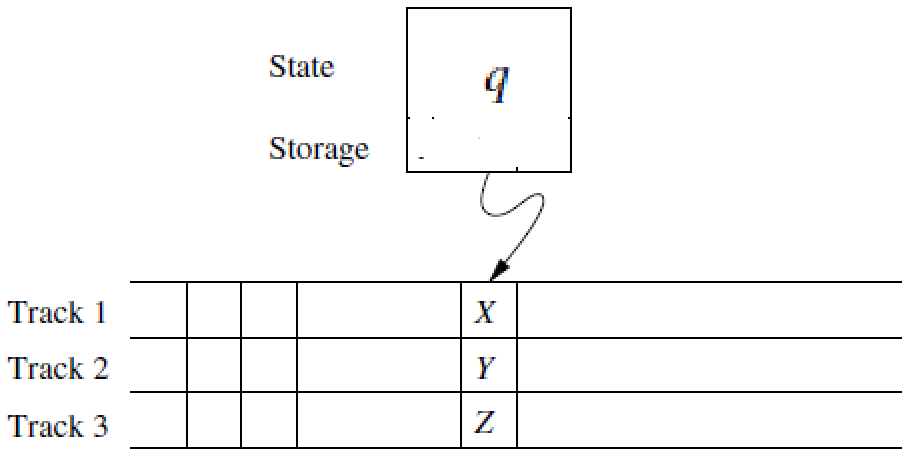
Variants of TM

Multi-track, multi-tape, NTM

https://www.andrew.cmu.edu/user/ko/pdfs/lecture-14.pdf

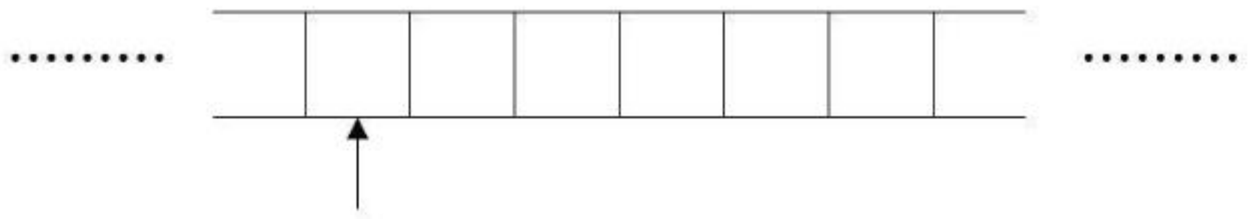
Multi-track



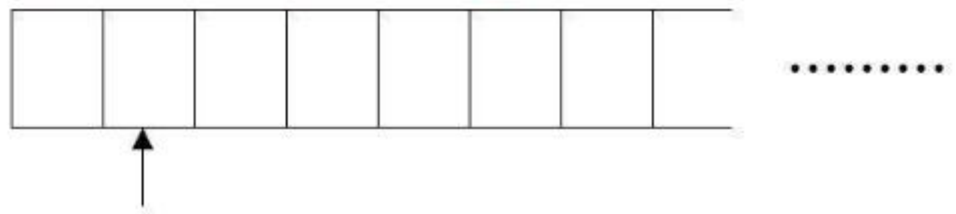
- In this example, the tape symbol is the triplet (X, Y, Z) and we can see the tape as a single-track one.

