Turing Machines

COMP1600 / COMP6260

Semester 2, 202

Alan Turing (1912–1954)

Turing's Scientific contributions

1936

- introduces *Turing machines* and the study of computability.

1950.

- intr
  pro
- curr
  (for the last 5 competitions – from 2013 to 2019) th
  18-year-old female from Leeds, England.

1952.

- Pioneering work on computation in *nature*;

Also. Key figure in the invention of the earliest modern computers.

Turing at War

Germany

- Germans used *enigma machines* – most sophisticated crypto equipment

The U.K.

- cod
- Tur

Achievements

- cracked secret Enigma code
- estimated to have shortened world war II by

Fallout

- may ideas and technologies discovered in Bletchley park fed directly into modern computers

Death and Legacy

1952

- less than 10 years after heroic war efforts for the UK
- Turing prosecuted for 'crime' of homosexuality
- sentenced to chemical castration

1954

- dea

Legacy Now

- widely recognised as the *father of compu*
- *Turing award* equivalent of Nobel prize
- UK government apologised for persecution in 2009
- 2012 officially named the Turing year
- Royal Pardon in 2013

Turing Machines – Introduction

Computability – 1936 paper

- *On computable numbers, with an application to the Entscheidungsproblem*
- solv
- cha
- sim
- this device instrumental in solving Hilbert's pro

Modern Computers

- didn't exist in 1936?
- ENIAC in 1946 generally considered to be the first

Photo c/o Early Office Museum Archives

A model for 'computers'

Computing in 1936

- computers not machines, it was a *profession*.
- Turing's contribution: mathematical *definition* of what computers can do.
- justification: references to 'state of mind' or similar
- see o
  //w 6.pdf

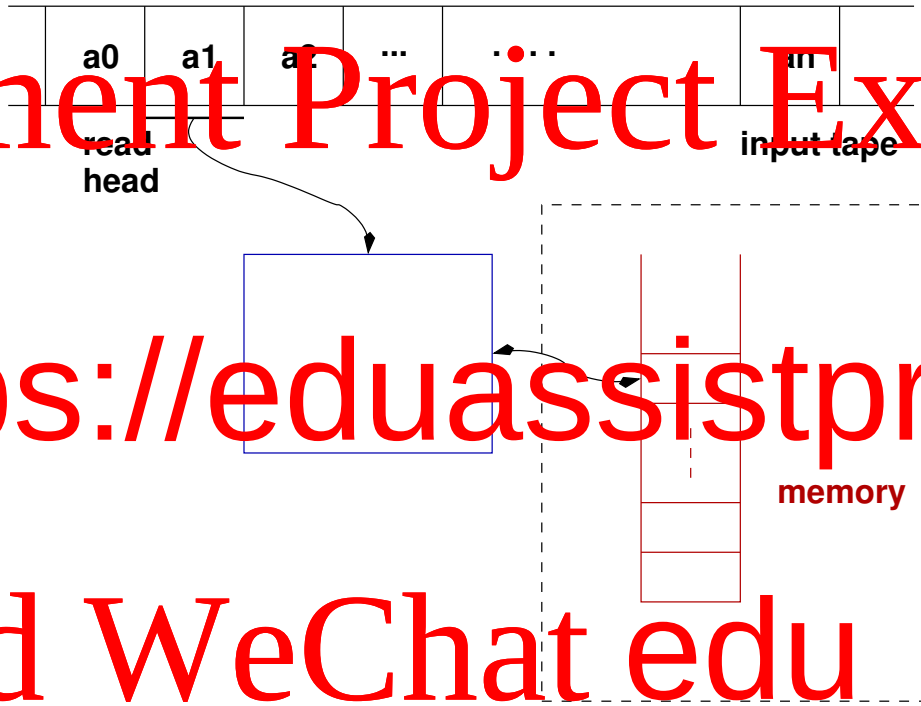
Turing's model today

- no computer has been built that is *m* odel
- Turing has discovered the *essence of co*

