Decidable and Semi-decidable

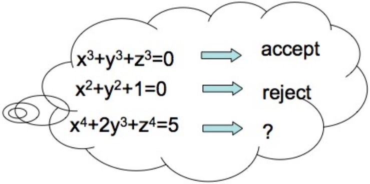
$$\textit{input} \xrightarrow{\textbf{machine}} \begin{cases} \bullet \text{ accept} \\ \bullet \text{ reject} \\ \bullet \text{ loop forever.} \end{cases}$$

For a language L

- if there is some Turing Machine that accepts every string in L and rejects every string not in L, then L is a decidable language
- if there is some Turing machine that accepts every string in L and either rejects or loops on every string not in L, then L is Semi-decidable or computably enumerable (CE)

L =all polynomial equations with integer coefficients that have a solution in the integers

This is CE!



if it were decidable, this would mean we had a method of determining whether any equation has a solution or not!

L =all C programs that crash on some input

CE as well!

If it were decidable, life would be sweet...

Accept=$\{\langle M, x\rangle : M$ is a Turing Machine that accepts string x$\}$

CE

Alternative definition of Computable Enumerability

- Why is "Semi-Decidable" called CE?
- Definition: an enumerator for a language $L \subset \Sigma^*$ is a TM that writes on its output tape

$$\#x_1\#x_2\#x_3\#\dots$$

and $L = \{x_1, x_2, x_3, \dots\}$.

- The output may be infinite

Computable Enumerability

Theorem

A language is Semi-decidable/CE iff some enumerator enumerates it.

Proof:

($\Leftarrow$) Let E be the enumerator for L. We create a semi-decider for L. On input w:

- Simulate E. Compare each string it outputs with w.
- If w matches a string output by E, accept.

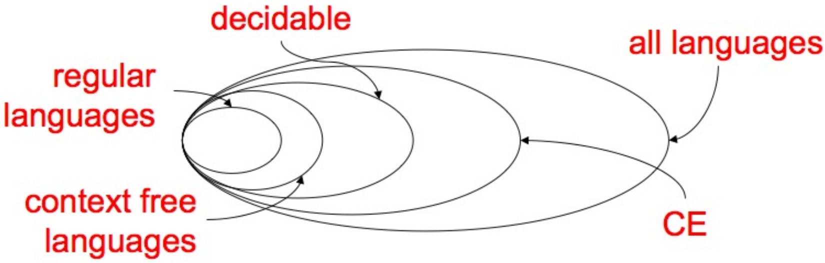
Theorem

A language is Semi-decidable/CE iff some enumerator enumerates it.

Proof:

(⇒) Let M recognise (semi-decide) language $L \subset \Sigma^*$. We create an enumerator for L.

- let $s_1, s_2, s_3, \ldots$ be enumeration of Σ^* in lexicographic order.
- for $i = 1, 2, 3, 4, \ldots$
  - simulate M for i steps on $s_1, s_2, s_3, \ldots, s_i$
- if any simulation accepts, print out that s_j



decidable $\subset$ CE $\subset$ all languages

our goal: prove these containments proper

Countable and Uncountable Sets

- the natural numbers $\mathbf{N}=\{1, 2, 3, \ldots\}$ are countable

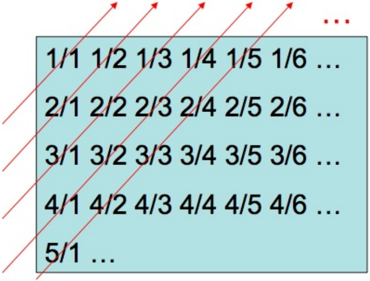
- Definition: a set S is countable if it is finite, or if it is infinite and there is an onto (surjective) function $f : \mathbf{N} \rightarrow S$

Equivalently: there is a function from S into $\mathbf{N}$

Theorem

The positive rational numbers $\mathbf{Q} = \{m/n \ : \ m, n \in \mathbf{N}\}$ *are countable.*

- Proof:



Theorem

The real numbers **R** *are NOT countable (they are "uncountable").*

How do you prove such a statement?

