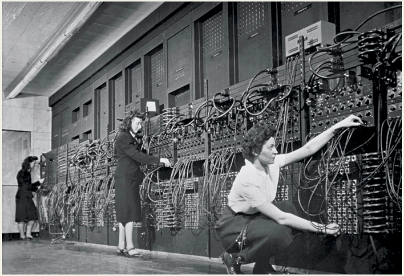
BBM401-Reading: Variants of Turing Machines

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs373/fa2010/lectures
https://courses.engr.illinois.edu/cs498374/lectures.html

Special purpose machines?

- Different DFA for different languages (duh)
- Different TMs for different languages, functions.
- Early computer programming was no different



Von Neumann Architecture

- stored-program computer
  - programs can be data!
  - program-as-data determines subcircuits to employ
- fetch-decode-execute cycle
- hence, one computer can behave like any

http://idiomzero.blogspot.com/2010/07/8-anecdotes-about-john-von-neumann.html

Original Idea was due to Turing

"I know that in or about 1943 or '44 von Neumann was well aware of the fundamental importance of Turing's paper of 1936 ... Von Neumann introduced me to that paper and at his urging I studied it with care. Many people have acclaimed von Neumann as the "father of the computer" (in a modern sense of the term) but I am sure that he would never have made that mistake himself. He might well be called the midwife, perhaps, but he firmly emphasized to me, and to others I am sure, that the fundamental conception is owing to Turing— in so far as not anticipated by Babbage ... "

- Stan Frankel – Los Alamos

Universal TM

- A *single* TM M_u that can compute anything computable!
- Takes as input
  - the ***description*** of some *other* TM M
  - data w for M to run on
- Outputs
  - the results of running $M(w)$

Need to make precise what the *description* of a TM is

Coding of TMs

- Show how to represent every TM as a natural number
- *Lemma*: If L over alphabet {0,1} is accepted by some TM M, then there is a one-tape TM M' that accepts L, such that
  - $\Gamma = \{0,1,B\}$
  - states numbered 1, ..., k
  - q_1 is the unique start state
  - q_2 is the unique halt/accept state
  - q_3 is the unique halt/reject state
- So, to represent a TM, we need only list its set of transitions – everything else is implicit by above

Listing Transition

- Use the following order:

  $\delta(q_1,0), \delta(q_1,1), \delta(q_1,B), \delta(q_2,0), \delta(q_2,1), \delta(q_2,B),\ldots$

  $\ldots\ \delta(q_k,0), \delta(q_k,1), \delta(q_k,B).$

- Use the following encoding:

  $111\ t_1\ 11\ t_2\ 11\ t_3\ 11\ \ldots\ 11\ t_{3k}\ 111$

where t_i is the encoding of transition *i* as given on the next slide.

Encoding a transition

Recall transition looks like $\delta(q,a) = (p, b, L)$

So, encode as

<state> 1 <input> 1 <new state> 1 <new-symbol> 1 <direction>

where

- state q_i represented by 0^i
- 0, 1, B represented by 0, 00, 000
- L, R, S represented by 0, 00, 000

$\delta(q_3, 1) = (q_4, 0, R)$ represented by $\underbrace{000}_{q_3}1\underbrace{00}_{1}1\underbrace{0000}_{q_4}1\underbrace{0}_{0}1\underbrace{00}_{R}$

Typical TM code:

1110101000010010011010010000010101 1.....11......11......111

- Begins, ends with 111
- Transitions separated by 11
- Fields within transition separated by 1
- Individual fields represented by 0s

TMs are (binary) numbers

- Every TM is encoded by a unique element of N
- Convention: elements of N that do not correspond to any TM encoding represent the "null TM" that accepts nothing.
- Thus, every TM is a number, and vice versa
- Let <*M*> mean the number that encodes *M*
- Conversely, let M_n be the TM with encoding *n.*

Universal TM M_u

Construct a TM M_u such that

$L(M_u) = \{ \langle M \rangle \# w \mid M \text{ accepts } w \}$

Thus, M_u is a stored-program computer.