Mathematical Models vs Reality

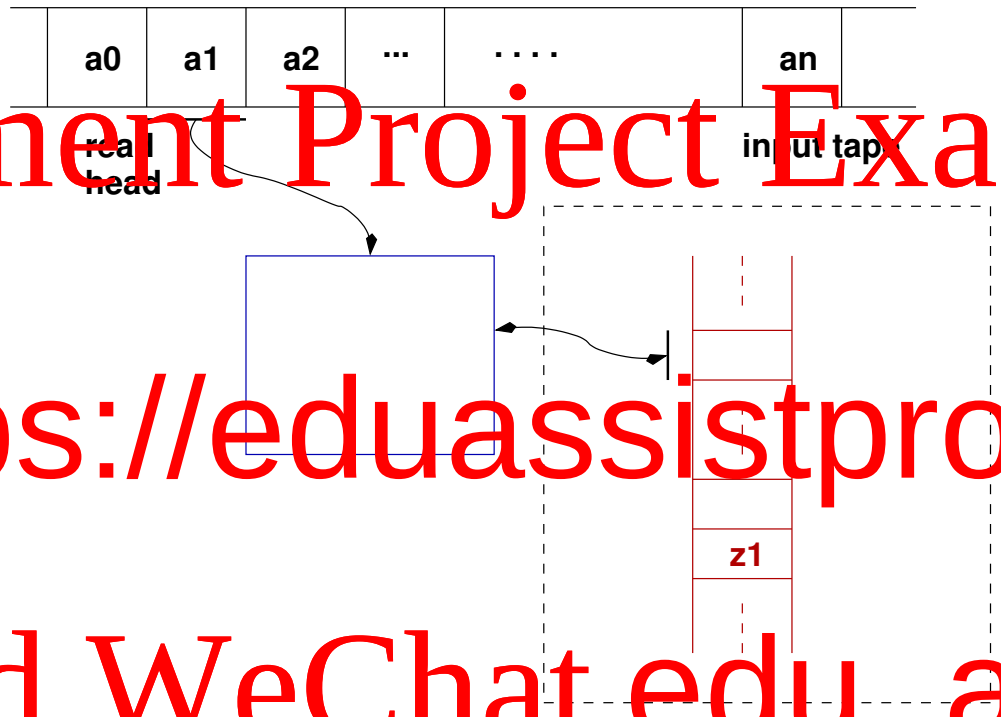
- no formal *proof* that the model is the "right" one
- but widely accepted
- new paradigms emerge, e.g. quantum computing

Push Down Automata, reloaded



- A PDA with its auxiliary store is *almost* a whole computer, except we can only directly access the symbol on the top of the stack.

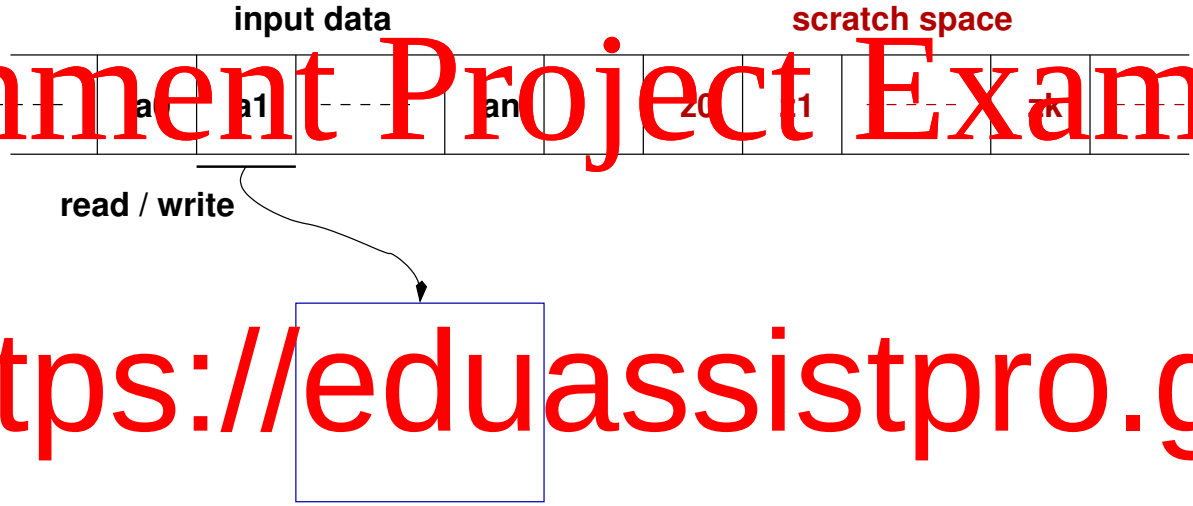
Turing Machines



Generalisation from PDA to Turing machine

- replace *stack memory* by *tape memory*
- can access *arbitrary* symbols on tape by moving tape head

Single tape Turing Machines



Simplification

- have *the same* tape both for input and for stor
- tape is assumed to be *infinite* in both directions
- analogy: "get more paper if you run out"

Turing Machines as language recognisers

Deterministic Machines for the time being

- at every time of the computation *at most* one action is possible
- If there is *no* action that is legal, the TM *halts*
- If a T
  set of s
- If it ha
  rejected.
- A TM may also loop forever...

Definition. If a language is recognised by a Turing machine, then it is called *recursively enumerable*.

