Computational Models Lecture 7, Spring 2009

- The Church-Turing Thesis
- The language classes $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$
- David Hilbert’s Tenth Problem
- Encoding of TMs
- Universal Turing Machines
- The Halting/Acceptance problem
- The Halting/Acceptance problems are undecidable
- Diagonalization

- Sipser’s book, 3.2, 3.3, 4.1, 4.2

Decidability vs. Enumerability

- Decidability is a stronger notion than enumerability.
- If a language L is decidable then clearly it is enumerable (the other dirction does not hold, as we'll show in a couple of lectures).
- It is also clear that if L is decidable then so is $\overline{L}$, and thus $\overline{L}$ is also enumerable.
- Let $\mathcal{RE}$ denote the class of enumerable languages, and let $co\mathcal{RE}$ denote the class of languages whose complement is enumerable.
- Let $\mathcal{R}$ denote the class of decidable languages. Then what we just argued implies $\mathcal{R} \subseteq \mathcal{RE} \cap co\mathcal{RE}$.

Decidability vs. Enumerability (2)

Theorem: $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$.

Proof: We should prove the $\supseteq$ direction. Namely if $L \in \mathcal{RE} \cap co\mathcal{RE}$, then $L \in \mathcal{R}$.

In other words, if both L and its complement are enumerable, then L is decidable.

Let M_1 be a TM that accepts L.
Let M_2 be a TM that accepts $\overline{L}$.
We describe a TM, M, that decides L.

On input x, M runs M_1 and M_2 in parallel.
If M_1 accepts, M accepts.
If M_2 accepts, M rejects.

Should now show that indeed M decides L.

Reformulation

Theorem: A language is decidable if and only if it is both enumerable and co-enumerable.

Proof: We must prove two directions:

- If L is decidable, then both L and $\overline{L}$ are enumerable.
- If L and $\overline{L}$ are both enumerable, then L is decidable,

One Direction

Claim: If L is decidable, then both L and $\overline{L}$ are enumerable.

Proof: Argued this three slides ago!

Other Direction

Claim: If L and $\overline{L}$ are both enumerable, then L is decidable,

- Let M_1 be the acceptor for L, and
- M_2 the acceptor for $\overline{L}$.

$M =$ On input w

1. Run both M_1 and M_2 in parallel.
2. If M_1 accepts, accept ; if M_2 accepts, reject.

In Parallel?

Question: What does it mean to run M_1 and M_2 in parallel?

- M has two tapes
- M alternates taking steps between M_1 and M_2.

Claim

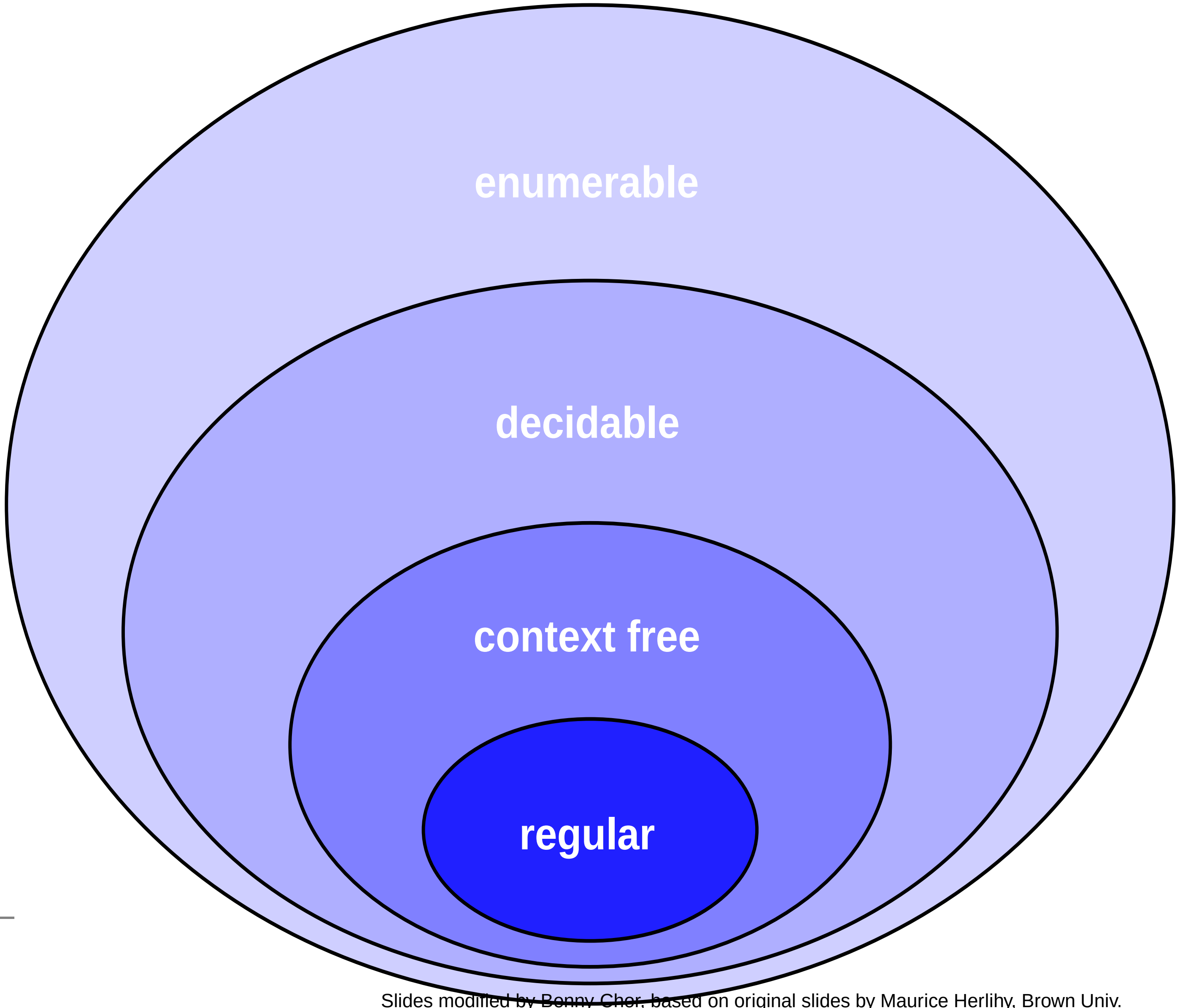
We claim that M decides L.

