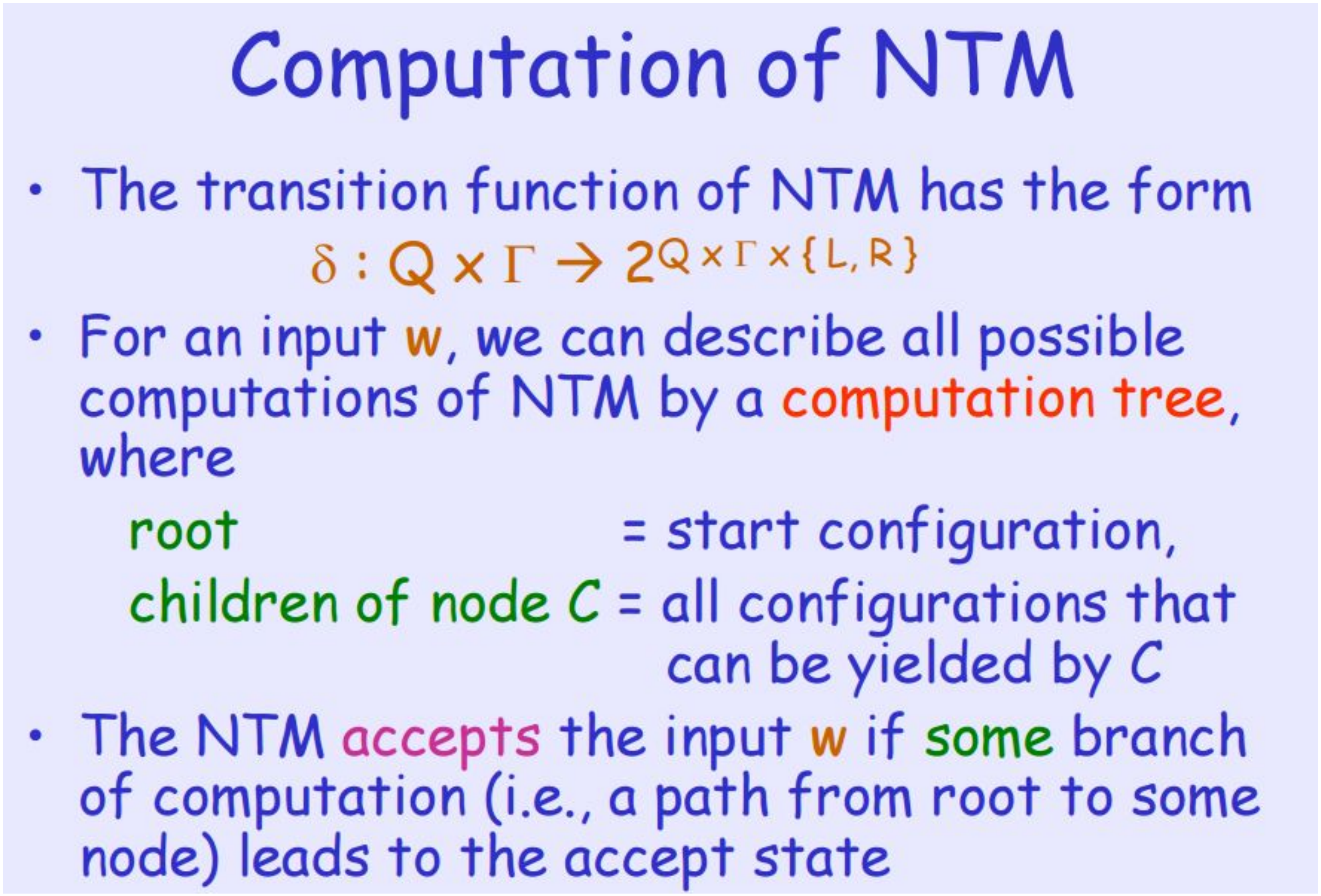
Time Complexity of NTM

Relationship with DTM

Recap of Converting a NTM to DTM



Computation of NTM

- The transition function of NTM has the form

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

- For an input w, we can describe all possible computations of NTM by a computation tree, where

  root = start configuration,

  children of node C = all configurations that can be yielded by C

- The NTM accepts the input w if some branch of computation (i.e., a path from root to some node) leads to the accept state

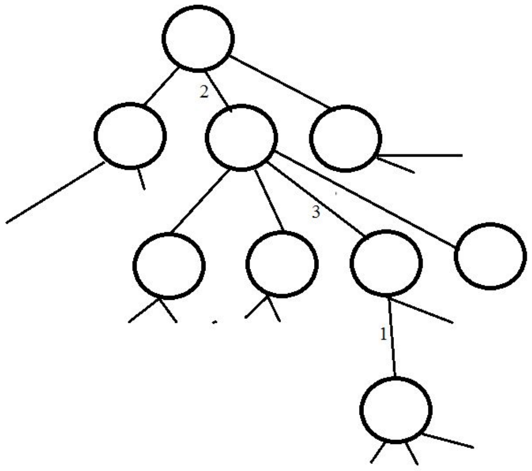
NTM = TM

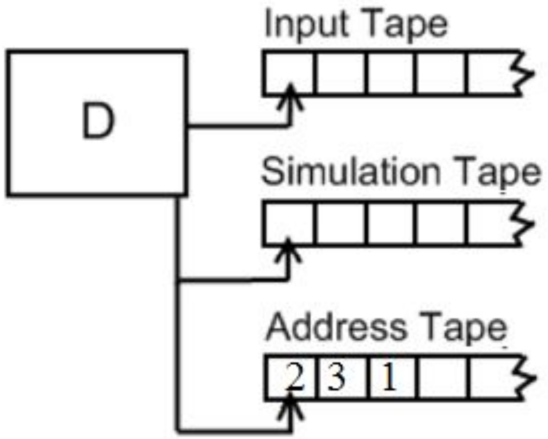
Theorem: Given an NTM that recognizes a language L, we can find a TM that recognizes the same language L.