Output

Turing Machines as Computing Devices

- TMs can calculate *any* computable function.
- inp
- out

Infinite Tapes aren't a problem

- computation halts after *finitely many* ste
- only finitely many tape cells will be written to

Turing Machine – formal definition

A Turing Machine has the form $(S, s_0, F, \Gamma, \Sigma, \Lambda, \delta)$, where

- S is the set of states, $s_0 \in S$ is the initial state and $F \subseteq S$ are the final states;
- Γ is th
  $\Lambda \in$
- δ is a (

$$\delta : S \times \Gamma \nrightarrow S \times \Gamma \times$$

(state, tape symbol) $\mapsto$ (new sta ction)

The direction tells the read/write head which way to go next: Left, Right, or Stay.

Running a TM

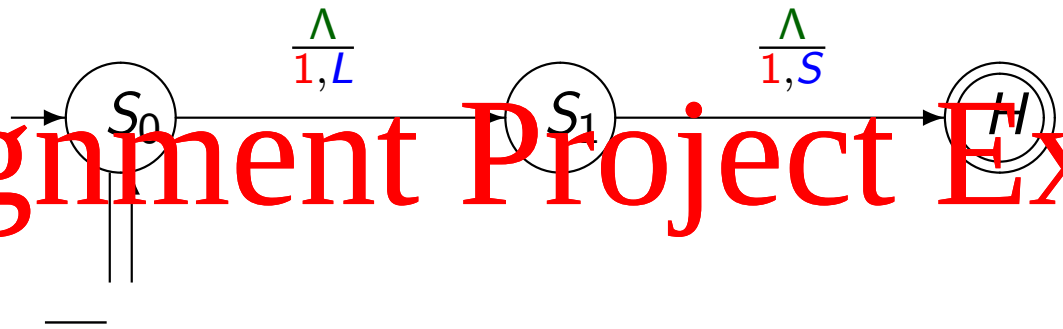
Initialisation.

- some input (a finite string over Σ) is written on the tape;
- every other tape cell is a blank – Λ;
- the r
  is the i
- the s

Running.

- in a cycle: read symbol and execute action (state de
  move/write/change state
- until a final state is reached (or the machine gets stuck)

Graphical Representation of the Transition Function

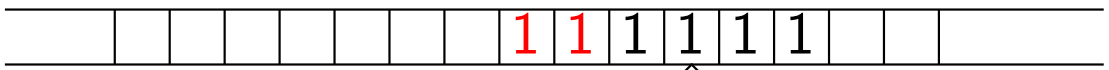


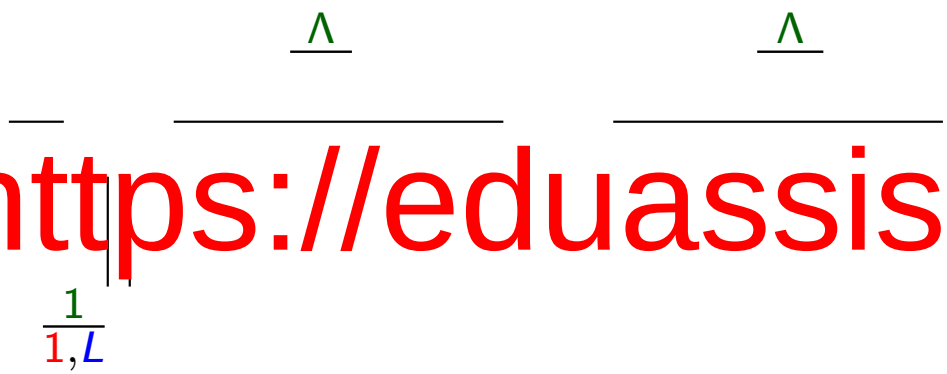
(Like FSA

Convention

- Numerator: *symbol read from tape*.
  - Λ means the tape is blank at that position.
- Denominator: *symbol written* / *direction of head movement*.
  - direction one of L, R, S for Left, Right, Stay.

What does it do?