Semi-infinite tape

Standard machine

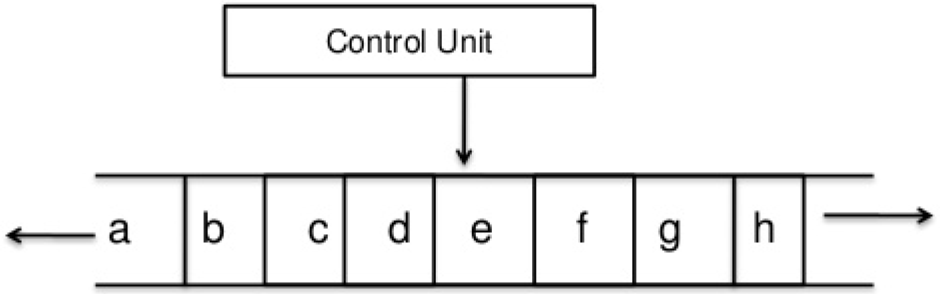


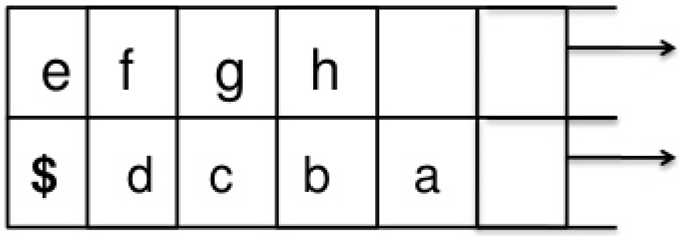
Semi-infinite tape machine



Simulation of two way infinite by semi-infinite tape

- Two way infinite tape simulated by semi -infinite tape





Multi-tape

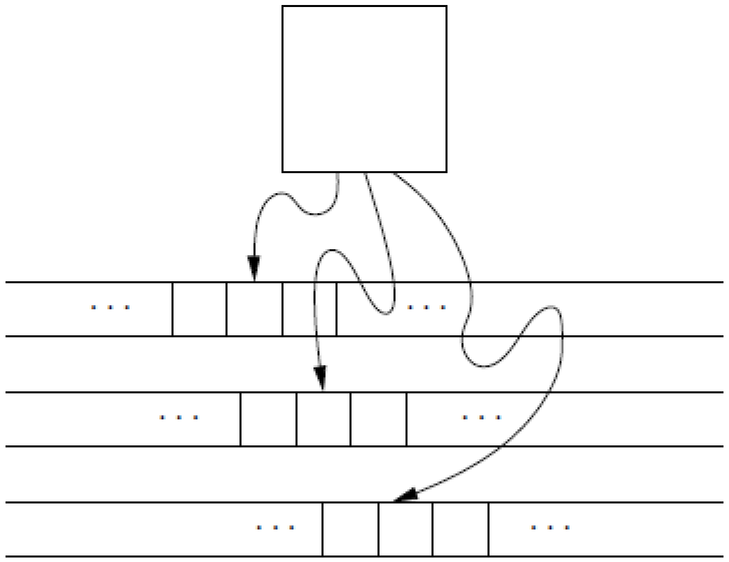
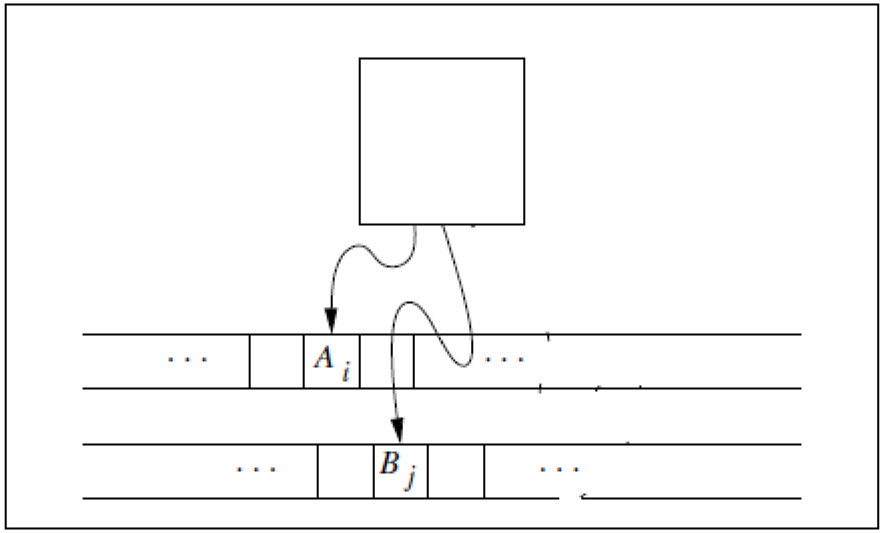


Figure 8.16: A multitape Turing machine

Simulation of multi-tape by one-tape



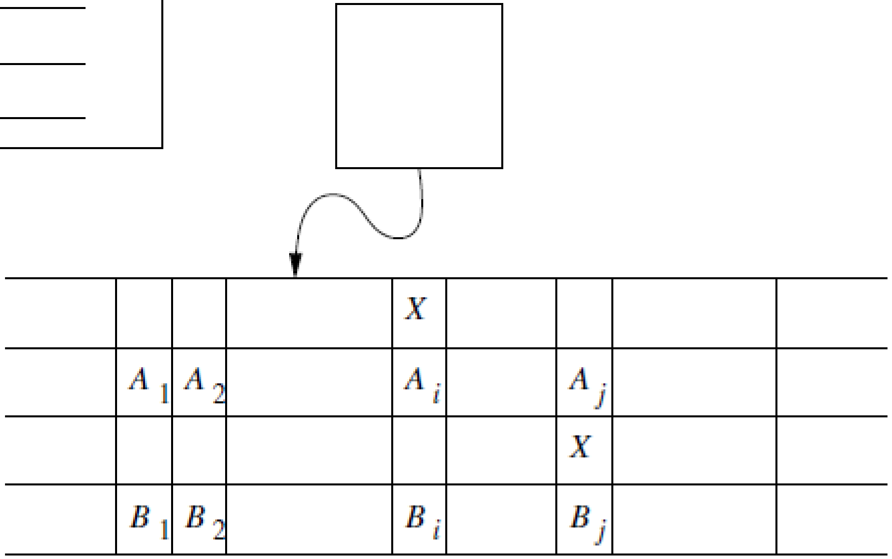


Figure 8.17: Simulation of a two-tape Turing machine by a one-tape Turing machine

NTM

NONDETERMINISTIC TM

- There is a choice in the next move.

