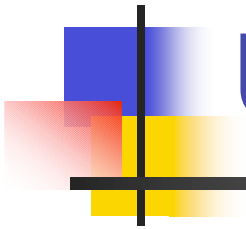
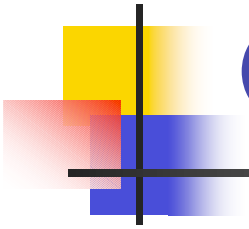


Universality

John E Savage

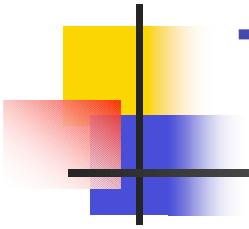
Computer Science
Brown University

CBSSS 2004
July 17, 2004



Overview

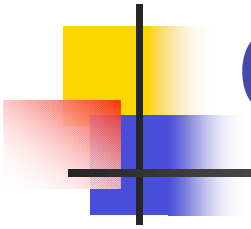
- Are some computational models more powerful than others?
  - When resources, such as time and space, are unlimited?
  - When resources are limited?
- Are there problems not solvable by any computer?



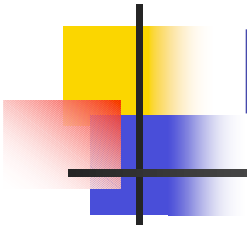
Task Descriptions

- Tasks defined by **functions** $f : B^* \alpha B^*$, $B = \{0,1\}$, that map binary strings to binary strings.
- Tasks also defined by **languages**. A language L is a set of strings over an alphabet $\Sigma = B^r$.
  - Can languages be defined by functions?

Equivalence Between Computational Models

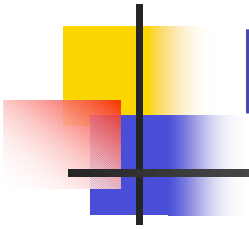


- Unlimited Resources
  - Two models are equivalent if they can both solve the same problem.
- Limited Resources
  - Two models are equivalent if they both solve the same problem P with “about the same resources”.
  - E.g. P is solvable in time polynomial in the length of its description on RAM and TM.



Models to be Compared

- Logic Circuit
  - Unstructured circuits
  - Crossbars
- Finite-State Machine
- Random Access Machine
- Turing Machine
- Multidimensional Mesh/Cellular Automaton
- PRAM



Logic Circuits

- The **basis** for a circuit is the set of logic gates used to realize it.
- A basis is **complete** if every Boolean function $f : B^n \alpha B$ can be realized.
  - {AND, OR, NOT} is complete
  - Is {NAND} complete?
  - Why is {AND, OR} not complete?

Circuit Complexity

- The **circuit complexity** of f : B^n α B^m over basis Ω, denoted C_Ω (f), is the smallest number of gates in circuit for f.

- If Ω_a and Ω_b are complete bases, then C_{Ω_a} (f) = $O(C_{\Omega_b})$ and C_{Ω_b} (f) = $O(C_{\Omega_a})$.

- h(n) = O(g(n)) if exists K and N such that h(n) · K g(n) for n ¸ N.

Monotone Circuits

- A monotone function $f(x_1, \Lambda , x_n)$ is constant or increases from 0 to 1 as inputs increase from 0 to 1.
- Each monotone function has a circuit over the monotone basis Ω_{mon} = {AND, OR}.
- Over Ω_{mon} some of them are very complex.
- Can NOTs reduce their complexity?

