Formal Languages, Automata and Computation

Turing Machines

Carnegie Mellon University in Qatar

TURING MACHINES-SYNOPSIS

- The most general model of computation
- Computations of a TM are described by a sequence of configurations.
    - Accepting Configuration
    - Rejecting Configuration
- Turing-recognizable languages
    - TM halts in an accepting configuration if *w* is in the language.
    - TM may halt in a rejecting configuration or go on indefinitely if *w* is not in the language.
- Turing-decidable languages
    - TM halts in an accepting configuration if *w* is in the language.
    - TM halts in a rejecting configuration if *w* is not in the language.

Nondeterministic Turing Machines

- We defined the state transition of the ordinary TM as

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

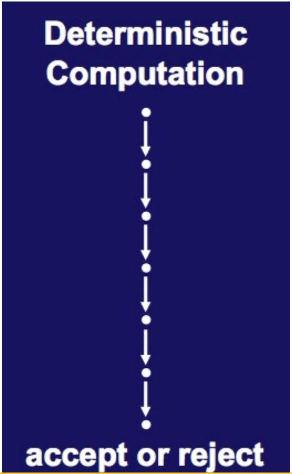
- A nondeterministic TM would proceed computation with multiple next cnfigurations. δ for a nondeterministic TM would be

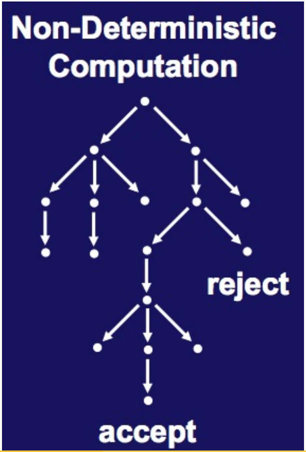
$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

  ($\mathcal{P}(S)$ is the power set of S.)
- This definition is analogous to NFAs and PDAs.

NONDETERMINISTIC TURING MACHINES

- A computation of a Nondeterministic TM is a tree, where each branch of the tree is looks like a computation of an ordinary TM.





Nondeterministic Turing Machines

- If a single branch reaches the accepting state, the Nondeterministic TM accepts, even if other branches reach the rejecting state.
- What is the power of Nondeterministic TMs?
  - Is there a language that a Nondeterministic TM can accept but no deterministic TM can accept?

Nondeterministic Turing Machines

Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing Machine.

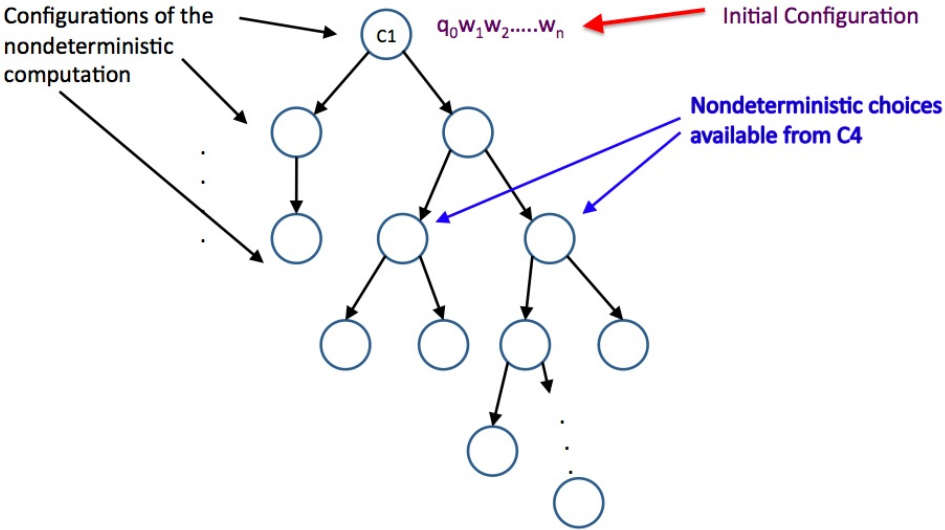
Proof Idea

- Timeshare a deterministic TM to different branches of the nondeterministic computation!
- Try out all branches of the nondeterministic computation until an accepting configuration is reached on one branch.
- Otherwise the TM goes on forever.

