CS372
FORMAL LANGUAGES & THE THEORY OF COMPUTATION

Nguyen Thi Thu Huong,PhD.
Phone: +84 24 38696121, Mobi: +84 903253796
Email: huongnt@soict.hust.edu.vn,
huong.nguyenthithu@hust.edu.vn

Unit 7

TM Variants, Church-Turing Thesis

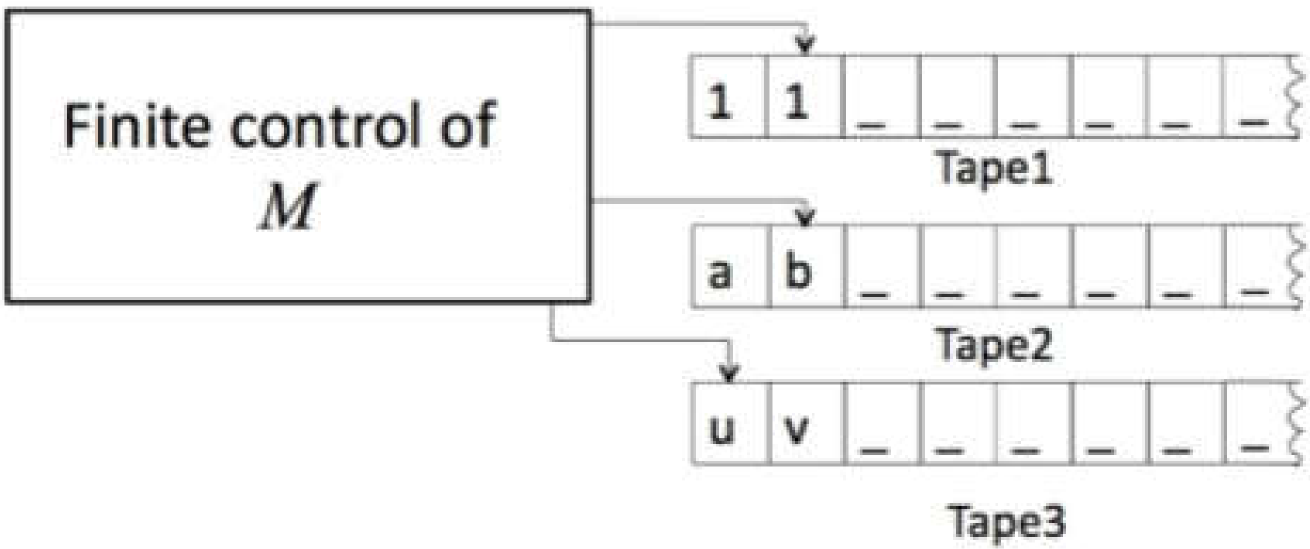
Variants of Turing Machines

- The basic Deterministic Turing Machine Single tape (infinite in one direction) was defined in previous unit
- Other variants:
  - Ordinary TMs which need not move after every action.
  - Multiple tapes – each with its own independent head
  - Nondeterministic Turing machine
  - Single tape infinite in both directions
  - Multiple tapes but with a single head
  - Multidimensional tape (move up/down/left/right)

Turing Machine with the stay option

- Add one more option of moving: Left, Right, Stay (S)….With option S, the TM to stay on the current cell, that is: $\delta : Q \times \Gamma = Q \times \Gamma \times \{L,R,S\}$
- Such a TM can easily simulate an ordinary TM: just do not use the S option in any move.
- An ordinary TM can easily simulate a TM with the stay option. For each transition with the S option, introduce a new state, and two transitions
  - One transition moves the head right, and transits to the new state.
  - The next transition moves the head back to left, and transits to the previous state.

