 is run with input $\langle DFA4 \rangle$
- There are two possibilities - either *DFA*4 accepts $\langle DFA4 \rangle$, or *DFA*4 rejects $\langle DFA4 \rangle$
- If *DFA*4 accepts $\langle DFA4 \rangle$ ....complete the argument....
- If *DFA*4 rejects $\langle DFA4 \rangle$ ....complete the argument....
- Therefore ....complete the argument....

Decision prob. SA_{TM}: Given a TM M, will it accept $\langle M \rangle$?

This is the Turing machine Self Acceptance decision problem.

Is there a Turing machine, $TM3$, that that given the encoding $\langle M \rangle$ of any Turing machine, M, will decide (yes/no) whether M accepts its own encoding....



We know SA_{DFA} is decidable. Later, we'll show

$$SA_{TM} = \{\langle M, \langle M \rangle\rangle \mid M \text{ is a TM and } M \text{ accepts } \langle M \rangle\}$$

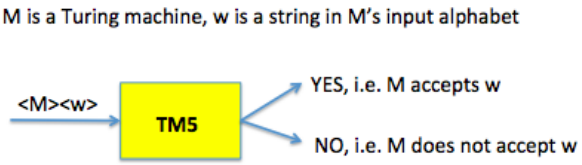
is undecidable.

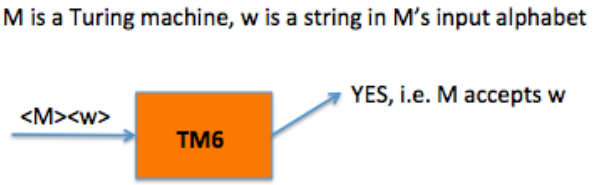
Why might the situation be different for DFAs and TMs?

Decision prob. A_{TM}: Given a TM M and w, will M accept w?

TM Acceptance decision problem

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





If there is such a TM5, A_{TM} would be a decidable problem

If there is such a TM6, A_{TM} would be a Turing recognisable problem

Note: The TM Acceptance decision problem is not the same as the TM Self Acceptance problem.

Sipser, pp 201 to (half way down) 202, and pp 207-210

TM Acceptance is Turing recognisable

Note that

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

is Turing recognisable.

To prove this, take advantage of the existence of a Universal Turing machine U as described earlier in this subject.

On input $\langle M, w\rangle$, U simulates M on input w.

- If M enters its accept state, U accepts.
- If M enters its reject state, U rejects.
- If M never halts, neither does U. (In this context, that's ok.)

So U is a recogniser for A_{TM}

TM Acceptance is undecidable

Theorem (Sipser Theorem 4.11):

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

is undecidable.

Proof: (Proof by contradiction) Suppose that A_{TM} is decided by a TM H:

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

Using H we construct a Turing machine D:

1. Input is $\langle M\rangle$, where M is some Turing machine.
2. Run H on $\langle M, \langle M\rangle\rangle$.
3. If H accepts, reject. If H rejects, accept.

TM Acceptance is undecidable

In summary:

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

But no machine can satisfy that specification!

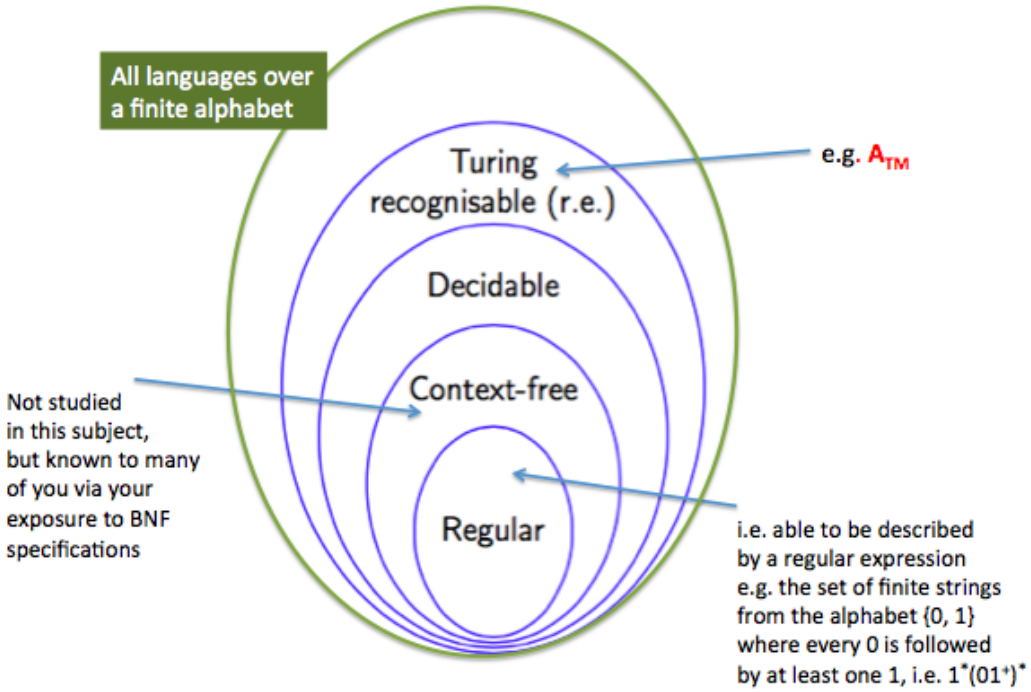
We obtain an absurdity when we investigate D's behaviour on its own encoding:

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

Hence neither D nor H can exist.

Sipser shows very nicely how this proof is really just a use of the technique termed diagonalisation that we saw earlier (in counting arguments).

A Hierarchy of Categories of Languages



Diagonalisation? Sipser - #1 of 3

Where is the diagonalization in the proof of Theorem 4.11? It becomes apparent when you examine tables of behavior for TMs H and D. In these tables we list all TMs down the rows, $M_1, M_2, \ldots$ and all their descriptions across the columns, $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$ The entries tell whether the machine in a given row accepts the input in a given column. The entry is *accept* if the machine accepts the input but is blank if it rejects or loops on that input. We made up the entries in the following figure to illustrate the idea.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$			$\vdots$		

FIGURE 4.19
Entry i, j is accept if M_i accepts $\langle M_j \rangle$

Diagonalisation? Sipser - #2 of 3

In the following figure the entries are the results of running H on inputs corresponding to Figure 4.19. So if M_3 does not accept input $\langle M_2 \rangle$, the entry for row M_3 and column $\langle M_2 \rangle$, is *reject* because H rejects input $\langle M_3, \langle M_2 \rangle \rangle$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

FIGURE 4.20
Entry i, j is the value of H on input $\langle M_i, \langle M_j \rangle \rangle$

Diagonalisation? Sipser - #3 of 3

In the following figure, we added D to Figure 4.20. By our assumption, H is a TM and so is D. Therefore it must occur on the list $M_1, M_2, \ldots$ of all TMs. Note that D computes the opposite of the diagonal entries. The contradiction occurs at the point of the question mark where the entry must be the opposite of itself.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$		$\vdots$			$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$		$\vdots$					$\ddots$

FIGURE 4.21
If D is in the figure, a contradiction occurs at "?"