It reads a program $\langle M \rangle$ and executes it on data w

M_u simulates the run of M on w

A single TM captures the notion of "computable" !!

How M_u works

3 tapes

- Tape 1: holds input *M* and *w*; never changes
- Tape 2: simulates *M*'s single tape
- Tape 3: holds *M*'s current state

1	1	1	t_1	1	1	t_2	1	1	...	t_{3k}	1	1	1	#	w

Input M

Input w

Universal TM M_u

Phase 1: Check if <*M*> is a valid TM on tape 1

- No four 1's in a row
- Three initial, ending 1's
- substring 110^i10^j1 doesn't appear twice
- appropriate number of 0's between 1's in transition codes: 11000010100000100001...
  (0000 does not encode a 0,1,or B to write)
- could check that transitions are in right order, and form a complete set (but not necessary)
- etc.

If not valid, then halt and reject

Phase 2: Set up

- copy *w* to tape 2, with head scanning first symbol
- write 0 on tape 3 indicating *M* is in start state q_1

Tape 1

111010100001001001101001000001010111......111 # 100110

Code for *M*

Tape 2

$<u>1</u>00110

Current contents of *M*'s tape

Tape 3

$<u>0</u>

Current state of *M*

If at any time, Tape 3 holds 00 (or 000), then halt and accept (or reject)

Phase 3: Repeatedly simulate steps of *M*

Where in code is next transition?

move tape 2 head to the right

what to do next

Tape 1

111010100001001001101001000001010011......111 # 100110

Code for *M*

copy new state 00000 to tape 3

Tape 2

$100110

Current contents of *M*'s tape

write a 0 under tape 2's head

Tape 3

$0

Current state of *M*

If tape 3 holds 0^i and tape 2 is scanning 1, then search for substring 110^i1001 on tape 1

Phase 3: After the single move

move tape 2 head to the right

Tape 1

111010100001001001101001000001010011......111 # 100110

Code for *M*

copy new state 00000 to tape 3

Tape 2

$000110

Current contents of *M*'s tape

write a 0 under tape 2's head

Tape 3

$00000

Current state of *M*

Check if 00 or 000 is on tape 3; if so, halt and accept or reject

Otherwise, simulate the next move by searching for pattern.

In this example, the next pattern = 1100000101

Towards "real" computers: RAMs

Random Access Machine:

- finite number of arithmetic registers
- infinite number of memory locations
- instruction set (next page)
- program instructions written in continuous block of memory starting at location 1 and all registers set to 0.

RAM instruction set

Instruction	Meaning
Add X, Y	Add contents of register X and Y, and place result in register X
LOADC X, num	Place constant num in register X
LOAD X, M	Put contents of memory loc M into register X
LOADI X, M	Indirect addressing: put value(value(M)) into register X
STORE X, M	Copy contents of reg X into mem location M
JUMP X, M	If register X = 0, then next instruction is at memory location M (otherwise, next instruction is the one following the current one, as usual)
HALT	Halt (duh)

TMs can simulate RAMs

- Can write a "TM-interpreter" of RAM code Thus, no more TM programming.
- Actual simulation has low overhead (though memory is not random-access).

TM tapes

- Instruction-location tape
  - stores memory location where next instruction is
  - initially contains only “1”
- Register tape
  - stores register numbers and their contents, as follows: # <reg-num> # <contents> # .. etc.
  - Example: suppose R1 has 11, and R4 has 101, and all other registers are empty. Then register tape:

$	#	1	,	1	1	#	1	0	0	,	1	0	1	#	.	.	.

