

CS3102 Theory of Computation

Is this language Regular? Context Free?

$$L = \begin{cases} a & \text{if UVA wins the NCAA Tournament} \\ b & \text{Otherwise} \end{cases}$$

$$L' = \begin{cases} a & \text{if there's life on Mars} \\ b & \text{Otherwise} \end{cases}$$

$$|L| = 1 = |L'|$$

L and L' are both finite, and therefore they're regular

On Computable Numbers, With an Application to the Entscheidungsproblem

- Alan Turing's seminal work
- Contributions
 - **Turing Machines**
 - Proof that some “numbers” (equivalently functions) are expressible, but not computable by **them**
 - Proof that the Entscheidungsproblem cannot be solved by **them**
 - **Proof that **they** are of equivalent power to the lambda calculus**
 - **Demonstration that **they** are “universal”**
 - There is one Turing Machine that can simulate all others
 - **A philosophical argument that **they** are equivalent to humans**

Universal Machine

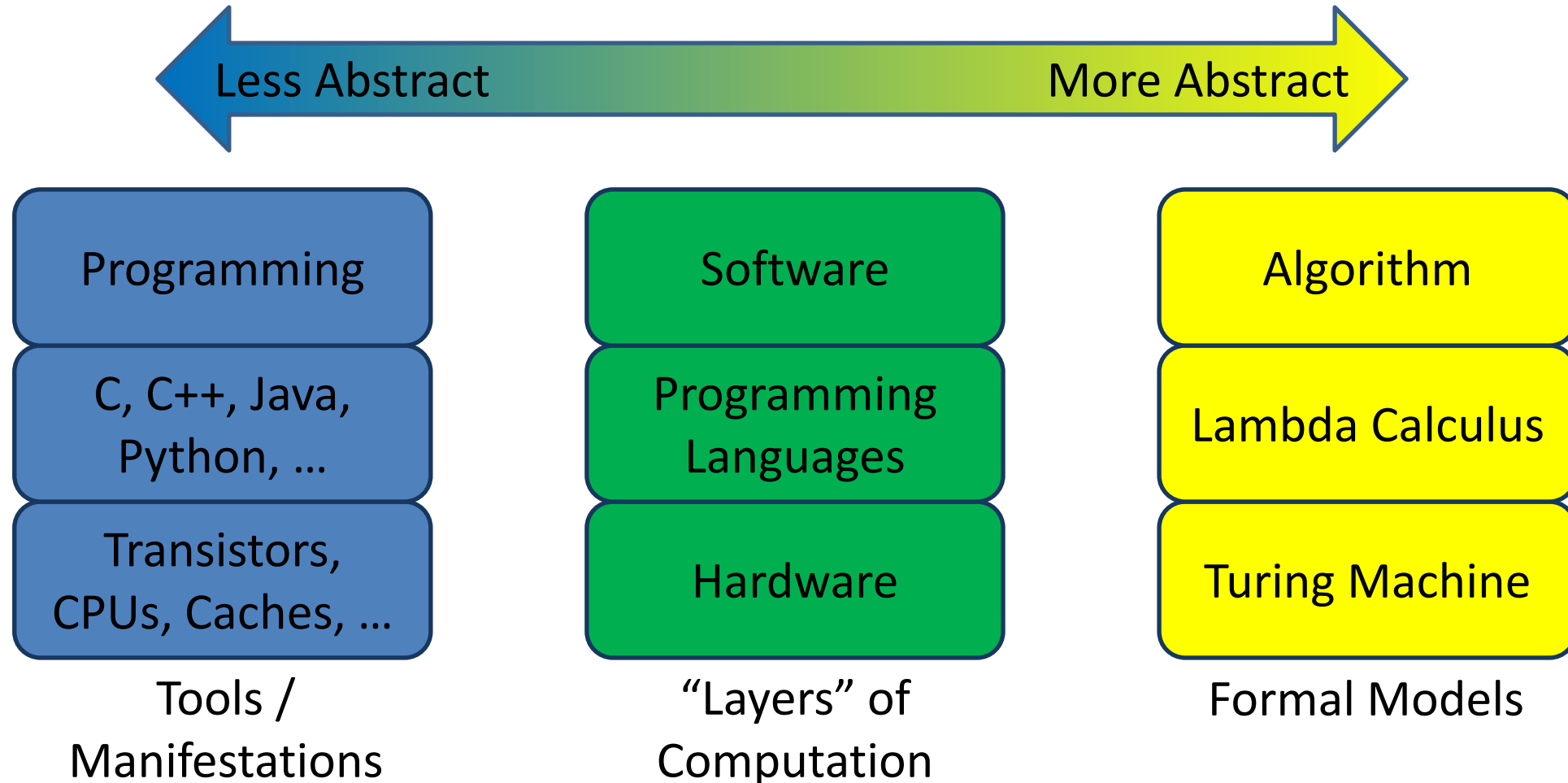
- We can have one Turing machine which can simulate any other Turing Machine
 - There is some machine M_u which given the description of another machine M' and an input w , it behaves the same way that M' would on w
- Why is this useful?
 - Turing machines are meant to be a model for all computers
 - Allows us to have one machine that we've built to simulate any other machine we describe