Disjunctive Normal Form (DNF) of f : $B^n \alpha B$

- DNF is OR of minterms, each an AND of all literals of f.

x	y	min_0	min_1	min_2	min_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

x	y	$f(x,y)$
0	0	1
0	1	0
1	0	0
1	1	1

$$min_0 = \bar{x} \wedge \bar{y},\ min_1 = \bar{x} \wedge y,\ min_2 = x \wedge \bar{y},$$
$$min_2 = x \wedge y$$
$$f(x,y) = min_0 \vee min_3$$

Conjunctive Normal Form (CNF) of f : $B^n \alpha B$

- CNF is AND of maxterms, each an OR of all literals of f.

x	y	max_0	max_1	max_2	max_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

x	y	$f(x,y)$
0	0	1
0	1	0
1	0	0
1	1	1

$$max_0 = x \vee y,\ max_1 = x \vee \bar{y},\ max_2 = \bar{x} \vee y,$$
$$max_2 = \bar{x} \vee \bar{y}$$
$$f(x,y) = max_1 \wedge max_2$$

Complexity of Normal Forms

- Some functions have very large CNF and DNF.
- Parity $f(x_1, \Lambda, x_n) = x_1 \copyright \Lambda \copyright x_n$ requires $2^{n/2}$ minterms in DNF and $2^{n/2}$ maxterms in CNF.
- Algebraic rules often used to simplify DNF and CNF. E.g. xy Ç xz = x(y Ç z).