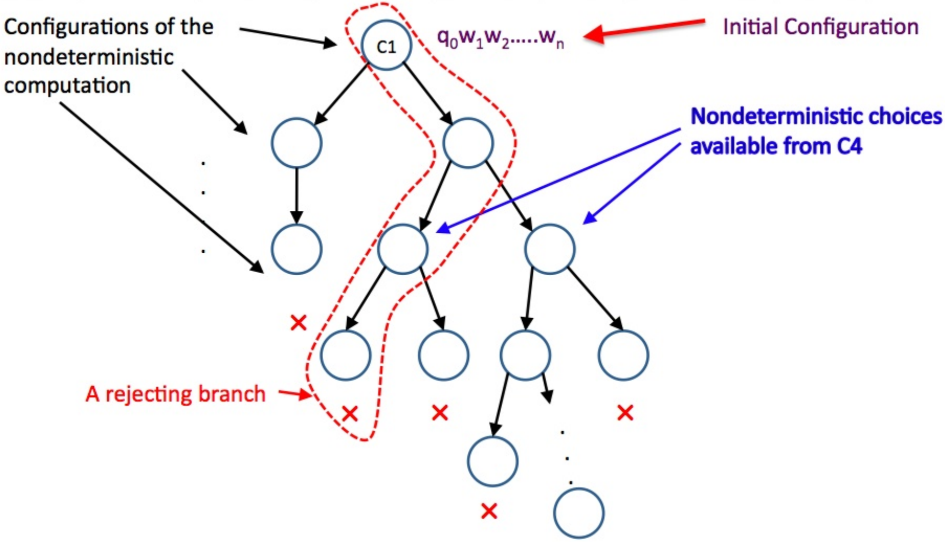
Nondeterministic Turing Machines

- Deterministic TM D simulates the Nondeterministic TM N.
- Some of branches of the N's computations may be infinite, hence its computation tree has some infinite branches.
- If D starts its simulation by following an infinite branch, D may loop forever even though N's computation may have a different branch on which it accepts.
- This is a very similar problem to processor scheduling in operating systems.
  - If you give the CPU to a (buggy) process in an infinite loop, other processes "starve".
- In order to avoid this unwanted situation, we want D to execute all of N's computations concurrently.