Diagonalisation?

We saw a similar pattern in an earlier lecture...showing the set of real numbers was not a countable set...

Bigger than countably infinite - a diagonalisation argument

$\mathbb{R}$ denotes the set of all numbers that have decimal representations - generally termed the *real numbers*

Theorem: There is no bijection $h : \mathbb{N} \to \mathbb{R}$. (i.e. "$\mathbb{R}$ is uncountable)

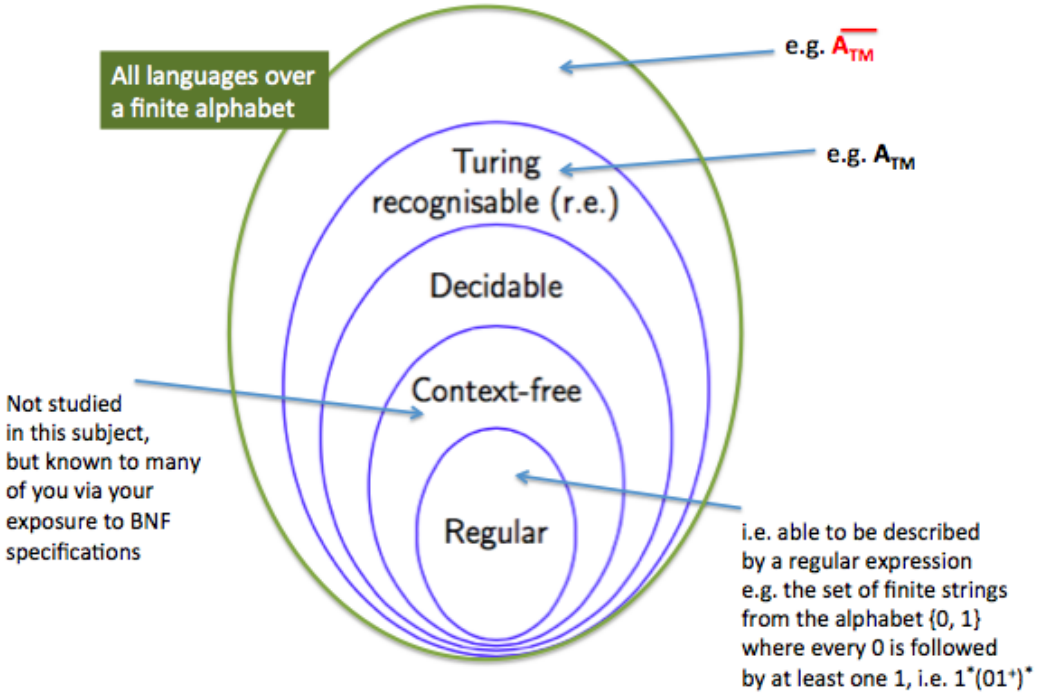
Proof: Assume h exists. Then $h(1), h(2), \ldots, h(n), \ldots$ contains every $r \in \mathbb{R}$, without duplicates. Construct $r' \in (0, 1)$ as follows: Its k'th digit is

'1' if the k'th digit (after the decimal point) of $h(k)$ is not 1
'2' if the k'th digit (after the decimal point) of $h(k)$ is 1

Then $r' \neq h(k)$ for all k, a contradiction.

$h(1)$	14.	4	4	3	7	9	...
$h(2)$	2.	6	1	6	9	3	...
$h(3)$	87.	0	0	0	0	0	...
$h(4)$	7.	7	7	7	7	6	...
$\vdots$							

A Hierarchy of Categories of Languages



A language that is not Turing-recognisa

Theorem (Sipser Theorem 4.22): A language is decidable iff it is Turing-recognisable and its complement is also Turing-recognisable.

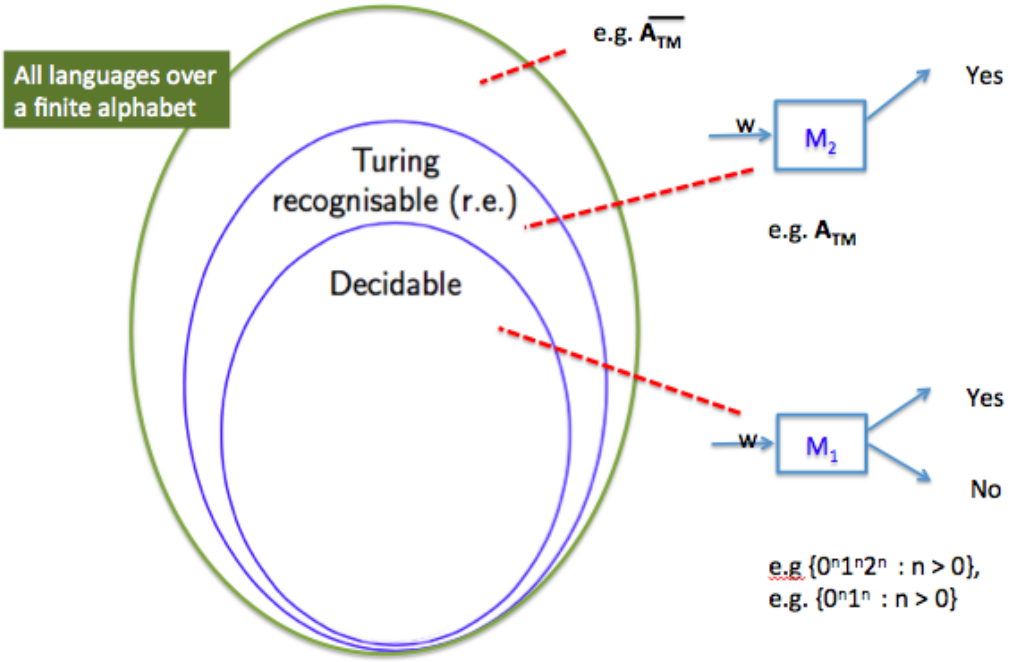
Proof:

A language that is not Turing-recognisable #2 of 2

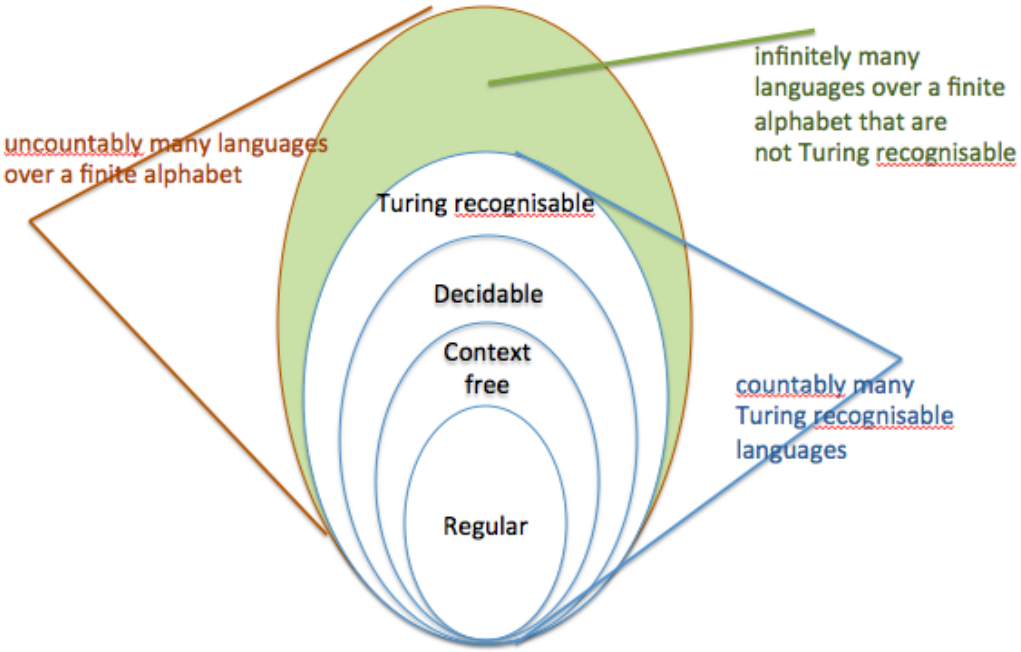
Corollary (Sipser Corollary 4.23): $\overline{A_{TM}}$, i.e. the complement of A_{TM}, is not Turing recognisable.

Proof:

A Hierarchy of Categories of Languages



Infinitely many languages are not Turing-recognisable



Context-Free Grammars in Compute

Backus used them to describe Algol 58 syntax, and they gained popularity with the Algol 60 Report, edited by Naur (1963).

They are frequently referred to as Backus-Naur Formalism (BNF).

Standard tools for parsing owe much to this formalism, which indirectly has helped make parsing a routine task.

It is extensively used to specify syntax of programming languages, document formats (XML's document-type definition), etc.

Derivations, Sentences and Sentential Forms

A grammar is a set of substitution rules, or productions. A simple example is grammar G:

$$\begin{aligned} A &\rightarrow 0\ A\ 1\ 1 \\ A &\rightarrow \epsilon \end{aligned}$$

Using the two rules as a rewrite system, we get derivations such as

$$\begin{aligned} A &\Rightarrow 0A11 \\ &\Rightarrow 00A1111 \\ &\Rightarrow 000A111111 \\ &\Rightarrow 000111111 \end{aligned}$$

A is called a variable. Other symbols (here 0 and 1) are terminals. We refer to a valid string of terminals (such as 000111111) as a sentence. The intermediate strings that mix variables and terminals are sentential forms.

Context-Free Languages

A grammar determines a formal language.

The language of G is written $L(G)$. Example on previous slide has

$$L(G) = \{0^n 1^{2n} \mid n \geq 0\}$$

A language that can be generated by some context-free grammar is a context-free language (CFL).

It should be clear that some of the languages that we found not to be regular are context-free, for example

$$\{0^n 1^n \mid n \geq 1\}$$

Interestingly, some superficially similar languages are not context-free, for example $\{0^n 1^n 2^n \mid n \geq 1\}$, $\{0^n 1^j \mid 0 \leq n \leq j^2\}$, $\{ww \mid w \in \{0,1\}^*\}$

Context-Free Grammars Formally

A context-free grammar (CFG) G is a 4-tuple (V, Σ, R, S), where

1. V is a finite set of variables,
2. Σ is a finite set of terminals,
3. R is a finite set of rules, each consisting of a variable (the left-hand side) and a sentential form (the right-hand side),
4. S is the start variable.

The binary relation $\Rightarrow$ on sentential forms is defined as follows.

Let u, v, and w be sentential forms. Then $uAw \Rightarrow uvw$ iff $A \to v$ is a rule in R. That is, $\Rightarrow$ captures a single derivation step.

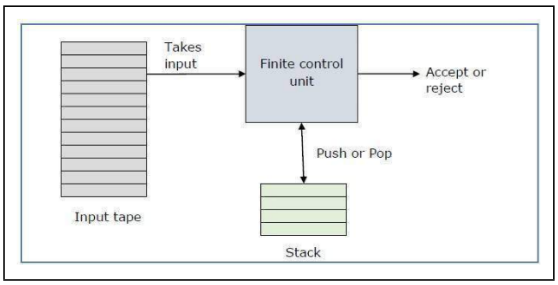
Let $\overset{*}{\Rightarrow}$ be the reflexive transitive closure of $\Rightarrow$. Definition of $L(G)$:

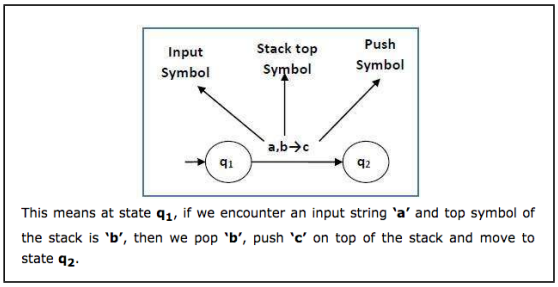
$$L(G) = \{s \in \Sigma^* \mid S \overset{*}{\Rightarrow} s\}$$

Pushdown Automata

Pushdown automata are to context-free grammars what finite-state automata are to regular languages.

Basically a pushdown automaton (PDA) is "Finite state machine" + "a stack". The stack head scans the top symbol of the stack. A stack has two operations: Push - a new symbol is added at the top; Pop - the top symbol is read and removed. In a PDA, the stack is unbounded in size, so a PDA has an unlimited memory - in contrast to a DFA that has only finite memory (via the finitely many states).