- Every string is in L or in $\overline{L}$ (of course not in both).
- Thus either M_1 or M_2 accepts the input w.
- Consequently, since M halts whenever M_1 or M_2 accepts, M always halts, and hence is a decider.
- Moreover, M accepts strings in L and rejects strings in $\overline{L}$.

Therefore, M decides L, so L is decidable. ♣

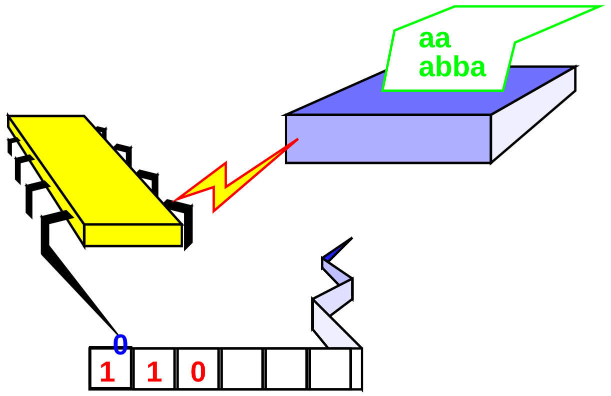
Revised View of the World of Languages

We'll see later today (or next week) that $\mathcal{R} \subsetneq \mathcal{RE}$



Enumerators

We said a language is enumerable if it is accepted by some Turing machine.
But why *enumerable*?



Definition: An enumerator is a TM with a printer.

- TM sends strings to printer
- may create infinite list of strings
- TM enumerates a language – all the strings produced.

Theorem

Theorem: A language is accepted by some Turing machine if and only if some enumerator enumerates it.

Will show

- If E enumerates language A, then some TM M accepts A.
- If M accepts A, then some enumerator E enumerates it.

Theorem

Claim: If E enumerates language A, then some TM M accepts A.

On input w, TM M

- Runs E. Every time E outputs a string v, M compares it to w.
- If $v = w$, M accept.
- If $v \neq w$, M continues running E.

Theorem

Claim: If M accepts A, then some enumerator E enumerates it.

Let $s_1, s_2, s_3, \ldots$ is a list of all strings in Σ^* (*e.g.* strings in lexicographic order).
The enumerator, E

- repeat the following for $i = 1, 2, 3, \ldots$
- run M for i steps on each input $s_1, s_2, \ldots, s_i$.
- if any computation accepts, print out the corresponding s. ♣

Note that with this procedure, each output is duplicated infinitely often.
How can this duplication be avoided?

Theorem

Claim: A language L is decidable if and only if there is some enumerator E that enumerates L in lexicographic order.

Proof: Left as an exercise.

What is an Algorithm???????

Indeed, What is an Algorithm?

Informally

- a recipe

- a procedure
- a computer program
- who cares? I know it when I see it :-(

Indeed, What is an Algorithm?

Historically,

- notion has long history in Mathematics (starting with Euclid’s gcd algorithm), but
- not precisely defined until 20th century
- informal notions rarely questioned,
- still, they were insufficient

Euclid of Alexandria, circa 300BC

Remarks

- Many models have been proposed for general-purpose computation.
- Remarkably, all “reasonable” models were shown to be equivalent to Turing machines.
- All “reasonable” programming languages (*e.g.* Java, Pascal, C, Python, Scheme, Mathematica, Maple, Cobol,. . .) are equivalent.
- The notion of an algorithm is model-independent!
- We don’t really care about Turing machines *per se*.
- We do care about understanding computation, and because of their simplicity, Turing machines are a good model to use.

Church-Turing Thesis

Formal notions appeared, starting in 1936:

- λ-calculus of Alonzo Church
- Turing machines of Alan Turing
- Recursive functions of Godel and Kleene
- Counter machines of Minsky
- Normal algorithms of Markov
- Unrestricted grammars
- Two stack automata
- Random access machines (RAMs)

⋮

These definitions look very different, but are provably

Church-Turing Thesis

These definitions look very different, but are provably equivalent.

The Church-Turing Thesis:

> "The intuitive notion of reasonable models of computation equals Turing machine algorithms".

Church-Turing Thesis

Why is this a “thesis” and not a theorem?

- Because the notion of “reasonable model of computing” is not well defined.
- This year (2009), Nachum Dershowitz (TAU), Udi Boker (a former TA in this course), and Yu. Gurevitch (MSFT) gave an axiomatization of this notion.
- This axiomatization seems to “reasonably capture” what a reasonable model of computing is.
- Under this axiomatization, DBG gave a proof of the Church-Turing thesis.

Wild Models

What about "wild" models of computation?

Consider MUntel's $\aleph$-AXP9$^{©}$ processor (to be released Labor Day 2009).

- Like a Turing machine, except
- Takes first step in 1 second.
- Takes second step in $1/2$ second.
- Takes i-th step in 2^{-i} seconds . . .

After 2 seconds, the $\aleph$-AXP$^{©}$ decides any enumerable language!

Question: Does the $\aleph$-AXP$^{©}$ invalidate the Church-Turing Thesis?

Hilbert's 10th Problem

In 1900, David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris, France.

- Presented 23 central mathematical problems
- challenge for the next (20th) century
- the 10th problem directly concerned algorithms

Hilbert

Hilbert's Tenth Problem

The Problem: Devise an algorithm that tests whether a multivariate polynomial has an integral root.