TM tapes

- Memory tape – similar to register tape, but can hold numbers, OR instructions:

numbers: # <mem-location> , <value> # ...

instructions:

example: mem location 101 holds ADD 3,6

	#	1	0	1	,	A D D	,	1	1	,	1	1	0	#	.	.	.

single symbol

- 5 work tapes

TM setup

- Blank register tape
- Memory tape holds RAM program, starting at memory location 1. No other data stored.
- 1 on instruction-location tape

TM step overview

(many TM steps for each RAM step)

- Read instruction-location tape
- search memory tape for the instruction
- execute the instruction, changing register and memory tapes as needed
- update the location-instruction tape

In other words, it goes through a fetch-decode-execute cycle

Example

- Suppose instruction location tape holds only:

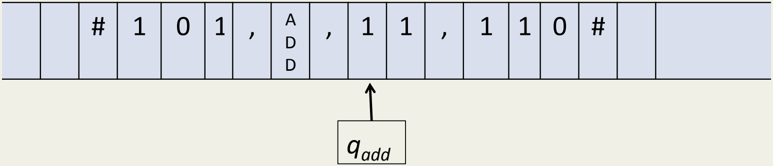
$	1	0	1														

- Scan memory tape, looking for "# 1 0 1 ,"
  Suppose it finds

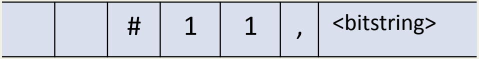
.	.	#	1	0	1	,	A D D	,	1	1	,	1	1	0	#		

- It finds "ADD" following "," and switches to special state q_{add} to handle the addition

Example (cont.)



- first argument is in register 11 so search register tape for:



- then copy <bitstring> to worktape 1
- similarly, search for, find, place value of register 110 onto worktape 2

Example (cont.)

- Now go to subroutine to add worktape 1 + worktape 2, place results on worktape 3.
- Result must go back into register 11
- Search register tape again for

		#	1	1	,	<bitstring>

- Replace <bitstring> with new value copied from worktape 3, shifting as necessary
- Add 1 to instruction-location tape

RAM simulation

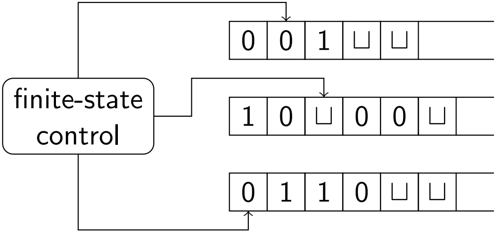
- *MANY* details left out
- Other types of instructions are similar
- Number of steps to simulate RAM?
- Delicate issue.... does RAM actually have constant-time access to *infinite* memory?
- Can show (beyond this course) for "reasonable" time model on a RAM, if $T(n)$ steps are required, then on a TM, only $T(n)^2$ steps. ($T(n)^3$ if RAM has mult. and div.)

Church-Turing thesis

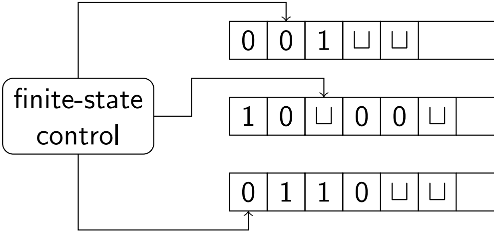
- TMs capture notion of "computable"
- Evidence
  - RAM computer
  - general recursive functions (Gödel & Herbrand)
    - constant/projection/successor/composition/recursion
  - λ-calculus (Church) for defining functions (CS 421)
  - general string-rewriting-system
    - unrestricted grammar, productions of form $\alpha \rightarrow \beta$ for any α and β
  - attempts to extend TMs

