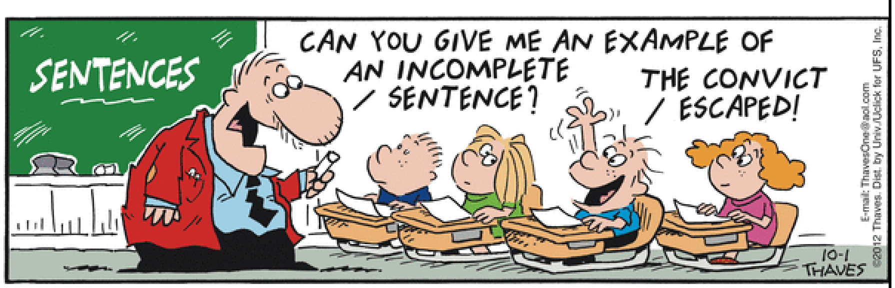
Variations:
- Multiple tracks
- Multiple tapes
- Non-deterministic

UNIVERSAL TURING MACHINE
CHURCH-TURING THESIS
THE HALTING PROBLEM

Your Questions?

- Previous class days' material
- Reading Assignments
- HW 14b problems
- Exam 3
- Anything else



An Encoding Example

Consider *M* = ({ *s*, *q*, *h* }, { a, b, c }, { Æ , a, b, c }, δ, *s*, { *h* }):

state	symbol	δ
s	Æ	(*q*, Æ , →)
s	a	(*s*, b, →)
s	b	(*q*, a, ←)
s	c	(*q*, b, ←)
q	Æ	(*s*, a,→)
q	a	(*q*, b, →)
q	b	(*q*, b, ←)
q	c	(*h*, a, ←)

state/symbol	representation
s	q00
q	q01
h	h10
Æ	a00
a	a01
b	a10
c	a11

Decision problem: Given a string w, is there a TM M such that w=<M> ?

Is this problem decidable?

<*M*> = (q00,a00,q01,a00,→), (q00,a01,q00,a10,→),
(q00,a10,q01,a01, ←), (q00,a11,q01,a10,←),
(q01,a00,q00,a01,→), (q01,a01,q01,a10,→),
(q01,a10,q01,a11,←), (q01,a11,h10,a01,←)

Encoding Multiple Inputs

Let:

$<x_1, x_2, \ldots x_n>$

represent a single string that encodes the sequence of individual values:

$x_1, x_2, \ldots x_n$.

The Specification of the Universal TM

On input <*M*, *w*>, *U* must:

- Halt iff *M* halts on *w*.

- If *M* is a deciding or semideciding machine, then:
    - If *M* accepts, accept.
    - If *M* rejects, reject.

- If *M* computes a function, then *U*(<*M*, *w*>) must equal *M*(*w*).

How *U* Works

U will use 3 tapes:

- Tape 1: *M*'s tape.

- Tape 2: <*M*>, the "program" that *U* is running.

- Tape 3: *M*'s state.

The Universal TM

	<M---------------------------M,				w------------------w>				
	1	0	0	0	0	0	0		
❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	❑	❑	❑	❑	❑	❑	❑		
	1	0	0	0	0	0	0		

Initialization of *U*:
1. Copy <*M*> onto tape 2.
2. Look at <*M*>, figure out what *i* is, and write the encoding of state *s* on tape 3.

After initialization:

	❑	❑	❑	❑	<w------------------w>				
	0	0	0	0	1	0	0		
❑	<M---------------------------M>				❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

The Operation of *U*

	❑	❑	❑	❑	<w------------------w>				
	0	0	0	0	1	0	0		
❑	<M---------------------------M>				❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

Simulate the steps of *M* :

1. Until *M* would halt do:
   - 1.1 Scan tape 2 for a quintuple that matches the current state, input pair.
   - 1.2 Perform the associated action, by changing tapes 1 and 3. If necessary, extend the tape.
   - 1.3 If no matching quintuple found, halt. Else loop.
2. Report the same result *M* would report.

How long does *U* take?

If A Universal Machine is Such a Good Idea …

Could we define a Universal Finite State Machine?

Such a FSM would accept the language:

$L = \{<F, w> : F \text{ is a FSM, and } w \in L(F)\}$

Enumerating Turing Machines

Theorem: There exists an infinite lexicographic enumeration of:

(a) All syntactically valid TMs.

(b) All syntactically valid TMs with specific input alphabet Σ.

(c) All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

Enumerating Turing Machines

Proof: Fix Σ = {`(`, `)`, a, q, y, n, `0`, `1`, comma, $\rightarrow$, $\leftarrow$}, ordered as listed. Then:

1. Lexicographically enumerate the strings in Σ^*.
2. As each string *s* is generated, check to see whether it is a syntactically valid Turing machine description. If it is, output it.