Actually, what he said (translated from German) was

> "to devise a process according to which it can be determined by a finite number of operations".

Note that

- Hilbert explicitly asks that algorithm be "devised"
- apparently Hilbert assumes that such an algorithm must exist, and someone "only" need find it.

Hilbert's Tenth Problem

- We now know no algorithm exists for this task.
- Mathematicians of 1900 could not have proved this, because they didn't have a formal notion of an algorithm.
- Intuitive notions work fine for constructing algorithms (we know one when we see it).
- Formal notions are required to show that no algorithm exists.

Hilbert’s Tenth Problem

In 1970, 23 years old Yuri Matijasevič, building on work of Martin Davis, Hilary Putnam, and Julia Robinson, proved that no algorithm exists for testing whether a multivariate polynomial has integral roots (a survey of the proof)

Reformulating Hilbert’s Tenth Problem

Consider the language:

$D = \{p \mid p$ is a multivariate polynomial
with an integral root$\}$

Hilbert’s tenth problem asks whether this language is decidable.

It is easy to see that this language is enumerable (why?). By Matijasevič theorem, it is not decidable, so it is in $\mathcal{RE} \setminus \mathcal{R}$.

Univariate Polynomials

Consider the simpler language:

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}$$

Here is a Turing machine that accepts D_1.
On input p,

- evaluate p with x set successively to $0, 1, -1, 2, -2, \ldots$.
- if p evaluates to zero, accept.

Univariate Polynomials (2)

$D_1 = \{p \mid p$ is a polynomial over x with an integral root$\}$

Note that

- If p has an integral root, the machine accepts.
- If not, M_1 loops.
- M_1 is an acceptor, but not a decider.

Univariate Polynomials (3)

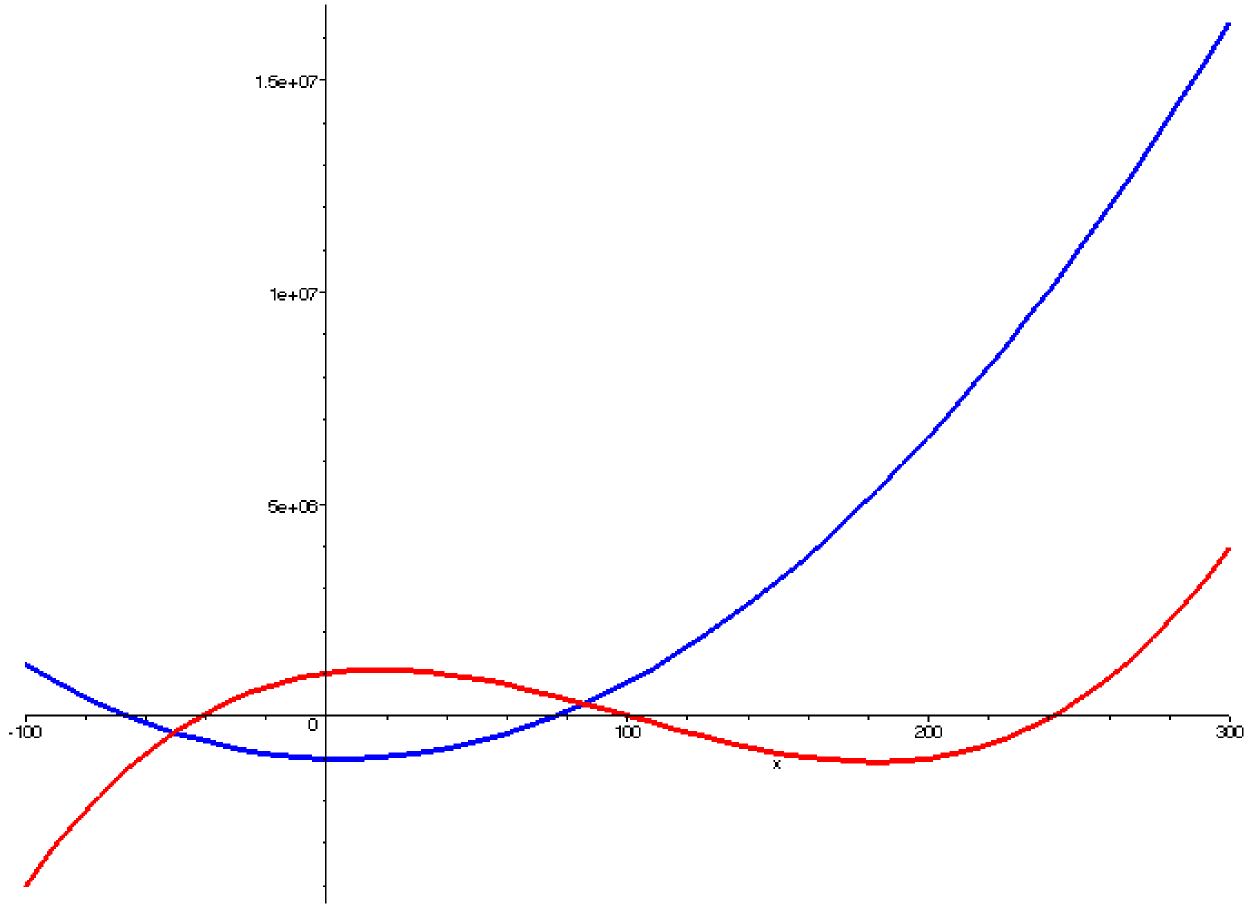
```
> f:=x->x^3-300*x^2+10000*x+1000000;
```

$$f := x \to x^3 - 300\,x^2 + 10000\,x + 1000000$$

```
> g:=x->200*x^2-2000*x-1000000;
```

$$g := x \to 200\,x^2 - 2000\,x - 1000000$$

```
> plot([f(x),g(x)],x=-100..300,color=[red,blue],thickness=3);
```



Univariate Polynomials (4)

In fact, D_1 is decidable.