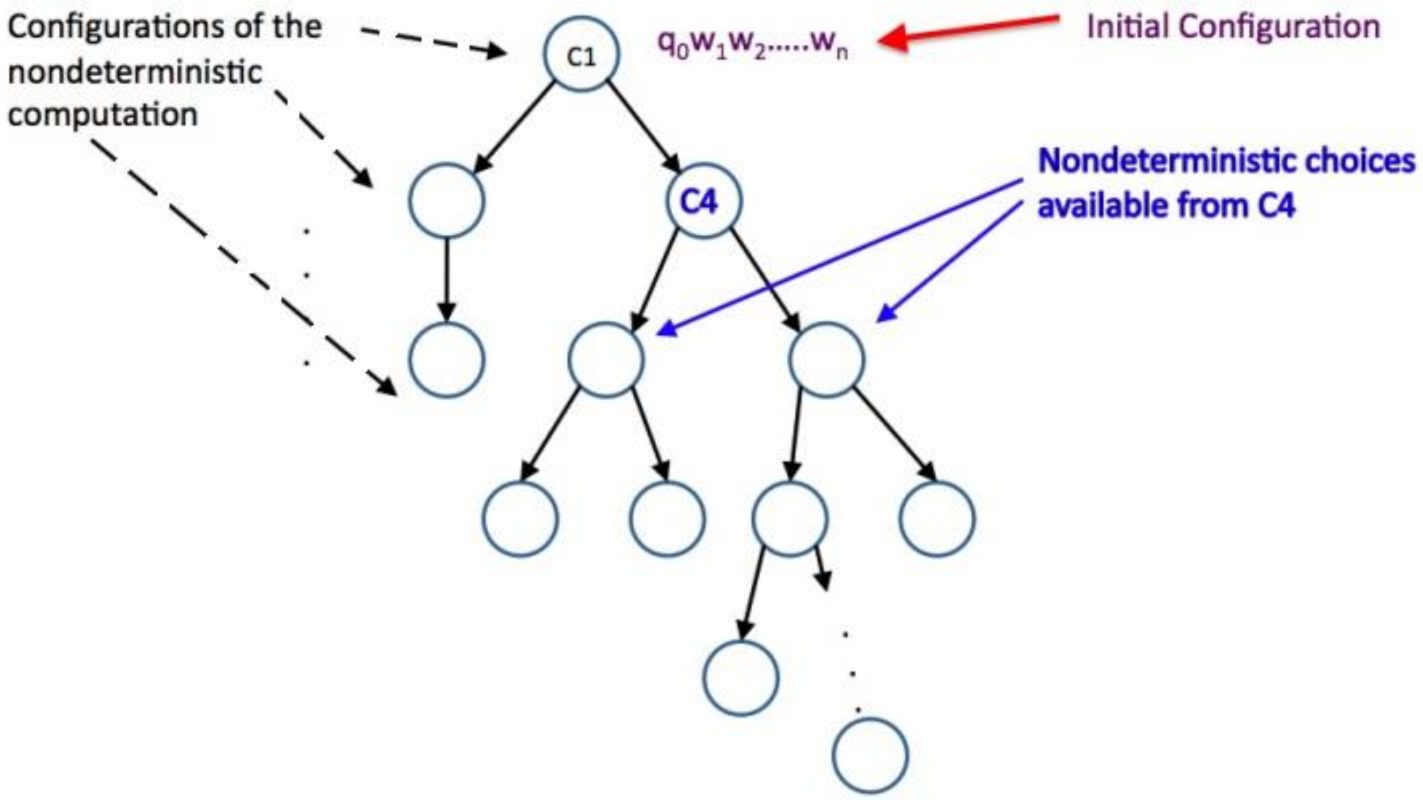
$$\delta(q, X) = \{(q_1, Y_1, D_1),\ (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

- Here, Y_i is a tape symbol, and D_i is one from $\{L, R\}$, the direction of movement of the head.