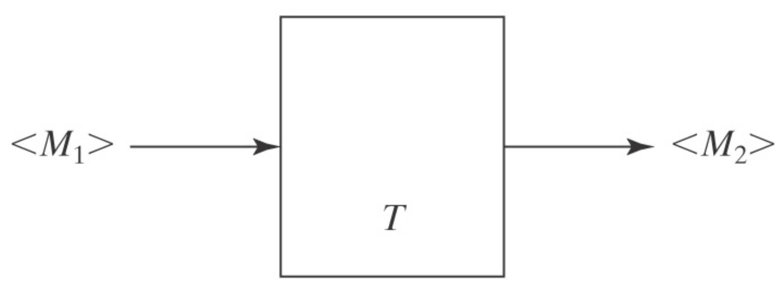
To restrict the enumeration to symbols in sets Σ and Γ, check, in step 2, that only alphabets of the appropriate sizes are allowed.

We can now talk about the i^{th} Turing machine.

Another Benefit of Encoding

Benefit of defining a way to encode any Turing machine *M*:

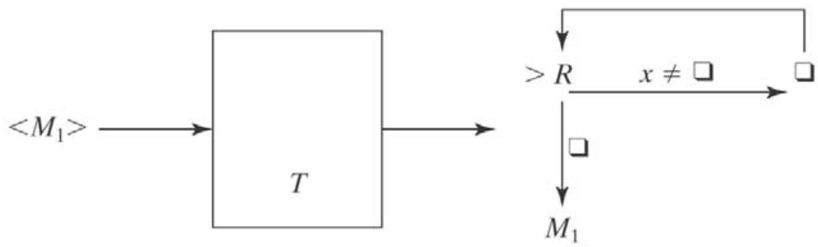
- We can talk about operations on programs (TMs).



Example of a Transforming TM *T*:

Input: a TM M_1 that reads its input tape and performs some operation P on it.

Output: a TM M_2 that performs P on an empty input tape.



The machine M_2 (output of T) empties its tape, then runs M_1.

The Church-Turing Thesis

Are We Done?

In this course: FSM $\Rightarrow$ PDA $\Rightarrow$ Turing machine

Is this the end of the line?

There are still problems that we cannot solve with a TM:

- There is a countably infinite number of Turing machines since we can lexicographically enumerate all the strings that correspond to syntactically legal Turing machines.
- There is an uncountably infinite number of languages over any nonempty alphabet.
- So there are more languages than there are Turing machines.

What Can Algorithms Do?

1. Can we come up with a system of axioms that makes all true statements be theorems (I.e. provable from the axioms)?

   The set of axioms can be infinite, but it must be decidable

2. Can we always decide whether, given a set of axioms, a statement is a theorem or not?

In the early 20th century, it was widely believed that the answer to both questions was "yes."

Gödel's Incompleteness Theorem

Kurt Gödel showed, in the proof of his Incompleteness Theorem [Gödel 1931], that the answer to question 1 is **no**. In particular, he showed that there exists no decidable axiomaitization of Peano arithmetic that is both consistent and complete.

Complete: All true statements in the language of the theory are theorems

The Entscheidungsproblem

From Wikipedia: The Entscheidungsproblem ("decision problem", David Hilbert 1928) asks for an algorithm that will take as input a description of a formal language and a mathematical statement in the language, and produce as output either "True" or "False" according to whether the statement is true or false. The algorithm need not justify its answer, nor provide a proof, so long as it is always correct.

Three equivalent formulations:

1. Does there exist an algorithm to decide, given an arbitrary sentence *w* in first order logic, whether *w* is valid?
2. Given a set of axioms *A* and a sentence *w*, does there exist an algorithm to decide whether *w* is entailed by *A*?
3. Given a set of axioms, *A*, and a sentence, *w*, does there exist an algorithm to decide whether *w* can be proved from *A*?

The Entscheidungsproblem

To answer the question, in any of these forms, requires formalizing the definition of an algorithm:

- Turing: Turing machines.
- Church: lambda calculus.

Turing proved that Turing machines and the lambda calculus are equivalent.

Church's Thesis (Church-Turing Thesis)

All formalisms powerful enough to describe everything we think of as a computational algorithm are equivalent.

This isn't a formal statement, so we can't prove it. But many different computational models have been proposed and they all turn out to be equivalent.