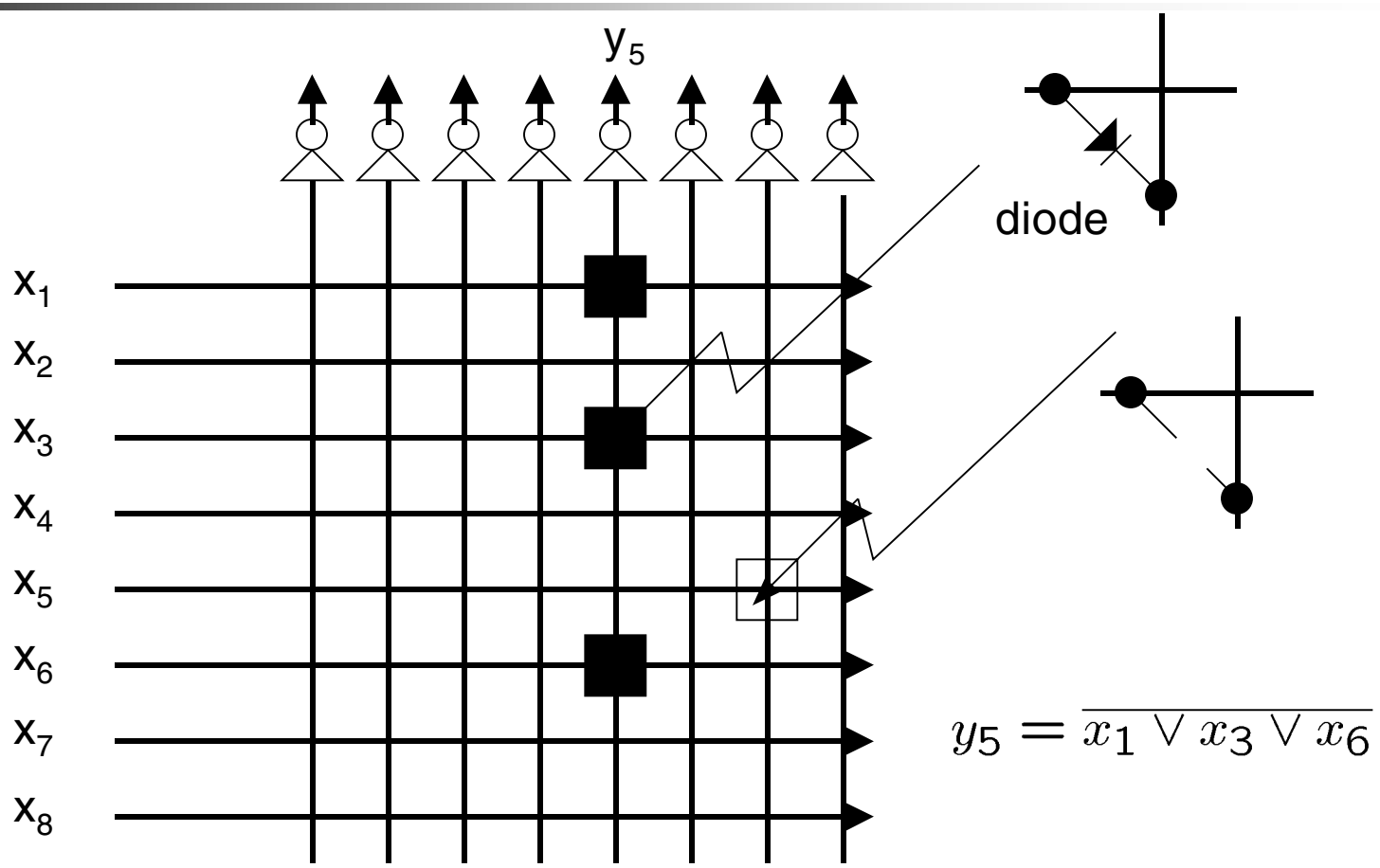
Bounded-Depth Circuits

- Bounded-depth circuits: alternate ANDs and ORs a fixed number of times.
- Can propagate NOTs to inputs without changing depth using DeMorgan's Rules

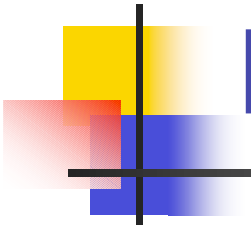
$$\overline{x \vee y} = \overline{x} \wedge \overline{y}$$

$$\overline{x \wedge y} = \overline{x} \vee \overline{y}$$

The Programmable Crossbar Realizes "Wired NORs"



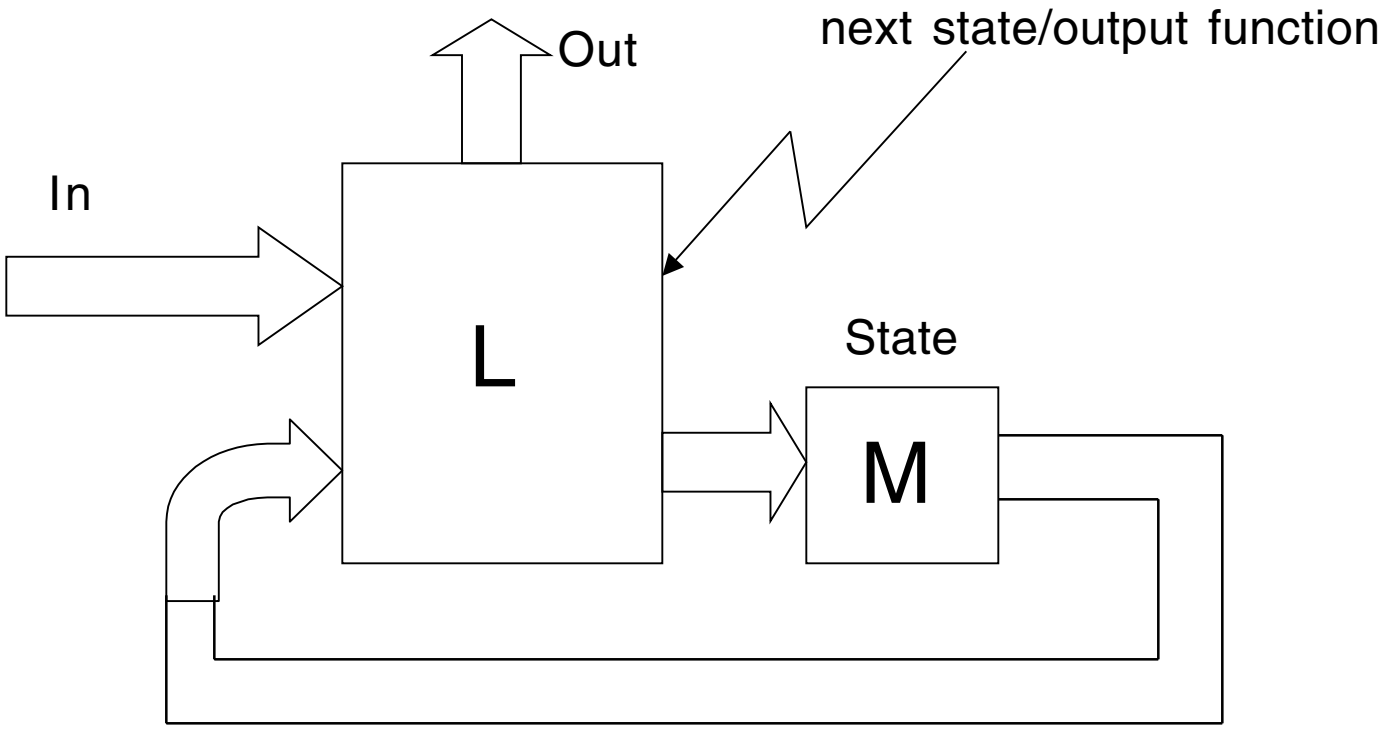
Bounded-Depth Circuits May Be Very Large



