Turing Machines, Continued

April 13, 2011

CS 81: Computability and Logic

TM Definition (Sipser)

A Deterministic TM consists of

- ✓ A finite set Q of control states
- ✓ A finite alphabet Σ
- ✓ A finite "tape alphabet" Γ $(\Sigma \subset \Gamma,\ \sqcup \in \Gamma \setminus \Sigma)$
- ✓ A starting state $q_0 \in Q$
- ✓ Accepting/Rejecting (halting) states $q_{accept}, q_{reject} \in Q$
- ✓ Transitions $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

TM Variations

The following yield no extra power:

- ✓ Adding the option to write or not
- ✓ Adding the option to stay-in-place rather than moving L/R.
- ✓ Making the tape infinite in both directions
- ✓ Adding an extra "Erase Tape" move.
- ✓ Multiple tapes with multiple (independent) read/write heads
- ✓ Finitely many registers with finite capacity
- ✓ 2-D infinite tape
- ✓ Nondeterminism (!)

Many attempts to define models of computation; all turn out to be equivalent to Turing Machines.

- ✓ If you can do it in C++, a TM can do it (slowly, encodedly)

TMs as Strings

A TM can be described by the finite list of transition rules:

```
q0, ␣ → q1, ␣, R
q1, ␣ → q0, ␣, L
q1, x → q2, x, L
q2, a → q1, x, R
```

We can devise a way to encode an arbitrary set of such rules into a fixed alphabet. Although the number of states and tape symbols can be arbitrary

- ✓ we can encode these by using strings of symbols like $\{0, 1\}$
- ✓ concatenate the symbols to describe rules
- ✓ concatenate rules to describe machines.

Configurations as Strings

A configuration can be expressed as a finite string, e.g.,

$011q01\sqcup110$

means

- ✓ The contents of the tape is $01101\sqcup110$ (padded on the right with blanks)
- ✓ The TM is in control state q
- ✓ The TM is pointing to the second 0.

Universal Turing Machines

UTMs can be shown to exist by constructing them.

Think about what would be required.

- ✓ The tape has to hold the tape of the machine being simulated.
- ✓ The tape has to hold the program of the machine being simulated.
- ✓ The tape has to hold the current state of the machine being simulated.

All this is possible, if somewhat laborious to construct.

Specific UTMs

- The first was constructed by Turing himself.

Specific UTMs

- ✓ The first was constructed by Turing himself.
- ✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).

Specific UTMs

- The first was constructed by Turing himself.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- Minsky (1960) gave a 7-state 6-symbol machine.

Specific UTMs

- The first was constructed by Turing himself.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- Minsky (1960) gave a 7-state 6-symbol machine.
- Watanabe (1961) gave an 8-state 5-symbol machine.

Specific UTMs

- ✓ The first was constructed by Turing himself.
- ✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- ✓ Minsky (1960) gave a 7-state 6-symbol machine.
- ✓ Watanabe (1961) gave an 8-state 5-symbol machine.
- ✓ Minsky (1962) gave a 7-state 4-symbol machine.

Specific UTMs

- The first was constructed by Turing himself.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- Minsky (1960) gave a 7-state 6-symbol machine.
- Watanabe (1961) gave an 8-state 5-symbol machine.
- Minsky (1962) gave a 7-state 4-symbol machine.
- Rogozhin (1996) gave a 4-state 6-symbol machine

Specific UTMs

- ✓ The first was constructed by Turing himself.
- ✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- ✓ Minsky (1960) gave a 7-state 6-symbol machine.
- ✓ Watanabe (1961) gave an 8-state 5-symbol machine.
- ✓ Minsky (1962) gave a 7-state 4-symbol machine.
- ✓ Rogozhin (1996) gave a 4-state 6-symbol machine
- ✓ Wolfram and Reed (2002) gave a 2-state 5-symbol machine.

Specific UTMs

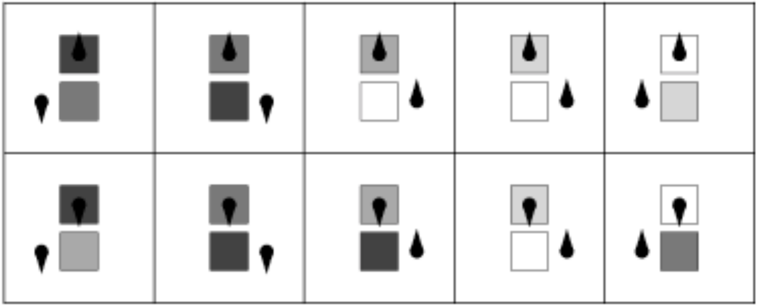
- ✓ The first was constructed by Turing himself.
- ✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- ✓ Minsky (1960) gave a 7-state 6-symbol machine.
- ✓ Watanabe (1961) gave an 8-state 5-symbol machine.
- ✓ Minsky (1962) gave a 7-state 4-symbol machine.
- ✓ Rogozhin (1996) gave a 4-state 6-symbol machine
- ✓ Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
- ✓ Smith and Wolfram (2007) gave a 2-state 3-symbol machine.