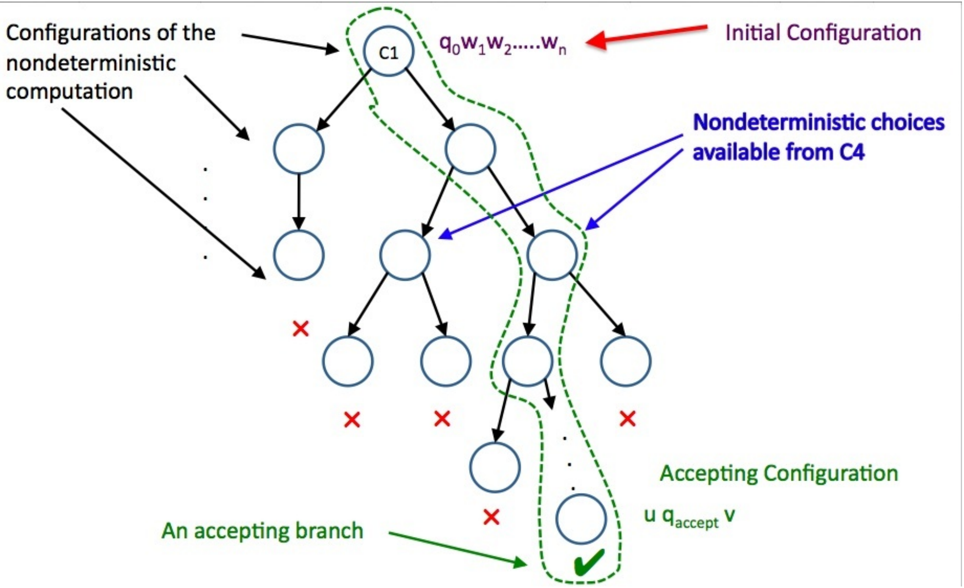
Nondeterministic Computation



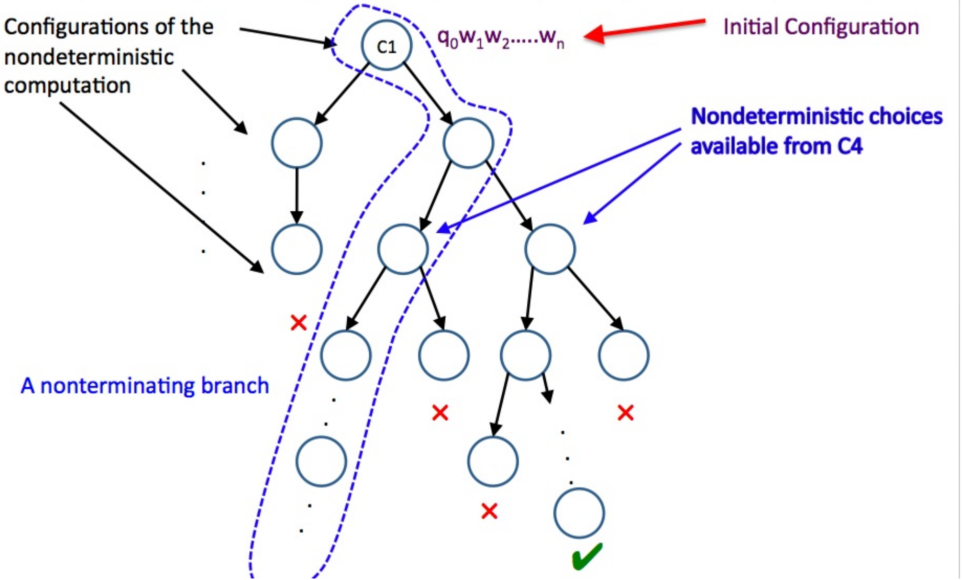
NONDETERMINISTIC COMPUTATION



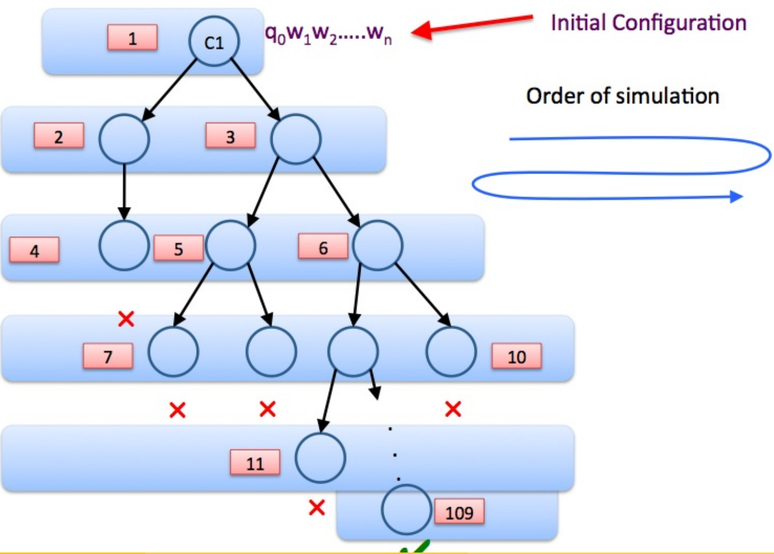
NONDETERMINISTIC COMPUTATION



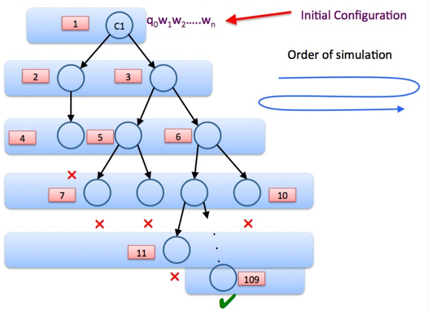
NONDETERMINISTIC COMPUTATION



Simulating Nondeterministic Computation



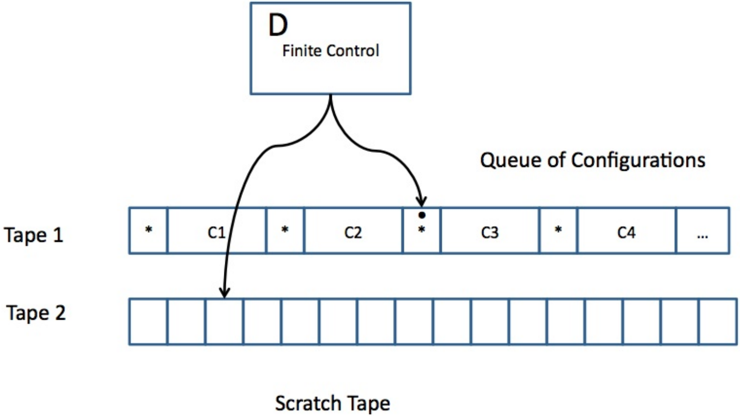
Simulating Nondeterministic Computation



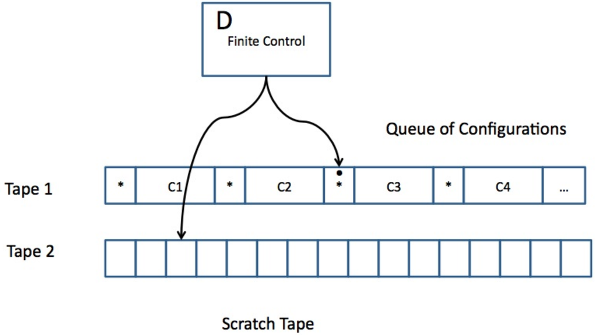
- During simulation, *D* processes the configurations of *N* in a breadth-first fashion.
- Thus *D* needs to maintain a queue of *N*'s configurations (Remember queues?)
- *D* gets the next configuration from the head of the queue.
- *D* creates copies of this configuration (as many as needed)
- On each copy, *D* simulates one of the nondeterministic moves of *N*.
- *D* places the resulting configurations to the back of the queue.

Structure of the Simulating DTM

- N is simulated with 2-tape DTM, D
  - Note that this is different from the construction in the book!

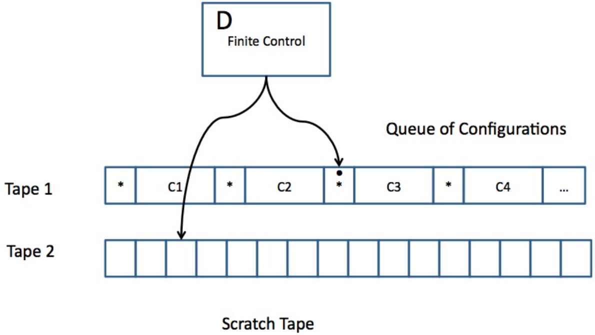


How D Simulates N



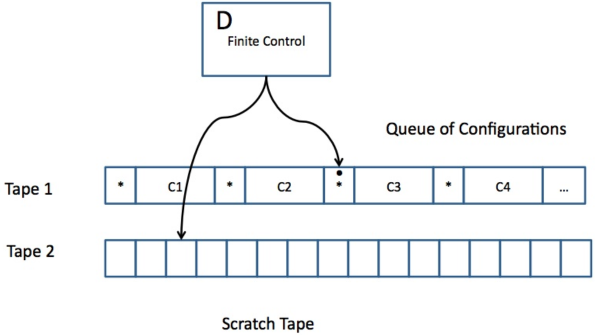
- Built into the finite control of D is the knowledge of what choices of moves N has for each state and input.

HOW *D* SIMULATES *N*



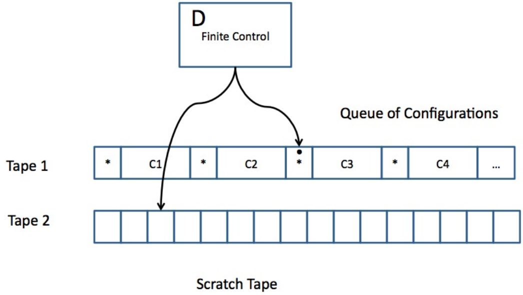