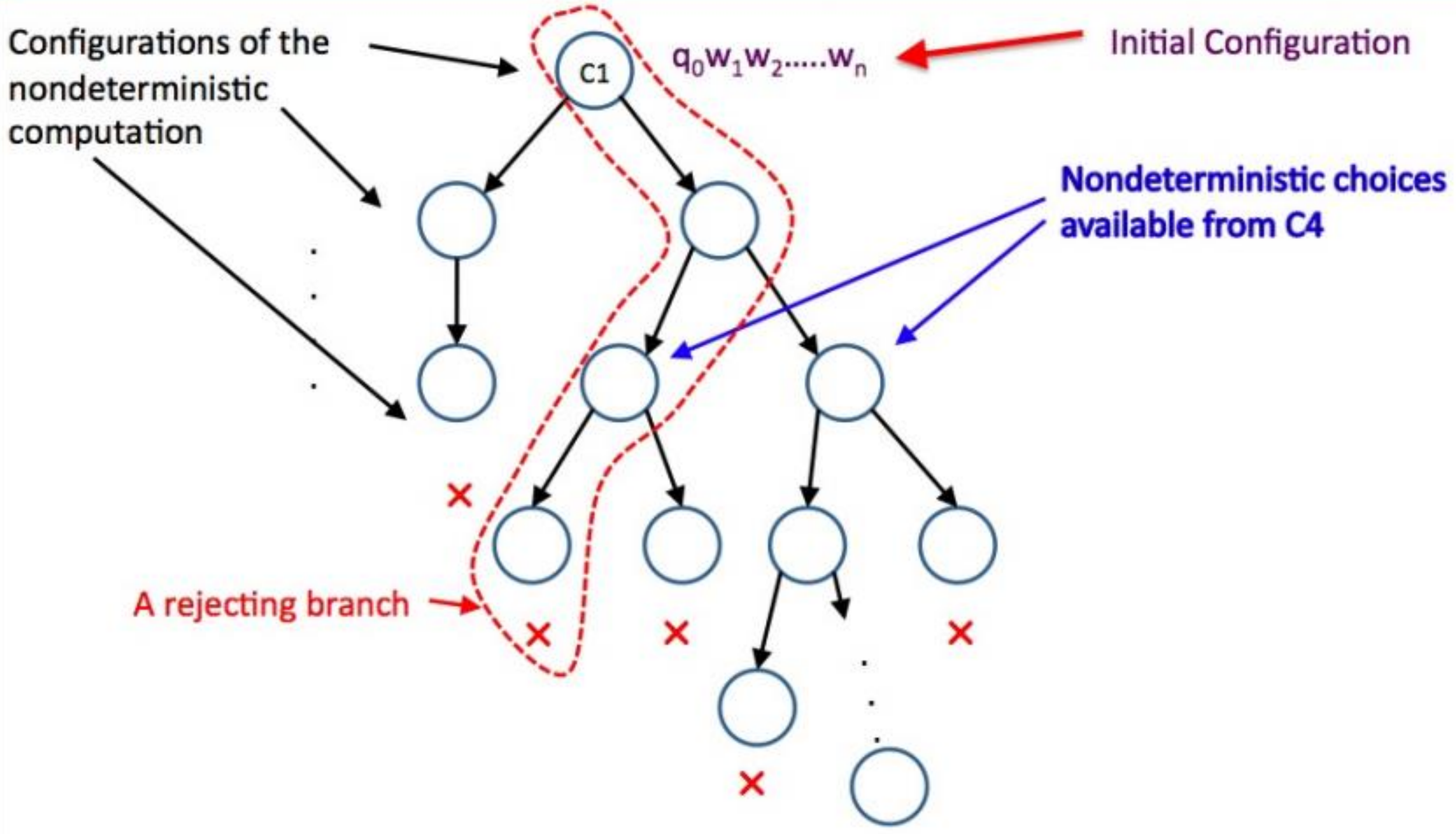
The transition function for a nondeterministic Turing machine has the form