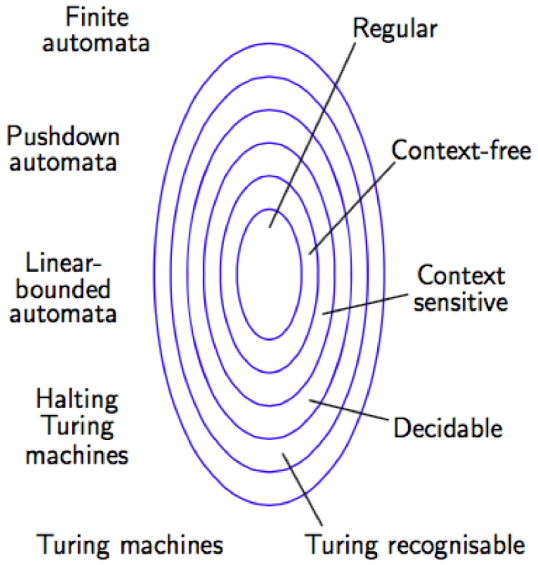




PDA mechanism is not examinable in this subject but FYI is supported in JFLAP.

Recall

Hierarchy of machines and languages



So...

- Whew!!
- Now, back to what Turing Machines can't do...

The Halting Problem

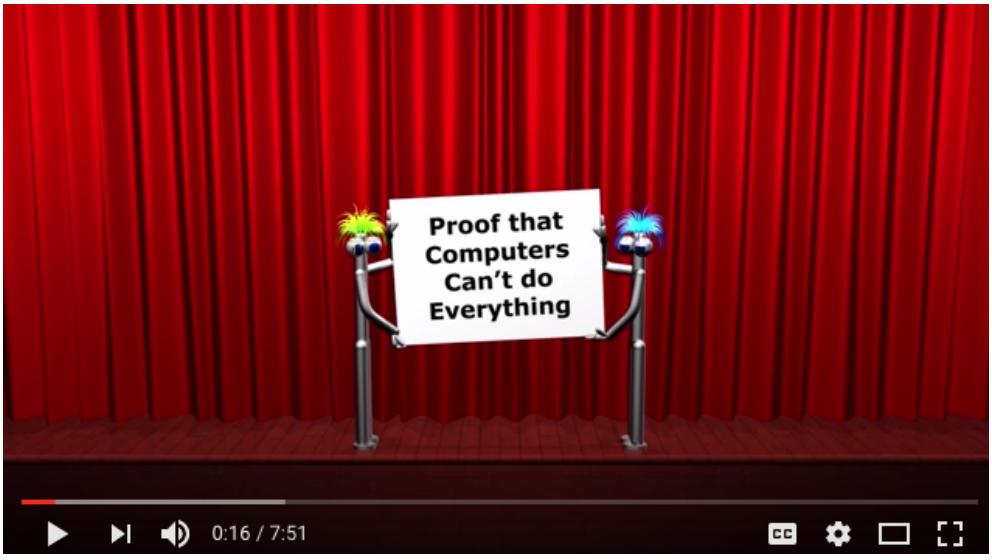
$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Theorem: $HALT_{TM}$ is undecidable.

Proof idea: Suppose $HALT_{TM}$ is decidable. Then there is a Turing machine...

– we will see two ways of proving this - one by a 'direct' argument, the other by a 'reduction' argument –

A (Direct) Proof by Contradiction of the Halting Problem



`https://www.youtube.com/watch?v=92WHN-pAFCs` (8 minutes)

A serious joke

A rendering of the proof by contradiction of the undecidability of the halting problem...in the style of Dr Seuss

`www.lel.ed.ac.uk/~gpullum/loopsnoop.html`
– SCOOPING THE LOOP SNOOPER

`www.youtube.com/watch?v=awjitzVU0Sk`
– Ryan Davis, reading Geoffrey Pullum's poem (3 mins)

Next...

- A very useful technique - reduction (in various forms used for (un)computability and complexity arguments

Reducibility

A reduction is a way of converting one problem to another, in a way that a solution to the second problem can be used to solve the first problem...

When P is reducible to P' (abbreviated $P \leq P'$), solving P cannot be harder than solving P' because a solution to P' yields a solution to P.

In terms of computability theory, when P is reducible to P' (i.e. $P \leq P'$)

- if P' is decidable then P is decidable
- if P is undecidable then P' is undecidable

The second of these points offers a major technique for establishing undecidability – as long as the reduction itself can be done 'computably'.

The general concept of one problem being reducible to another also is used later in the subject in the study of complexity. There are several notions of reducibility - one that is more stringent than what we have used above - mapping reducibility- is used later when discussing algorithmic complexity, as it is a way to measure the difficulty of computing a reduction.

Sipser, chapter 5 section 5.1.

Recall - our first undecidable language:

We showed that it is undecidable whether a Turing machine accepts a given input string.

That is,

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

is undecidable.

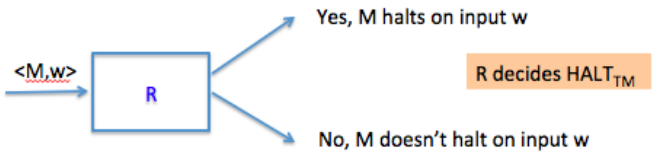
By reducing from A_{TM}, we can now show that other languages are undecidable., i.e. if we show $A_{TM} \leq X$, then we can conclude X is undecidable.

The Halting Problem Is Undecidable - proof by reduction

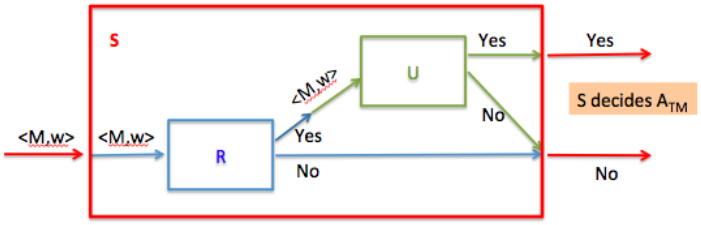
Theorem: $HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ is undecidable.

Proof idea: A_{TM} is reducible to $HALT_{TM}$.

Suppose we have a Turing machine R that decides $HALT_{TM}$



Then we can construct from R and a universal TM U a Turing machine S that decides A_{TM} :



But A_{TM} was undecidable, so $HALT_{TM}$ must also be undecidable.

