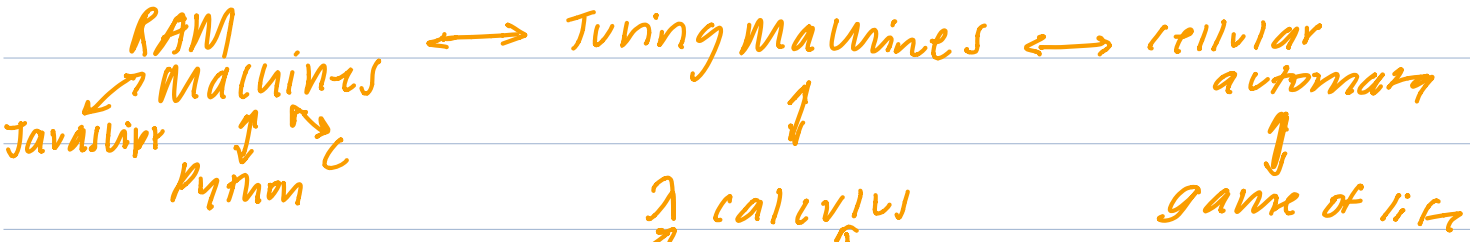
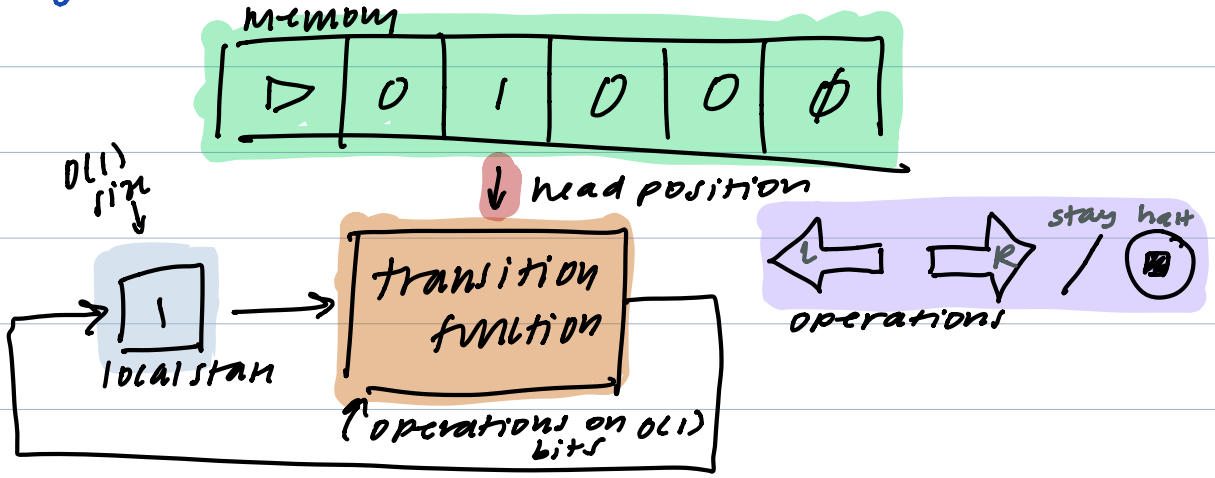
CS12| Lecture 8