$$\delta \colon Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$

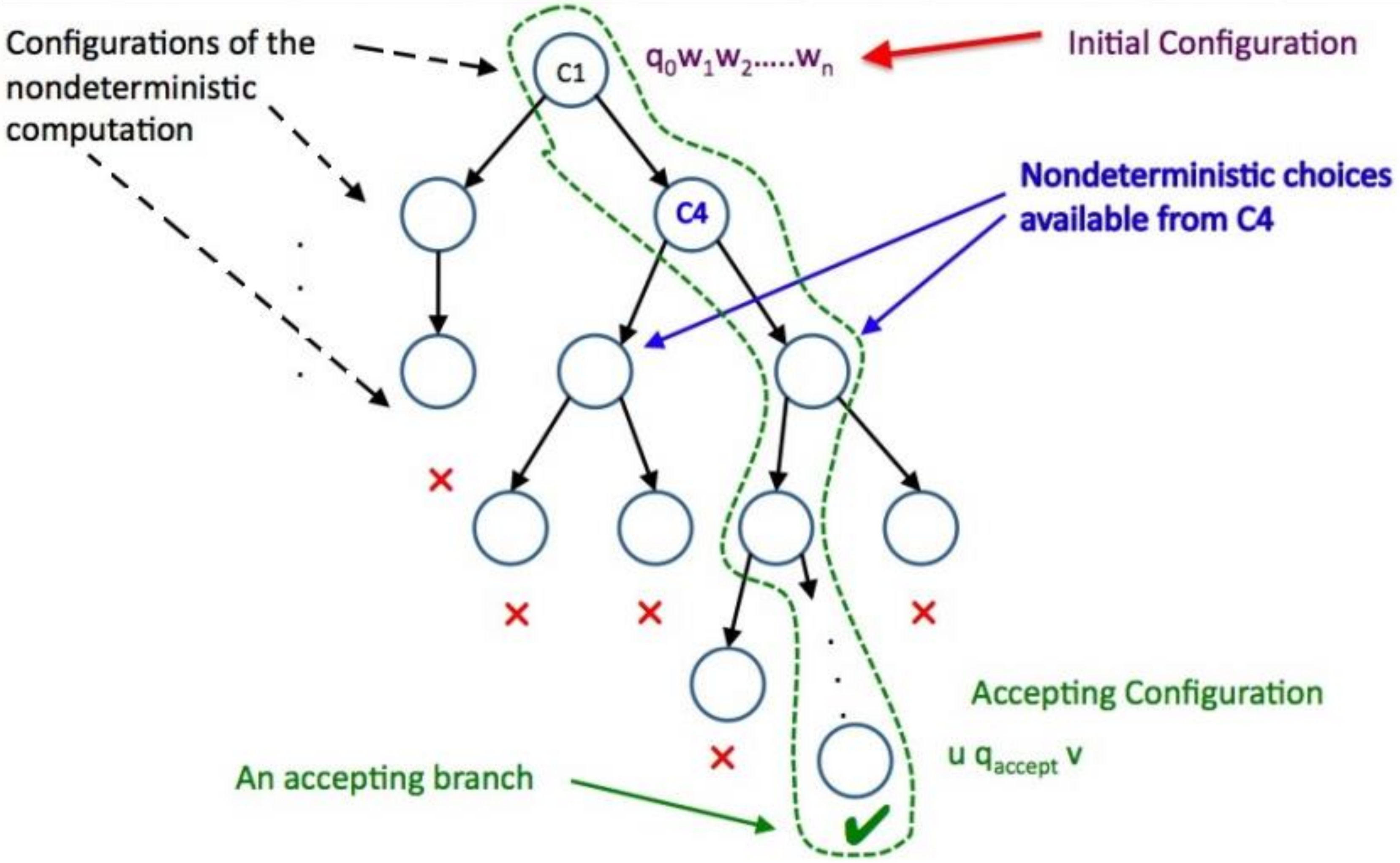
NONDETERMINISTIC COMPUTATION


Configurations of the nondeterministic computation
C1
$q_0w_1w_2.....w_n$
Initial Configuration
C4
Nondeterministic choices available from C4

Nondeterministic Computation

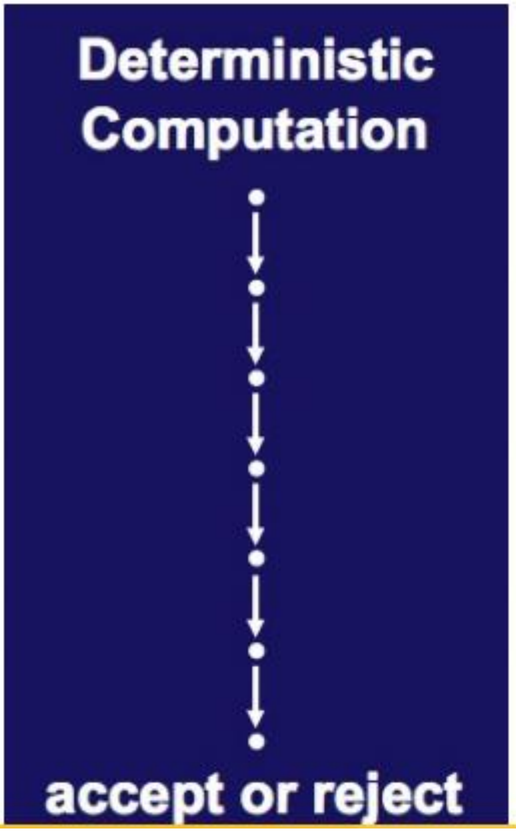

Configurations of the nondeterministic computation
C1
$q_0w_1w_2.....w_n$
Initial Configuration
Nondeterministic choices available from C4
A rejecting branch
×

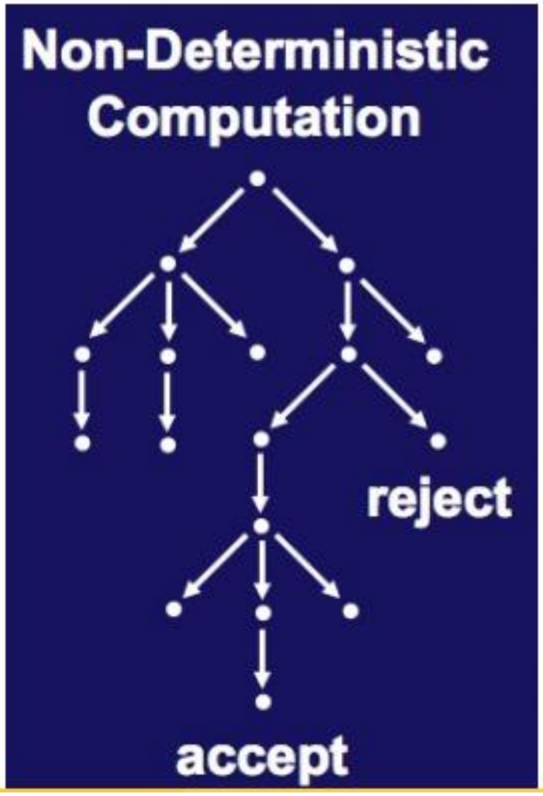
NONDETERMINISTIC COMPUTATION



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.