The Halting Problem Is Undecidable - proof from A_{TM}

Theorem: $HALT_{TM} =$

$$\{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

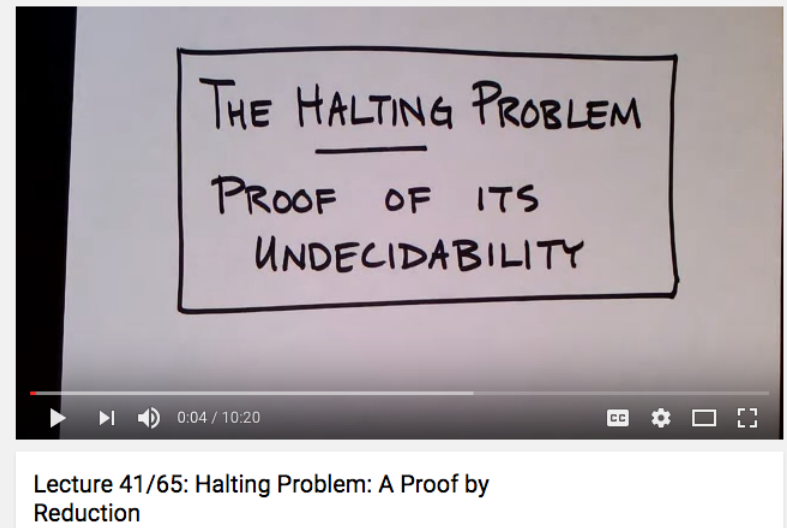
is undecidable.

Proof: A_{TM} is reducible to $HALT_{TM}$. Namely, if we have a Turing machine R that decides $HALT_{TM}$, we can construct a Turing machine that decides A_{TM} as follows:

1. Take input $\langle M, w\rangle$ and run R on this.
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M accepted, accept, otherwise reject.

But A_{TM} was undecidable, so $HALT_{TM}$ must also be undecidable.

The Halting Problem Is Undecidable - proof by reduction from A_{TM}



`https://www.youtube.com/watch?v=dP7fTn0pLvw` (10 minutes)

Harry Porter's Theory of Computation Videos: `http://web.cecs.pdx.edu/~harry/videos/`

TM Emptiness Is Undecidable

Theorem:

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

is undecidable.

Proof: A_{TM} is reducible to E_{TM}.

TM Emptiness Is Undecidable

Theorem:

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

is undecidable.

Proof: A_{TM} is reducible to E_{TM}.

First notice that, given $\langle M, w \rangle$, a Turing machine can create a machine M' that recognises $L(M) \cap \{w\}$.

Here is what the new machine M' does:

1. If input x is not w, reject.
2. Otherwise run M on w and accept if M does.

Notice how w has been "hard-wired" into M': This machine is like M, but it has extra states added to perform the test against w.

TM Emptiness Is Undecidable

Here then is a decider, S, for A_{TM}, using a decider R for E_{TM}:

1. From input $\langle M, w \rangle$ construct $\langle M' \rangle$.
2. Run R on $\langle M' \rangle$.
3. If R rejects, accept; if it accepts, reject.

TM Equality Is Undecidable

Theorem: $EQ_{TM} =$

$$\{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

is undecidable.

Proof: E_{TM} is reducible to EQ_{TM}.

TM Equality Is Undecidable

Theorem: $EQ_{TM} =$

$$\{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

is undecidable.

Proof: E_{TM} is reducible to EQ_{TM}.

Assume that R decides EQ_{TM}. Here is a decider for E_{TM}:

1. Input is $\langle M\rangle$.
2. Construct a Turing machine $M_\emptyset$ that rejects all input.
3. Run R on $\langle\langle M\rangle, \langle M_\emptyset\rangle\rangle$.
4. If R accepts, accept; if it rejects, reject.

But we know that E_{TM} is undecidable. So EQ_{TM} is undecidable.

TM Regularity Is Undecidable

Theorem:

$$REG_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$$

is undecidable.

Proof? Reducibility proof **not** part of this subject *if interested - see Sipser pp 219-220*. Later we will see how to establish this result by applying Rice's Theorem.

Next

- We look at a problem involving string manipulation that is undecidable (the Post Correspondence Problem (PCP))
- read the description of PCP - page 227 of Sipser
- PCP has been a very helpful tool for proving problems in logic and in formal language theory to be undecidable

Post's Correspondence Problem (PCP)

You are given an alphabet Σ and a finite set $\{d_1, \ldots\ d_n\}$ of "dominos" (e.g.)

$$\left\{ \left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right] \right\}$$

formally, a set of n pairs (t_i, b_i) with $t_i, b_i \in \Sigma^+$.

Assuming there is an unbounded supply of each domino, is there a finite sequence of dominos (repetitions permitted) that produces a match: i.e. where the top halves spell the same string as the bottom halves?
In this case, yes:

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right] \left[\frac{\mathtt{b}}{\mathtt{ca}}\right] \left[\frac{\mathtt{ca}}{\mathtt{a}}\right] \left[\frac{\mathtt{a}}{\mathtt{ab}}\right] \left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$$

PCP

How about this case?

$$\left\{ \left[\frac{\mathtt{a}}{\mathtt{cb}}\right], \left[\frac{\mathtt{bc}}{\mathtt{ba}}\right], \left[\frac{\mathtt{c}}{\mathtt{aa}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right] \right\}$$

And this?

$$\left\{ \left[\frac{\mathtt{ab}}{\mathtt{aba}}\right], \left[\frac{\mathtt{bba}}{\mathtt{aa}}\right], \left[\frac{\mathtt{aba}}{\mathtt{bab}}\right] \right\}$$

And this?

$$\left\{ \left[\frac{\mathtt{baa}}{\mathtt{abaaa}}\right], \left[\frac{\mathtt{aaa}}{\mathtt{aa}}\right] \right\}$$

a solver: `https://github.com/dcatteeu/PCPSolver`
some short problems with not-so-short solutions:
`http://webdocs.cs.ualberta.ca/~games/PCP/list.htm`

Deceptively simple

Shortest solution - has length 206

$$\left\{ \left[\frac{1000}{0}\right], \left[\frac{01}{0}\right], \left[\frac{1}{101}\right], \left[\frac{00}{001}\right] \right\}$$

Shortest solution - two with length 75

$$\left\{ \left[\frac{100}{1}\right], \left[\frac{0}{100}\right], \left[\frac{1}{0}\right] \right\}$$

... difficult to bound search while looking to solve a general PCP instance!

PCP

First described by mathematician Emil Post in 1946 as an example of an undecidable problem (under the condition that the alphabet has at least two symbols).

Frequently used in reducibility arguments to prove the undecidability of other problems.

The Bounded PCP (BPCP) is when we have as input not only a set of dominos, but also an integer k and ask for the existence of a solution involving at most k dominos (including repeats). BPCP is trivially decidable, but can be shown to be NP-complete

Write PCP(n) for the problem involving n dominos. It was shown in 1981 that PCP(2) is decidable. In 2006, PCP(2) was shown to have a polynomial time algorithm.