1. *D* examines the state and the input symbol of the current configuration (right after the dotted separator)
2. If the state of the current configuration is the accept state of *N*, then *D* accepts the input and stops simulating *N*.

How D Simulates N



1. D copies k copies of the current configuration to the scratch tape.
2. D then applies one nondeterministic move of N to each copy.

How *D* Simulates *N*



3. *D* then copies the new configurations from the scratch tape, back to the end of tape 1 (so they go to the back of the queue), and then clears the scratch tape.
4. *D* then returns to the marked current configuration, and "erases" the mark, and "marks" the next configuration.
5. *D* returns to step 1), if there is a next configuration. Otherwise rejects.

How D Simulates N

- Let m be the maximum number of choices N has for any of its states.
- Then, after n steps, N can reach at most $1 + m + m^2 + \cdots + m^n$ configurations (which is at most nm^n)
- Thus D has to process at most this many configurations to simulate n steps of N.
- Thus the simulation can take exponentially more time than the nondeterministic TM.
- It is not known whether or not this exponential slowdown is necessary.

Implications

Corollary

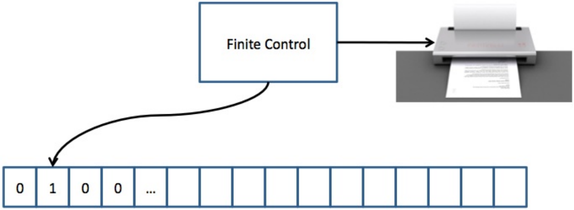
A language is Turing-recognizable if and only if some nondeterministic TM recognizes it.