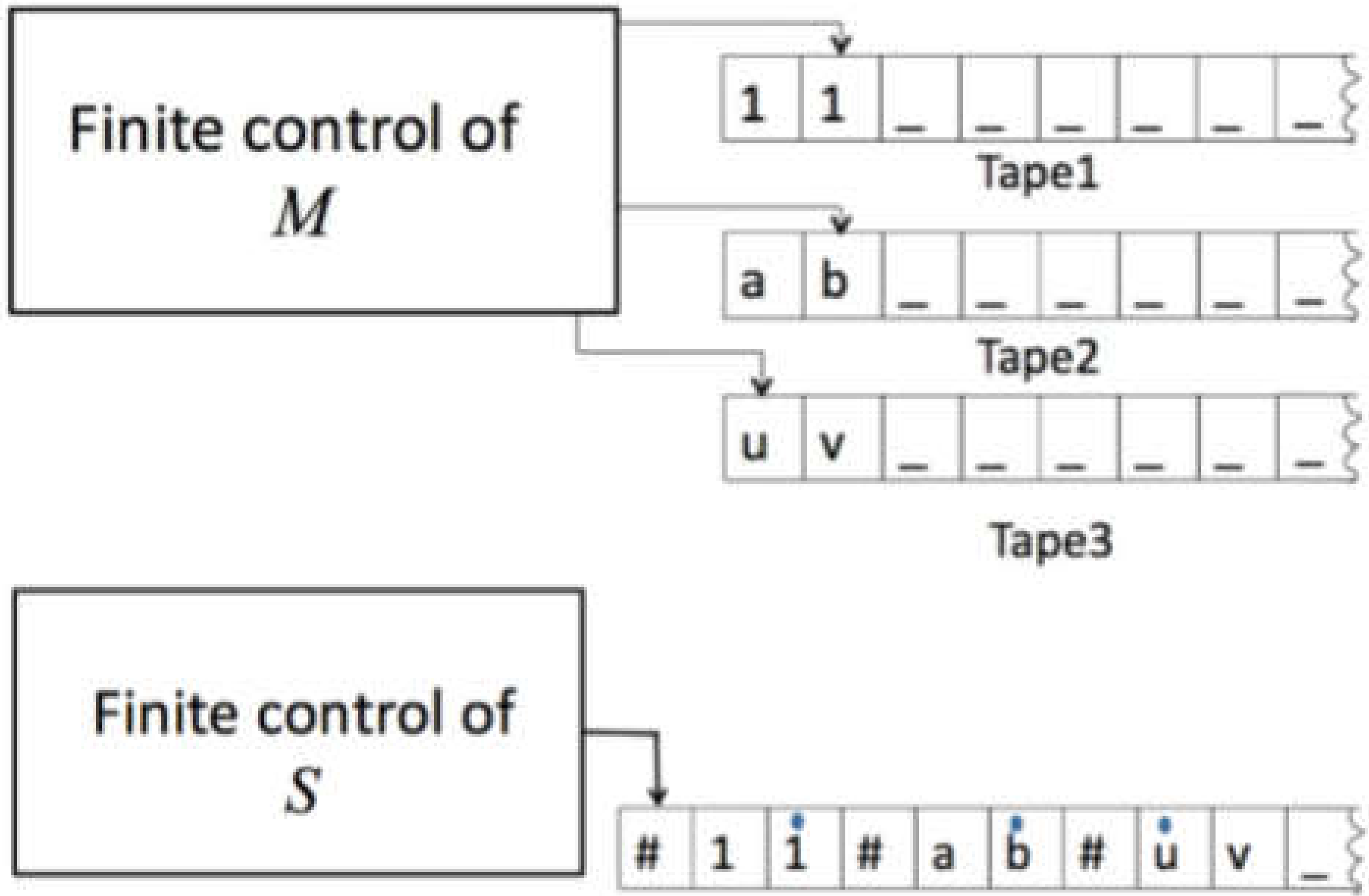
Multitape Turing Machine



Multitape Turing Machine

- A multitape Turing Machine has k tapes Each tape has its own independent read/write head.
- The state transition function.
- $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$ The δ entry $\delta(q_i,a_1,\ldots,a_k) = (q_j,b_1,\ldots,b_k, L, R, L,\ldots L)$

Simulating a Multitape TM with an ordinary TM



Simulating a Multitape TM with an ordinary TM

- Use # as a delimiter to separate out the different tape contents.
- To keep track of the location of heads, we use additional symbols Each symbol in Γ has a “dotted” version. A dotted symbol indicates that the head is on that symbol.
- Between any two #’s there is only one symbol that is dotted. Thus we have 1 real tape with k “virtual’ tapes, and 1 real read/write head with k “virtual” heads.

Simulating a Multitape TM with an ordinary TM

- Given input w = $w_1 \cdots w_n$, S puts its tape into the format that represents all k tapes of M

$$\# \dot{w}_1 \; w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#$$

- S starts at the leftmost # and scans the tape to the rightmost #.
- It determines the symbols under the “virtual” heads. This is remembered in the finite state control of S.
- S makes a second pass to update the tapes according to M.
- If one of the virtual heads, moves right to a #, the rest of tape to the right is shifted to “open up” space for that “virtual tape”. If it moves left to a #, it just moves right again.

Simulating a Multitape TM with an ordinary TM

- Thus from now on, whenever needed or convenient we will use multiple tapes in our constructions.
- It is possible to assume that these can always be converted to a single tape standard TM.