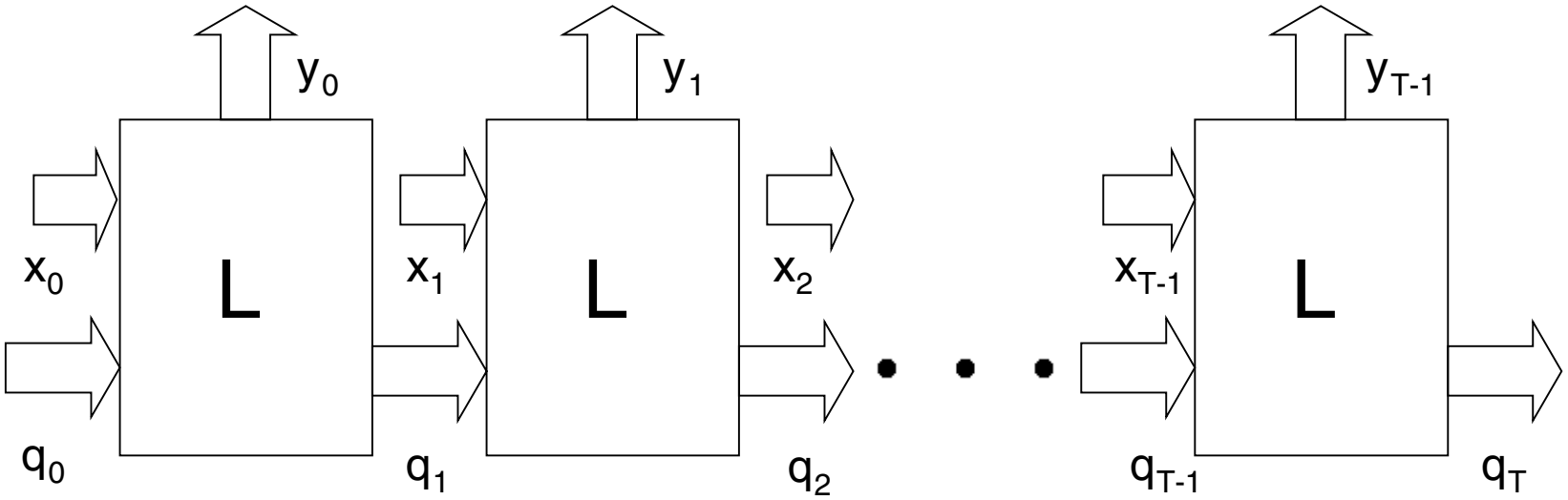
- Cascading programmable crossbars generates a bounded-depth circuit.

