Formal Languages and Computational Models

Programming Languages

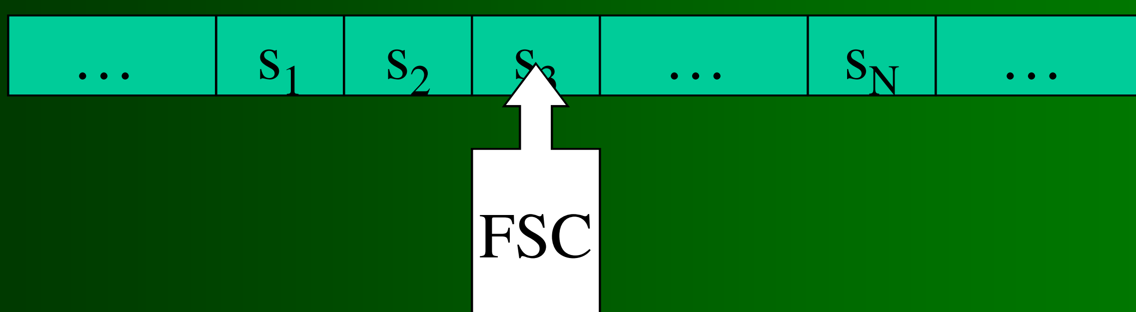
CS 214

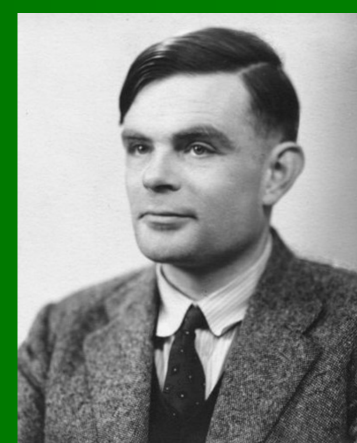
Dept of Computer Science Calvin College

Turing Machines

In ________ (years before the first programmable computer), ____________created a model for the process of computation known today as the ______________________, consisting of:

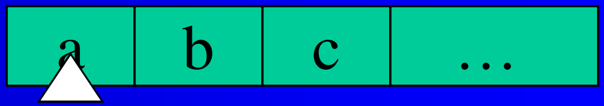
- An *I/O tape* consisting of an arbitrary number of *cells*, each able to store an arbitrary symbol;
- A *tape head* able to read/write a cell; and
- A *finite-state control* that governs movement of the head over the cells.





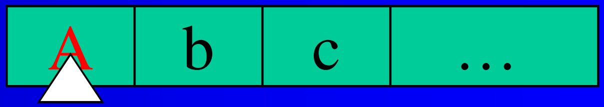
Turing Machines (ii)

Each "execution cycle", a TM ________ a *symbol* from the tape.

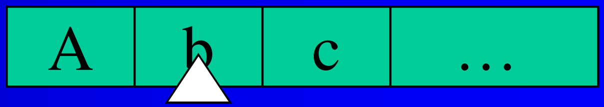


Depending on that *symbol* and its current state, it may then:

- ________________________________;



- ________________________________; and



- ________________________________.

The finite state controller starts in state 0: the ________________, and continues execution until it enters an ____________________, at which point it halts and its I/O tape contains the result of the computation.

Example: TM Addition

To add two numbers *m* and *n*:

- Precond: I/O tape contains *m* ones, a zero, and *n* ones.
- Postcond: I/O tape contains *m*+*n* ones.

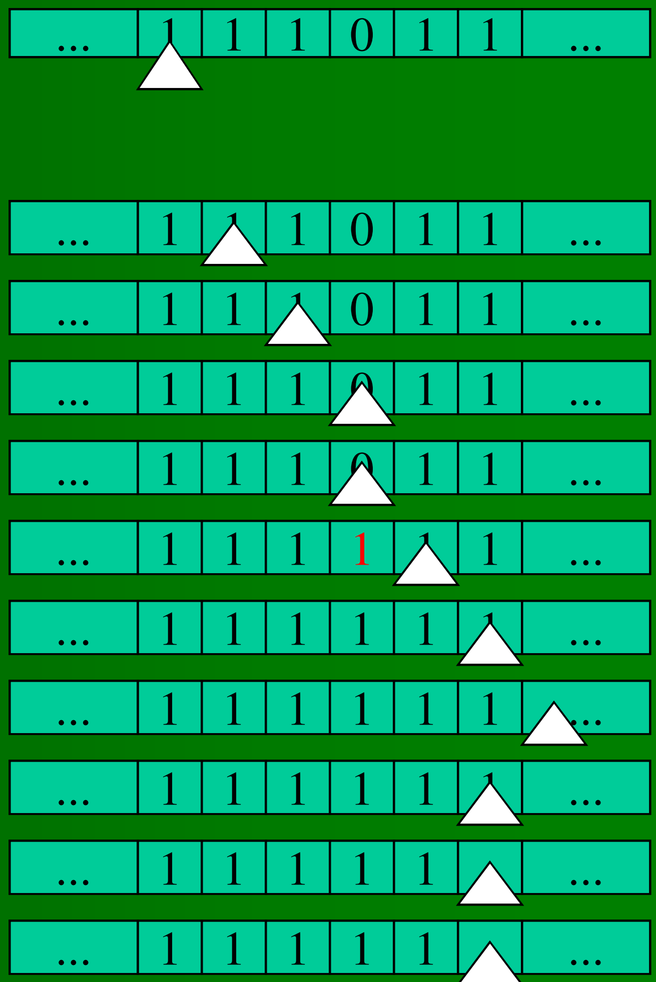
Our finite state controller uses these states and rules:

State 0:	If *symbol* is 1 or blank: move head right; goto State 0. If *symbol* is 0: goto State 1
State 1:	Write 1; move head right; goto State 2.
State 2:	If *symbol* is 1: move head right; goto State 2. If *symbol* is blank: move head left; goto State 3
State 3:	Write blank; goto State 4.
State 4:	Accept.

Example: 3 + 2

To compute 3 + 2, we start with:

Step	$State_i$	Read	Write	Move	$State_j$
1	0	1	____	____	____
2	0	1	____	____	____
3	0	1	____	____	____
4	0	0	____	____	____
5	1	-	____	____	____
6	2	1	____	____	____
7	2	1	____	____	____
8	2	blank	____	____	____
9	3	-	____	____	____
10	4	-	____	____	____



TMs and Computability

In 1931, _______________ proved that there exist easily-described functions that cannot be computed.

In 1936, __________ proved that a TM can be built for any computable function.

He later proved that a ___________________ can be built that can perform the task of any single-function TM, implying:

→ Since it is independent of any particular hardware details, a proof about a UTM applies to <u>every</u> computer that will <u>ever</u> be built!

→ If a function *f* can be computed, then a UTM can compute *f*.

→ If a UTM cannot compute a function *g*, then ____.

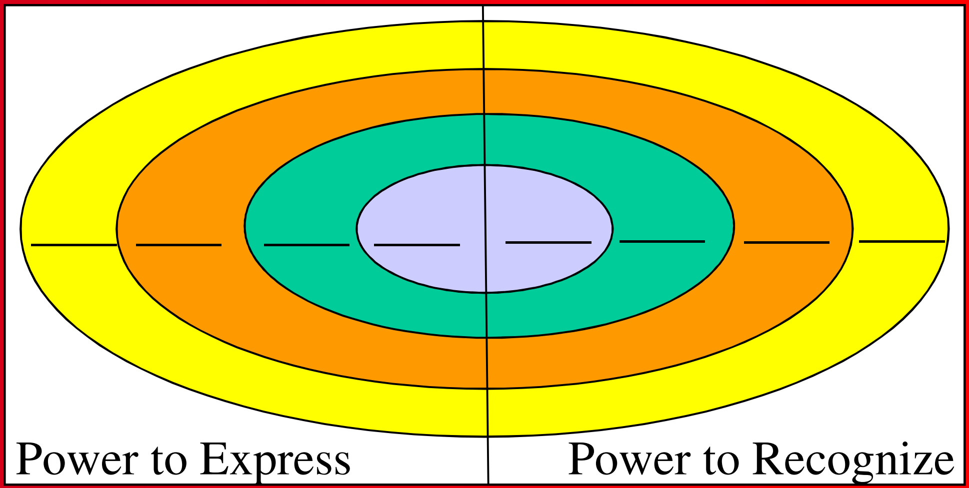
Turing proved the ____________ cannot be solved by a UTM.

The Chomsky Hierarchy

In 1956, _________________ classified languages as follows:

Level	Language	Recognizer
___	_______________ (REs)	___________________ (FSM)
___	______________ (CFLs)	___________________ (PDA)
___	_______________ (CSLs)	___________________ (LBA)
___	______________ (ULs)	__________________ (TM)

Chomsky's categories form a *hierarchy*, organized by their power of expression (language) and power of recognition (automaton):



Chomsky and BNFs

The *Chomsky Hierarchy* specifies that:

- A ____ can recognize any language able to be recognized.
- A ____ can recognize CSLs, CFLs, & REs but not ULs.
- A ____ can recognize CFLs & REs but not CSLs or ULs.
- A ____ can recognize REs but not CFLs, CSLs or ULs.