The Church-Turing Thesis

Examples of equivalent formalisms:

- Modern computers (with unbounded memory)
- Lambda calculus
- Partial recursive functions
- Tag systems (FSM plus FIFO queue)
- Unrestricted grammars:

  $aSa \rightarrow B$
- Post production systems
- Markov algorithms
- Conway's Game of Life
- One dimensional cellular automata
- DNA-based computing
- Lindenmayer systems

The Lambda Calculus

The successor function:

$(\lambda\ x.\ x + 1)\ 3 = 4$

Addition: $(\lambda\ x.\ \lambda\ y.\ x + y)\ 3\ 4$

This expression is evaluated by binding 3 to x to create the new function ($\lambda\ y.\ 3 + y$), which is applied to 4 to return 7.

In the pure lambda calculus, there is no built in number data type. All expressions are functions. But the natural numbers can be defined as lambda calculus functions. So the lambda calculus can effectively describe numeric functions.

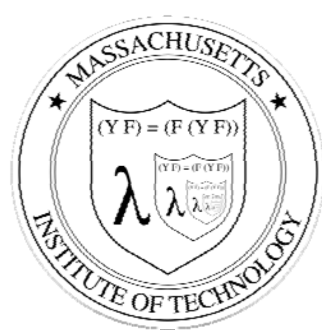
The Lambda Calculus

```
> (define Y
    (lambda (f)
      ((lambda (x) (f (lambda (y) ((x x) y))))
       (lambda (x) (f (lambda (y) ((x x) y)))))))
> (define H
    (lambda (g)
      (lambda (n)
        (if (zero? n)
            1
            (* n (g (- n 1)))))))
> ((Y H) 5)
120
>
```

Λ-Calculus in Scheme

The Applicative Y Combinator

```
> (((lambda (f)
      ((lambda (x) (f (lambda (y) ((x x) y))))
       (lambda (x) (f (lambda (y) ((x x) y))))))
    (lambda (g)
      (lambda (n)
        (if (zero? n)
            1
            (* n (g (- n 1)))))))
   5)
120
```



Tag Systems

A tag system (or a Post machine) is an FSM augmented with a FIFO queue.

Simple for WW:
Not so simple for PalEven

The Power of Tag Systems

Tag systems are equivalent in power to Turing machines because the TM's tape can be simulated with the FIFO queue.

Suppose that we put abcde into the queue:

	a	b	c	d	e	

To read the queue, we must remove the a first.

But suppose we want to remove e first:

The Power of Tag Systems

Tag systems are equivalent in power to Turing machines because the TM's tape can be simulated with the FIFO queue.

Suppose that we push abcde onto the queue:

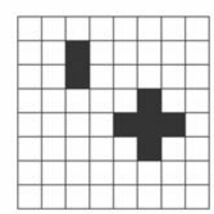
	a	b	c	d	e	

To read the queue, we must remove the a first.

But suppose we want to remove e first:

Treat the queue as a loop.

The Game of Life



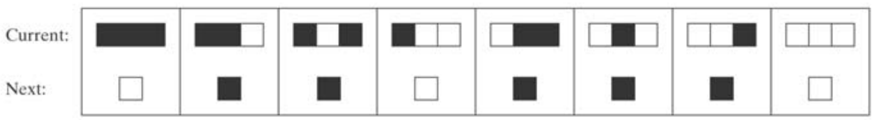
At each step of the computation, the value for each cell is determined by computing the number of neighbors (up to a max of 8) it currently has, according to the following rules:

- A dead cell with exactly three live neighbors becomes a live cell (birth).
- A live cell with two or three live neighbors stays alive (survival).
- In all other cases, a cell dies or remains dead (overcrowding or loneliness).

We'll say that a game halts iff it reaches some stable configuration.

Elementary Cellular Automata

Wolfram's Rule 110 is a universal computer, if you can figure out how to encode the program and the input in the initial configuration:

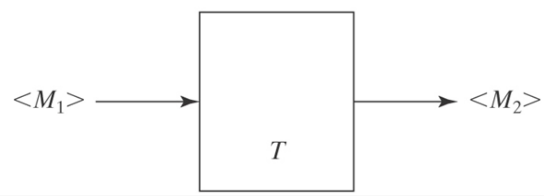


For some fascinating pictures, look up Rule 110.
Conjectured in 1985 to be Turing complete, proved in 2000 by Matthew Cook.
Also: http://en.wikipedia.org/wiki/A_New_Kind_of_Science

Background for The Halting Problem

Key ideas so far

- Let Σ_U be
  `{(, ), a, q, y, n, h, 0, 1, comma,` →, ←},
  ordered as listed
- Any TM *M* may be encoded as a string <*M*> over alphabet Σ_U
- We can design a TM *T* to take as input <M_1>, an encoding of TM M_1, and produce as output <M_2>, an encoding of TM M_2



Key ideas so far 2

- We can lexicographically enumerate:
  - All TM encodings
  - All TM encodings with a given input alphabet
  - All TM encodings with a given input alphabet and a given tape alphabet
- For any TM *M* and any string w over *M*'s input alphabet, we can encode the pair *M*, *w* as a single string <*M*, *w*>