All give you exactly the TM-computable functions

Multi-Tape Turing Machine

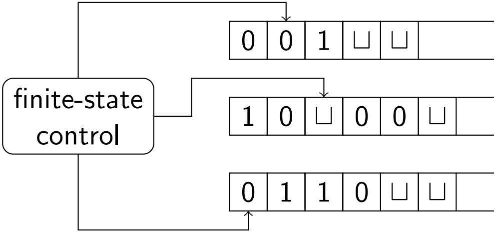


Multi-Tape Turing Machine



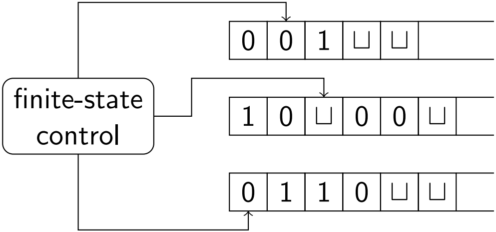
- Input on Tape 1

Multi-Tape Turing Machine



- Input on Tape 1
- Initially all heads scanning cell 1, and tapes 2 to k blank

Multi-Tape Turing Machine



- Input on Tape 1
- Initially all heads scanning cell 1, and tapes 2 to k blank
- In one step: Read symbols under each of the k-heads, and depending on the current control state, write new symbols on the tapes, move the each tape head (possibly in different directions), and change state.

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ where

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set of control states

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state
- $q_{acc} \in Q$ is the accept state

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state
- $q_{\mathrm{acc}} \in Q$ is the accept state
- $q_{\mathrm{rej}} \in Q$ is the reject state, where $q_{\mathrm{rej}} \neq q_{\mathrm{acc}}$

Multi-Tape Turing Machine

Formal Definition

A k-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state
- $q_{acc} \in Q$ is the accept state
- $q_{rej} \in Q$ is the reject state, where $q_{rej} \neq q_{acc}$
- $\delta : (Q \setminus \{q_{acc}, q_{rej}\}) \times \Gamma^k \to Q \times (\Gamma \times \{L, R\})^k$ is the transition function.

Computation, Acceptance and Language

- A configuration of a multi-tape TM must describe the state, contents of all k-tapes, and positions of all k-heads. Thus, $\mathrm{C} \in Q \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the head position.

Computation, Acceptance and Language

- A configuration of a multi-tape TM must describe the state, contents of all k-tapes, and positions of all k-heads. Thus, $\mathrm{C} \in Q \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the head position.
- Accepting configuration is one where the state is q_{acc}, and starting configuration on input w is $(q_0, *w, *\sqcup, \ldots, *\sqcup)$

Computation, Acceptance and Language

- A configuration of a multi-tape TM must describe the state, contents of all k-tapes, and positions of all k-heads. Thus, $\mathrm{C} \in Q \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the head position.
- Accepting configuration is one where the state is q_{acc}, and starting configuration on input w is $(q_0, *w, *\sqcup, \ldots, *\sqcup)$
- Formal definition of a single step is skipped.

Computation, Acceptance and Language

- A configuration of a multi-tape TM must describe the state, contents of all k-tapes, and positions of all k-heads. Thus, $\mathrm{C} \in Q \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the head position.
- Accepting configuration is one where the state is q_{acc}, and starting configuration on input w is $(q_0, *w, *\sqcup, \ldots, *\sqcup)$
- Formal definition of a single step is skipped.
- w is accepted by M, if from the starting configuration with w as input, M reaches an accepting configuration.

Computation, Acceptance and Language

- A configuration of a multi-tape TM must describe the state, contents of all k-tapes, and positions of all k-heads. Thus, $\mathrm{C} \in Q \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the head position.
- Accepting configuration is one where the state is q_{acc}, and starting configuration on input w is $(q_0, *w, *\sqcup, \ldots, *\sqcup)$
- Formal definition of a single step is skipped.
- w is accepted by M, if from the starting configuration with w as input, M reaches an accepting configuration.
- $L(M) = \{w \mid w \text{ accepted by } M\}$

Expressive Power of multi-tape TM

Expressive Power of multi-tape TM

Theorem

For any k-tape Turing Machine M, there is a single tape TM $single(M)$ such that $L(single(M)) = L(M)$.