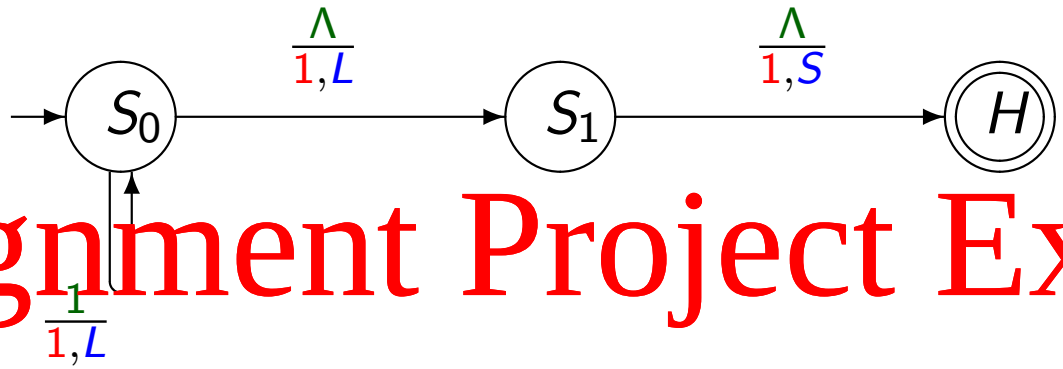




$\frac{1}{1,L}$

- Adds two to a unary number!
- Assume the head starts over the input data.
- First phase scans left.
- Second phase writes two extra *1*s.

The Convention for Errors



TMs getti

- sup
- TM

  we meet such a token (this is the job of the rightward 0
- this is an error - the input is *rejected*.
- S_0 *not* an accepting state!

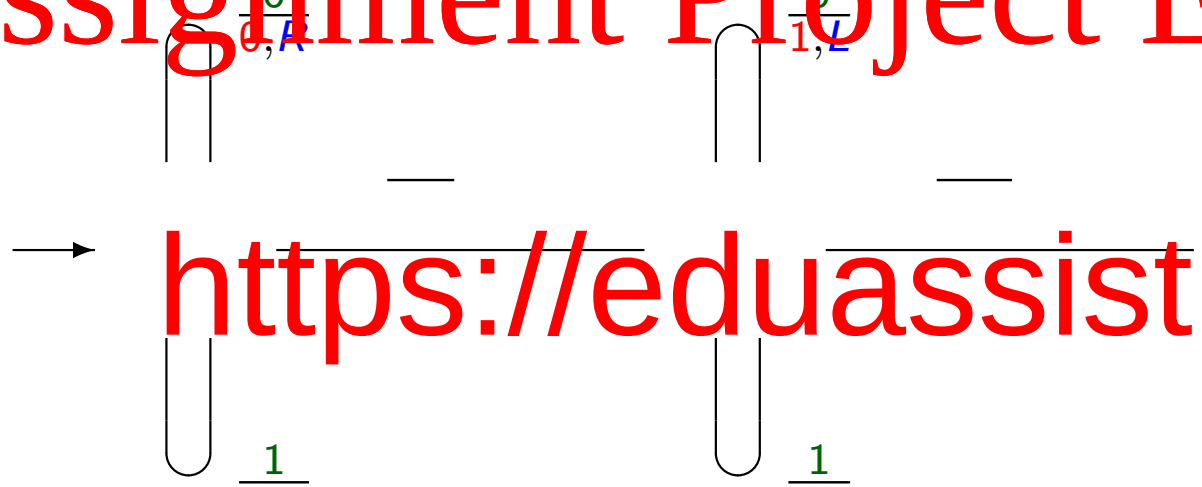
Language

- TM accepts precisely $\{1^n \mid n \in \mathbb{N}\}$.

Alternative Formulation (not used here)

- could add an error state that the machine transitions to
- error state *not* accepting

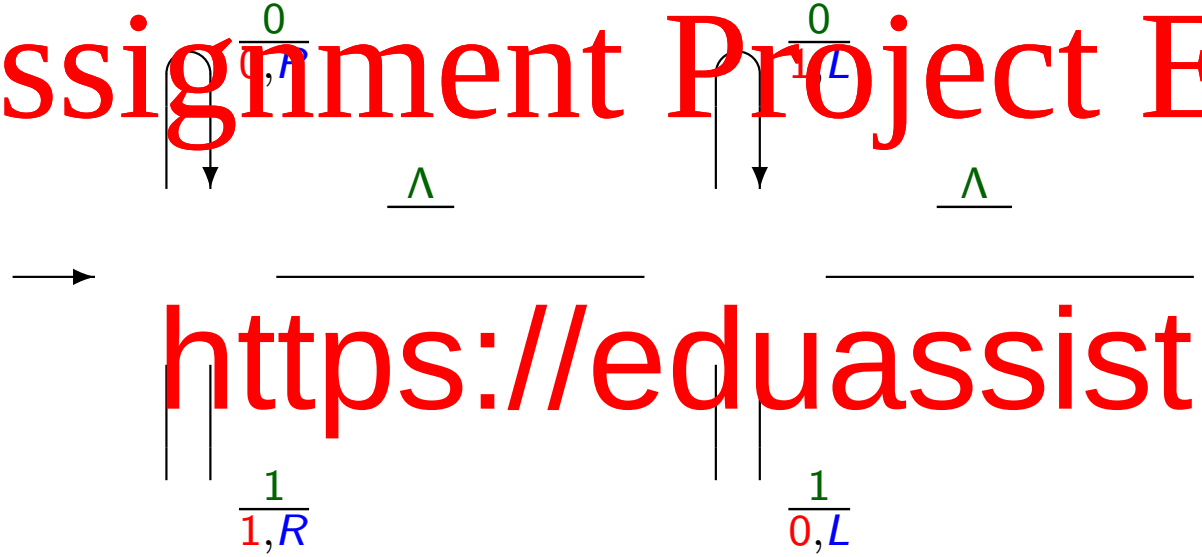
What does this one do?