- assume countable (so there exists function f from **N** onto **R**)
- derive contradiction ("construct" an element not mapped to by f)
- technique is called diagonalization (Cantor)

Proof:

- suppose **R** is countable
- list **R** according to the bijection f:

n	$f(n)$
1	$3.14159\ldots$
2	$5.55555\ldots$
3	$0.12345\ldots$
4	$0.50000\ldots$
$\ldots$	

Proof:

- suppose **R** is countable
- list **R** according to the bijection f:

n	$f(n)$
1	3.14159 ...
2	5.55555 ...
3	0.12345 ...
4	0.50000 ...
...	

set $x = 0 \cdot a_1 a_2 a_3 a_4 \ldots$

where digit $a_i \neq i$-th digit after decimal point of $f(i)$

e.g. $x = 0.2641\ldots$

x cannot be in the list!

Theorem

There exist languages that are not Computably Enumerable.

Proof outline:

- the set of all TMs is countable (and hence so is the set of all CE languages)
- the set of all languages is uncountable
- the function $L : \{\text{TMs}\} \rightarrow \{\text{all languages}\}$ cannot be onto

non-CE languages

Lemma

The set of all TMs is countable.

Proof:

- each TM M can be described by a finite-length string $\langle M \rangle$
- can enumerate these strings, and give the natural bijection with $\mathbf{N}$

Lemma

The set of all languages is uncountable.

Proof:

- fix an enumeration of all strings s_1, s_2, s_3, . . . (for example, lexicographic order)
- a language L is described by an infinite string in $\{\text{In}, \text{Out}\}^*$ whose i-th element is In if s_i is in L and Out if s_i is not in L.

non-CE languages

- suppose the set of all languages is countable
- list membership strings of all languages according to the bijection f:

n	$f(n)$
1	0101010 ...
2	1010011 ...
3	1110001 ...
4	0100011 ...
...	

0 =Out
1 =In

non-CE languages

- suppose the set of all CE languages is countable
- list characteristic vectors of all languages according to the bijection f:

n	$f(n)$
1	0101010 . . .
2	1010011 . . .
3	1110001 . . .
4	0100011 . . .
. . .	

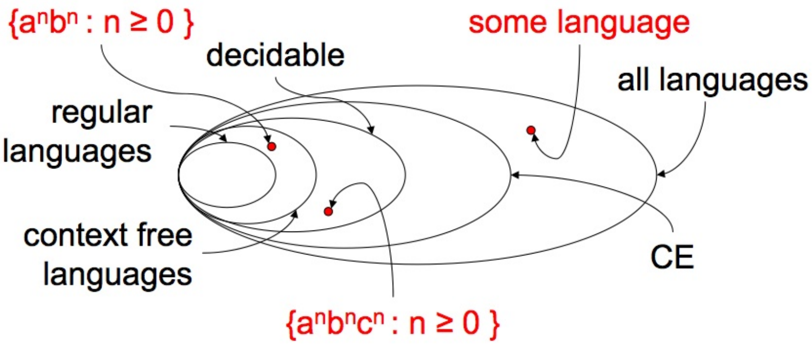
create language L with membership string x

where i-th digit of $x \neq i$-th digit of $f(i)$

x cannot be in the list!

therefore, the language L is not in the list.

So far...



- This language might be an esoteric, artificially constructed one. So who cares?
- We will show a natural undecidable L next.

The Halting Problem

- Definition of the "Halting Problem":
  $$\text{HALT} = \{\langle M, x\rangle \ : \ \text{TM } M \text{ halts on input } x\}$$
  $\langle M, x\rangle$ denotes coding of machine and input as a string (pick some coding – doesn't matter for this argument)
- HALT is computably enumerable.
  (proof?)
- Is HALT decidable?

HALT is a generic software-testing challenge, so genuinely interesting!

Theorem

HALT is not decidable (undecidable).