The BNF is a tool for specifying _____ syntax.

– Programming language syntax is relatively “easy”, linquistically.

It can also be used to specify RE syntax
(but doing so is overkill -- simpler tools are available).

Different tools are needed to specify CFL and/or UL syntax.

PDAs and (BNF) Parsing

A PDA is a FSM with a ______ on which it can save things...

Recall our basic parsing algorithm (for BNFs):

0. Push *S* (the starting symbol) onto a stack.
1. Get the first terminal symbol *t* from the input file.
2. Repeat the following steps:
    a. Pop the stack into *topSymbol*;
    b. If *topSymbol* is a nonterminal:
        1) Choose a production *p* of *topSymbol* based on *t*
        2) If *p* != ε:
            Push *p* right-to-left onto the stack.
    c. Else if *topSymbol* is a terminal && *topSymbol* == *t*:
        Get the next terminal symbol *t* from the input file.
    d. Else
        Generate a 'parse error' message.

while the stack is not empty.

A FSM cannot parse a CFL/ BNF because it has __________.

The Random Access Machine (RAM)

Proving things about TMs was a bit clumsy…

1963: *Shepherdson and Sturgis* devise the RAM as a model that is equivalent to a TM but more convenient to use:

The RAM has four components

- A *memory*: an integer array, indexed from zero.
- A *program*: a sequence of numbered instructions.
- An *input file*.
- An *output file*.

Shepherdson and Sturgis proved ______________________________ ______________________________, and vice versa.

The RAM Instruction Set

- `M[i] = n` → store n at index i
- `M[i] = M[j]` → copy value at j to i
- `M[i] = M[j] + M[k]` → add and store
- `M[i] = M[j] - M[k]` → subtract and store
- `M[M[j]] = M[k]` → indirection
- `read M[i]` → input (destructive)
- `write M[i]` → output
- `goto s` → unconditional branch
- `if M[i] >= 0 goto s` → conditional branch
- `halt` → terminate execution

Later extensions added other operators (arithmetic, relational)

The result was quite similar to a ______________________.

Example 1

Here is a RAM for a computation…

What does it do (try some sample inputs)?

program

```
1. M[0] = 0.
2. read M[1].
3. if M[1] >= 0 goto 5.
4. M[1] = M[0] - M[1].
5. write M[1].
6. halt.
```

memory

[0]
[1]
[2]
[3]
…

input

output

Example 2

Here is a different RAM. What does it compute?

program

```
1. M[0] = 64.
2. M[1] = 91.
3. M[2] = 32.
4. read M[3].
5. if M[3] >= 0 goto 7.
6. goto 14.
7. M[4] = M[0] - M[3].
8. M[5] = M[3] - M[1].
9. if M[4] >= 0 goto 12.
10. if M[5] >= 0 goto 12.
11. M[3] = M[3] + M[2].
12. write M[3].
13. goto 4.
14. halt.
```

memory

[0]
[1]
[2]
[3]
[4]
[5]
...

input

output

RAM Extensions

Like a TM, a RAM can compute anything that is computable.

With these simple extensions:

- ______________________ instead of memory locations
- ________________ and ____________ operators
- other ______________ (==, !=, <, >, >=) operators
- ____________ within arithmetic expressions

it becomes a convenient tool for studying HLL constructs, as a "portable assembly language" to study how a compiler can translate HLL constructs.

RAM Extension Examples

Example 1 program

```
1. read val.
2. if val >= 0 goto 4.
3. val = 0 - val.
4. write val.
5. halt.
```

Example 2 program

```
1. read ch.
2. if ch < 0 goto 10.
3. lo = ch - 65.
4. hi = ch - 90
5. if lo < 0 goto 8.
6. if hi > 0 goto 8.
7. ch = ch + 32.
8. write ch.
9. goto 1.
10. halt.
```

Even with the improvements, such programs are hard to read because of their coding style (aka ________________)...

Summary

The ______________________ names four "levels" of language, plus the weakest machine able to recognize at each level:

3 Regular Expressions	→ Finite State Machine
2 Context Free Languages	→ Pushdown Automata
1 Context Sensitive Languages	→ Linear Bounded Automata
0 Unrestricted Languages	→ Turing Machine

The ______ is the most powerful of the machines, able to

- recognize any language capable of being recognized.
- compute any function capable of being computed.

The _______ is a computational model that is

- as powerful as the TM
- more convenient than the TM for studying HLL constructs.