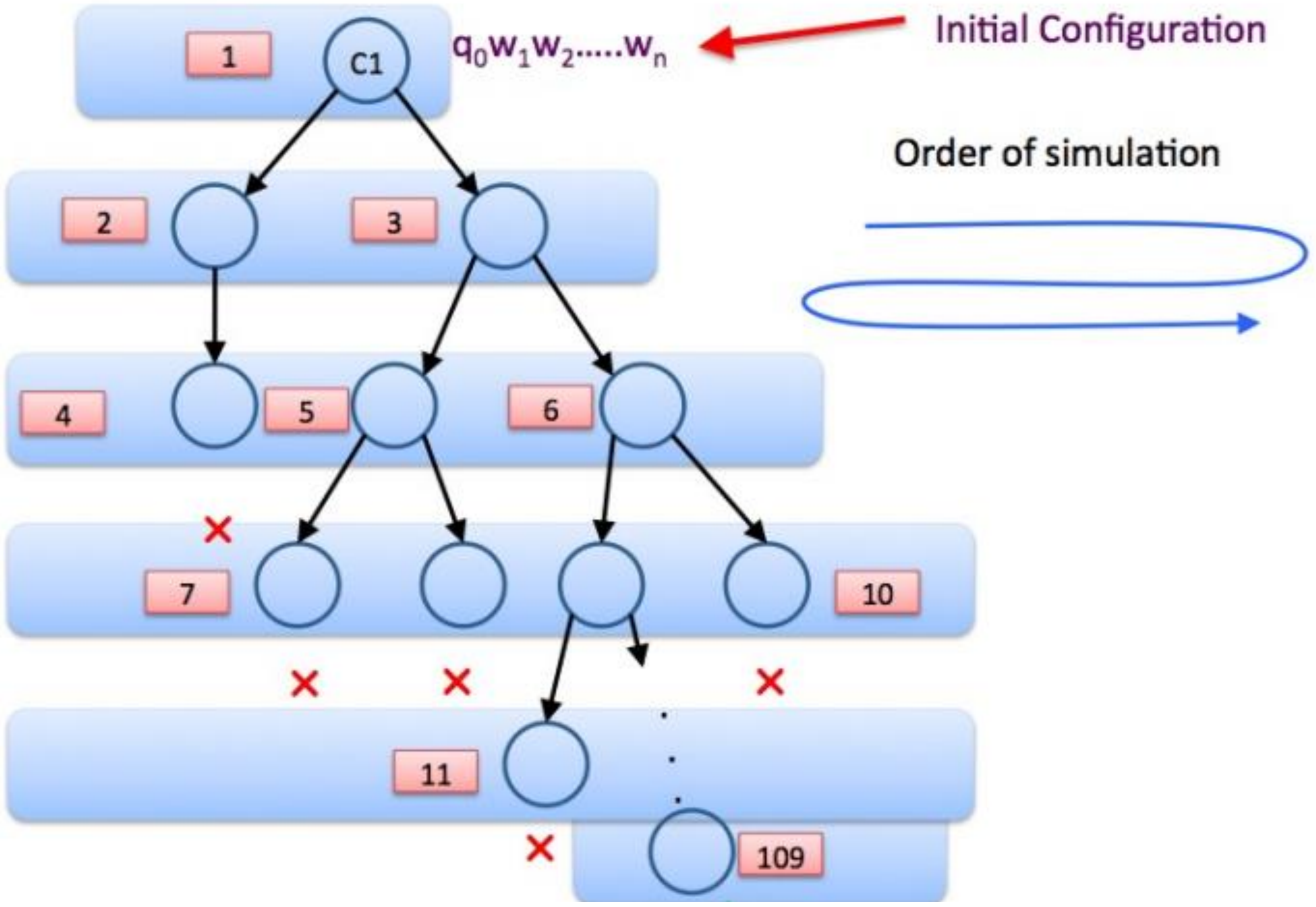
Proof Idea

- 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.

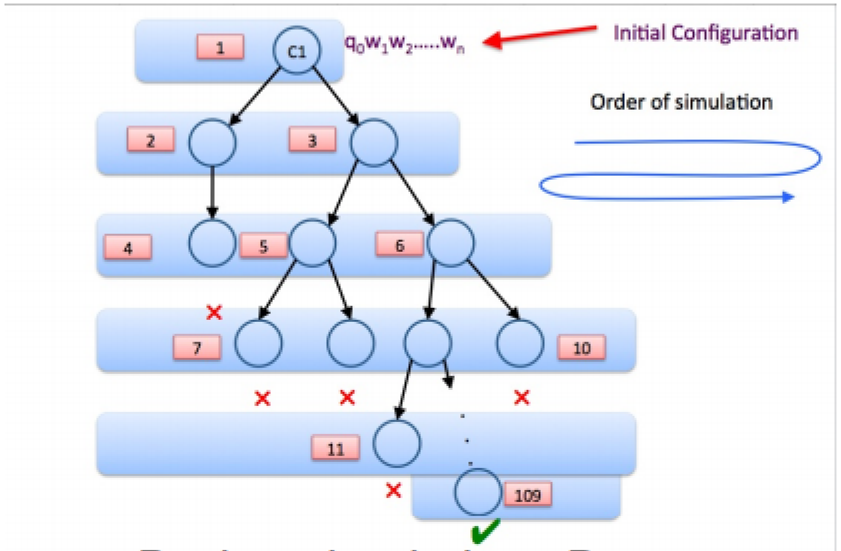
NONDETERMINISTIC TURING MACHINES

- 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.