Proof will involve the following

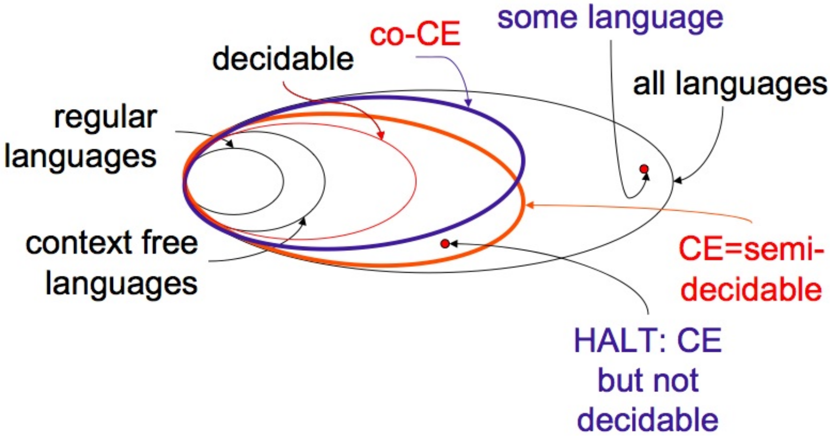
- Suppose there's some TM H that decides HALT. Using this we will get a contradiction.
- You'll need to believe that TMs can simulate other TMs, also can be composed with each other.

Proof

- For simplicity, assume input alphabet is one-letter, so inputs to machines are unary integers.
- Assume that HALT were decidable. We create a new TM H' that is *different from every other Turing machine* (clearly a contradiction, since H' would have to be different from itself!)
- Let $M_1, \ldots, M_n, \ldots$ enumerate all the Turing Machine descriptions. Suppose H decides HALT.
- Definition of H':
  On input n (i.e. 1^n), H' runs machine H on $\langle M_n, n \rangle$
  - if H returns ACCEPT (so M_n halts on n), then H' goes into a loop (alternatively: runs M_n on n, and then H' returns ACCEPT iff M_n rejects n.
  - If H returns REJECT (so M_n does not halt on n), then H' ACCEPTS.

H' is a TM, but is different from every TM (since disagrees with i-th TM in its behaviour on input 1^i $\rightarrow$ contradiction!)

Language Classes: Current Summary



Q: any interesting language that is not CE?

Theorem

A language L is decidable if and only if L is CE and L is co-CE.

Proof:
($\Rightarrow$) we already know decidable implies CE

- if L is decidable, then complement of L is decidable by flipping accept/reject.
- so L is co-CE.

Theorem

A language L is decidable if and only if L is CE and L is co-CE.

Proof:
($\Leftarrow$) we have TM M that recognises L, and TM M' recognises complement of L.

- on input x, simulate M, M' in parallel
- if M accepts, accept; if M' accepts, reject.

A concrete language that is not CE

Theorem

A language L is decidable if and only if L is CE and L is co-CE.

Corollary

The complement of HALT is not CE.

Proof:

- we know that HALT is CE but not decidable
- if complement of HALT were CE, then HALT is CE and co-CE hence decidable. Contradiction.

Bottom line: For every "strictly semi-decidable language", its complement cannot be semi-decidable.

Reductions

- Given a new problem NEW, want to determine if it is easy or hard
    - right now, easy typically means decidable
    - right now, hard typically means undecidable
- One option:
    - prove from scratch that the problem is easy (decidable), or
    - prove from scratch that the problem is hard (undecidable) (e.g. dream up a diag. argument)

Reductions

- A better option:
  - to prove NEW is decidable, show how to transform it (effectively) into a known decidable problem OLD so that solution to OLD can be used to solve NEW.
  - to prove NEW is undecidable, show how to transform a known undecidable problem OLD into NEW so that solution to NEW could be used to solve OLD.
- called a **reduction**. Reduction from problem A to problem B shows that "A is no harder than B", and also that "B is at least as hard as A".
- to get a positive result on NEW, create a reduction from NEW to OLD, where OLD is known to be easy.
- To get a negative result on NEW, create a reduction from OLD to NEW, where OLD is known to be hard.

Example reduction

- Try to prove undecidable:
  $ACC_{TM} = \{\langle M, w\rangle \ : \ M \text{ accepts input } w\}$
- We know this language is undecidable:
  $HALT = \{\langle M, w\rangle \ : \ M \text{ halts on input } w\}$
- Idea:
  - suppose ACC_{TM} is decidable
  - show that we can use ACC_{TM} to decide $HALT$ (*reduction*)
  - conclude $HALT$ is decidable. Contradiction.