Questions.

- Do you see two phases?
- What does each phase accomplish?

What does this one do?

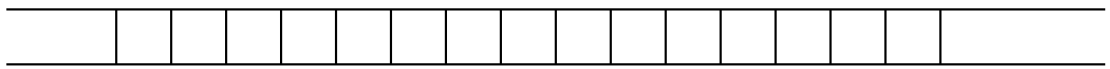


Answer.

- Phase 1: initialisation.
- Phase 2: computation, in this case, complement a binary number.

Harder Problems?

- Incrementing a binary number



Yo

- Add

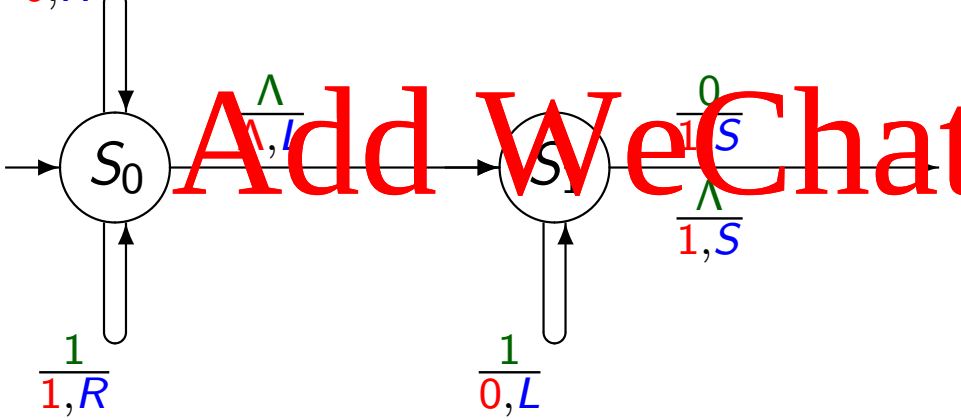
Conv[illegible] write the result before the data

- Multiplication - and so on!

Incrementing a binary number

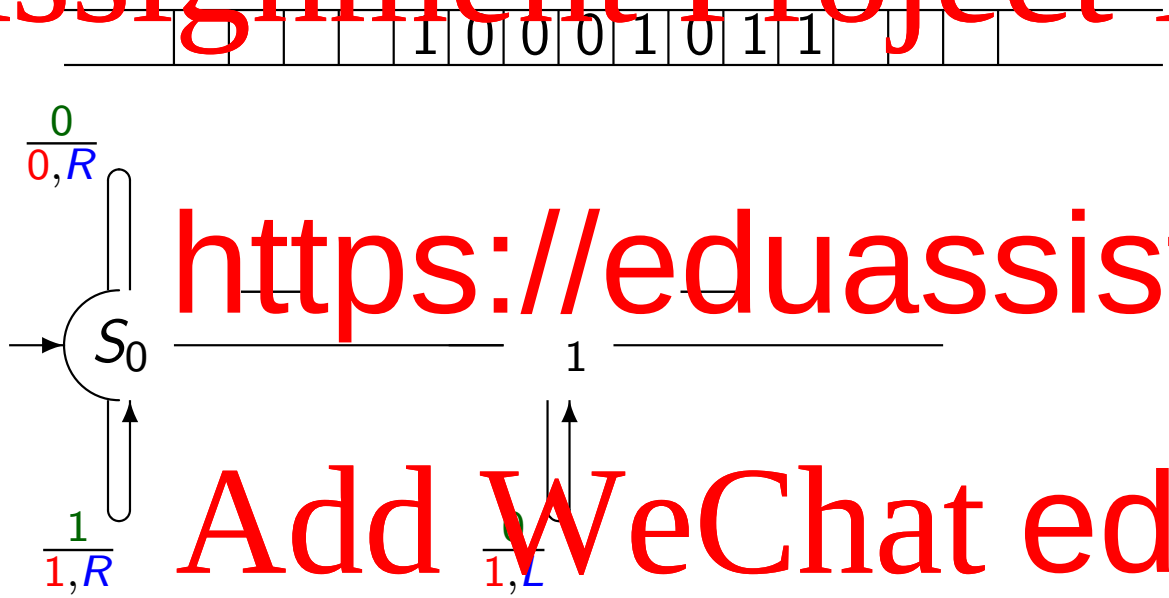
Example. Number increment

Solution



Decrement

Example. Number decrement (similar)



Failure (or non-acceptance): If given number is 0 it fails (at state S_1),

How to add two numbers?

