CSE 311: Foundations of Computing

Lecture 29: Turing Machines
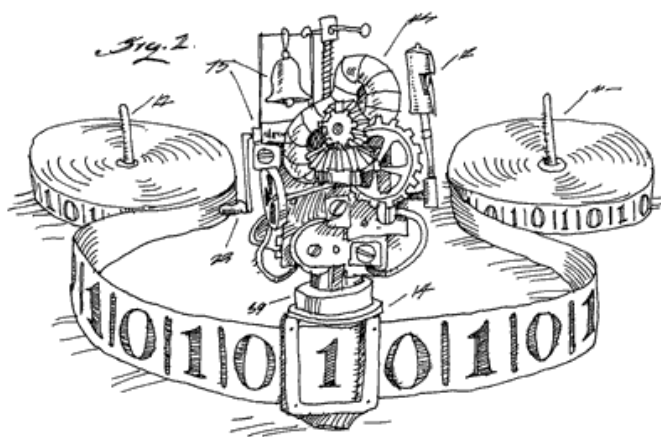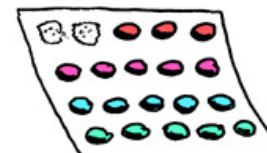
Final exam

- **Monday** at either **2:30-4:20 p.m.** or **4:30-6:20 p.m.**
  - **Kane Hall 220**
  - You need to fill out **Catalyst Survey** to say which you are taking by **midnight Sunday** night.
  - Bring your **UW ID** and have it out and ready during the exam
- **Comprehensive** coverage. If you had a homework question on it, it is fair game. See link on webpage.
  - Includes pre-midterm topics, e.g. formal proofs. Will contain the same sheets at end.
- **Review session: Sunday 4:00-6:00 p.m. EEB 105**
  - **Bring your questions !!**

Homework

- Homework 9
  - Due today at 5:00 p.m.
  - Grading won't be finished until after the final

- Look for `cse311` e-mail with pointer to solutions for Homework 9 overnight tonight
  - Will be on password-protected webpage

Computers and algorithms

- Does Java (or any programming language) cover all possible computation? Every possible algorithm?

- There was a time when computers were people who did calculations on sheets paper to solve computational problems
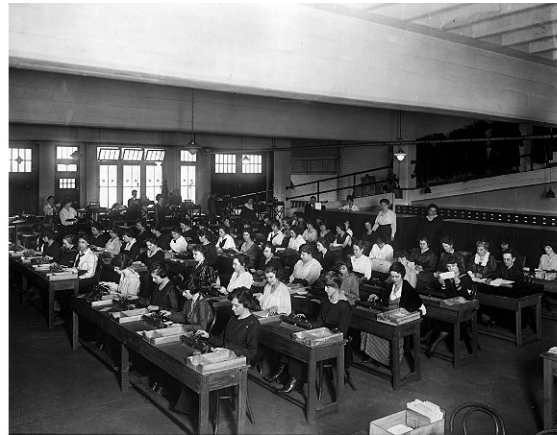
- Computers as we known them arose from trying to understand everything these people could do.

Before Java

1930's:

How can we formalize what algorithms are possible?

- **Turing machines** (Turing, Post)
  - basis of modern computers
- **Lambda Calculus** (Church)
  - basis for functional programming, LISP
- **µ-recursive functions** (Kleene)
  - alternative functional programming basis

Turing machines

Church-Turing Thesis:

> Any reasonable model of computation that includes all possible algorithms is equivalent in power to a Turing machine

Evidence

– Intuitive justification

– Huge numbers of models based on radically different ideas turned out to be equivalent to TMs

Turing machines

- ## Finite Control
  - Brain/CPU that has only a **finite # of possible "states of mind"**
- ## Recording medium
  - **An unlimited supply of** blank **"scratch paper"** on which to **write & read symbols,** each chosen from a **finite set of possibilities**
  - **Input** also supplied **on** the **scratch paper**
- ## Focus of attention
  - **Finite control** can only **focus on** a **small portion of the recording medium at once**
 - **Focus of attention** can only **shift a small amount at a time**

Turing machines

- **Recording medium**
  - An **infinite** read/write **"tape"** marked off into **cells**
  - Each **cell** can store **one symbol or** be **"blank"**
  - **Tape** is **initially** all **blank except** a few **cells** of the tape **containing the input string**
  - **Read/write head** can **scan one cell** of the tape - **starts on input**

- **In each step,** a Turing machine
  1. **Reads** the **currently scanned cell**
  2. **Based on current state and scanned symbol**
     i. **Overwrites symbol in scanned cell**
     ii. **Moves read/write head left or right one cell**
     iii. **Changes to a new state**

- **Each Turing Machine is specified by its finite set of rules**

Turing machines

	_	0	1
s_1	$(1, L, s_3)$	$(1, L, s_4)$	$(0, R, s_2)$
s_2	$(0, R, s_1)$	$(1, R, s_1)$	$(0, R, s_1)$
s_3			
s_4			