Key ideas so far 3

- There is a **universal TM** *U* whose input alphabet is $\Sigma_{U.}$
- If *U* is started with input <*M*,*w*>, it simulates the behavior of *M*, started with input *w*:
  - If *M* does not halt, *U* does not halt
  - If *M* halts and accepts, so does *U*
  - If *M* halts and rejects, so does *U*
  - If *M* is a "function computing" TM, then *U* leaves the same string on the tape that *M* would leave, so that *U*(<*M*, *w*>) = *M*(*w*)
- **Church-Turing Thesis (brief version):** "Computable" is equivalent to "computable by a Turing machine"

Recap: D and SD

- A TM *M* with input alphabet Σ ***decides*** a language $L \subseteq \Sigma^*$ iff, for any string $w \in \Sigma^*$,
  - if $w \in L$ then *M* accepts *w*, and
  - if $w \notin L$ then *M* rejects *w*.

A language *L* is ***decidable*** (an element of **D**) iff there is a Turing machine *M* that decides it.

- A TM *M* with input alphabet Σ ***semidecides*** *L* iff for any string $w \in \Sigma^*$,
  - if $w \in L$ then *M* accepts *w*
  - if $w \notin L$ then *M* does not accept *w*. *M* may reject or loop.

A language *L* is ***semidecidable*** (an element of **SD**) iff there is a Turing machine that semidecides it.

Defining the Universe

What is the complement of:

•$A^nB^n = \{a^nb^n : n \geq 0\}$?

Depends on the universe:
That universe may be {a, b}*, or even {a, b, c, d}*, or could be $\{a^kb^m\}$

•{<*M*, *w*> : TM *M* halts on input string *w*}

Universe may be Σ_U*, or could be

{<*M*, *w*> : M is a TM and w is a string over M's input alphabet}

Defining the Universe

L_1 = {<*M*, *w*> : TM *M* halts on input string *w*}.
L_2 = {<*M*> : *M* doesn't halt on any input string}.
L_3 = {<M_a, M_b> : M_a and M_b halt on the same strings}.

For a string *w* to be in L_1, it must:

- be syntactically well-formed.
- encode a machine *M* and a string *w* such that *M* halts when started on *w*.

Define the universe from which we are drawing strings to contain only those strings that meet the syntactic requirements of the language definition.

This convention has no impact on the decidability of any of these languages since the set of all syntactically valid strings is clearly in D.

A Different Definition of Complement

Our earlier definition:

$\neg L_1$ = {x: x is not a syntactically well formed <M, w> pair} $\cup$ {<M, w> : TM M does not halt on input string w}.

We will use a different definition:

Define the ***complement*** of any language L whose member strings include at least one Turing machine description to be with respect to a universe of strings that are of the same syntactic form as L.

Now we have:

$\neg L_1$ = {<M, w> : TM M does not halt on input string w}.

Does This Program Always Halt?

times3(x: positive integer) =
while $x \neq 1$ do:
if x is even then $x = x/2$.
else $x = 3x + 1$

times3(25) …

Lothar Collatz, 1937, conjectured that times3 halts for all positive integers n. Still an open problem.

Paul Erdős: "Mathematics is not yet ready for such confusing, troubling, and hard problems."

http://mathworld.wolfram.com/CollatzProblem.html

```
max = 100000
maxCount = 0
for i in range(1, max+1):
    current = i
    count = 0

    while current != 1:
        count += 1
        if current % 2 == 0:
            current /= 2
        else:
            current = 3 * current + 1

    print "%7d %7d" % (i, count)
    if count > maxCount:
        maxCount = count

print "maxCount = ", maxCount
```

Collatz function example

27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, ***9232***, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

The Halting Problem

(Expressed as The Halting Language)

The Language H

Theorem: The language:

H = {<*M*, *w*> : TM *M* halts on input string *w*}

- is semidecidable, but
- is not decidable.

Proof soon! *via* two lemmas ..

We know that we can decide the halting question for *specific* simple TMs.

Or can we … ?

H is Semidecidable

Lemma: The language:

H = {<*M*, w> : TM *M* halts on input string *w*}

is semidecidable.

Proof: The TM M'_H semidecides H:

M'_H(<*M*, *w*>) =

1. Use U to run *M* on *w*
2. Accept.

M'_H accepts <*M*, *w*> iff *M* halts on *w*. Thus M'_H semidecides H.

H is Not Decidable

Lemma: The language:

H = {<*M*, *w*> : TM *M* halts on input string *w*}

is not decidable.

Contradiction
Specification of *halts*
Trouble ***[in (Wabash) River City)]***
halts(<*Trouble*, *Trouble*>) - what happens?