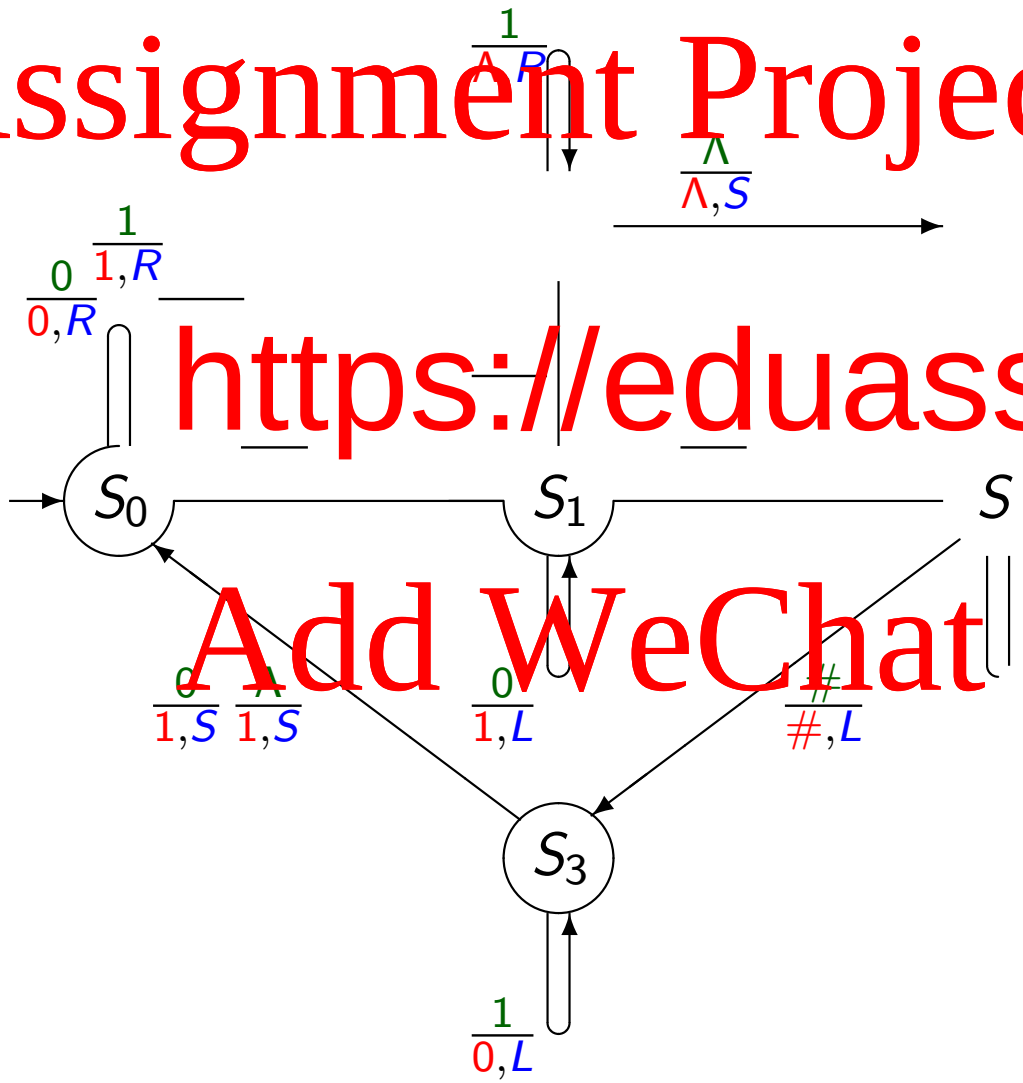
Input. Binary numbers separated by #, say

$$\underbrace{10100101}_{n} \# \underbrace{100101010}_{m}$$

Operation

- Go back and forth between m and n, decrementing one (until this fails) and incrementing the other.
- decrement m, and increment n becaus
- m gets changed to $00\ldots0$, n is replace
- Finally, delete the $\#00\cdots0$ on the right.

How to add two numbers? ctd.



How to multiply two numbers?

Input. as for addition

$$\underbrace{\quad}_{n}\#\underbrace{10100101}_{n}\#\underbrace{100101010}_{m}$$

Operation.