Expressive Power of multi-tape TM

Theorem

For any k-tape Turing Machine M, there is a single tape TM $single(M)$ such that $L(single(M)) = L(M)$.

Challenges

- How do we store k-tapes in one?

Expressive Power of multi-tape TM

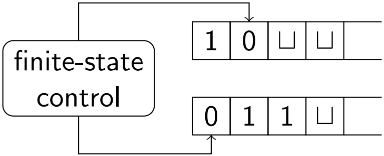
Theorem

For any k-tape Turing Machine M, there is a single tape TM $single(M)$ such that $L(single(M)) = L(M)$.

Challenges

- How do we store k-tapes in one?
- How do we simulate the movement of k independent heads?

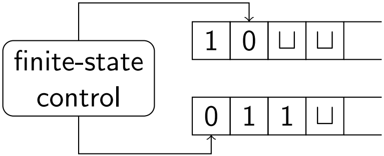
Storing Multiple Tapes



Multi-tape TM M

Store in cell i contents of cell i of all tapes.

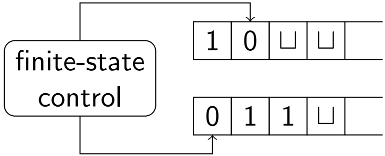
Storing Multiple Tapes



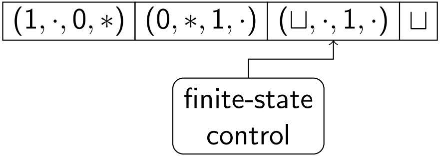
Multi-tape TM M

Store in cell i contents of cell i of all tapes. “Mark” head position of tape with $*$.

Storing Multiple Tapes



Multi-tape TM M



1-tape equivalent single(M)

Store in cell i contents of cell i of all tapes. "Mark" head position of tape with $*$.

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the state, as we encounter them.

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the state, as we encounter them.

Challenge 2: After this scan, 1-tape TM knows the next step of k-tape TM. How do we change the contents and move the heads?

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the state, as we encounter them.

Challenge 2: After this scan, 1-tape TM knows the next step of k-tape TM. How do we change the contents and move the heads?

- Once again, scan the tape, change all relevant contents, "move" heads (i.e., move $*$s), and change state.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
   - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
   - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
   - Scan back from right-to-left moving the heads that need to be moved left.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
    - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
    - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
    - Scan back from right-to-left moving the heads that need to be moved left.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
    - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
    - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
    - Scan back from right-to-left moving the heads that need to be moved left.

Formal construction in notes.

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, where

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$, where

- $Q, \Sigma, \Gamma, q_0, q_{\text{acc}}, q_{\text{rej}}$ are as before for 1-tape machine

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, where

- $Q, \Sigma, \Gamma, q_0, q_{acc}, q_{rej}$ are as before for 1-tape machine
- $\delta : (Q \setminus \{q_{acc}, q_{rej}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\})$

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\vdash$ is defined similarly. $X_1X_2\cdots X_{i-1}qX_i\cdots X_n \vdash X_1X_2\cdots pX_{i-1}Y\cdots X_n$, if $(p, Y, L) \in \delta(q, X_i)$

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\vdash$ is defined similarly. $X_1X_2\cdots X_{i-1}qX_i\cdots X_n \vdash X_1X_2\cdots pX_{i-1}Y\cdots X_n$, if $(p, Y, L) \in \delta(q, X_i)$; case for right moves is analogous.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\vdash$ is defined similarly. $X_1X_2\cdots X_{i-1}qX_i\cdots X_n \vdash X_1X_2\cdots pX_{i-1}Y\cdots X_n$, if $(p, Y, L) \in \delta(q, X_i)$; case for right moves is analogous.
- w is accepted by M, if from the starting configuration with w as input, M reaches an accepting configuration, for some sequence of choices at each step.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\vdash$ is defined similarly. $X_1X_2\cdots X_{i-1}qX_i\cdots X_n \vdash X_1X_2\cdots pX_{i-1}Y\cdots X_n$, if $(p, Y, L) \in \delta(q, X_i)$; case for right moves is analogous.
- w is accepted by M, if from the starting configuration with w as input, M reaches an accepting configuration, for some sequence of choices at each step.
- $L(M) = \{w \mid w \text{ accepted by } M\}$