- The parity function on n inputs has exponential circuit size in bounded-depth circuits. (Furst, Saxe & Sipser, Atjai, Yao, Hastad)

FSM Model

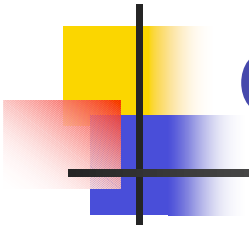


Simulating FSM Executing T Steps with a Circuit



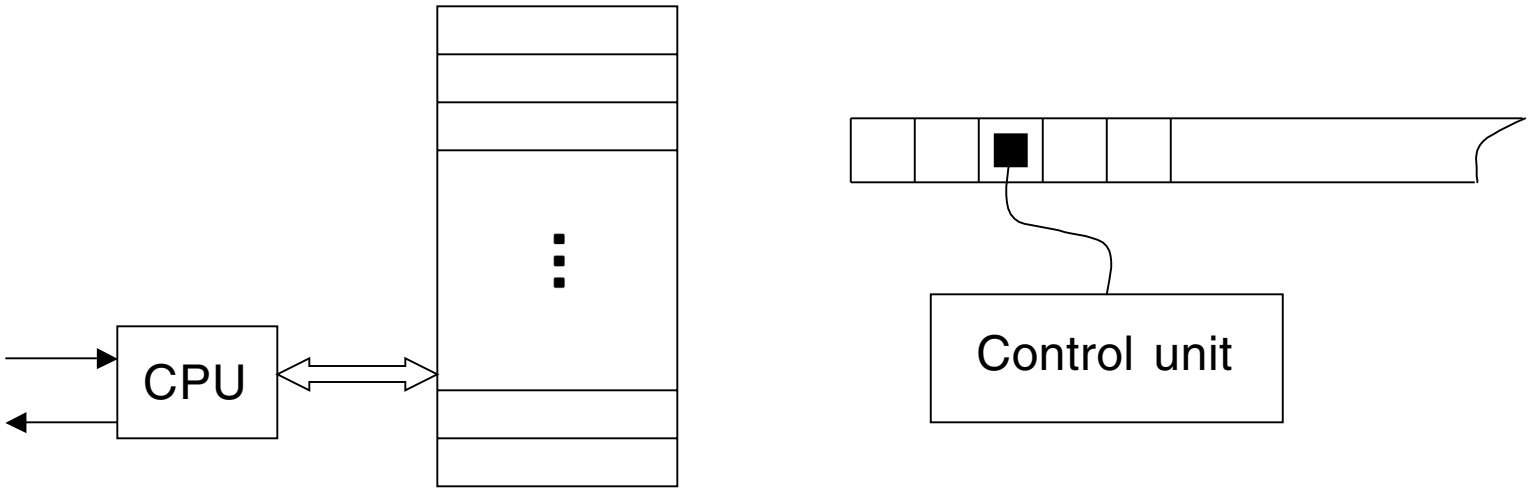
- For finite memory-based computations, {AND, OR, NOT} are universal.
- Circuit simulation of FSM not feasible when time is unbounded.

Simulating Circuits and FSMs on RAM



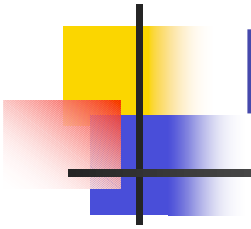
- A circuit is equivalent to a straight-line program – no branching or looping
- FSM is circuit with a loop.
- Both are easily simulated on a RAM, the model assumed by imperative programs

Random Access Machine Versus Turing Machine

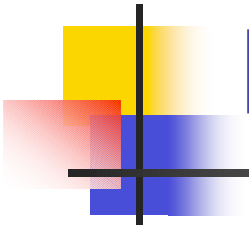


- Simulating TM on RAM straightforward
- How to simulate RAM on TM?

