“Computer Science is no more about computers than astronomy is about telescopes.”

- Often attributed to Edsger Dijkstra

Foundations of Computing

“I fear - as far as I can tell - that most undergraduate degrees in computer science these days are basically Java vocational training.” - Alan Kay

Turing machines

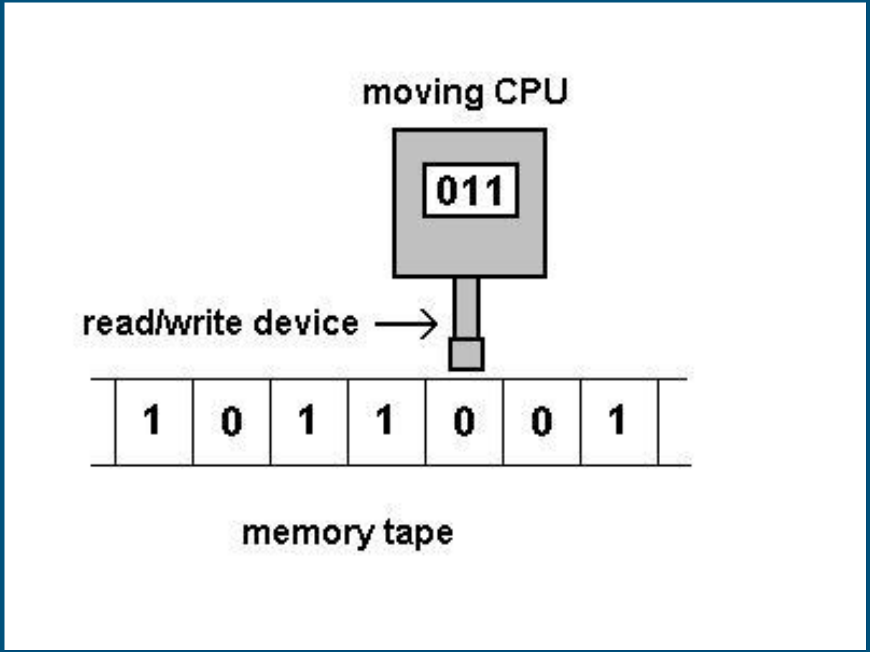
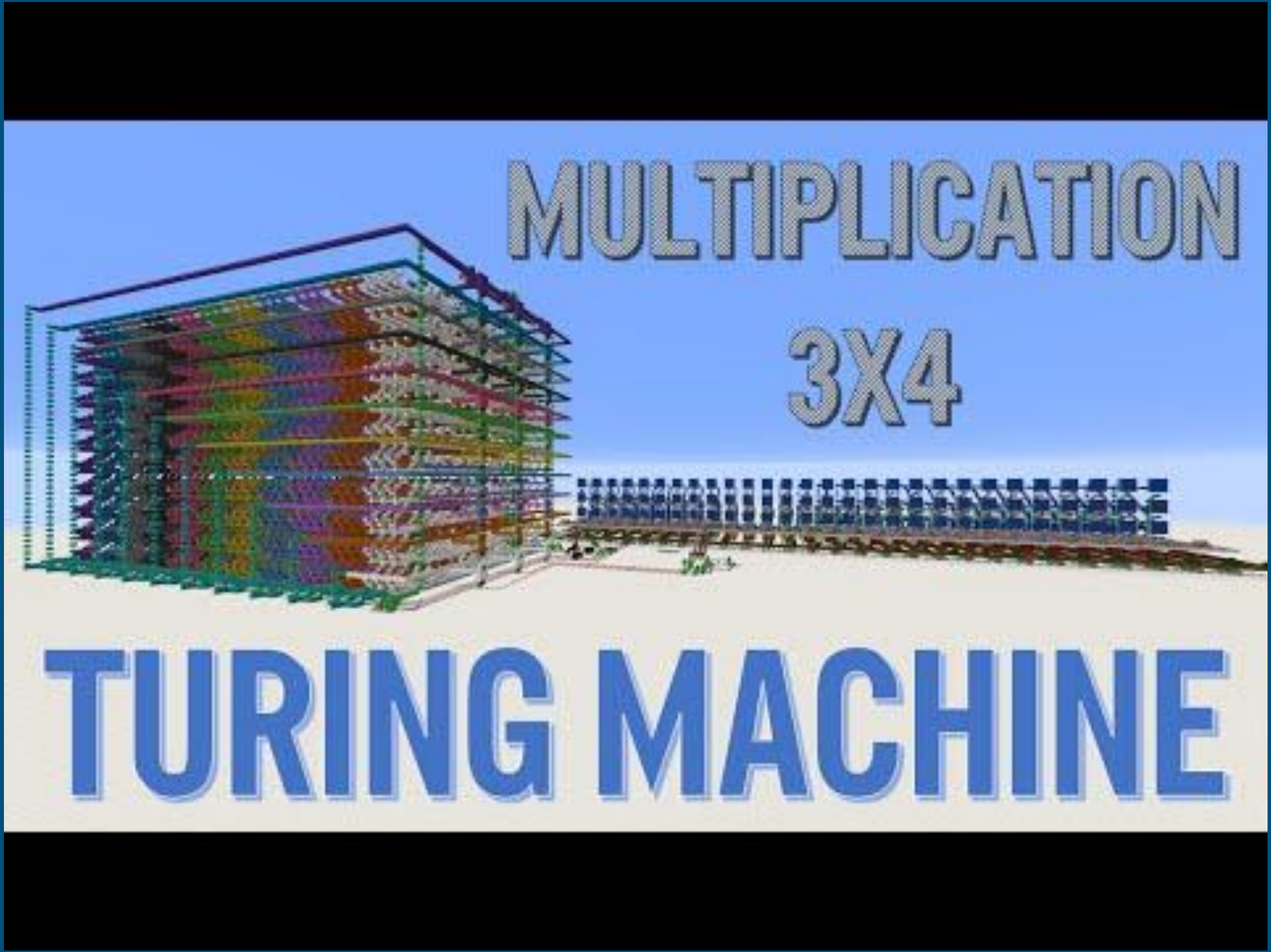