Example reduction

- How could we use procedure that decides ACC_{TM} to decide $HALT$?
  - given input to $HALT$: $\langle M, w \rangle$
- Some things we can do:
  - check if $\langle M, w \rangle \in ACC_{TM}$
  - construct another TM M' and check if $\langle M', w \rangle \in ACC_{TM}$

Example reduction

Deciding $HALT$ using a procedure that decides ACC_{TM} ("reducing $HALT$ to ACC_{TM}").

- on input $\langle M, w \rangle$
- check if $\langle M, w \rangle \in ACC_{TM}$
  - if yes, then know M halts on w; **ACCEPT**
  - if no, then M either rejects w or it loops on w
- construct M' by swapping $q_{\text{accept}}/q_{\text{reject}}$ in M
- check if $\langle M', w \rangle \in ACC_{TM}$
  - if yes, then M' accepts w, so M rejects w; **ACCEPT**
  - if no, then M neither accepts nor rejects w; **REJECT**

Want to prove language L is undecidable.
Let $L_{\text{impossible}}$ be some problem that we already know is undecidable (e.g. Halting).

Proof by contradiction: Assume that there were some TM M_L that decides L. Show that using M_L we could decide $L_{\text{impossible}}$, a contradiction.

How to do this?
Create a Turing Machine N that decides $L_{\text{impossible}}$; N has "subroutines" calling M_L.

Simplest version, "many-one reduction": N takes an input I to $L_{\text{impossible}}$, and construct a new input I' to test against M_L.

Another example

Try to prove undecidable:

$$\text{NEMP} = \{\langle \text{M} \rangle \ : \ L(\text{M}) \neq \emptyset\}$$

Reduce from

$$\text{HALT} = \{\langle \text{M}, w \rangle \ : \ \text{M halts on input } w\}$$

OK, we want to decide HALT using NEMP

Create a machine N that decides HALT on input $\langle M, w \rangle$ using "subroutines" for NEMP.

N wants to check if $\langle M, w \rangle \in$HALT

- N constructs another TM M' and checks if $\langle M' \rangle \in$NEMP
- M′ constructed so that $\langle \text{M}, w \rangle \in$HALT $\Leftrightarrow \langle \text{M}' \rangle \in$ NEMP

Reducing HALT to NEMP

idea of N (function it computes):

- Given $\langle M, w \rangle$, construct $\langle M' \rangle$; on any input i, M' runs M on w and accepts i if M halts

construction of M':

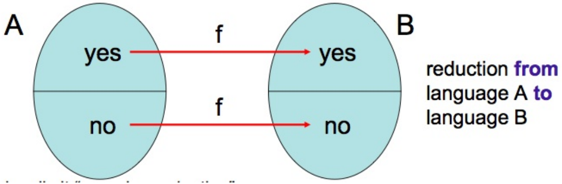
1. Use 3 states to delete any input (make tape blank)
2. $|w|$ states print w on input tape
3. Use copies of M's states to simulate M on w
4. ...make sure all states accept.

N constructs M' as above (can be done automatically, i.e. N is doing something computable!)

Extra note: this reduction also proves that the problem of recognising whether a TM accepts an infinite number of distinct inputs, is undecidable.

Definition: $A \leq_m B$ (A many-one reduces to B) if there is a computable (using a TM) function f such that for all w

$$w \in A \Leftrightarrow f(w) \in B$$



Book calls it "mapping reduction".

Example: to show NEMP undecidable, constructed computable f so that $\langle M, w\rangle \in \text{HALT} \Leftrightarrow f(\langle M, w\rangle) \in \text{NEMP}$

In this notation: $\text{HALT} \leq_m \text{NEMP}$

many-one reductions

Definition: $A \leq_m B$ (A many-one reduces to B) if there is a computable function f such that for all w

$$w \in A \Leftrightarrow f(w) \in B$$

Theorem

If $A \leq_m B$ and B is decidable then A is decidable.

Proof:

- decider for A: on input w compute $f(w)$, run decider for B, do whatever it does.

Using many-one reductions

Theorem

If $A \leq_m B$ and B is CE, then A is CE.

Proof:

- TM for recognizing A: on input w compute $f(w)$, run TM that recognises B, do whatever it does.