Hardware Realizations of RAM and TM

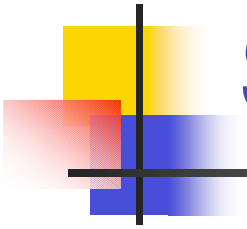


- If size of memory is bounded, both RAM and TM are FSMs and can be realized by circuits with memory over a complete basis.



Equivalence of TM and RAM

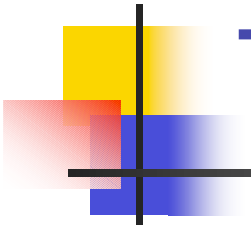
- TM can simulate RAM and vice versa
- Simulations can be done “efficiently.”



Simulating RAM on TM

- Must simulate CPU – equivalent to implementing loop of fixed size.
  - Shuttling back on forth on tape using markers will simulate CPU.
  - Use two tape tracks, one for data, another for markers
- How to simulate memory as it grows?
  - Store address with data!

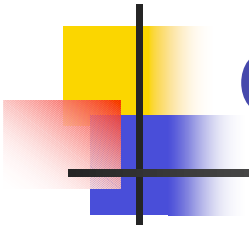
Storing RAM Data on TM Tape



- If value **v** is written at address **a** in RAM for first time, write (**a**, **v**) at end of non-blank portion of TM tape.

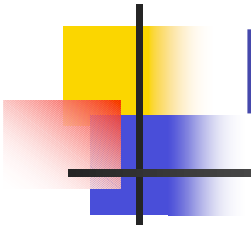
- If writing second time at **a**, invalidate old value and write (**a**, **v'**) at end of non-blank portion of TM tape where **v'** is new value to be written.

Time to Simulate T-Step RAM on TM

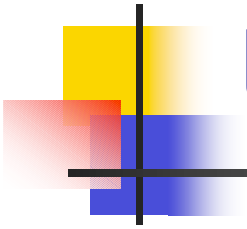


- To find one bit of address, entire tape of TM may have to be scanned.
- The length of the tape grows with T.
  - If RAM can only add words or shift them by one place (no multiplication), the length of words grows by at most constant amount on each step
- Each step of RAM requires $O(T^2)$ TM steps.
- $O(T^3)$ steps on TM to simulate T-step RAM

Polynomial Equivalence of RAM and TM



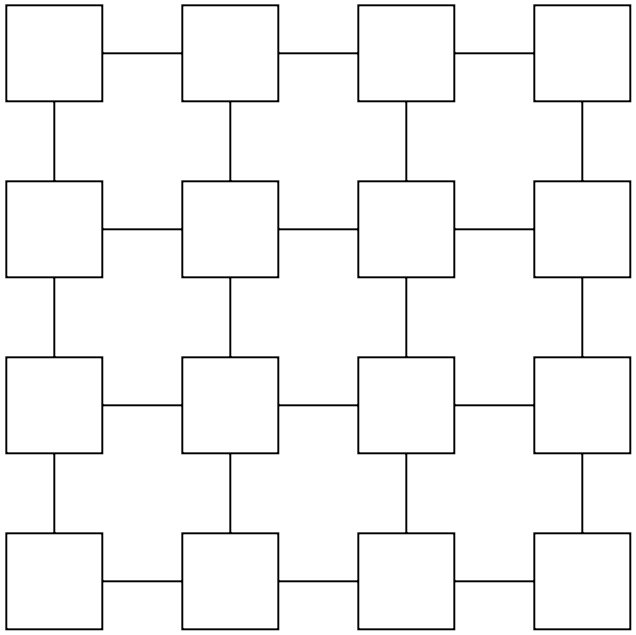
- A T-step problem on RAM can be simulated in time polynomial in T, on a TM and vice versa.
- **P**, the problems requiring polynomial time on RAM, require polynomial time on TM and vice versa.
- TM is model of choice for theoretical CS



Universal Turing Machine

- A **universal TM U** is a TM that can simulate an arbitrary TM on its input
- First step: How to encode a TM?
  - TM described by its FSM control unit
  - FSM can be described by tables
- For **U** to simulate T, it uses value under head and current state to obtain new value under head and next state.
  - **U** repeatedly reads input, does state transition, writes new tape value, and moves head.