_	_	1	1	0	1	1	_	_
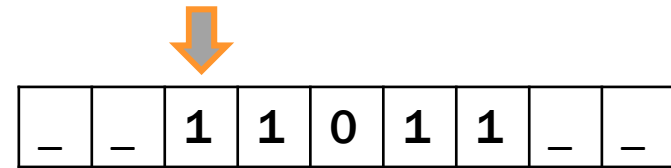
UW CSE's Steam-Powered Turing Machine

Original in Sieg Hall stairwell

Turing machines

Ideal Java/C programs:

- Just like the Java/C you're used to programming with, except you never run out of memory
  - Constructor methods always succeed
  - **malloc** in C never fails

Equivalent to Turing machines except a lot easier to program:

- Turing machine definition is useful for breaking computation down into simplest steps
- We only care about high level so we use programs

Turing's big idea part 1: Machines as data

Original Turing machine definition:

– A different "machine" **M** for each task

– Each machine **M** is defined by a finite set of possible operations on finite set of symbols

– So... **M** has a finite description as a sequence of symbols, its "code", which we denote **\<M\>**

You already are used to this idea with the notion of the program code or text but this was a new idea in Turing's time.

Turing's big idea part 2: A Universal TM

- **A Turing machine interpreter U**
  - On input **<M>** and its input **x**,
    U outputs the same thing as **M** does on input **x**
  - At each step it decodes which operation **M** would have performed and simulates it.

- **One Turing machine is enough**
  - Basis for modern stored-program computer

    Von Neumann studied Turing's UTM design

input   output

x ⟶ **M** ⟶ **M(x)**

x ⟶   output

<M> ⟶ **U** ⟶ **M(x)**
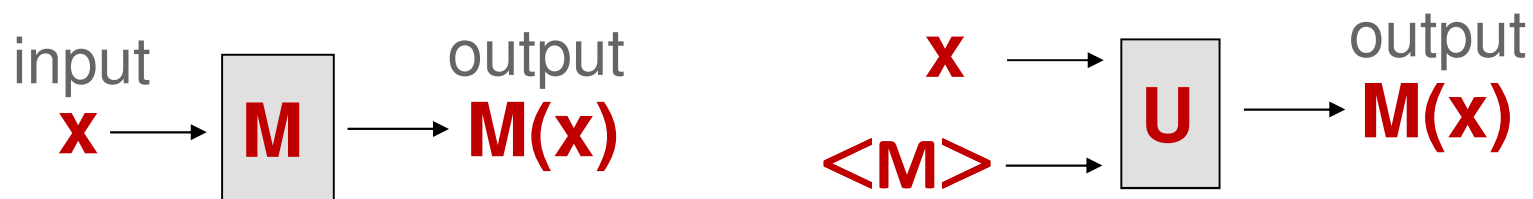
Takeaway from undecidability

- **You can't rely on the idea of improved compilers and programming languages to eliminate major programming errors**
  - truly safe languages can't possibly do general computation

- **Document your code**
  - there is no way you can expect someone else to figure out what your program does with just your code; since in general it is provably impossible to do this!

We've come a long way!

- Propositional Logic.

- Boolean logic and circuits.

- Boolean algebra.

- Predicates, quantifiers and predicate logic.

- Inference rules and formal proofs for propositional and predicate logic.

- English proofs.

- Set theory.

- Modular arithmetic.

- Prime numbers.

- GCD, Euclid's algorithm, modular inverse, and exponentiation.

We've come a long way!

- Induction and Strong Induction.

- Recursively defined functions and sets.

- Structural induction.

- Regular expressions.

- Context-free grammars and languages.

- Relations and composition.

- Transitive-reflexive closure.

- Graph representation of relations and their closures.

We've come a long way!

- DFAs, NFAs and language recognition.
- Product construction for DFAs.
- Finite state machines with outputs at states.
- Minimization algorithm for finite state machines
- Conversion of regular expressions to NFAs.
- Subset construction to convert NFAs to DFAs.
- Equivalence of DFAs, NFAs, Regular Expressions
- Finite automata for pattern matching.
- Method to prove languages not accepted by DFAs.
- Cardinality, countability and diagonalization
- Undecidability: Halting problem and evaluating properties of programs.

What's next? ...after the final exam...

- **Foundations II** (312)
  - Fundamentals of counting, discrete probability, applications of randomness to computing, statistical algorithms and analysis
  - Ideas critical for machine learning, algorithms

- **Data Abstractions** (332)
  - Data structures, a few key algorithms, parallelism
  - Brings programming and theory together
  - Makes heavy use of induction and recursive defns

Course Evaluation Online

- **Fill this out by Sunday night!**
  - Your ability to fill it out will disappear at 11:59 p.m. on Sunday.
  - It will be worth your while to fill it out!

- **Look for an important message on the cse311 mailing list about course evaluations this evening**

Final exam

- **Monday** at either **2:30-4:20 p.m.** or **4:30-6:20 p.m.**
  - **Kane Hall 220**
  - You need to fill out **Catalyst Survey** to say which you are taking by **midnight Sunday** night.
  - Bring your **UW ID** and have it out and ready during the exam
- **Comprehensive** coverage. If you had a homework question on it, it is fair game. See link on course webpage.
  - Includes pre-midterm topics, e.g. formal proofs. Will contain the same sheets at end.
- **Review session: Sunday 4:00-6:00 p.m. EEB 105**
  - **Bring your questions !!**