On the other hand, it is known, that PCP(n) is undecidable if $n \geq 5^{\dagger}$

Therefore, the borderline between decidability and undecidability lies somewhere between 3 and 4, when considering the subproblems of PCP defined by the number of available dominos.

† known since 1996 that PCP(n) undecidable for $n \geq 7$, but only established in 2015 that PCP(5) and PCP(6) are undecidable: `http://drops.dagstuhl.de/opus/volltexte/2015/4948/pdf/48.pdf`

PCP Is Undecidable

Theorem: (*Sipser p* 228) *PCP* is undecidable.

There are several ways of proving the Theorem, but the strategy is more or less the same: Reduce the TM acceptance problem or the TM halting problem to the PCP, by encoding sequences of TM configurations as partial solutions of the PCP.

Whatever approach is used, the proof has tedious details, but the idea is simple.

In Sipser, they reduce A_{TM} to *PCP* via computation histories.

That is, for given TM M and input w it is sufficient to construct an instance P of *PCP* such that P has a solution iff M accepts w.

A solution to P will, in effect, simulate the running of M on w.

Using PCP

PCP is often used to establish (by reduction) that various properties of formal languages are undecidable, e.g.

- it is undecidable whether a context free grammar is ambiguous (ie has a string with more than one leftmost derivation)
- for arbitrary context free grammars G_1, G_2, it is undecidable whether $L(G_1) \cap L(G_2)$ is empty
- it is undecidable whether an arbitrary context free language is regular, i.e. $REG_{CFG} = \{\langle G \rangle : G \text{ is a CFG and } L(G) \text{ is regular}\}$ is undecidable

Most will be familiar with context free grammars from other subjects, because of the relationship with programming language definitions; formal definitions are in Chapter 4 of Sipser. CFGs and CFLs are not examinable in this subject.

Next...

- Another **very** useful way of proving undecidability - Rice's Theorem
- statement of Rice's Theorem (Problem 5.28, Sipser pg 241)
- 4 minute introduction `https://www.youtube.com/watch?v=nP2phzkNzwE`

Rice's Theorem - Undecidability is (almost) everywhere

- We've seen many undecidability results for properties of *TM*s, e.g., for:
  - $E_{TM} = \{\langle M \rangle | \ L(M) \in \emptyset)\}$
  - $REGULAR_{TM} = \{\langle M \rangle | \ L(M)$ is a regular language$\}$
  - $A_{TM} = \{\langle M, w \rangle \mid M$ is a TM and $w \in \mathrm{L}(M)\}$
  - $E_{TM} = \{\langle M \rangle \mid M$ is a TM and $L(M) = \emptyset\}$
  - $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2$ are TMs and $L(M_1) = L(M_2)\}$
- These are all properties of the language recognized by the machine.
- Contrast with:
  - $\{\langle M \rangle | \ M$ never tries to move left off the left end of the tape$\}$
  - $\{\langle M \rangle | \ M$ has more than 20 states$\}$
- Rice's theorem says (essentially) that any nontrivial property of the language recognized by a *TM* is undecidable.

Rice's Theorem

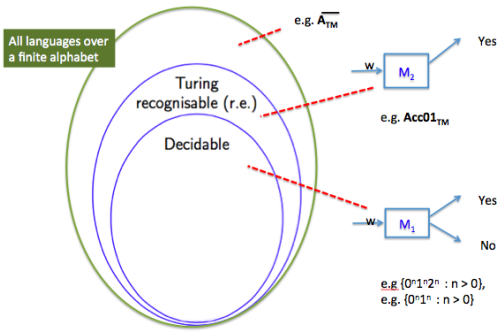
- Define a set P of languages consisting of Turing Machine descriptions, to be a property of Turing-recognisable languages, (abbreviated as "language property") provided that whenever $L(M_1) = L(M_2)$ then either both $\langle M_1 \rangle$ and $\langle M_2 \rangle$ are in P, or neither is in P.
- Define a set P of languages consisting of Turing Machine descriptions, to be a nontrivial property of Turing-recognisable languages, provided that P is a language property **and**
  - There is some Turing-recognisable language L_1 in P, and
  - There is some Turing-recognisable language L_2 not in P.

- Rice's theorem: Let P be a nontrivial property of Turing-recognisable languages. Let $M_P = \{\langle M \rangle | \ L(M) \in P\}$. Then M_P is undecidable.

The proof of Rice's Theorem is not part of this subject.

Rice's Theorem

- Note
  - Applying Rice's Theroem allows to prove undecidability, but doesn't establish non-Turing-recognisability.
  - The sets M_P may be Turing-recognisable.
- Example:
  P = languages that contain 01
  - Then $M_P = \{\langle M \rangle | \ 01 \in L(M)\}$ – let's call this $Acc01_{TM}$
  - Rice tells us that $Acc01_{TM}$ is undecidable.
  - But we know that $Acc01_{TM}$ is Turing-recognisable.
    - For a given input $\langle M \rangle$, a universal TM can simulate M on 01 and accept iff this simulation accepts.



Recall: undecidable simply means not decidable

Applications of Rice's Theorem

- Example 1: Using Rice
  - $\{\langle M \rangle | \ M$ is a TM that accepts at least 37 different strings$\}$
  - Rice implies that this is undecidable.
  - This set $= M_P$, where $P =$ "the language contains at least 37 different strings"
  - P is a language property.
  - Nontrivial, since some TM-recognizable languages satisfy it and some don't.

Applications of Rice's Theorem

- Example 2: *(Property that isn't a language property and is decidable)*
  - $\{\langle M \rangle | \ M$ is a TM that has at least 37 states$\}$
  - Not a language property, but a property of a machine's structure.
  - So Rice doesn't apply.
  - Obviously decidable, since we can determine the number of states given the TM description.

Applications of Rice's Theorem

- Example 3:
  - $\{\langle M \rangle | \ M$ is a TM that runs for at most 37 steps on the input 01$\}$
  - Not a language property, but a property of a machine's structure.
  - So Rice doesn't apply.
  - Obviously decidable, since, given the TM description, we can just simulate it for 37 steps.