How do we compute?

- We start with a universal Turing machine that we've built (a CPU)
- We use the lambda calculus (programming languages) to write algorithms
- Since the lambda calculus is equivalent to Turing machines, and our machine was universal, we can perform the operation described

What is Computer Science?

Computer Science studies all layers of abstractions of computing machines



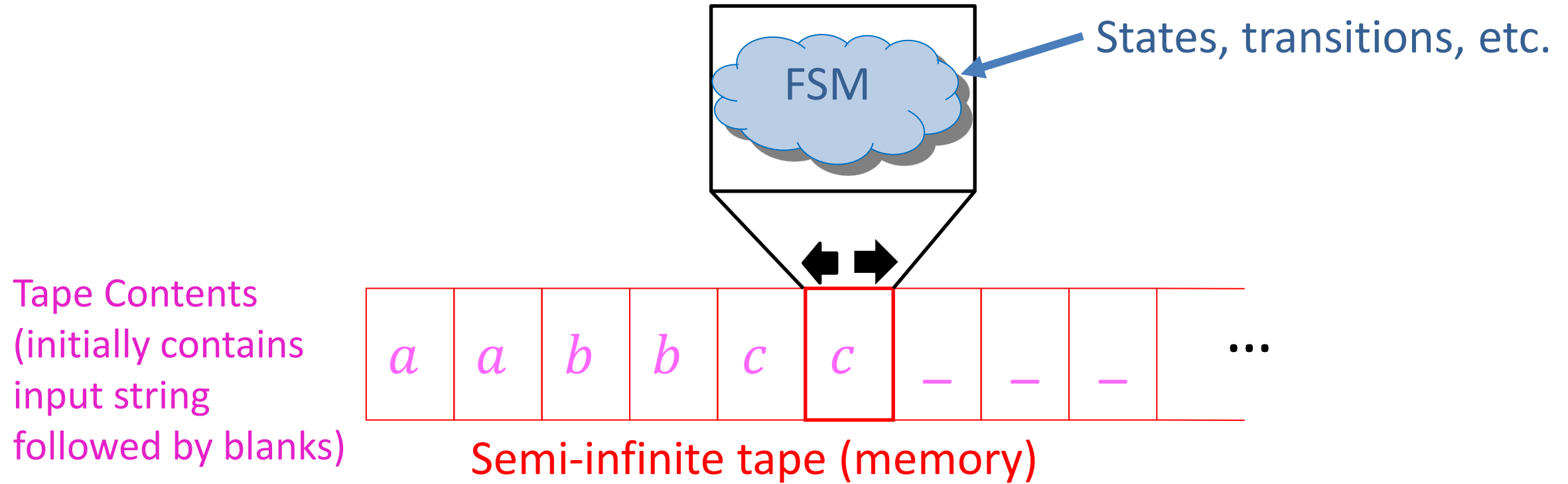
Can We do better?

- Church-Turing Thesis:
 - No!
 - Turing Machines can compute anything a human can compute
- Why is this reasonable?
 - How do humans calculate?
 - Brains = finite state machine
 - Using scratchwork = tape

Turing Machines

- NFA/DFA:
 - Finite number of states,
 - read-once input,
 - transition using input character and state
- PDA:
 - Finite number of states,
 - read-once input,
 - stack (memory)
 - Transition using input character, state, and stack
 - Can push to the stack on transition
- Turing Machine:
 - Finite number of states,
 - ~~– Read-once input,~~
 - Semi-infinite tape (memory)
 - Transition using input character, state, and “current symbol” on tape
 - Can overwrite current symbol, move left/right on tape

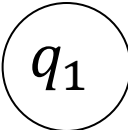
Turing Machine



Operation: transitions outgoing from each state match on current character on the tape, when transitioning you can overwrite that character and move which cell you're reading

Turing Machine

Basic idea: a **pushdown automaton** is a finite automaton that can optionally write to an unbounded **stack**.