Simulating 2D Mesh/Cellular Automaton (CA) on RAM/TM



Cells are FSMs

I/O on periphery

- Mesh can be simulated as double loop on RAM

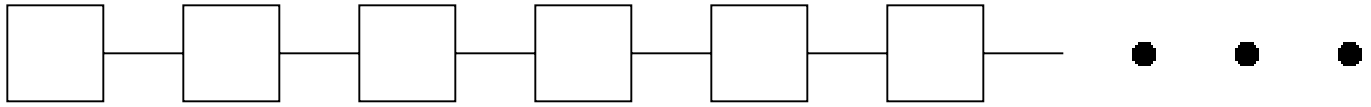
Simulating TM on CA

- TM "state" captured by configuration **c**.
- Let **t** = (t_1, t_2, Λ, t_s) be non-blank tape.
- **c** = $(t_1, t_2, \Lambda, t_{j-1}, \langle q, t_j \rangle, t_{j+1}, \Lambda, t_s)$ if head over jth cell and control in state q.
  - **c**'s alphabet contains t_i's and $\langle q, t_j \rangle$.
- If TM has configuration **c**, how is new configuration **c**' determined?

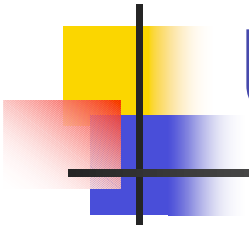
Simulating TM on CA (cont.)

- If TM in state q reading t_j, writes t'_j, goes to state q* and moves head, right: $\mathbf{c}'=(t_1, t_2,\Lambda, t_{j-1}, t'_j, \langle q^*,t_{j+1}\rangle, t_{j+2},\Lambda, t_s)$
- Similar changes if it moves left.
- Non-blank symbol t_{s+1} may be added.
- Can cellular automaton simulate TM?

CA Simulates TM



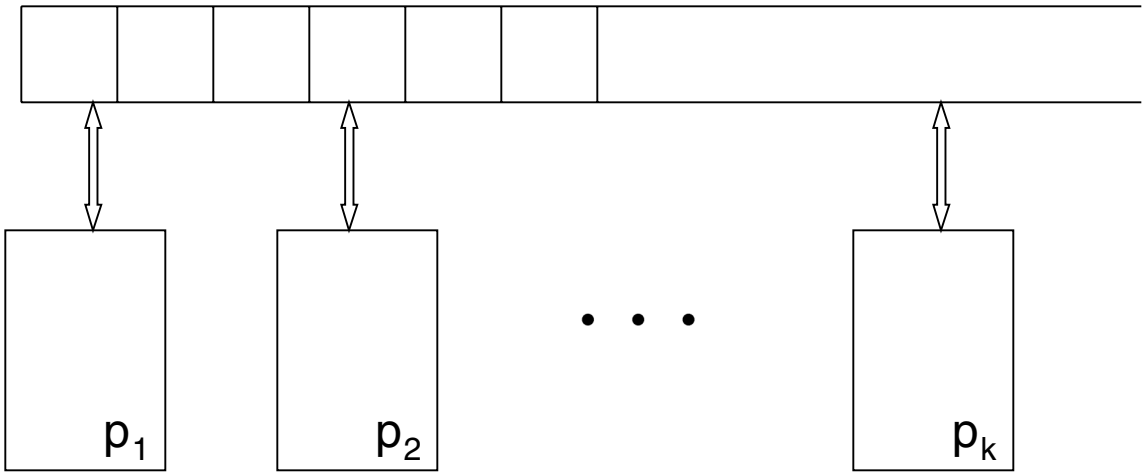
- Let each cell contain one component of configuration.
- Which cell changes state?