Nondeterministic Turing Machine

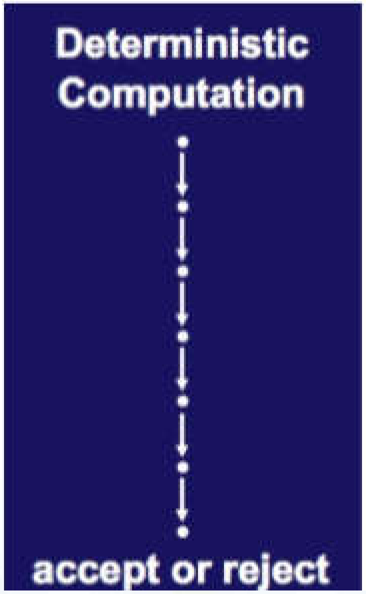
- A nondeterministic TM would proceed computation with multiple next configurations.
- δ for a nondeterministic TM would be

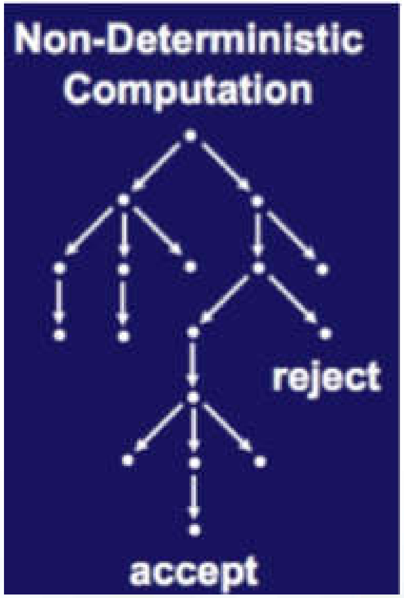
  $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$

  (P(S) is the power set of S.)

Nondeterministic Turing Machine

- 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 Machine

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

Nondeterministic Turing Machine

- Theorem

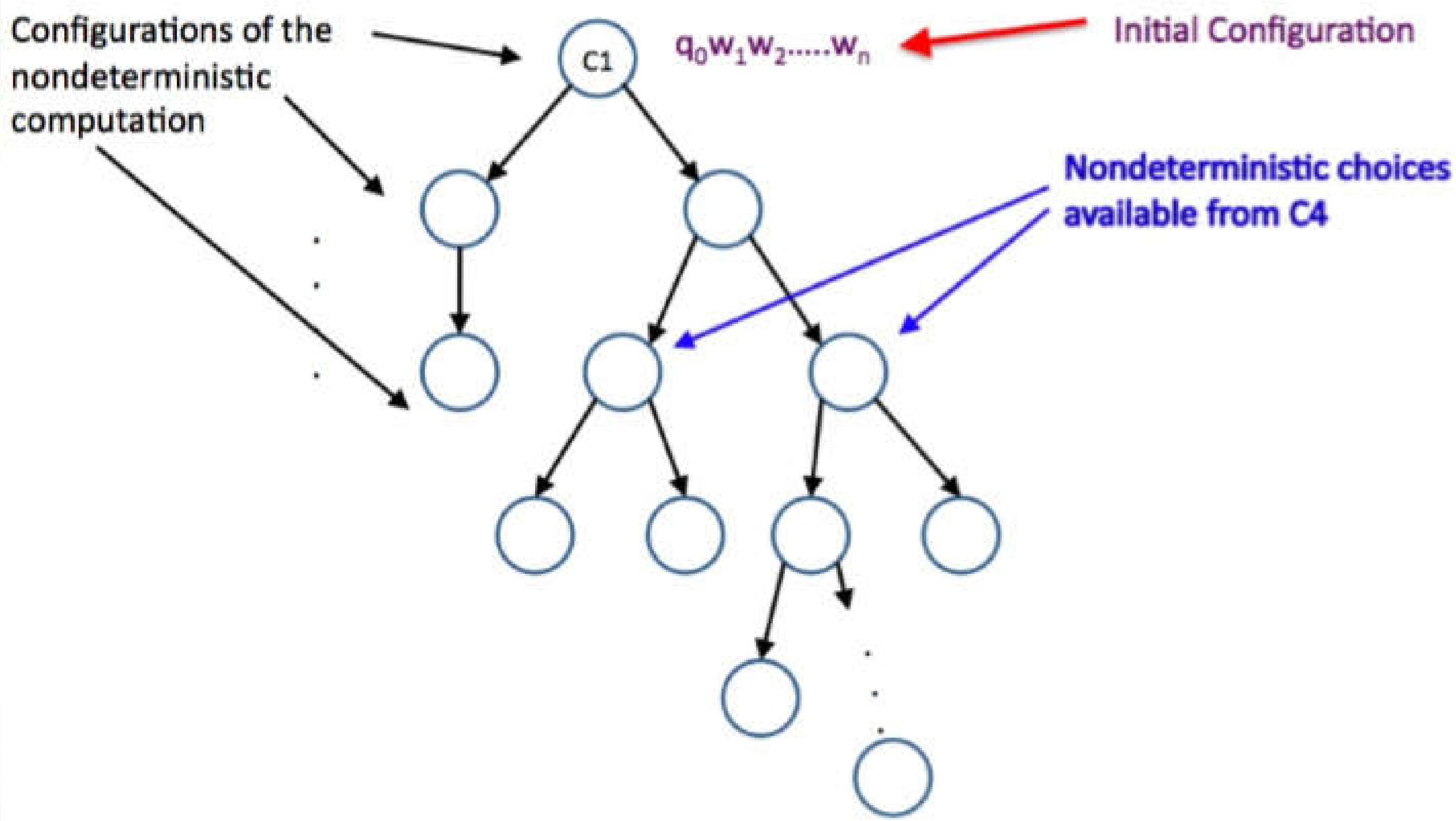
Every nondeterministic Turing machine has an equivalent deterministic Turing Machine.