Simulating Nondeterministic Computation



Simulating Nondeterministic Computation

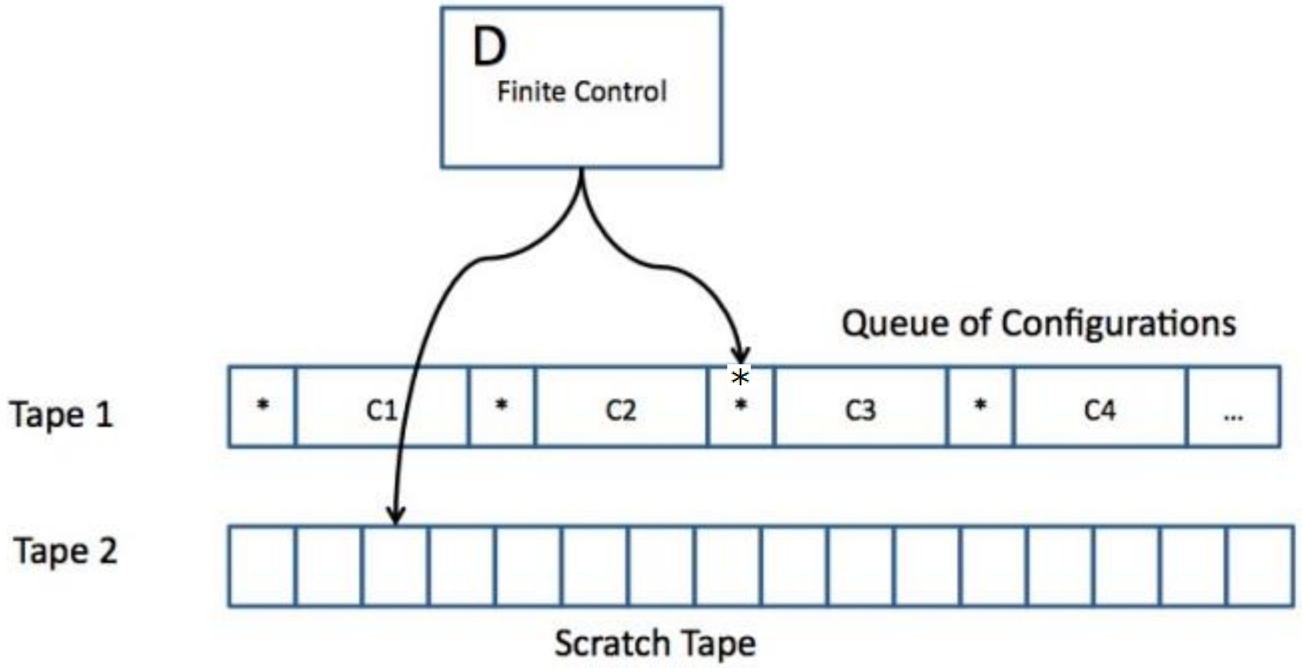


- During simulation, *D* processes the configurations of *N* in a breadth-first fashion.
- Thus *D* needs to maintain a queue of *N*'s configurations (Remember queues?)

- *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.