Applications of Rice's Theorem

- Example 4: *(Undecidable problem for which Rice's Theorem doesn't work to prove undecidability)*
  - $Acc01SQ = \{\langle M \rangle |\ M$ is a TM that accepts the string 01 in exactly a perfect square number of steps$\}$
  - Not a language property, Rice doesn't apply.
  - Can prove undecidable by showing $Acc01_{TM} \leq_m Acc01SQ$
    - $Acc01_{TM}$ is the set of TMs that accept 01 in any number of steps
    - $Acc01SQ_{TM}$ is the set of TMs that accept 01 in a perfect square number of steps

Applications of Rice's Theorem

- Example 4: continued
  - Design mapping f so that M accepts 01 iff $f(M) = \langle M' \rangle$ where M' accepts 01 in a perfect square number of steps
  - $f(M) = \langle M' \rangle$ where, M': on input x
    - If $x \neq 01$, then reject
    - If $x = 01$, then simulate M on 01. If M accepts 01, then accept, but just after doing enough extra steps to ensure that the total number of steps is a perfect square.
  - $\langle M \rangle \in Acc01_{TM}$ iff M' accepts 01 in a perfect square number of steps, iff $f(\langle M \rangle) \in Acc01SQ$
  - So $Acc01_{TM} \leq_m Acc01SQ$, so $Acc01SQ$ is undecidable.

Applications of Rice's Theorem

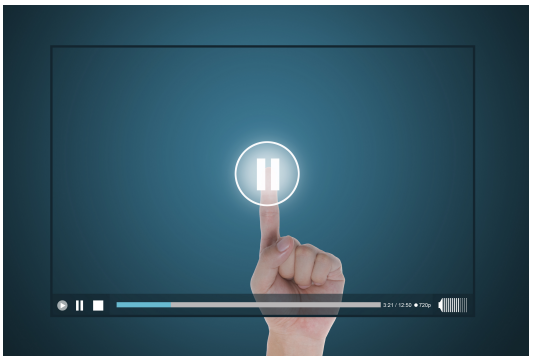
- Example 5:
  - $\{\langle M \rangle |\ M$ is a TM and $L(M)$ is recognized by some TM having an even number of states$\}$
  - This is a language property because....
  - So it might seem that Rice should apply
  - But, it's a trivial language property: Every Turing-recognizable language is recognized by some TM having an even number of states.
    - Could always add an extra, unreachable state.
  - Decidable or undecidable?
  - Decidable (of course), since it's the set of all TMs.

Applications of Rice's Theorem

- Example 6
  - $\{\langle M \rangle |\ M$ is a TM and $L(M)$ is recognized by some TM having at most 37 states and at most 37 tape symbols$\}$
  - A language property because....
  - Is it nontrivial?
    - Yes, some languages satisfy it and some don't.
  - So Rice applies, showing that it's undecidable.
- Note: This isn't $\{\langle M \rangle |\ M$ is a TM that has at most 37 states and at most 37 tape symbols$\}$
  - That's decidable.
- What about: $\{\langle M \rangle |\ M$ is a TM and $L(M)$ is recognized by some TM having at least 37 states and at least 37 tape symbols$\}$
  - Trivial – all Turing-recognizable languages are recognized by some such machine.

Applications of Rice's Theorem

- Example 7
  - $REG_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$
  - A language property because....
  - Nontrivial because...
  - So Rice applies, showing that it's undecidable.



Some of Alan Turing's contributions

www.turing.org.uk/publications/dnb.html
Turing abbreviated biography by *Andrew Hodges*

www.turing.org.uk/scrapbook/machine.html
The Alan Turing Internet Scrapbook: Computable Numbers and the Turing Machine, 1936

www.turing.org.uk/scrapbook/ww2.html
The Alan Turing Internet Scrapbook: Critical Cryptanalysis: The Enigma War, 1939-1942

www.turing.org.uk/scrapbook/manmach.html
The Alan Turing Internet Scrapbook: The World's First Working Universal Turing Machine, 1948-1950

www.turing.org.uk/scrapbook/test.html
The Alan Turing Internet Scrapbook: The Turing Test, 1950

www.charlespetzold.com/blog/2014/12/
The-Imitation-Game-and-Alan-Turings-Real-Contribution-to-Computing.html

Alan Turing's contributions - 1936

Turing and the Entscheidungsproblem - the paper

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

Copy available on the LMS - see 'Subject Resources' - also available from http://www.turingarchive.org/

Adv. Theoretical Computer Science (2016) Lecture 1: Getting started 16 / 35

Alan Turing's contributions, 1939-1945

http://www.rutherfordjournal.org/article030108.html

Enigma

Bombe

The total number of possible ways in which a standard German army-issue **Enigma** machine could be set up was approximately 158 million million million. Many of the settings were changed every 24 hours. Turing and colleagues developed a special purpose electronic machine the **Bombe** to reduce the work of exploring possible encryption keys. Even with this machinery, a deal of manual work was needed. It is widely accepted that the codebreaking work at Bletchley Park in England shortened the length of the war in Europe by around 2 years.

Alan Turing's contributions, 1950

COMPUTING MACHINERY AND INTELLIGENCE

By A. M. Turing

1. The Imitation Game

I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous, If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

The new form of the problem can be described in terms of a game which we call the 'imitation game." It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart front the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either "X is A and Y is B" or "X is B and Y is A." The interrogator is allowed to put questions to A and B thus:

The Turing Test – the original imitation game: www.turing.org.uk/scrapbook/test.html

Computing Machinery and Intelligence (1950), ctd

to light in one of the positions of the wheel. This machine could be described abstractly as follows. The internal state of the machine (which is described by the position of the wheel) may be q_1, q_2 or q_3. There is an input signal i_0 or i_1 (position of lever). The internal state at any moment is determined by the last state and input signal according to the table

		Last State		
		q_1	q_2	q_3
Input	i_0	q_2	q_3	q_1
	i_1	q_1	q_2	q_3

The output signals, the only externally visible indication of the internal state (the light) are described by the table

State	q_1	q_2	q_3
Output	o_0	o_0	o_1

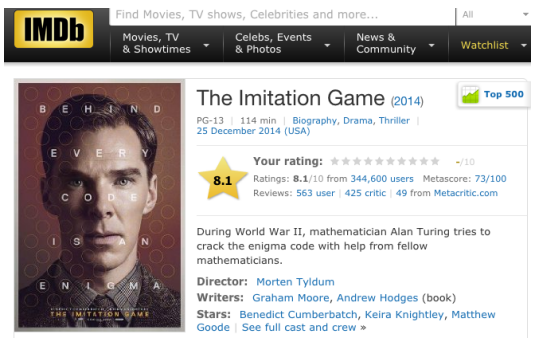
This example is typical of discrete state machines. They can be described by such tables provided they have only a finite number of possible states.