- Repeatedly decrement m (until this fails) and add n to p (p is initially bla
- Mu p being add

Modification of addition routine

- Two new tape symbols, $0'$ and $1'$.
- Before each addition stage, change all the 1s in
- When decrementing n, swap 0s and 1s as usua
- When finished adding n to p, go back and use the primes to restore n.

Observation.

- this is *very tedious* – but the model is simple and easy to analyse
- tricks that you see here are typical

Programming Issues – Data

Data Types and Gadgets

- not present in the model
- but can be simulated . . .

Numbers

- Usu

Vocabulary.

- Can be arbitrary, and $\{0, 1\}$ suffice. Ch
  strings of bits.

Variables, Arrays, Files

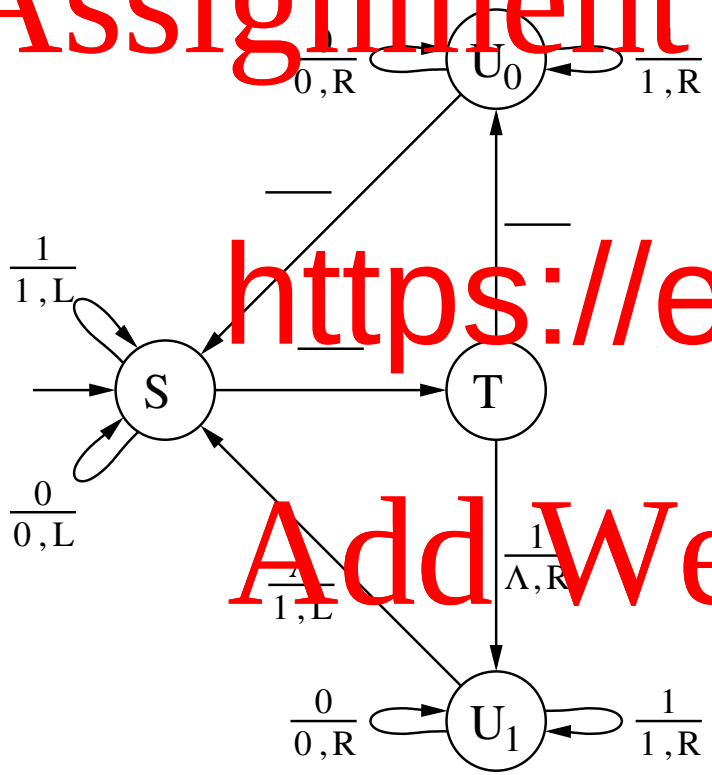
- Use markers on the tape to separate values.

Programming Issues – Control

Common Idioms.

- Scan to blank or to find, insert, delete a symbol
- Use control states to remember information
  In pa
  else
- Co
  If you have a TM to multiply by 3 and one to multiply by
  together to multiply by 15.
- Decisions (conditional computation)
  As we have seen, we can branch on 0 or 1 (or T or F).
- Loops — of course.

Using States to Remember a Tape Symbol



Given a string of 0 or 1 surrounded by blanks, this machine repeatedly forever

We use the cho
U_0 or U
symbol has been erased and is to be written

Universal Turing Machines and Turing Completeness

- We can construct a TM to simulate any conventional computer. (Cost effectiveness is not brilliant.)
- From this point, just think of a TM as like a computer program (with a ma
- We c
  TM a
  Haskell program whose purpose is to read any other Haskell program and run that)
- Turing machines can simulate themselves.
- Turing machines can *compute properti*
- Any computing device which can simulate a universal Turing Machine is also called *universal* or *Turing Complete*.

Church-Turing Thesis

Church-Turing Thesis.

> If a function is computable, then it can be calculated by a Turing machine.

Equivalent Formulation.

- if a pr
  Tur

This is a Thesis.

- more a definition than a theorem
- can never be *proved*: what does algorithm
- however, there's lots of *evidence*

Evidence.

- all *other* definitions of the term computable give the same class of computable functions
- there are many: λ-calculus, register machines, while programs, etc.

Church and λ-calculus

Example. The (untyped) λ-calculus

- proposed by Alonzo Church (the Church in the Church-Turing thesis)
- in 19
- Ros

Equivale

- if a function is computable by a Turing machine, th computable in the λ-calculus

- . . . and vice versa
- simulation of the respective formalism in the other approach.

Can real computers simulate Turing Machines?

Differences between computers and Turing machines

- A Turing Machine has an *infinite tape*.
- "real" computers have finite memory (sometimes a lot, but nowhere near infinite)
- phy
  finit
- . . . bu
  Ho
- physical computers are an *approxima*

Intuitive Understanding

- Turing machine model *feels* closer to what we can program
- e.g. we *can* recognise the language $\{a^n b^n \mid n \geq 0\}$
- if the real machines runs out of memory . . . we can always buy more
- this is about computation *in principle*

Church-Turing Thesis: Argument 1

Turing Machines emulate humans.