- Finite set of states: $Q = \{q_0, q_1, q_2, \dots, q_k\}$ 

- Input alphabet: Σ

- **Tape** alphabet (includes $_$): Γ

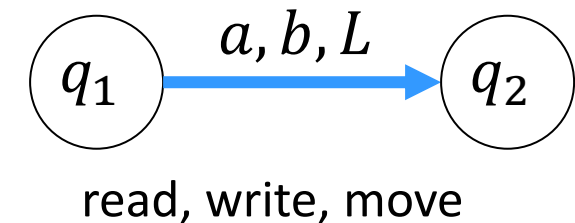
- **Transition** function: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

Read
Write
Move

- **Initial** state: $q_0 \in Q$ 

- **Accept** state: $A \in Q$ 

- **Reject** state: $R \in Q$ 



Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, A, R)$

Configurations

- DFA:
 - Which state is currently active
 - At start: Start state is active
- PDA:
 - Which state is currently active
 - What's in the stack
 - At start: start state is active, stack is empty
- Turing Machine
 - Which state is currently active
 - What's on the tape
 - Where the read head is on the tape
 - At start: start state is active, input is on the tape (with blanks after), read head is at the first cell

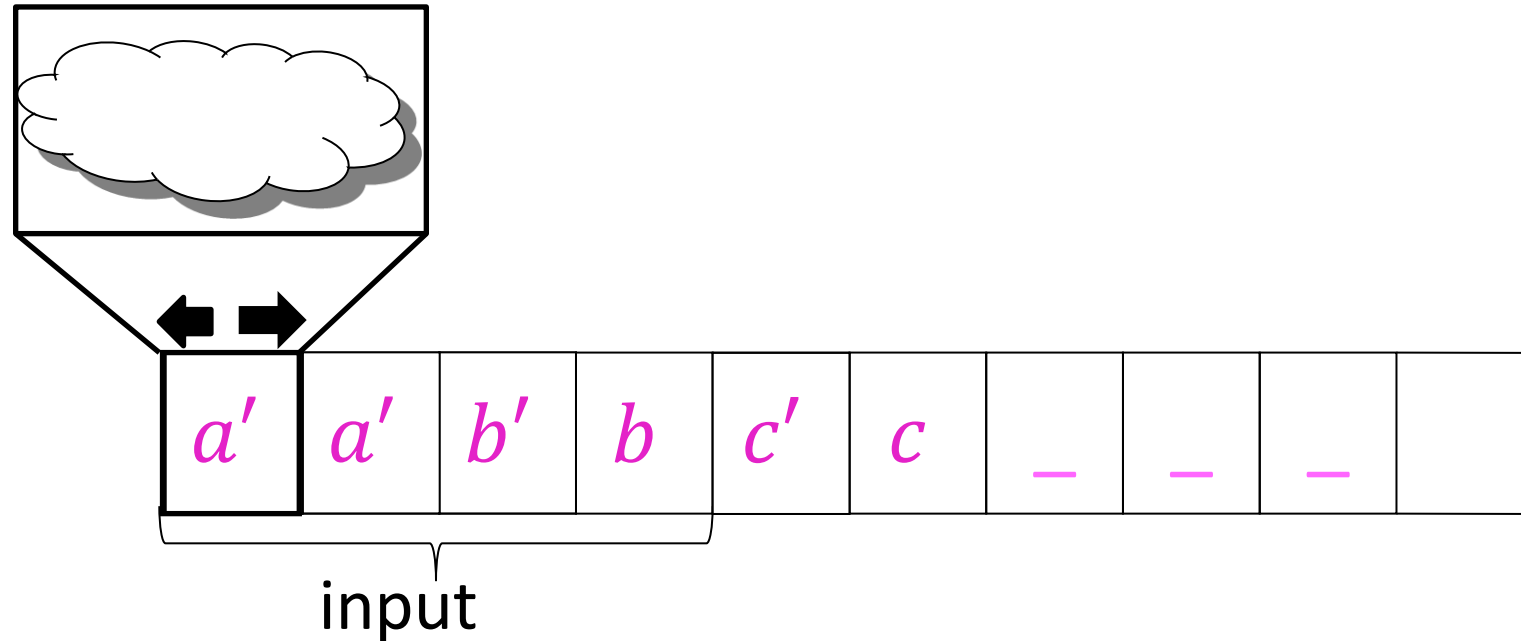
Transition behavior

- DFA:
 - Input: State and input symbol
 - Result: state
- PDA:
 - Input: State, input symbol, symbol popped from the stack
 - Result: state, symbol pushed to the stack
- Turing Machine:
 - Input: State, symbol at current location in the tape
 - Result: State, symbol written to current location in the tape, current location moved 1 to the left/right

Example Turing Machine

- For language $a^n b^n c^n$
- “mark off” each a with the next b and the next c
 - “marked” $a = a'$
 - “marked” $b = b'$
 - “marked” $c = c'$
- If everything is marked, accept

$$a^n b^n c^n$$



1. Mark an a
2. Scan to first unmarked b
3. Mark the b
4. Scan to first unmarked c
5. Mark the c
6. Go back to last marked a
7. Go right by one
8. If that's an unmarked a
 - Go to step 1
9. If that's a marked b
 - Check that all characters are marked

What does the machine look like?

Acceptance Condition

- DFA
 - Input has all been read
 - We're in a final/accepting state
- PDA
 - Input has all been read
 - We're in a final/accepting state
 - The stack is empty
- TM
 - We're in the accepting state

Rejection Condition

- DFA
 - Input has all been read
 - We're not in a final/accepting state
- PDA
 - Input has all been read
 - We're not in a final/accepting state
 - The stack isn't empty