Can show that all real roots of $p[x]$ lie inside interval

$$(\ -|kc_{max}/c_1|, |kc_{max}/c_1| \ ) \ ,$$

where k is number of terms, c_{max} is max coefficient, and c_1 is high-order coefficient.

By Matijasevič theorem, such effective bounds on range of real roots cannot be computed for multivariable polynomials.

Furthermore, it was shown that the problem is undecidable even when restricted to polynomials of degree ≤ 4 in just nine variables.

Encoding

- Input to a Turing machine is a string of symbols.
- But we want algorithms that work on graphs, matrices, polynomials, Turing machines, etc.
- Need to choose an encoding for objects.
- Can often be done in many reasonable ways.
- Sometimes it is helpful to distinguish between X, the object, and $\langle X \rangle$, its encoding.

Encoding

Consider strings representing undirected graphs.

A graph is connected if every node can be reached from any other node by traveling along edges.

Define the language:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

Clearly the language should be decidable.

High-Level Description

High-level description of a machine that decides

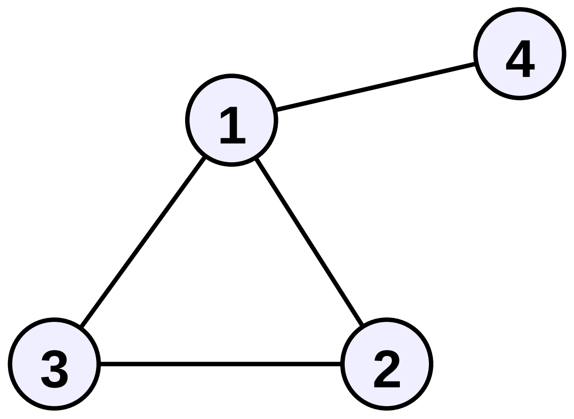
$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

On input $\langle G \rangle$, encoding of graph G

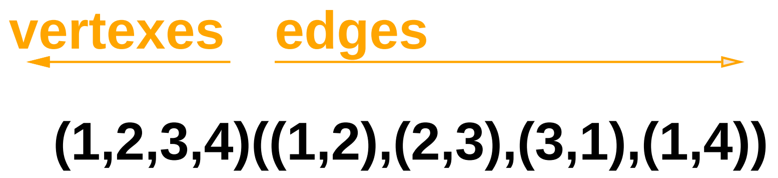
- Select first node of G and mark it.
- Repeat until no new nodes marked:
- For each unmarked node in G, mark it if attached by an edge to a node already marked.
- Scan nodes of G to determine whether they are all marked. If so, accept, otherwise reject.

Technical Details

Question: How is G encoded?



G



(1,2,3,4)((1,2),(2,3),(3,1),(1,4))

<G>

One possible answer: List of nodes, followed by list of edges.
(Note: Other encodings are also possible and feasible)

More Details

M checks that the input is valid encoding of a graph, consisting of:

- two lists
- first is list of numbers ("nodes")
- second is list of pairs ("edges")
- every node in second list appears in first
- first list contains no duplicates (element distinctness subroutine)
- second list contains no duplicates (element distinctness subroutine)

Now ready to start “step one”.

Detailed Algorithm

On input $\langle G \rangle$, encoding of graph G.

1. M dots first node.
2. loop:
   - M scans list and "underlines" first undotted node n_1.
   - M rescans and "underlines" first dotted node n_2.
   - M scans edges.
   - M tests each edge if it equals (n_1, n_2).
   - If so, dot n_1, remove underlines, goto Step 2.
   - If not, check next edge. When no more edges, move underline from n_2 to next dotted node, naming this new n_2.
   - When exhausting all dotted vertices, move underlines: new n_1 is next dotted node and new n_2 is first undotted node. Repeat Step 2. When no more undotted nodes, go to Step 3.
3. M scans list of nodes. If all are dotted, accept, else reject.

Standard Encoding of Turing Machines

- The encoding $\langle M\rangle$ of a TM, M, will use a binary alphabet.
- Blocks of 0's will be used as delimiters.
- A set Q with m states will be indicated by m in unary.
- By conventions states will be 1 through m.
- By convention, q_0 is indicated by state 1.
- Indicate the accept and the reject states i and j (& delimiters!).
- Indicate the sizes of Σ, ℓ, and of Γ, k, (again in unary, plus delimiters). $\ell < k$. Letters in Σ encoded by $1 \ldots \ell$ (in unary), while letters in Γ encoded by $\ell + 1 \ldots k$ (in unary). By convention, blank is encoded by k.
- Finally, the transition function δ is encoded as a list of 5-tuples with correct size and no duplications in q, γ entries.
- Important and Easy: An algorithm (TM) can check that a given string is legal encoding of a TM.

Univeral Turing Machine

Univeral Turing Machines

We now define the universal Turing machine, U.
On input $\langle M, w\rangle$, where M is a TM and w a string

1. Checks that $\langle M, w\rangle$ is a proper encoding of a TM, followed by a string from Σ^*.
2. Simulates M on input w (HOW???)
3. If M on input w enters its accept state, U accept, and if M on input w ever enters its reject state, U reject.

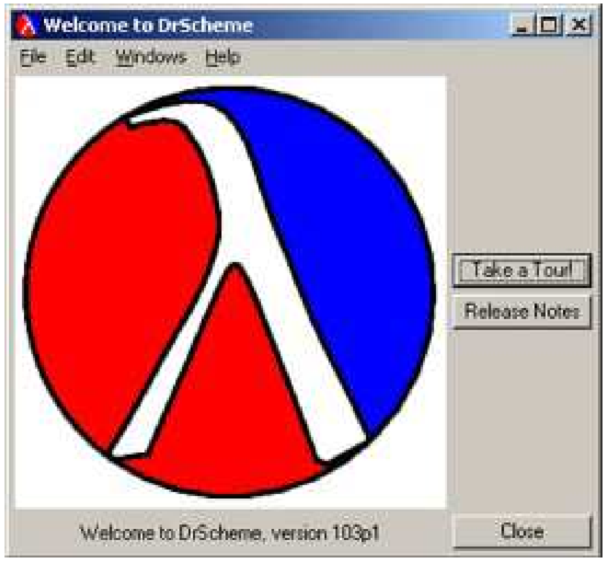
Notice that as a consequence, if M on input w enters an infinite loop, so does U on input $\langle M, w\rangle$.