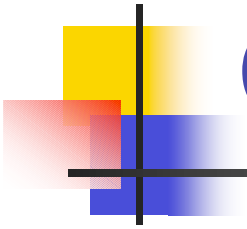
Universal Machines

- Every problem computable on a TM, RAM, or CA can be computed on one of the other machines.
- The time and space used can differ substantially from machine to machine.
  - The CA is parallel!

Simulating EREW PRAM on RAM/TM



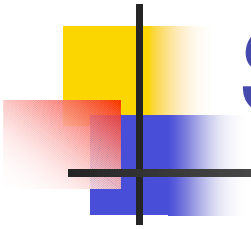
- Intersimulation possible of PRAM, RAM & TM
- Computation times will be very different.



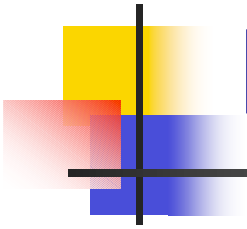
Church-Turing Hypothesis

- Every computable function can be computed by a Turing machine.
- Are there well-defined non-computable functions?

Enumerating TMs and Strings



- Can the FSMs be enumerated?
  - Fix input/output alphabets
  - Enumerate FSMs by number of states
- Can TMs be enumerated, T_1, T_2, Λ ?
  - Control units are TMs
- How about input strings, $\mathbf{x}_1$, $\mathbf{x}_2$, Λ ?
  - Enumerate by length



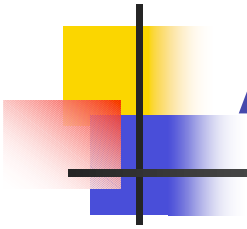
Language Recognition by TM

- A TM recognizes language L if it accepts each string in L by halting.
  - A TM can enter an infinite loop!
- For a TM M to process a string, place it left-adjusted on M's tape. If it is accepted, the TM halts and writes 1 left-adjusted on otherwise blank tape.

Diagonalization

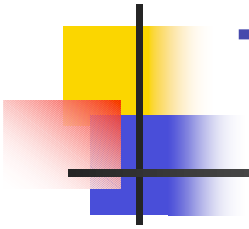
	T_1	T_2	T_3	T_4	$\ldots$
x_1	0	1	0	0	1
x_2	1	1	1	1	0
x_3	0	0	0	1	1
x_4	1	1	1	1	0
x_5	1	0	0	0	0
$\vdots$	$\ldots$				

- Define language L_1 by diagonalization
  - L_1 contains $\mathbf{x}_i$ if T_i does not accept it.
  - In example, L_1 contains $\mathbf{x}_1$, $\mathbf{x}_3$ and $\mathbf{x}_5$.



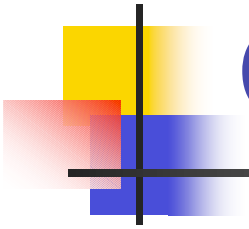
A Non-Computable Language

- Assume a TM M exist recognizing L_1.
  - M is T_j for some j.
  - If $\mathbf{x}_j$ is in L_1, T_j accepts it. But if T_j accepts $\mathbf{x}_j$, it cannot be in L_1.
  - If $\mathbf{x}_j$ is **not** in L_1, $\mathbf{x}_j$ is not accepted by T_j. But this this implies that it must be in L_1.
- The assumption that M exists is incorrect. L_1 is not recognizable by any TM.



The Halting Problem

- Can there exist a TM that **H** that when given a description of a TM M and its input string **w** (as with a universal TM), **H** can tell whether M will halt on **w**?
- If **H** exists, we can use it together with a universal TM **U** to recognize L_1.
  - How? Find j such that $\mathbf{x}_j = \mathbf{w}$. Also find T_j.
- This implies that **H** does not exist!



Conclusions

- Computer science has many models.
  - How can we model I/O latency?
- As new technologies are introduced, they often result in new models.
- Fascinating new questions arise for which modeling and analysis help to provide the answers.