- 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.

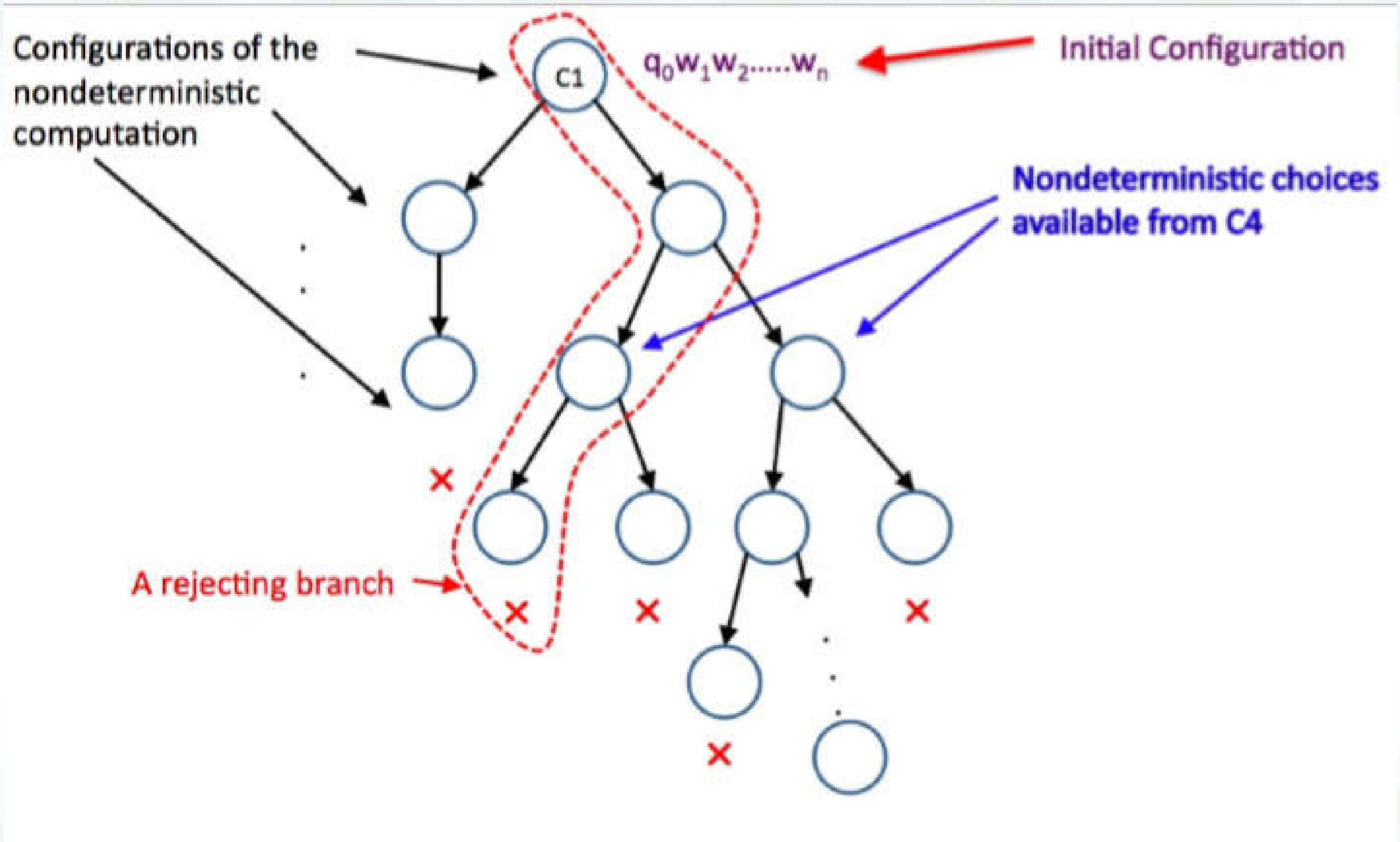
Nondeterministic Turing Machine

- Deterministic TM D simulates the Nondeterministic TM N.
- Some of branches of the N’s computations may be infinite. If D starts its simulation by following an infinite branch, D may loop forever.
- In order to avoid this unwanted situation, we want D to execute all of N’s computations concurrently.