Universal Turing Machines (2)

- The universal machine U obviously has a fixed number of states (100 should do) .
- Despite this, it can simulate machines M with many more states.
- Universal machines inspired the development of stored-program computers in the 40s and 50s.
- Most of you have seen a universal machine, and have even used one!

Universal Turing Machines (3)

- For example, *Dr. Scheme* (interpreter) is a universal Scheme machine.



- It accepts a two part input: “Above the line” – the program (corresponding to $\langle M \rangle$), and “below the line” the input to run it on (corresponding to w).

The Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

Many common problems are unsolvable, *e.g.*

- Does a program sort an array of integers?
  - Note that this problem is well defined: Both program and specification are precise mathematical objects.
- Hey, proving program $\cong$ specification should be just like proving that triangle 1 $\cong$ triangle 2 ...
- Well, this is not the case!

CFG, NFA, DFA Reminders

Let $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

- We saw that E_{CFG} is a decidable language.
- $A_{\text{CFG}} = \{\langle M, w\rangle \mid M$ is a PDA accepting the string $w\}$. Saw that the language A_{CFG} is decidable.
- $A_{\text{NFA}} = \{\langle M, w\rangle \mid M$ is an NFA accepting the string $w\}$.
- $A_{\text{DFA}} = \{\langle M, w\rangle \mid M$ is a DFA , accepting the string $w\}$.
- Saw both A_{NFA} and A_{DFA} are also decidable.
- What would happen with Turing Machines?

Acceptance Problem

Does a Turing machine accept a string?

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

Theorem: A_{TM} is undecidable.

Recall that the corresponding languages for DFAs, NFAs, and CFGs, namely A_{DFA}, A_{NFA}, and A_{CFG}, are decidable.

The Acceptance Problem

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

Before approaching the proof of undecidability, we first prove

Theorem: A_{TM} is recursively enumerable (namely in $\mathcal{RE}$).

Proof: The universal machine accepts A_{TM}. ♣

Acceptance, Again

We are now able to prove the undecidability of

$$A_{\textsf{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\} .$$

Proof: By contradiction. Suppose a TM, H, is a decider for $A_{\textsf{TM}}$.

On input $\langle M, w\rangle$, where M is a TM and w is a string, H halts and accepts if and only if M accepts w. Furthermore, H halts and rejects if M fails to accept w.

Acceptance (2)

On input $\langle M, w\rangle$, where M is a TM and w is a string, H halts and accepts if and only if M accepts w. Furthermore, H halts and rejects if M fails to accept w.

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

Acceptance (3)

Now we construct a new TM, D, with H as a subroutine.

D does the following

- Calls H to determine what TM, M, does when the input to M is its own description, $\langle M \rangle$.
- When D determines this, it does the opposite.
- So D rejects if M accepts $\langle M \rangle$, and accepts if M does not accept $\langle M \rangle$.

Acceptance (4)

More precisely, D does the following:

- Run H on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what H outputs:
  - If H accepts, reject, and
  - If H rejects, accept.

Self Reference (4)

Don’t be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of self-reference comes up again and again in diverse areas.
- Read “Gödel, Escher, Bach, an Eternal Golden Braid”, by Douglas Hofstadter.
- This notion of self-reference is the basic idea behind Gödel’s revolutionary result.

Compilers do this all the time . . . .

The Punch Line

So far we have,

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

What happens if we run D on its own description?

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

Oh, oh...
Or, more accurately, a contradiction (to what?) ♣

Once Again

- Assume that TM H decides A_{TM}.
- Then use H to build a TM, D, that when given $\langle M\rangle$, accepts exactly when M does not accept.
- Run D on its own description.
- D does:
  - H accepts $\langle M, w\rangle$ when M accepts w.
  - D rejects $\langle M\rangle$ exactly when M accepts $\langle M\rangle$.
  - D rejects $\langle D\rangle$ exactly when D accepts $\langle D\rangle$.
- Last step leads to contradiction.
- Therefore neither TM D nor H can exist.
- So A_{TM} is undecidable!

A Non-enumerable Language

- We already saw a non-decidable language: A_{TM}.
- Can we do better (i.e., worse)?
- Mais, oui!
- We now display a language that isn't even recursively enumerable . . . .

A Non-enumerable Language

Earlier we saw

Theorem: If L and $\overline{L}$ are both enumerable, then L is decidable.

Corollary: If L is not decidable, then either L or $\overline{L}$ is not enumerable.

Definition: A language is co-enumerable if it is the complement of an enumerable language.