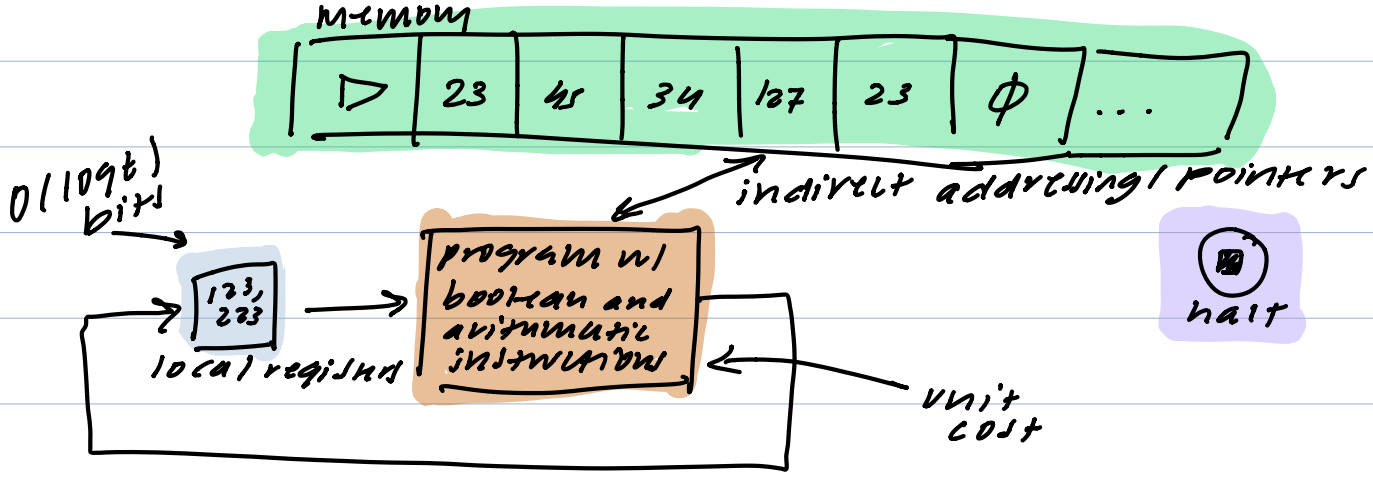
September 26



Turing Machine



RAM Machine



- unbounded memory - holding ints
- finite # of int registers
- indirect addressing: Memory[register]
- Finite # of logic instructions
- Looping / halting

NAND-RAM

- Scalar variables (foo) hold ints $\{0, 1, \ldots T\}$
- Array of ints Bar
- can do foo = Bar[blah] or Bar[foo] = blah
- NAND-RAM arithmetic operations
- if statements
- loops: while, do, etc.

Thm: F is computable by NAND-RAM iff F computable by NAND-TM

Proof: Syntactic sugar

Internal Loops:

```
def f(n):
    i = 0
    print("-")
    while (i<n):
        print("*")
        i += 1
    print("-")
```

```
def f(n):
    i = 0
    a, b, c = False, False, False
    while not a:
        if not b: print("-"); b=True
        while (i<n):
            print("*"); i+=1;
        if not c: print("-"); c=True
```

Load Foo into i:

```
def Load(Bar):
    i = 0
    while(Bar[i] == 0):
        i += 1
    return i
```

Foo[Bar]:

```
def Get(Foo, Bar):
    j = Load(Bar)
    return Foo[j]
```

2-dim arrays:

```
def Mult(Foo, Bar):
    i = 0; k = 0
    while Foo[i] == 0:
        j = 0
        while Foo[j] == 0:
            j += 1
            k += 1
        i += 1
    Blah[k] = 1
    return Blah
```

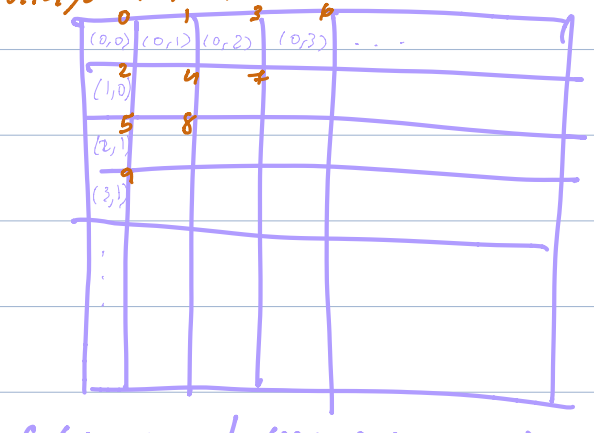
ex: Give a one-to-one function $f: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

Have: 1-dim arrays

Want: 2-dim arrays

$A[i,j] \longleftrightarrow B[f(i,j)]$

diagonization:



$f(x,y) = \frac{1}{2}(x+y)(x+y+1) + x$

Integer Arrays

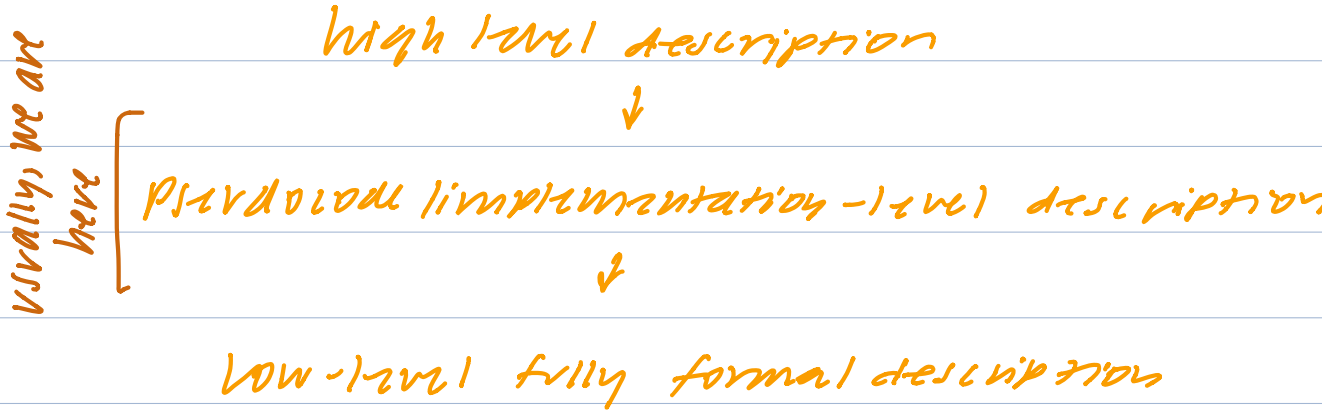
Show we can encode any list of ints as list of list of bits:

A: List of bits = string, can encode integers as strings

Cor: 2-dim bit array = 1-dim integer array

When we want to prove F can be computed use most powerful model

When we want to prove F cannot be computed use weakest model



Computable

$f(a) = a^2$, $a \in \mathbb{N}$

$$g(x,y) = \begin{cases} 1 & x \text{ prefix of } y \\ 0 & \text{o/w} \end{cases}$$

Cellular Automata

Configuration: coloring of pixels/cells in 1D, 2D, or 3D space.

- Every pixel assigned one of finitely many colors/symbols

Every step: New color at pixel p is set to RULE(color of p's neighborhood)

- If p is blank and all neighbors are blank, it stays blank

Example: Game of life

- <2 neighbors or >3 neighbors : die
- alive and 2-3 neighbors : survival
- not alive and 3 neighbors : reproduction

Thm: 1-D cellular automata A computes $F: \{0,1\}^* \to \{0,1\}^*$ if when initial A in conf $E(\triangleright)E(x_0)\cdots E(x_{n-1})$

1. eventually pixel 0 gets color 1
2. when that happens, pixel 1 gets color $F(x)$

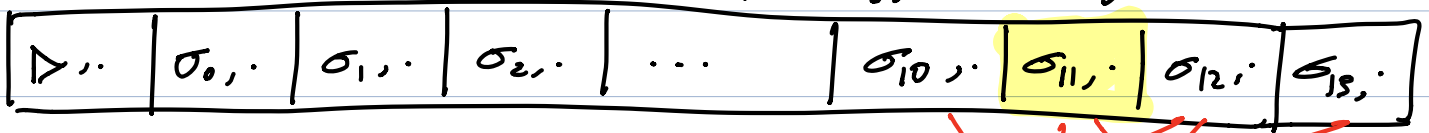
Thm: F is computable iff computable by a 1-D CA

configuration

Def: If M is TM with k states and tape alphabet Σ, configuration of M is string α over $\Gamma \cdot \Sigma^* (\cdot V[k])$ s.t. $a_i = (a, \cdot)$ for exactly 1

- a string, the ith of it = size of memory a TM has used so far. Encode state of TM

string over alphabet $\Sigma \times (\{\cdot\} \cup V[k])$ encoding config:



config in next step