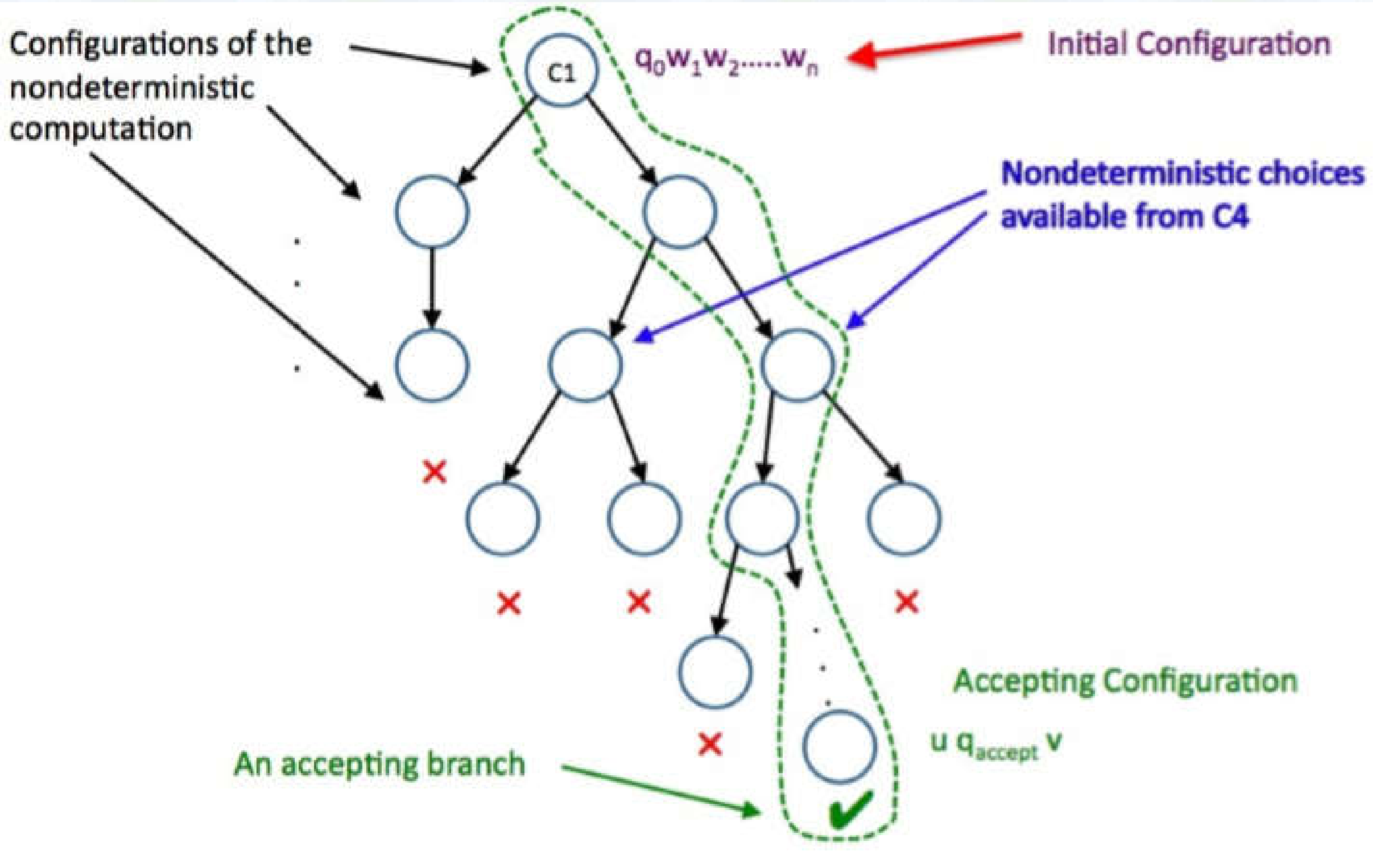
Nondeterministic Computation



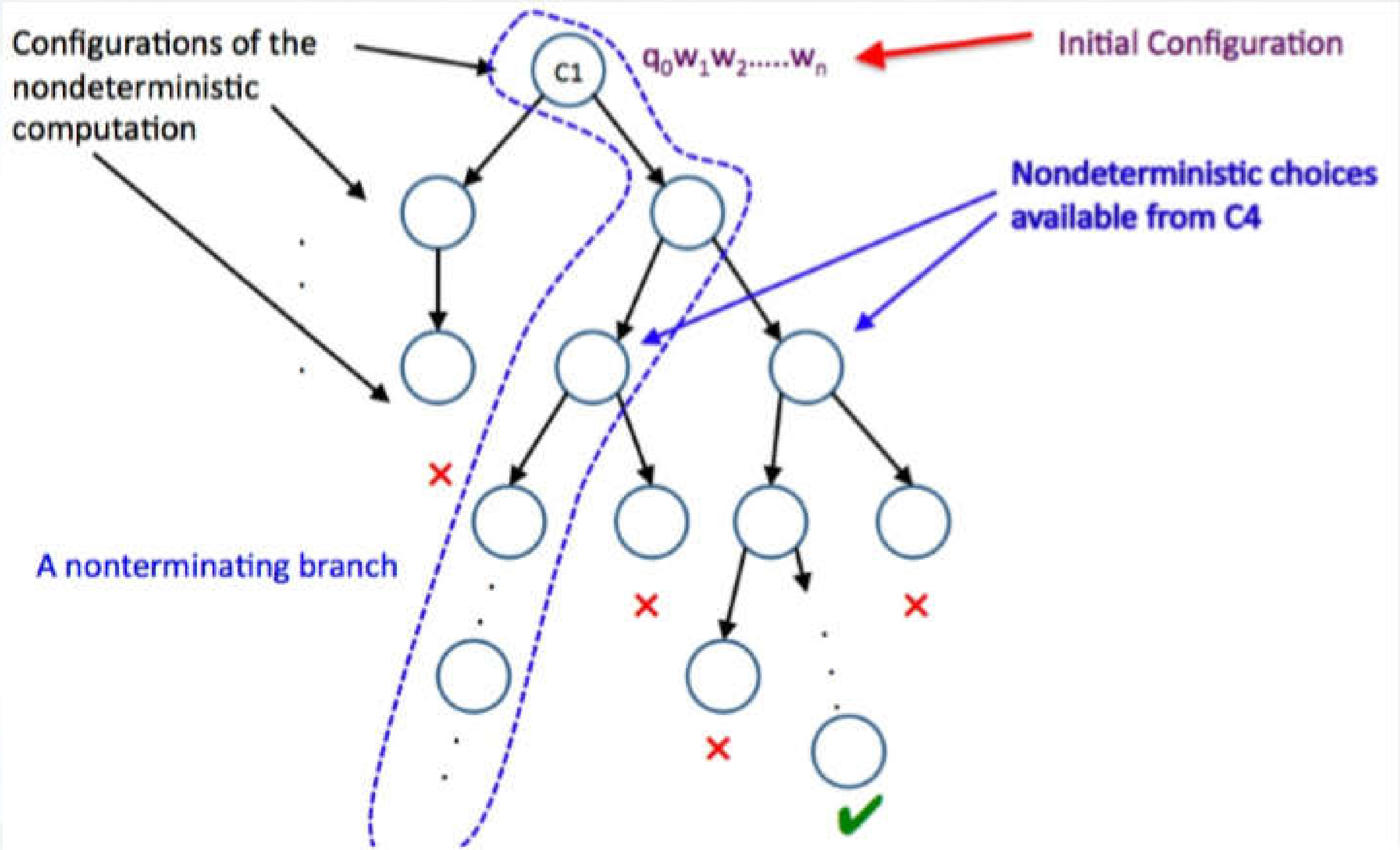
Nondeterministic Computation



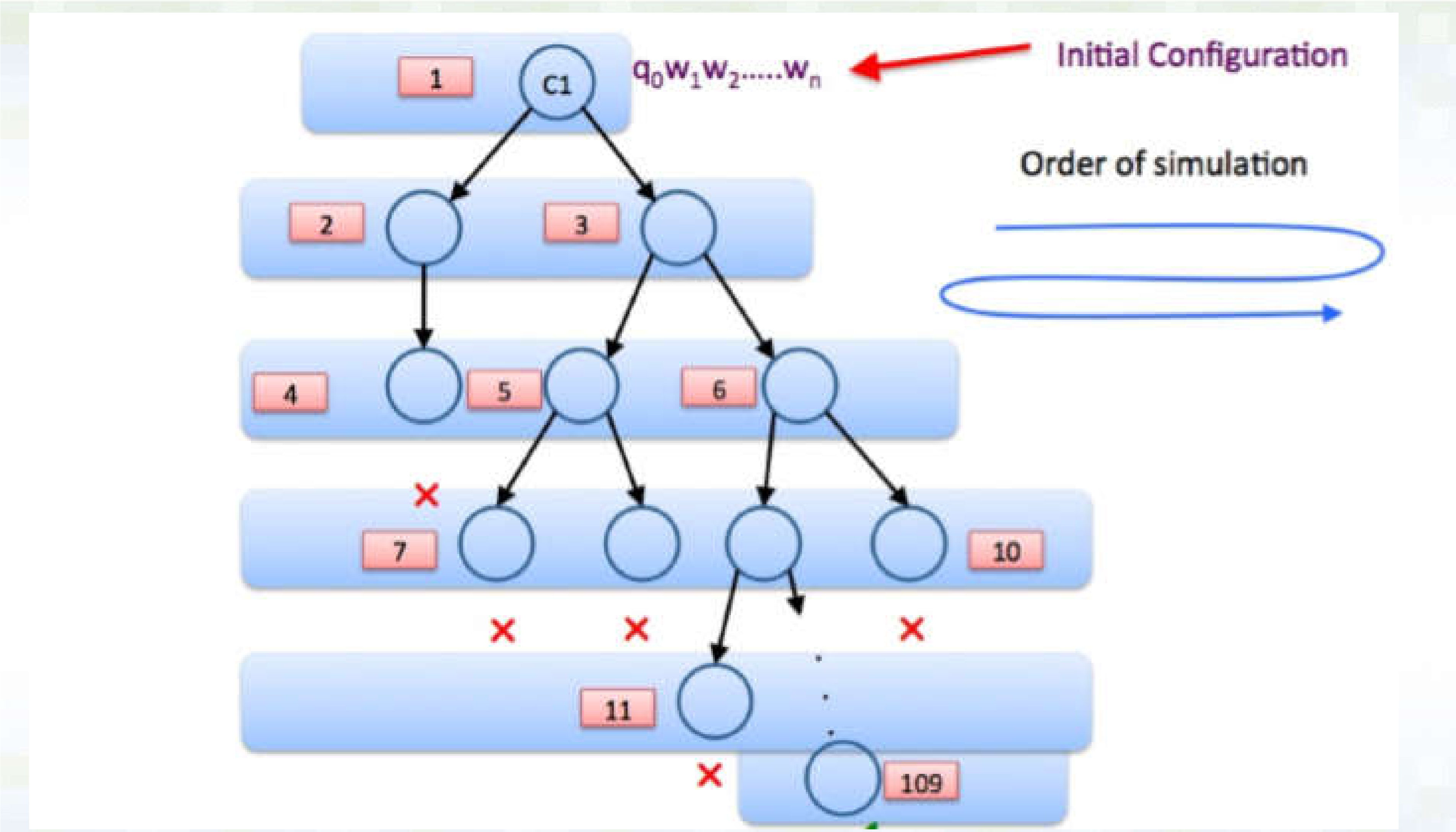
Nondeterministic Computation



Nondeterministic Computation



Simulating Nondeterministic Computation

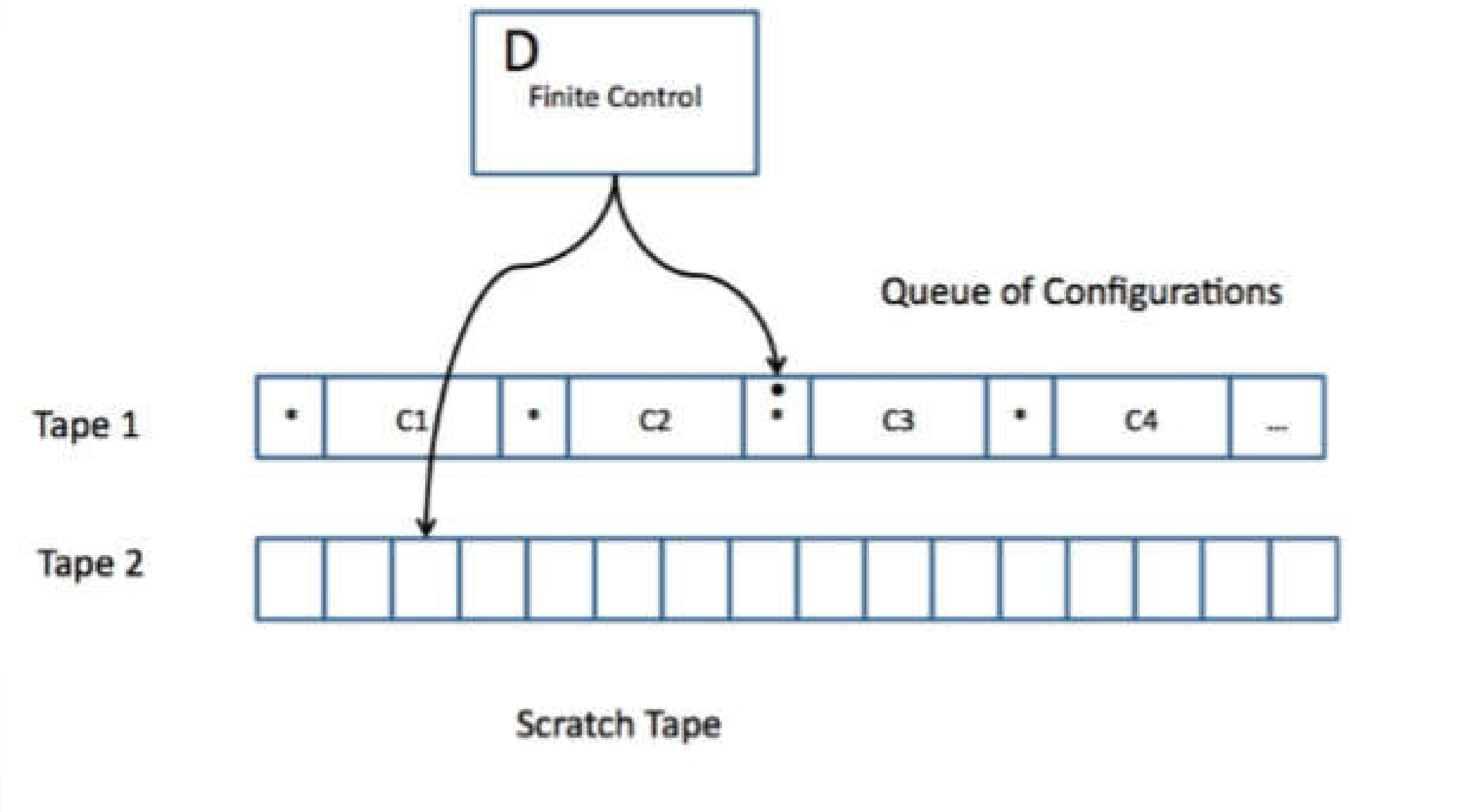


Simulating Nondeterministic Computation

- During simulation, D processes the configurations of N in a breadth-first fashion. Thus D needs to maintain a queue of N’s configurations
- 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



How D simulates M

- 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 M

1) D examines the state and the input symbol of the current configuration