Image from 2009.igem.org

A Turing Machine

- If you are in state 1 and reading a 0, move right and change to state 2.
- If you are in state 1 and reading a 1, write a 0.
- If you are in state 2 and reading a 0, write a 1 and change to state 3.
- If you are in state 2 and reading a 1, move right.
- If you are in state 3 and reading a 0, move right and halt.
- If you are in state 3 and reading a 1, move left.


MULTIPLICATION
3X4
TURING MACHINE

Register Machines

Sample Program:

Add the contents of register 1 to the contents of register 2

STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	1	2	3
2.	Inc	2	1	
3.	End			

Equivalence Theorem

A function is computed by some Turing machine if and only if it is computed by some register machine.

The Lambda Calculus

Idea: introduce a notation for functions

$\lambda x.x^2$ - the function that squares any number

$\lambda x.\lambda y.x+y$ - the function that, given two numbers, returns their sum

The Lambda Calculus

Expression $s,t ::= x \mid st \mid \lambda x.t$

Rule α: λx.---x--- is equal to λy.---y---

So $\lambda x.x^2 = \lambda y.y^2$

Rule β: $(\lambda x.s)t$ is equal to $s[t/x]$, the result of substituting t for x in s

So $(\lambda x.x^2)4 = 4^2$

Coding for Numbers

We can code numbers as lambda-calculus expressions:

$0 = \lambda x.\lambda y.y$
$1 = \lambda x.\lambda y.xy$
$2 = \lambda x.\lambda y.x(xy)$
$3 = \lambda x.\lambda y.x(x(xy))$

Now, what do these do?

$\lambda x.\lambda y.\lambda z.\lambda w.xz(yzw)$
$\lambda x.\lambda y.xy$

Equivalence Theorem

The following are all equal:

- The set of functions computed by Turing machines
- The set of functions computed by register machines
- The set of functions computed by lambda-calculus expressions
- The set of functions computed by Post canonical systems
- The set of functions computed by Petri nets
- .....

Church-Turing Thesis

A function is computable by a human being following some **algorithm** if and only if it is computable by a Turing machine.

The Halting Problem

Given a Turing machine M and input n, decide if Turing machine M will halt when started with input n.

More precisely:

Assign a natural number to every Turing machine T_0, T_1, T_2, ...

Given numbers m, n, decide if Turing machine T_m will halt when started with input n

The Halting problem is **not** Turing computable!

Suppose Turing machine M:

- given input m and n
- outputs 1 if T_m halts with input n and 0 if it does not

Let H be the machine which, given input n:

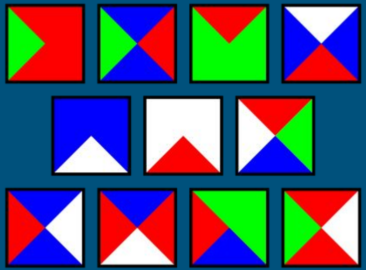
1. Creates a copy, so the tape is n 1s, then 0, then n 1s
2. Follows the operations of M
3. If the tape has a 1, go into an infinite loop. If the tape has a 0, halt.

Let H be Turing machine T_h. Does H halt when given input h?

Other uncomputable functions

The following problems are uncomputable:

- The Halting problem
- Given a set of Wang tiles, can they cover the plane?
- Given a Diophantine equation, does it have a solution?
- The **Busy Beaver** function:
  BB(n) = the largest number k such that
  there exists a Turing machine with n states
  that outputs k when started with a blank tape



$$3x^2 - 2xy - y^2 z - 7 = 0$$

Images from Wikipedia

P vs NP

A decision problem is a function with outputs 0 and 1.