- comparison how humans perform tasks
- Turing's 'computers' only perform TM operations: writing, reading, refo

TMs are fin

- can
- can't read the whole infinite tape

Taken together

- convincing (in the 1930s) that Turing's model is correct
- but no mathematical evidence (yet)

Church-Turing Thesis: Argument 2

Stability of the TM Model.

- adding 'features' doesn't make more functions computable

Multi-tape

- have extra tapes to store data
- easier to program, but no extra "power"
- (sin

Multi-he

- have more than one head, heads can move indepe
- heads can access multiple symbols at once
- again, no extra power

Non-determinism. (as for nfa's)

- tm can make one of several possible next moves
- tm can "guess" the right next move
- *may* make it faster, but cannot compute more functions.

Church-Turing Thesis: Argument 3

Many Models of Computation

- plenty of *different* definitions of what computable may mean
- different purposes, different contexts

Main Insight.

- *any* *isely* the sam
- "re

Examples.

- Lambda-Calculus (Church, 1932)
- Post-Systems (Post, 1939)
- Register Machines
- ...

Doesn't include

- models based on physical phenomena
- ... or biology, or ...

Argument 3B: Grammars

Theorem. Any language that is generated by a grammar can be recognised by a Turing machine.

Proof (Sketch)

- writ
- sear
- eac

Acceptance

- if the grammar finds the input, we will find a derivati
- if the grammar does *not* generate the input, we may loop forever (and not accept)

Argument 3B – grammars ctd

Unrestricted Grammars. Have productions of the form

$$\alpha \to \beta$$

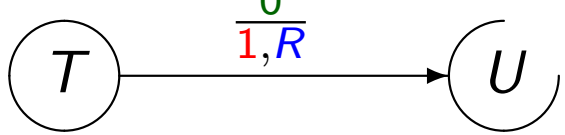
where β is arbitrary, and α contains at least one non-terminal.

Theorem *precisely* the words tha

Proof (Sk

- non-terminals (of the grammar) are states of th
- run TM "backwards" (we are interested in input

TM Transition.



Grammar Production. $1U \to T0$.

(Detail missing, e.g. how to handle blanks)

Argument 3C – Computers & Programming Languages

Observation.

- no computer ever invented could do things that a TM can't do
- no programming language can do more than a TM
- bac

Commo

A progra

computed by a Turing machine is called *T*

Examples

- The languages that you know: Haskell, Java, Py
- even the simple while language that we used for Hoare logic
- implement TM simulator in your favourite programming language

Argument 3D – John Conway's Game of Life

Game of Life.

- infinite 2d grid
- finitely many cells marked *alive*, all others are *dead*

Rules of the

- live c opulation);
- live c opulation);
- dead cells with exactly three live neighbours com *(reproduction)*;
- all other cells stay as they are.

Emergent Behaviour.

- visualised by GoL, many interesting forms
- analogy of complex behaviour emerging from simple rules

Argument 3D – John Conway's Game of Life ctd.

From TM's to Conway's Game

- can implement Game of Life on a Turing machine
- lots o

From Con

- showed that GoL can simulate Turing machine
- comes down to clever choice of initial configurat
  `http://rendell-attic.org/gol/t`

Metaprograms

Metaprograms are programs that have other programs as input or output.

Examples

- *Lexi* hich aut SP

- *Code generation* which automatically pr specification or a formal proof

Next Goal. Scrutinise TMs that take TMs as input.

- main goal: *halting problem*

Coding a TM onto tape

Coding of a TM as binary strings

- can be written onto a tape
- just code the transition function

States are ordered $S_1, S_2, \ldots, S_n$, where S_1 is the start state and S_2 the unique fin

Tape Sym is 1, and X_3 is Λ.

Directions $\{L, R, S\}$ as D_1, D_2, D_3 respecti

Transitions $\delta(S_i, X_j) = (S_k, X_l, D_m)$ mapp

$$0^i\ 1\ 0^j\ 1\ 0^k\ 1\ 0^l\ 1\ 0^m$$

(the 0s carry information, the 1s act as separators.)

Coding a TM onto tape ctd.

Coding of all transitions. A TM with transitions $C_1, \ldots, C_n$ is codes as

(11 is used a

Additional Input.

- code for a TM, and additional string
- use 111 to separate TM and string input

Coding a TM onto tape – example

The transi

0	1	00	1	0	1	00	1	00
0	1	000	1	000	1	00	1	00
000	1	000	1	00	1	00	1	000

Recall: States, Symbols, Directions.

So the TM as a whole is encoded by the binary string

01001010010011 01000100010010011 00010001001001000