p 440

As we have mentioned, digital computers fall within the class of discrete state machines. But the number of states of which such a machine is capable is usually enormously large. For instance, the number for the machine now working at Manchester it about $2^{165,000}$, *i.e.* about $10^{50,000}$. Compare this with our example of the clicking wheel described above, which had three states. It is not difficult to see why the number of states should be so immense. The computer includes a store corresponding to the paper used by a human computer. It must be possible to write into the store any one of the combinations of symbols which might have been written on the paper. For simplicity suppose that only digits from 0 to 9 are used as symbols. Variations in handwriting are ignored. Suppose the computer is allowed 100 sheets of paper each containing 50 lines each with room for 30 digits. Then the number of states is $10^{100\times50\times30}$, *i.e.* $10^{150,000}$. This is about the number of states of three Manchester machines put together. The logarithm to the base two of the number of states is usually called the 'storage capacity' of the machine. Thus the Manchester machine has a storage capacity of about 165,000 and the wheel machine of our example about 1·6. If two machines are put together their capacities must be added to obtain the capacity of the resultant machine. This leads to the possibility of statements such as 'The Manchester machine contains 64 magnetic tracks each with a capacity of 2560, eight electronic tubes with a capacity of 1280. Miscellaneous storage amounts to about 300 making a total of 174,380.'

p 441

The Imitation Game

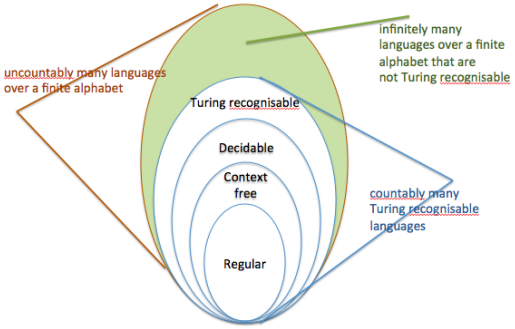


Writers Graham Moore, Andrew Hodges (book), released 2014

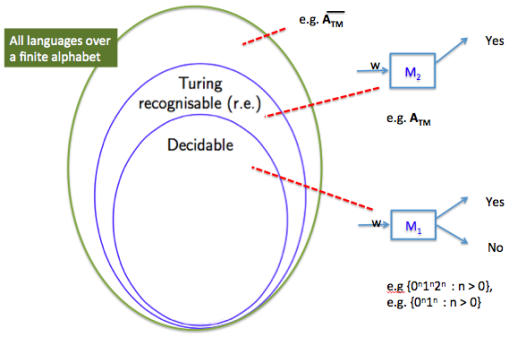
Writer: Ian McEwan, TV play broadcast BBC April 1980

Alan Turing: A multitude of lives in fiction, Graham Moore, 2012
www.bbc.com/news/technology-18472563
The Imitation Game: is it history, drama or myth?
theconversation.com/the-imitation-game-is-it-history-drama-or-myth-35849
www.charlespetzold.com/blog/2015/01/Pushback-on-The-Imitation-Game.html –some of the historical inaccuracies of the movie *The Imitation Game* - a title borrowed from Turing's 1950 paper.

In summary...



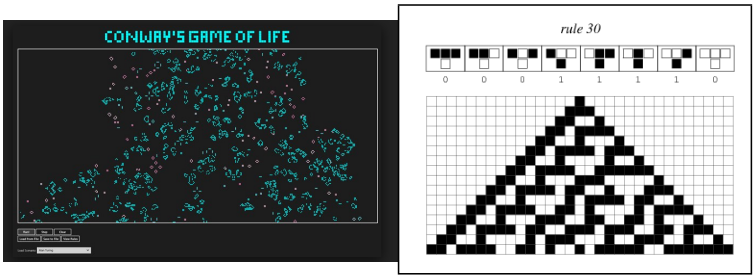
From Lectures 7-11, slide 27



From Lectures 7-11, slide 26

In summary... 2 of 4

Turing, Gödel, Church



Conway, Wolfram

(also Minsky's Counter Machines...)

In summary... 3 of 4

Key concepts & techniques - Phil Wadler, University of Edinburgh

`https://www.youtube.com/watch?v=GnpcMCWORUA` (8 mins)

A brief introduction to the hilarious subject of computability theory. Performed as part of Bright Club at The Stand in Edinburgh on Tuesday 28 April 2015.

In summary... 4 of 4

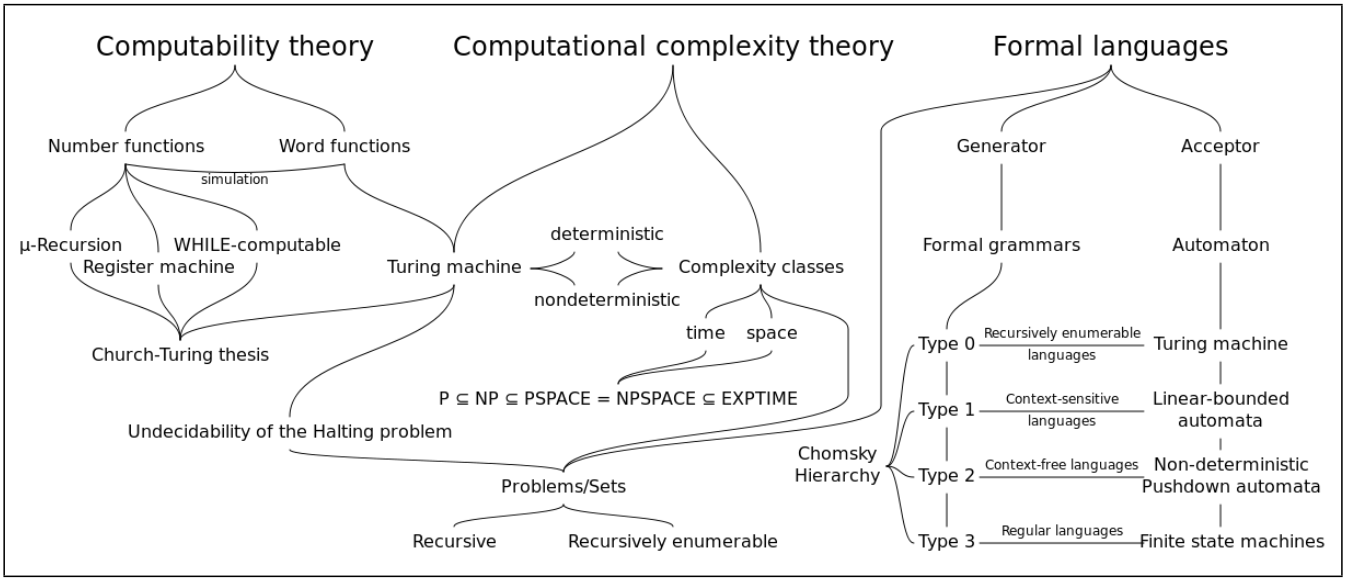


image source: `https://upload.wikimedia.org/wikipedia/en/thumb/6/64/Theoretical_computer_science.svg/1200px-Theoretical_computer_science.svg.png`
GNU Free Document License