Specific UTMs

- The first was constructed by Turing himself.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- Minsky (1960) gave a 7-state 6-symbol machine.
- Watanabe (1961) gave an 8-state 5-symbol machine.
- Minsky (1962) gave a 7-state 4-symbol machine.
- Rogozhin (1996) gave a 4-state 6-symbol machine
- Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
- Smith and Wolfram (2007) gave a 2-state 3-symbol machine.
- No 2-state 2-symbol UTM exists.

A Specific UTM

2-state, 5 symbol UTM published by Wolfram in 2002



adapted from Wolfram, S. *A New Kind of Science.* Wolfram Media, p. 707, 2002.

Turing Machines as Enumerators

Several variant definitions. Each specify a language $\mathbf{L}$.

1. A TM that prints out all the members of L, *one at a time* (but not necessarily in any particular order)
2. A TM that prints out all the members of L, *one at at time* (but...) with arbitrarily many repeats.
3. A TM that, *given an integer n*, returns the nth element of a sequence like (1) above.
4. A TM that, *given an integer n*, returns the nth element of a sequence like (2) above.

For a fixed language, all these are interconvertable.

Theorem

A language is recognizable $\iff$ *it can be enumerated.*

Computability and Uncomputability

April 13, 2011

CS 81: Computability and Logic

Question

How is my iPad more like a Finite State Machine than like a Turing Machine?

How is my iPad more like a Turing Machine than like a Finite State Machine?

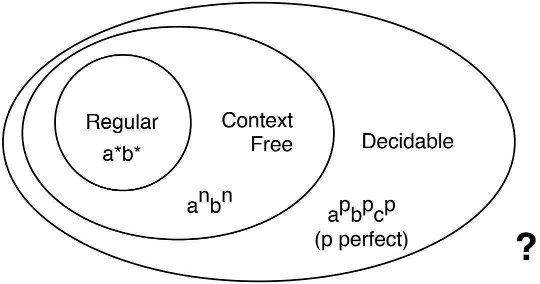
Church-Turing Thesis

If it can be done at all, then it can be done by

- ✓ A Turing Machine
- ✓ Lambda Calculus
- ✓ An Unrestricted Grammar
- ✓ A 2-register machine
- ✓ C
- ✓ …

(Note: assumes suitably coded inputs and outputs)

Is There More?



Some Languages Aren't Decidable

Given a finite Σ, how many strings are there?

Some Languages Aren't Decidable

Given a finite Σ, how many strings are there?

Given a finite Σ, how many languages are there?

Some Languages Aren't Decidable

Given a finite Σ, how many strings are there?

Given a finite Σ, how many languages are there?

How many TMs are there?

Some Languages Aren't Decidable

Given a finite Σ, how many strings are there?

Given a finite Σ, how many languages are there?

How many TMs are there?

QED

But Wait…

How many languages over Σ could I describe (say, in a LaTeX document)?

But Wait...

How many languages over Σ could I describe (say, in a LaTeX document)?

How many TMs are there?

But Wait…

How many languages over Σ could I describe (say, in a LaTeX document)?

How many TMs are there?

QED?

Languages of Acceptance

Which are recognizable (by a TM)? Decidable?

- ✓ $A_{\mathrm{DFA}} = \{\, \langle \mathrm{D}, w \rangle \mid \mathrm{D}$ a DFA, D accepts $w \,\}$
- ✓ $A_{\mathrm{NFA}} = \{\, \langle \mathrm{N}, w \rangle \mid \mathrm{N}$ an NFA, N accepts $w \,\}$
- ✓ $A_{\mathrm{RE}} = \{\, \langle \mathrm{R}, w \rangle \mid \mathrm{R}$ a regexp, R matches $w \,\}$
- ✓ $A_{\mathrm{CFG}} = \{\, \langle \mathrm{G}, w \rangle \mid \mathrm{G}$ a CFG, G produces $w \,\}$
- ✓ $A_{\mathrm{TM}} = \{\, \langle \mathrm{M}, w \rangle \mid \mathrm{M}$ a TM, M accepts $w \,\}$

Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

- ✓ Gnu C Compiler (used in CS 105) is written in C

Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

- ✓ Gnu C Compiler (used in CS 105) is written in C
- ✓ Glasgow Haskell Compiler (used in CS 131) is in Haskell etc.

Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

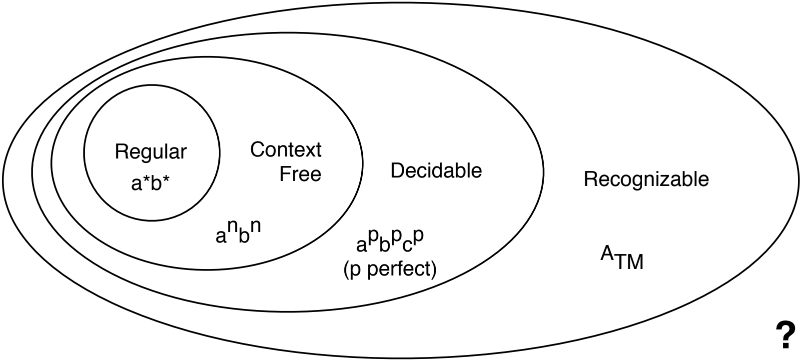
- ✓ Gnu C Compiler (used in CS 105) is written in C
- ✓ Glasgow Haskell Compiler (used in CS 131) is in Haskell etc.

Practical reasons to run programs on their own source code!

A_{TM} IS NOT DECIDABLE

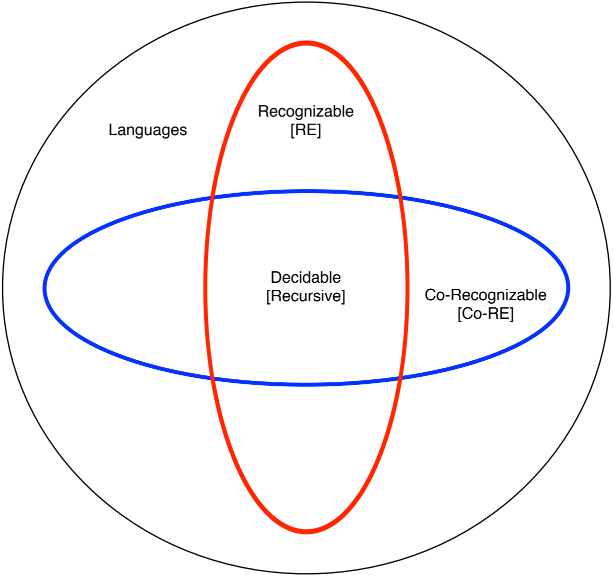
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_0	Acc.		Acc.	Acc.		
M_1						
M_2	Acc.			Acc.		
M_3	Acc.	Acc.	Acc.	Acc.	Acc.	
M_4			Acc.	Acc.		

Is There More?

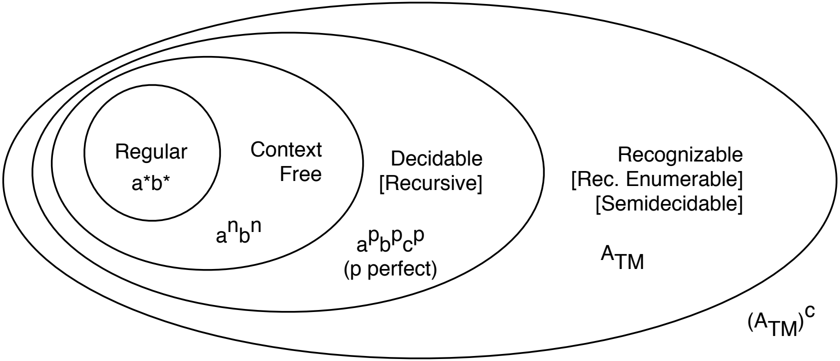


What is the complement of A_{TM}?

What is the complement of A_{TM}?



We'll Stop Here



Obligatory Corollary

Theorem

The language

$$H = \{ \langle M, w \rangle \mid M \text{ a TM}, M \text{ halts on } w \}$$

is not decidable.

Obligatory Corollary

Theorem

The language

$$H = \{ \langle M, w \rangle \mid M \text{ a TM}, M \text{ halts on } w \}$$

is not decidable.

Proof.

Suppose there were a halt-checking TM… □