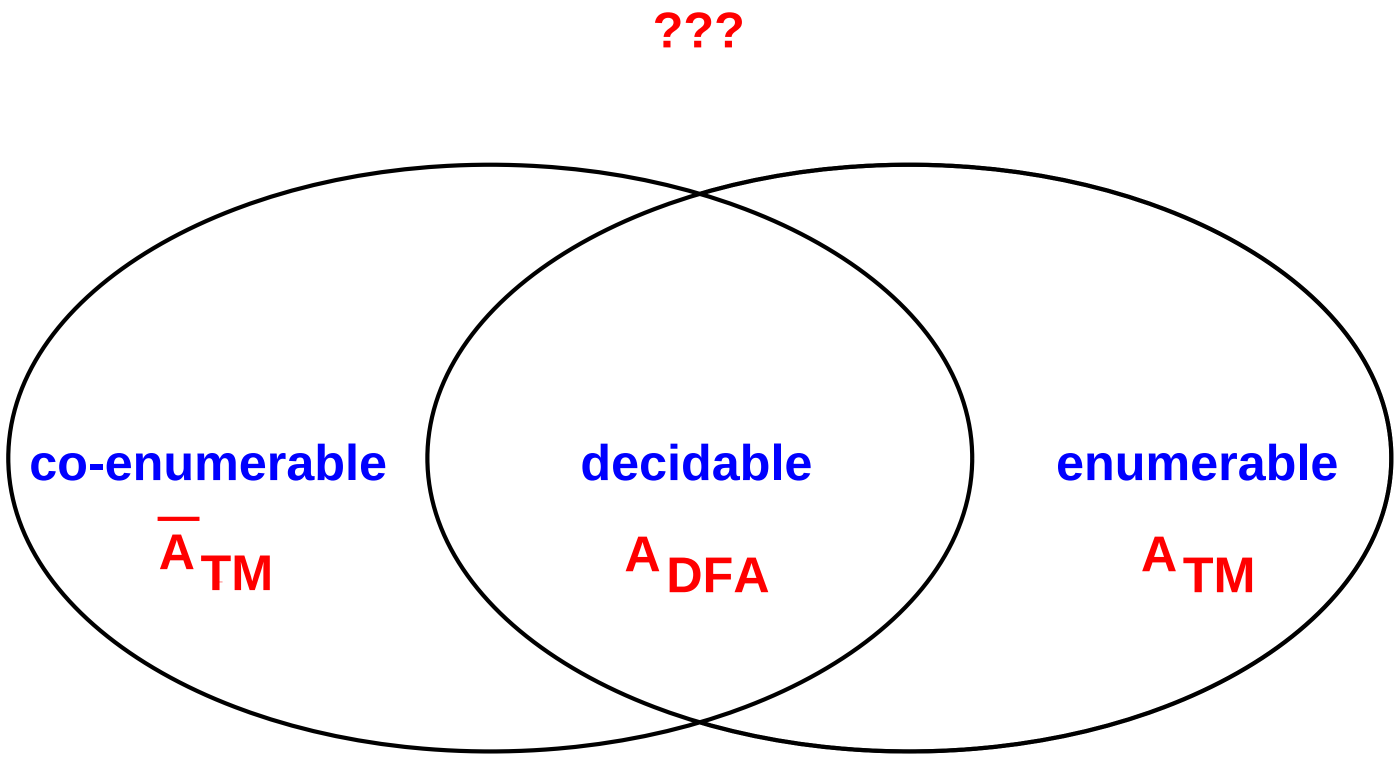
Reformulating theorem Theorem: A language is decidable if and only if it is both enumerable and co-enumerable.

$\overline{A}_{\mathsf{TM}}$ is not Enumerable

Theorem: If L and $\overline{L}$ are both enumerable, then L is decidable.

- We proved that A_{TM} is undecidable.
- On the other hand, we saw that the universal TM, U, accepts A_{TM}.
- Therefore A_{TM} is enumerable.
- If $\overline{A}_{\mathsf{TM}}$ were also enumerable, then by theorem A_{TM} was decidable.
- Therefore $\overline{A}_{\mathsf{TM}}$ is not enumerable. ♣

Languages



Question: Are there any languages in the area marked ??? ?

Answer: Yes, heaps (why?)

The Acceptance Problem (again)

We saw

$A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a TM that accepts $w\}$

Our proof that A_{TM} is undecidable was actually a diagonalization proof.

To see this, we start with a short “diagonalization reminder”.

Comparing Sizes of Sets

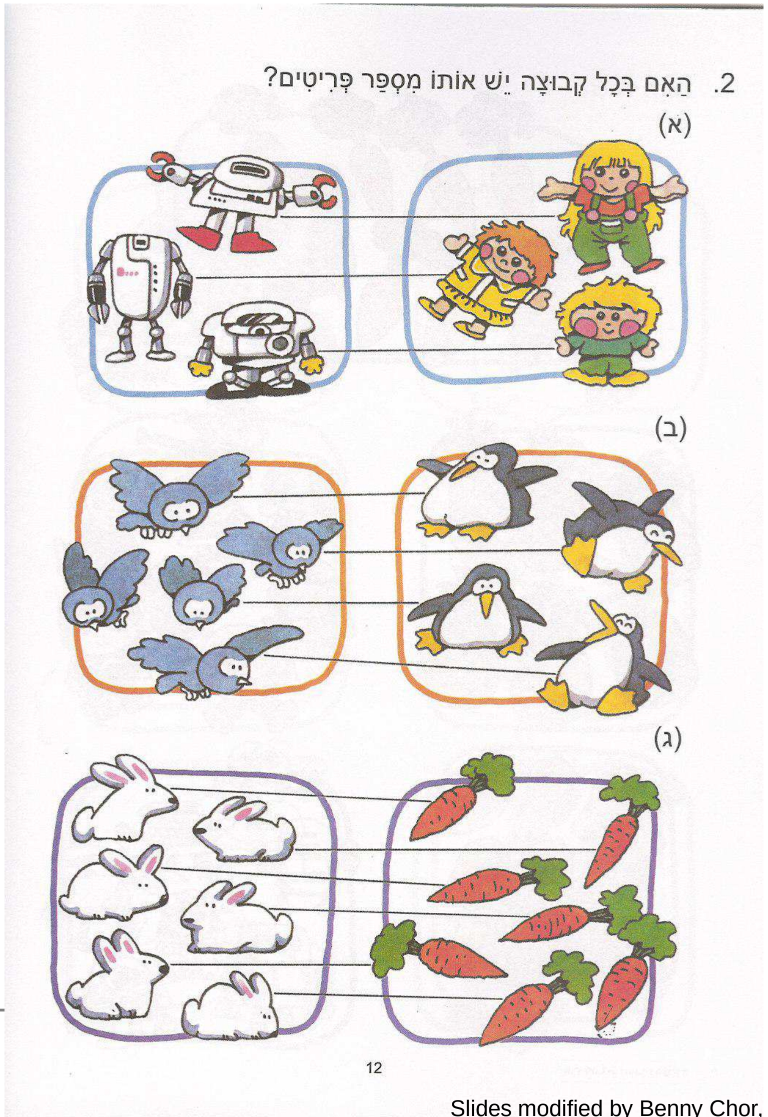
Suppose A and B are two sets, and we wish to compare their sizes.