Expressive Power of Nondeterministic TM

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine M, there is a (deterministic) TM $det(M)$ such that $L(det(M)) = L(M)$.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine M, there is a (deterministic) TM $det(M)$ such that $L(det(M)) = L(M)$.

Proof Idea

$\det(M)$ will simulate M on the input.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine M, there is a (deterministic) TM $det(M)$ such that $L(det(M)) = L(M)$.

Proof Idea

$\det(M)$ will simulate M on the input.

- Idea 1: $\det(M)$ tries to keep track of all possible "configurations" that M could possibly be after each step.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine M, there is a (deterministic) TM $det(M)$ such that $L(det(M)) = L(M)$.

Proof Idea

$\det(M)$ will simulate M on the input.

- Idea 1: $\det(M)$ tries to keep track of all possible "configurations" that M could possibly be after each step. Works for DFA simulation of NFA

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine M, there is a (deterministic) TM $det(M)$ such that $L(det(M)) = L(M)$.

Proof Idea

$\det(M)$ will simulate M on the input.

- Idea 1: $\det(M)$ tries to keep track of all possible "configurations" that M could possibly be after each step. Works for DFA simulation of NFA but not convenient here.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine M, there is a (deterministic) TM $det(M)$ such that $L(det(M)) = L(M)$.

Proof Idea

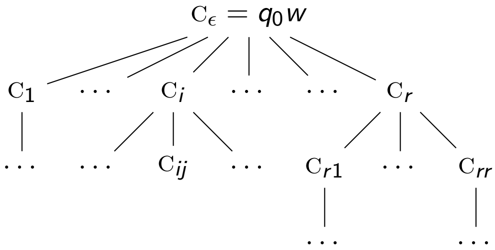
det(M) will simulate M on the input.

- Idea 1: det(M) tries to keep track of all possible "configurations" that M could possibly be after each step. Works for DFA simulation of NFA but not convenient here.
- Idea 2: det(M) will simulate M on each possible sequence of computation steps that M may try in each step.

Nondeterministic Computation

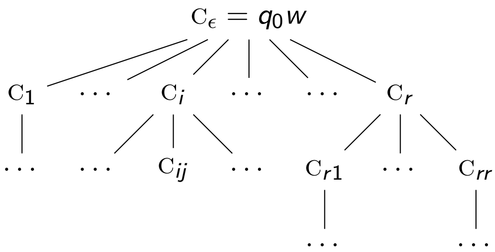
- If $r = \max_{q,X} |\delta(q, X)|$ then the runs of M can be organized as an r-branching tree.

Nondeterministic Computation



- If $r = \max_{q,X} |\delta(q, X)|$ then the runs of M can be organized as an r-branching tree.
- $\mathrm{C}_{i_1 i_2 \cdots i_n}$ is the configuration of M after n-steps, where choice i_1 is taken in step 1, i_2 in step 2, and so on.

Nondeterministic Computation



- If $r = \max_{q,X} |\delta(q, X)|$ then the runs of M can be organized as an r-branching tree.
- $\mathrm{C}_{i_1 i_2 \cdots i_n}$ is the configuration of M after n-steps, where choice i_1 is taken in step 1, i_2 in step 2, and so on.
- Input w is accepted iff $\exists$ accepting configuration in tree.