Corollary

A language is decidable if and only of some nondeterministic TM decides it.

ENUMERATORS

- Remember we noted that some books used the term recursively enumerable for Turing-recognizable.
- This term arises from a variant of a TM called an enumerator.



- TM generates strings one by one.
- Everytime the TM wants to add a string to the list, it sends it to the printer.

Enumerators

- The enumerator E starts with a blank input tape.
- If it does not halt, it may print an infinite list of strings.
- The strings can be enumerated in any order; repetitions are possible.
- The language of the enumerator is the collection of strings it eventually prints out.

ENUMERATORS

THEOREM

A language is Turing recognizable if and only if some enumerator enumerates it.

PROOF.

The If-part: If an enumerator E enumerates the language A then a TM M recognizes A.

$M =$ "On input w

1. Run E. Everytime E outputs a string, compare it with w.
2. If w ever appears in the output of E, *accept.*"

Clearly M accepts only those strings that appear on E's list.

□

ENUMERATORS

THEOREM

A language is Turing recognizable if and only if some enumerator enumerates it.

PROOF.

The Only-If-part: If a TM M recognizes a language A, we can construct the following enumerator for A. Assume $s_1, s_2, s_3, \ldots$ is a list of possible strings in Σ^*.

$E =$ "Ignore the input

1. Repeat the following for $i = 1, 2, 3, \ldots$
2.   Run M for i steps on each input $s_1, s_2, s_3, \ldots s_i$.
3.   If any computations accept, print out corresponding s_j."

If M accepts a particular string, it will appear on the list generated by E (in fact infinitely many times)

The Definition of Algorithm - History

- in 1900, Hilbert posed the following problem:

  "Given a polynomial of several variables with integer coefficients, does it have an integer root – an assignment of integers to variables, that make the polynomial evaluate to 0"

- For example, $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3, z = 0$.
- Hilbert explicitly asked that an algorithm/procedure to be "devised". He assumed it existed; somebody needed to find it!
- 70 years later it was shown that no algorithm exists.
- The intuitive notion of an algorithm may be adequate for giving algorithms for certain tasks, but was useless for showing no algorithm exists for a particular task.

The Definition of Algorithm - History

- In early 20^{th} century, there was no formal definition of an algorithm.
- In 1936, Alonzo Church and Alan Turing came up with formalisms to define algorithms. These were shown to be equivalent, leading to the

Church-Turing Thesis

Intutitive notion of algorithms $\equiv$ Turing Machine Algorithms

The definition of an Algorithm

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$
- Hilbert's 10^{th} problem in TM terminology is "Is D decidable?" (No!)
- However D is Turing-recognizable!
- Consider a simpler version
  $D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with integral roots}\}$
- M_1 = "The input is polynomial p over x.
  1. Evaluate p with x successively set to 0, 1, -1, 2, -2, 3, -3, ...
  2. If at any point, p evaluates to 0, *accept*."
- D_1 is actually decidable since only a finite number of x values need to be tested (math!)
- D is also recognizable: just try systematically all integer combinations for all variables.

Describing Turing Machines and Their Inputs

- For the rest of the course we will have a rather standard way of describing TMs and their inputs.
- The input to TMs have to be strings.
- Every object O that enters a computation will be represented with an string $\langle O \rangle$, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges $\langle O \rangle = \texttt{(1,2,3,4)((1,2),(2,3),(3,1),(1,4))}$
- Then we can define problems over graphs,e.g., as:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

Describing Turing Machines and Their Inputs

- A TM for this problem can be given as:
- $M =$ "On input $\langle G \rangle$, the encoding of a graph G:
  1. Select the first node of G and mark it.
  2. Repeat 3) until no new nodes are marked
  3. For each node in G, mark it, if there is edge attaching it to an already marked node.
  4. Scan all the nodes in G. If all are marked, the *accept*, else *reject*"

OTHER OBJECT ENCODINGS

- DFAs: Represent as a graph with 4 components, q_0, F, δ as a list of labeled edges.
- TMs: Represent as a string encoding δ with blocks of 5 components, e.g., q_i, a, q_j, b, L. Assume that q_0 is always the start state and q_1 is the final state.
  - Individual symbols can even be encoded using only two symbols e.g. just $\{0, 1\}$.