If both A and B are finite, we can count how many elements each of them has, and compare the numbers.

This method does not generalize to infinite sets.

Comparing Sizes of Sets (2)

Alternatively, we can pair the elements of A and B. If they pair perfectly, they have equal sizes.


2. הַאִם בְּכָל קְבוּצָה יֵשׁ אוֹתוֹ מִסְפַּר פְּרִיטִים?

(א)

(ב)

(ג)

12

Correspondence

Question: What does it mean to say that two infinite sets are the *same size*?

Answered by Georg Cantor in 1873: Pair them off.

A map $f : A \to B$ is a *correspondence* if f satisfies

- f one-to-one: if $a_1 \neq a_2$ then $f(a_1) \neq f(a_2)$.
- f onto: for every $b \in B$, there is an $a \in A$ such that $f(a) = b$.

Question: What does it mean to say that sets A and B are the *same size*?

Answer: A and B are the *same size* if there is a correspondence from A to B.

Correspondence (2)

Question: In a crowded room, how can we tell if there are more people than chairs, or more chairs than people?

Answer Establish a correspondence: ask everyone to sit down.

(c.f., Mathematician's trick for counting a herd of cows . . .)

Correspondence

Claim: The set $\mathcal{N}$ of natural numbers has the same size as the set $\mathcal{E}$ of even numbers

Proof: Let $f(i) = 2i$.

Remark: The set $\mathcal{E}$ is a proper subset of the set $\mathcal{N}$, yet they are the same size!

Countable Sets

Definition A set A is *countable* if

- either A is finite, or
- A has the same size as $\mathcal{N}$, the natural numbers.

We have just seen that $\mathcal{E}$ is countable.

A countable set is sometimes said to have size $\aleph_0$.

Claim: The set $\mathcal{Z}$ of integers is countable.

Proof: Define $f : \mathcal{N} \to \mathcal{Z}$ by

$$f(i) = \begin{cases} i/2 & \text{if } i \text{ is even} \\ -(\lfloor i/2 \rfloor + 1) & \text{if } i \text{ is odd} \end{cases}$$

Pop Quiz

In Heaven, there is a hotel with a countable number of rooms.

One day, the society of Prophets, Oracles, and AI Researchers holds a 3-day convention that books every room in the hotel.

Then one more guest arrives, claiming he invented Lisp, and angrily demanding a room.

You are the manager. What do you do?

Answer: Ask the guest in room i to move to room $i+1$, and put the newcomer in room 1.

Pop Quiz #2

Then a countable number of guests arrive, all angrily demanding rooms. (What a noise!)

Now what do you do?

Answer: Ask the guest in room i to move to room $2i$, and put the newcomers in the odd-numbered rooms.

Rational Numbers

Let

$$\mathcal{Q} = \left\{ \frac{m}{n} \mid m, n \in \mathcal{N} \right\}$$

Theorem: $\mathcal{Q}$ is countable.

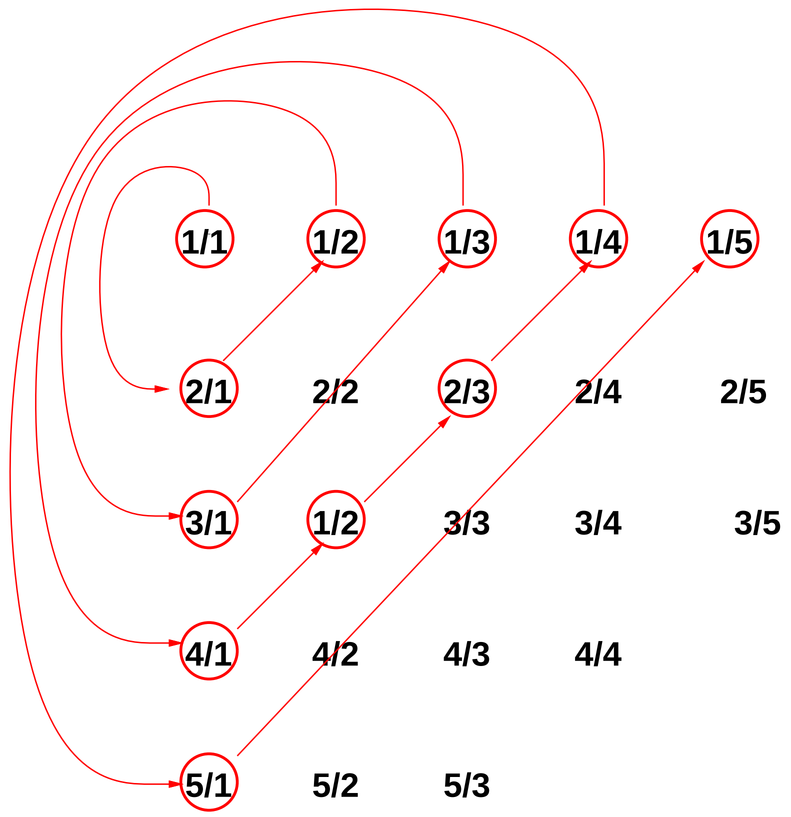
This claim may seem counterintuitive.

Idea

- list $\mathcal{Q}$ as 2-dim array
- begin counting with the first row . . .

Why doesn't this work?

Rational Numbers (2)



Enumerate numbers along northeast and diagonals, skipping duplicates.

Does this mean that every infinite set is countable?