Proof Idea

The machine det(M) will search for an accepting configuration in computation tree

Proof Idea

The machine $\det(M)$ will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating M on the appropriate sequence of nondeterministic choices

Proof Idea

The machine $\det(M)$ will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating M on the appropriate sequence of nondeterministic choices
- $\det(M)$ will perform a BFS on the tree.

Proof Idea

The machine det(M) will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating M on the appropriate sequence of nondeterministic choices
- det(M) will perform a BFS on the tree. Why not a DFS?

Proof Idea

The machine det(M) will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating M on the appropriate sequence of nondeterministic choices
- det(M) will perform a BFS on the tree. Why not a DFS?

Observe that det(M) may not terminate if w is not accepted.

Proof Details

det(M) will use 3 tapes to simulate M

Proof Details

det(M) will use 3 tapes to simulate M

Proof Details

det(M) will use 3 tapes to simulate M (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called input tape, will always hold input w

Proof Details

det(M) will use 3 tapes to simulate M (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called input tape, will always hold input w
- Tape 2, called simulation tape, will be used as M's tape when simulating M on a sequence of nondeterministic choices

Proof Details

det(M) will use 3 tapes to simulate M (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called input tape, will always hold input w
- Tape 2, called simulation tape, will be used as M's tape when simulating M on a sequence of nondeterministic choices
- Tape 3, called choice tape, will store the current sequence of nondeterministic choices

Execution of det(M)

1. Initially: Input tape contains w, simulation tape and choice tape are blank
2. Copy contents of input tape onto simulation tape
3. Simulate M using simulation tape as its (only) tape
   1. At the next step of M, if state is q, simulation tape head reads X, and choice tape head reads i, then simulate the ith possibility in $\delta(q, X)$; if i is not a valid choice, then goto step 4
   2. After changing state, simulation tape contents, and head position on simulation tape, move choice tape's head to the right. If Tape 3 is now scanning $\sqcup$, then goto step 4
   3. If M accepts then accept and halt, else goto step 3(1) to simulate the next step of M.
4. Write the lexicographically next choice sequence on choice tape, erase everything on simulation tape and goto step 2.

Deterministic Simulation

In a nutshell

- det(M) simulates M over and over again, for different sequences, and for different number of steps.

Deterministic Simulation

In a nutshell

- det(M) simulates M over and over again, for different sequences, and for different number of steps.
- If M accepts w then there is a sequence of choices that will lead to acceptance. det(M) will eventually have this sequence on choice tape, and then its simulation M will accept.

Deterministic Simulation

In a nutshell

- $\det(M)$ simulates M over and over again, for different sequences, and for different number of steps.
- If M accepts w then there is a sequence of choices that will lead to acceptance. $\det(M)$ will eventually have this sequence on choice tape, and then its simulation M will accept.
- If M does not accept w then no sequence of choices leads to acceptance. $\det(M)$ will therefore never halt!

Deciding a Language

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}
- A Turing machine may keep running forever on some input

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}
- A Turing machine may keep running forever on some input
- Then the machine does not accept that input

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}
- A Turing machine may keep running forever on some input
- Then the machine does not accept that input
- So two ways to not accept: reject or never halt

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}
- A Turing machine may keep running forever on some input
- Then the machine does not accept that input
- So two ways to not accept: reject or never halt

Definition

A Turing machine M is said to decide a language L if $L = L(M)$ and M halts on every input

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}
- A Turing machine may keep running forever on some input
- Then the machine does not accept that input
- So two ways to not accept: reject or never halt

Definition

A Turing machine M is said to decide a language L if $L = L(M)$ and M halts on every input

Deciding a language is more than recognizing it.

Deciding a Language

- Only halting configurations are those with state q_{acc} or q_{rej}
- A Turing machine may keep running forever on some input
- Then the machine does not accept that input
- So two ways to not accept: reject or never halt

Definition

A Turing machine M is said to decide a language L if $L = L(M)$ and M halts on every input

Deciding a language is more than recognizing it. There are languages which are recognizable, but not decidable.