- TM
 - We're in the rejecting state

Some Turing Machines never accept/reject

- In this case they run forever
- 3 “reporting” behaviors
 - Accept and halt
 - Reject and halt
 - Run forever (implicit reject)
- This is necessary for computation

```
while(true){  
    twiddle(thumbs);  
}
```

```
while(x != 1){  
    if(x%2 == 0){  
        x = x / 2;  
    }  
    else{  
        x = 3x+1;  
    }  
}
```

Running forever

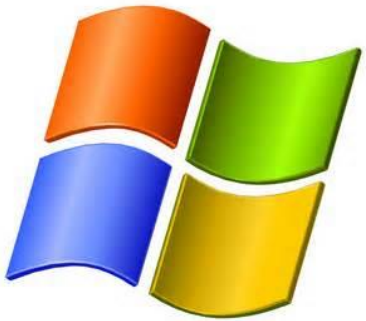
- Is it a bad thing?

Programs we want to halt:



Your 2150
Homework

Programs we want to run forever:



Turing Machine Outcomes

1. Running TM M on input w eventually leads to A
 - Result: Accept
2. Running TM M on input w eventually leads to R
 - Result: Reject
3. Running TM M on input w runs forever (never terminates).
 - Result: Reject

Recognizing vs. Deciding

Turing-recognizable: A language L is “Turing-recognizable” if there exists a TM M such that for all strings w :

If $w \in L$ eventually M enters A (the accept state)

If $w \notin L$ either M enters R (*the reject state*) **or** M never terminates

Turing-decidable: A language L is “decidable” if there exists a TM M such that for all strings w :

If $w \in L$ eventually M enters A (the accept state)

If $w \notin L$ either M enters R (*the reject state*) **or** M never terminates

Turing -Recognizable vs. -Decidable



A language is **Turing-decidable** iff it is exactly the set of strings accepted by some **always-halting** TM.

$w \in \Sigma^*$	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb	aaaa	...
$M(w) \Rightarrow$	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	...
$L(M) =$	{ <i>a</i> ,		<i>aa</i> ,				<i>aaa</i> ,								<i>aaaa</i>	...}

M must **always halt** on every input.

Turing -Recognizable vs. -Decidable



A language is **Turing-recognizable** iff it is exactly the set of strings accepted by some Turing machine.

$w \in \Sigma^*$	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb	$aaaa$...
$M(w) \Rightarrow$	✓	×	✓	∞	×	∞	✓	∞	∞	×	×	×	∞	×	✓	...
$L(M) =$	{ a ,		aa ,				aaa ,								$aaaa$... }

M can **run forever** on an input, which is implicitly a reject (since it is not an accept).