2) If the state of the current configuration is the accept state of N, then D accepts the input and stops simulating N.
3) D copies the new configurations from the scratch tape, back to the end of tape 1
4) D clears the scratch tape.
5) D returns to the marked current configuration, and "erases" the mark, and "marks" the next configuration.
6) D returns to step 1), if there is a next configuration. Otherwise rejects.

How D simulates M

- 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.
- 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.

Implications

- Corolarry

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

- Corolarry

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

Church Turing Thesis

- History of algorithm
- Church - Turing thesis
- Solvability of a problem
- Encoding problems

History of algorithms

- In 1900, Hilbert raised the first concept of algorithm
- In early 20th century, there was no formal definition of an algorithm.
- In 1936, Alonzo Church (Lambda calculus) and Alan Turing (Turing machine) came up with formalisms to define algorithms.
- These were shown to be equivalent, leading to the Church Turing thesis

Church - Turing thesis

Intutitive notion of algorithms ≡

Turing Machine Algorithms

Hilbert's 10^{th} problem

- Let D = {p | p is a polynomial with integral roots}
- D is recognizable: just try systematically all integer combinations for all variables.
- Consider a simpler version

  D_1 = {p |p is a polynomial over x with integral roots}
- M_1 = The input is polynomial p over x.
  - 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.
- In TM terminology is "Is D decidable?" (No!)

Encoding problems for TM

- The input to TMs have to be strings. Every object O that enters a computation will be represented with an string <O>, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges

  <O> = (1,2,3,4)((1,2),(2,3),(3,1),(1,4))
- Then we can define problems over graphs, e.g., as:

  A = {<G> | G is a connected undirected graph}

Encoding problems for TM

- A TM for this problem can be given as:
- M =

Input <G>, 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, then accept, else reject