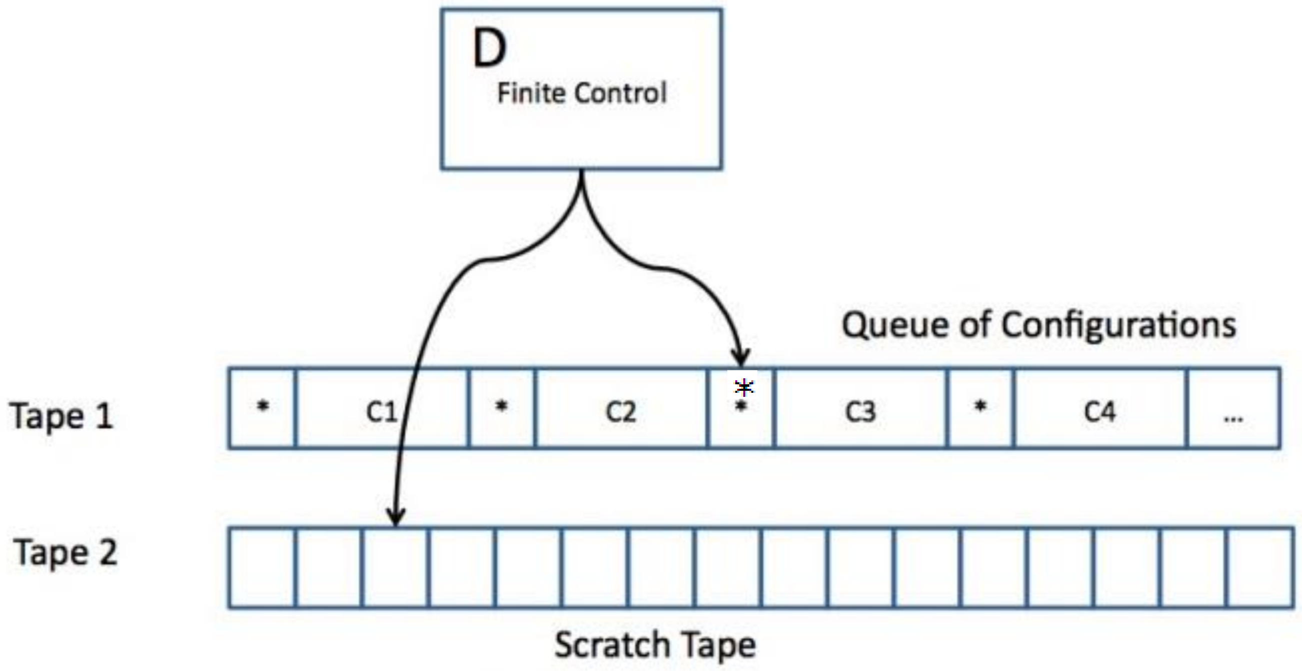
STRUCTURE OF THE SIMULATING DTM

- *N* is simulated with 2-tape DTM, *D*



Structure of the Simulating DTM

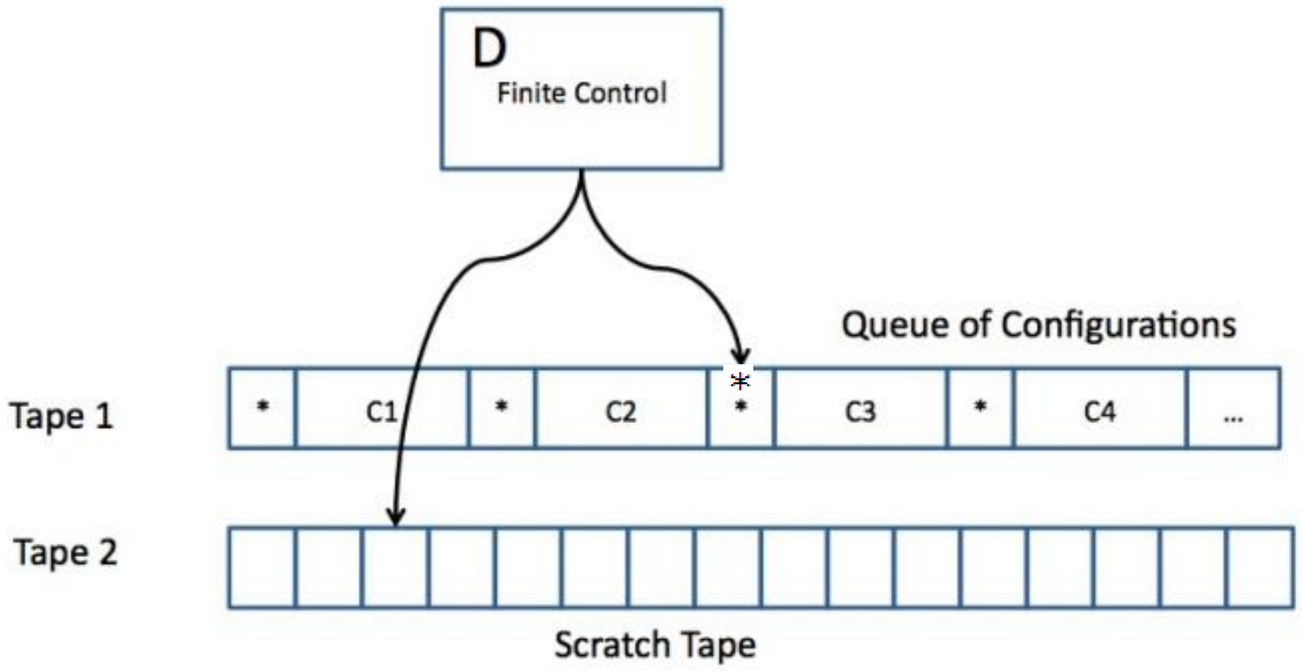
- *N* is simulated with 2-tape DTM, *D*



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

Structure of the Simulating DTM

- *N* is simulated with 2-tape DTM, *D*



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

- 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

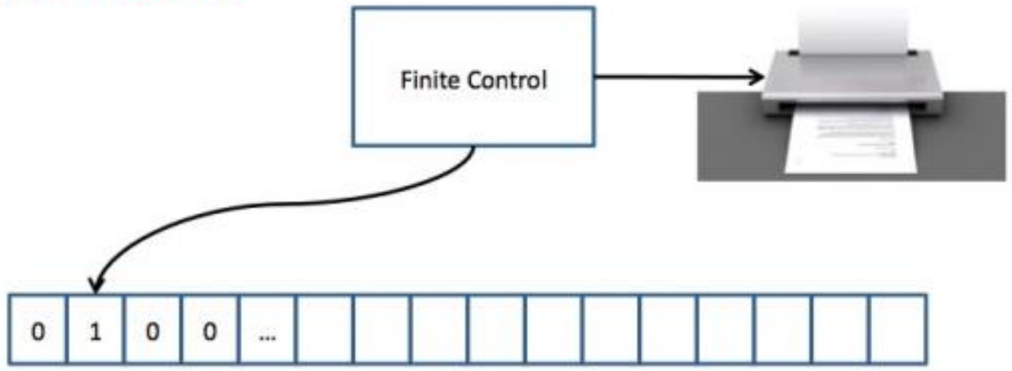
A language is Turing-recognizable if and only if some nondeterministic TM recognizes it.

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 20th 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 10th 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.