Main use: given language NEW, prove it is not CE by showing $OLD \leq_m NEW$, where OLD known to be not CE.

Applying Reductions to Get Negative Results on Decidability

Theorem

The language
$REGULAR = \{\langle M\rangle \ : \ M \text{ is a TM and } L(M) \text{ is regular}\}$
is undecidable.

Proof:

- reduce from ACC_{TM} (i.e. show $ACC_{TM} \leq_m REGULAR$)
- i.e. want
  M accepts $w \quad \Leftrightarrow \quad f(\langle \mathrm{M}, w\rangle)$ is code of regular language
- what should $f(\langle \mathrm{M}, w\rangle)$ produce?

Proof:

- $f(\langle M, w\rangle) = \langle M'\rangle$ described below

M' takes input x:

- if x has form 0^n1^n, accept
- else simulate M on w and accept x if M accepts

$M' = \{0^n1^n\}$ if $w \notin L(M)$
$=\Sigma^*$ if $w \in L(M)$

What would a formal proof of this look like?

- is f computable?
- YES maps to YES?
  $\langle M, w\rangle \in ACC_{TM} \Rightarrow$
  $f(M, w) \in REGULAR$
- NO maps to NO?
  $\langle M, w\rangle \notin ACC_{TM} \Rightarrow$
  $f(M, w) \notin REGULAR$

general idea: write pseudo-code that takes description of M as input and produces description of M'.
Argue that this pseudo-code could be implemented as a Turing machine with output tape.

Decidable and Undecidable problems

The boundary between decidability and undecidability is often quite delicate

- seemingly related problems
- one decidable
- other undecidable

We will cover most examples in the problem sheet

Problem: Given a context free grammar G, is the language it generates empty?
Decidable: i.e. language $\{\langle G \rangle \ : \ L(G) \text{ empty}\}$ is a decidable language.

See problem sheets.

Decidable and Undecidable problems

The boundary between decidability and undecidability is often quite delicate

- seemingly related problems
- one decidable
- other undecidable

We will cover most examples in the problem sheet

Problem: Given a context free grammar G, is the language it generates empty?
Decidable: i.e. language $\{\langle G \rangle \ : \ L(G) \text{ empty}\}$ is a decidable language.
See problem sheets.

Problem: Given a context free grammar G, does it generate every string?
Undecidable: i.e. language $\{\langle G \rangle \ : \ L(G) = \Sigma^*\}$ is an undecidable language.
In next problem set.

Decidable and Undecidable problems

Problem: Given a NPDA, is the language it accepts empty?

- Decidable. Convert to CFG and use previous result.

Note: reduction *to* a known decidable problem is device to prove decidability

Decidable and Undecidable problems

Problem: Given a NPDA, is the language it accepts empty?

- Decidable. Convert to CFG and use previous result.

Note: reduction *to* a known decidable problem is device to prove decidability

Problem: Given a two-stack NPDA, is the language it accepts empty?

- Undecidable. In current problem set.

Post Correspondence Problem

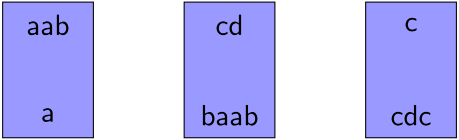
Undecidability can find its way into problems that are not "obviously" about TMs/computation in general. E.g. some puzzle-like problems; PCP is as follows:

$$PCP = \{\langle (x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k) \rangle \ : \ x_i, y_i \in \Sigma^*$$

$$\text{and there exists } (a_1, a_2, \ldots, a_n)$$

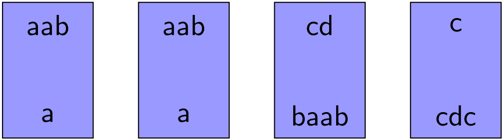
$$\text{for which } x_{a_1} x_{a_2} \ldots x_{a_n} = y_{a_1} y_{a_2} \ldots y_{a_n}\}$$

PCP example



← Input

Solution:



PCP

Idea is a many-one reduction from ACC to PCP:
given a TM M and input w, we have an effective procedure that creates a set of tiles $T = f(M, w)$ such that:
M accepts $w \Leftrightarrow$ there is some way of producing a tiling with T.
(I won't cover it in lectures.)