Let P be the set of all decision problems that can be computed by a Turing machine in **polynomial time**.

Let NP be the set of all decision problems that can be computed by a non-deterministic Turing machine in polynomial time.

Is P = NP?

$1,000,000 if you can find the answer...

Type Theory

The Typed Lambda-Calculus

Add a notion of types (sets) to the lambda calculus.

$x_1{:}A_1 , \ldots , x_n{:}A_n \vdash t{:}B$

Example:

$x : A \longrightarrow B, y{:}A \vdash xy : B$

What rules should these judgements obey?

Rules for the Typed Lambda Calculus

1. If $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t{:}C$ then $x_1{:}A_1, \ldots, x_n{:}A_n \vdash \lambda y.t : B \rightarrow C$

2. If $x_1{:}A_1, \ldots, x_n{:}A_n \vdash s{:}B \rightarrow C$ and $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t{:}B$ then $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash st{:}C$

The same as the rules for IF...THEN in logic

“A remarkable correspondence” - Curry, 1958

More Rules!

1. If $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash s{:}C$ and $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t{:}D$ then $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash (s,t){:}C \times D$

2. If $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t{:}C \times D$ then $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t_1 : C$

3. If $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t{:}C \times D$ then $x_1{:}A_1, \ldots, x_n{:}A_n, y{:}B \vdash t_2 : D$

The same as the rules of AND in logic!

The Curry-Howard Isomorphism

Logic	Type Theory
IF A THEN B	Functions from A to B
A AND B	Pairs AxB
A OR B	Disjoint union $A \uplus B$
For all x, P(x)	Dependent function type $\Pi x.P(x)$
Proposition	Type
Proof	Program
...	...

Type theory-based languages

- **Agda** (developed here at Chalmers)
- also Coq, Idris, HOL, …

Both a programming language and a theorem prover!

Formalization of Mathematics

```
proof
  let "?S = {x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where fy: "?S = f y" ..
    show False
    proof cases
      assume "y ∈ ?S"
      hence "y ∉ f y"     by simp
```

Image from vdash.org

Correct-by-construction Programming

$$
\begin{array}{llcl}
\multicolumn{4}{l}{\textbf{interp} \;:\; \textbf{REnv}\ ty_{in} \rightarrow \mathsf{Lang}\ ty_{in}\ ty_{out}\ T \rightarrow \mathsf{IO}\ (\mathsf{Pair}\ (\textbf{REnv}\ ty_{out})\ (\textbf{interpTy}\ T))} \\
\textbf{interp}\ env & (\mathsf{READ}\ p) & \mapsto & \underline{\mathrm{do}}\ val \leftarrow \textbf{readIORef}\ (\textbf{rlookup}\ p\ env) \\
 & & & \quad \textbf{return}\ (\mathsf{MkPair}\ env\ val) \\
\textbf{interp}\ env & (\mathsf{WRITE}\ v\ p) & \mapsto & \underline{\mathrm{do}}\ \textbf{writeIORef}\ (\textbf{rlookup}\ p\ env)\ v \\
 & & & \quad \textbf{return}\ (\mathsf{MkPair}\ env\ ()) \\
\textbf{interp}\ env & (\mathsf{LOCK}_i\ p\ pri) & \mapsto & \underline{\mathrm{do}}\ \textbf{lock}\ (\textbf{llookup}\ i\ env) \\
 & & & \quad \textbf{return}\ (\mathsf{MkPair}\ (\textbf{lockEnv}\ p\ env)\ ()) \\
\textbf{interp}\ env & (\mathsf{UNLOCK}_i\ p) & \mapsto & \underline{\mathrm{do}}\ \textbf{unlock}\ (\textbf{llookup}\ i\ env) \\
 & & & \quad \textbf{return}\ (\mathsf{MkPair}\ (\textbf{unlockEnv}\ p\ env)\ ()) \\
\textbf{interp}\ env & (\mathsf{ACTION}\ io) & \mapsto & \underline{\mathrm{do}}\ io \\
 & & & \quad \textbf{return}\ (\mathsf{MkPair}\ env\ ()) \\
\textbf{interp}\ env & (\mathsf{RETURN}\ val) & \mapsto & \textbf{return}\ (\mathsf{MkPair}\ env\ val) \\
\textbf{interp}\ env & (\mathsf{CHECK}\ (\mathsf{Just}\ a)\ j\ n) & \mapsto & \textbf{interp}\ env\ (j\ a)
\end{array}
$$

Do you want to know more?

- TMV028/DIT322 Finite automata theory
  Bachelor course given in LP3
- DAT350/DIT233 Types for Programs and Proofs
  Master course given in LP1
- DAT060/DIT201 Logic in Computer Science
  Master course given in LP1
- DAT415/DIT311 Computability
  Master course given in LP2