The Real Numbers

Every *real number* has a decimal representation.

For example, $\pi = 3.1415926\ldots$, $\sqrt{2} = 1.4142136\ldots$, and $0 = 0.0000000\ldots$.

Let $\mathcal{R}$ be the set of real numbers.

Theorem: $\mathcal{R}$ is uncountable.

$\mathcal{R}$ is sometimes said to have size $\aleph_1$.

- This is Cantor's historic proof, which
- introduced the *diagonalization* method.

The Real Numbers

Assume there is a correspondence between $\mathcal{N}$ and $\mathcal{R}$.
Write it down:

n	$f(n)$
1	`3.14159...`
2	`55.55555...`
3	`40.18642...`
4	`15.20601...`

We now show that there is a number x not in this list.

Diagonalization

Pick $0 \leq x \leq 1$, so its significant digits follow decimal point.
Will ensure $x \neq f(n)$ for all n.

n	$f(n)$
1	3.14159...
2	55.55555...
3	40.18643...
4	15.20607...

Diagonalization

n	$f(n)$
1	3.**1**4159...
2	55.5**5**555...
3	40.18**6**43...
4	15.206**0**7...

- First fractional digit of $f(1)$ is 1, so pick first fractional digit of x to be something else (say, 2).
- Second fractional digit of $f(2)$ is 5, so pick second fractional digit of x to be something else (say, 6).
- and so on $\ldots$
- $x = 0.2691\ldots$

Diagonalization

A similar proof shows there are languages that are not enumerable.

  - the set of Turing machines is countable, but
  - the set of languages is uncountable!
- Ergo,
  - there exist languages that are not enumerable (why?)
  - indeed, “most” languages are not enumerable (explain)

$\exists$ Countably Many Turing Machines

Claim: The set of strings, Σ^*, is countable.

Proof: List strings of length 0, then length 1, then 2, and so on. This exhausts all of Σ^*.

The union of countably many finite sets is countable.

$\exists$ Countably Many Turing Machines (2)

Claim: The set of all Turing machines is countable.

Proof: Each TM M has an encoding as a string $\langle M \rangle$. Therefore there is a one-to-one mapping from the set of all TMs into (but not onto) Σ^*.

Since Σ^* is countable, so is the set of all TMs.

The Set of All Languages is Uncountable

Let $\mathcal{B}$ be the set of of infinite binary sequences.

Claim $\mathcal{B}$ is uncountable.

Proof Diagonalization argument, essentially identical to the proof that $\mathcal{R}$ is uncountable.

(additional helpful clue: think of binary sequence as binary expansion!)

The Set of Languages is Uncountable (2)

Let $\mathcal{L}$ be the set of all languages over alphabet Σ.
Recall $\mathcal{B}$ is the set of of infinite binary sequences.
We give a correspondence

$$\chi : \mathcal{L} \rightarrow \mathcal{B}$$

called the language's *characteristic sequence*.

- Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$ (in lexicographic order).
- Each language $L \in \mathcal{L}$ is associated with a unique sequence $\chi(L) \in \mathcal{B}$:
- the i-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$.

The Set of Languages is Uncountable (3)

Each language $L \in \mathcal{L}$ has a unique sequence $\chi(L) \in \mathcal{B}$: the i-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$.

Example:

$$\begin{array}{llllllllll}
\Sigma^* & \{\varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000 & \ldots\} \\
A & \{ & 0, & & 00, & 01, & & & 000 & \ldots\} \\
\chi(A) & \{0, & 1, & 0 & 1, & 1, & 0, & 0, & 1 & \ldots\}
\end{array}$$

The map $\chi : \mathcal{L} \to \mathcal{B}$

- is one-to-one and onto (why?),
- and is hence a correspondence.
- It follows that $\mathcal{L}$ is uncountable.

TMs vs. Languages

We saw that the set of all Turing machines is countable.

We saw that the set $\mathcal{L}$ of all languages over alphabet Σ is uncountable.

Therefore there are languages that are not accepted by any TM.

This is an existential proof – it does not explicitly show any such language.

Reflections on Diagonalization

This proof that the acceptance problem is undecidable is actually diagonalization in transparent disguise.
To unveil this, let's start by making a table.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					
M_4	accept	accept			
$\vdots$					

Entry (i, j) is accept if M_i accepts $\langle M_j \rangle$, and blank if M_i rejects or loops on $\langle M_j \rangle$.

Diagonalization (2)

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\dots$
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					
M_4	accept	accept			
$\vdots$					

Run H on on corresponding inputs. In new table, entry (i, j) states whether H accepts $\langle M_i, \langle M_j\rangle\rangle$.

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\dots$
M_1	accept	reject	accept	reject	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	
M_4	accept	accept	reject	reject	
$\vdots$		$\vdots$	$\vdots$	$\vdots$	

Diagonalization (3)

Now we add D to the table.

- By assumption, H is a TM, and therefore so is D.
- D occurs on the list $M_1, M_2, \ldots$ of all TMs.
- D computes the opposite of the diagonal entries.
- At diagonal entry, D computes its own opposite!

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\ldots$	$\langle D \rangle$
M_1	$\boxed{\text{accept}}$	reject	accept		
M_2	accept	$\boxed{\text{accept}}$	accept		
M_3	reject	reject	$\boxed{\text{reject}}$		
M_4	accept	accept	reject		
$\vdots$		$\vdots$		$\ddots$	
D	reject	reject	accept		$\boxed{\text{???}}$
$\vdots$		$\vdots$		$\ldots$	

♣

Halting vs Acceptance Problem

We have already established that A_{TM} is undecidable.

Here is a closely related problem.

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

Clarification: How does H_{TM} differ from A_{TM}?

Undecidable Problems

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

Theorem: H_{TM} is undecidable.

Proof idea:

- Again, proof by diagonalization.
- Will do this on the blackboard.
- Next time (Friday!!) will prove this differently, by a reduction from A_{TM}.