Proof: Let N be the NTM. We show how to convert N into some TM D. The idea is to simulate N by trying all possible branches of N's computation. If one branch leads to an accept state, D accepts. Otherwise, D's simulation will not terminate.

NTM = TM (Proof)

- To simulate the search, we use a 3-tape TM for D
  - first tape stores the input string
  - second tape is a working memory, and
  - third tape "encodes" which branch to search
- What is the meaning of "encode"?





NTM = TM (Proof)

- Let $b = |Q \times \Gamma \times \{ L, R \}|$, which is the maximum number of children of a node in N's computation tree.
- We encode a branch in the tree by a string over the alphabet {1,2,…,b}.
  - E.g., 231 represents the branch:

    root r → r's 2nd child c → c's 3rd child d → d's 1st child

NTM = TM (Proof)

On input string w,

Step 1. D stores w in Tape 1 and □ in Tape 3

Step 2. Repeat

2a. Copy Tape 1 to Tape 2

2b. Simulate N using Tape 2, with the branch of computation specified in Tape 3.

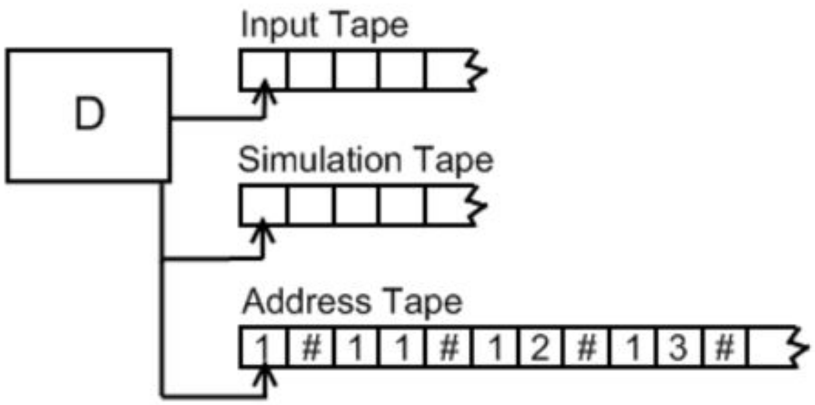
Precisely, in each step, D checks the next symbol in Tape 3 to decide which choice to make. (Special case …)

NTM = TM (Proof)

2b [Special Case].

1. If this branch of N enters accept state, accepts w
2. If no more chars in Tape 3, or a choice is invalid, or if this branch of N enters reject state, D aborts this branch

2c. Copy Tape 1 to Tape 2, and update Tape 3 to store the next branch (in Breadth-First Search order)



NTM = TM (Proof)

- In the simulation, D will first examine the branch ε (i.e., root only), then the branch 1 (i.e., root and 1st child only), then the branch 2, and then 3, 4, …, b, then the branches 11, 12, 13, …, 1b, then 21, 22, 23, …, 2b, and so on, until the examined branch of N enters an accept state (what if N enters a reject state?)
- If N does not accept w, the simulation of D will run forever
- Note that we cannot use DFS (depth-first search) instead of BFS (why?)

TIME COMPLEXITY RELATION BETWEEN DTM & NTM

NTM decider

An NTM is a decider if all its computation branches halt on all inputs.

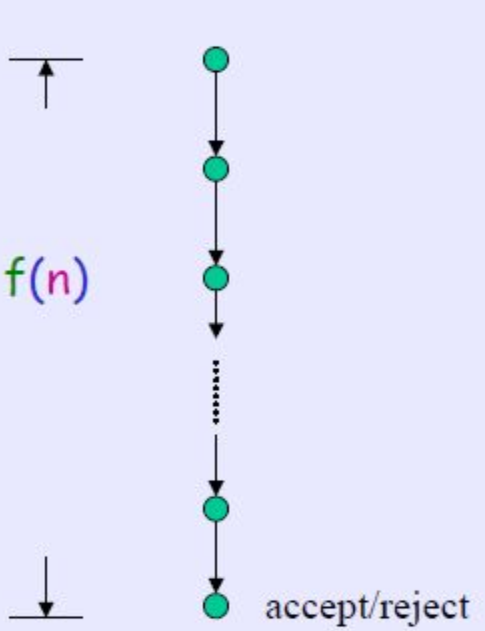
Definition: Let M be an NTM decider. The running time of M is the function $f: N \rightarrow N$, where $f(n)$ is the maximum number of steps that M uses on any branch of its computation on any input of length n

DTM versus NTM decider

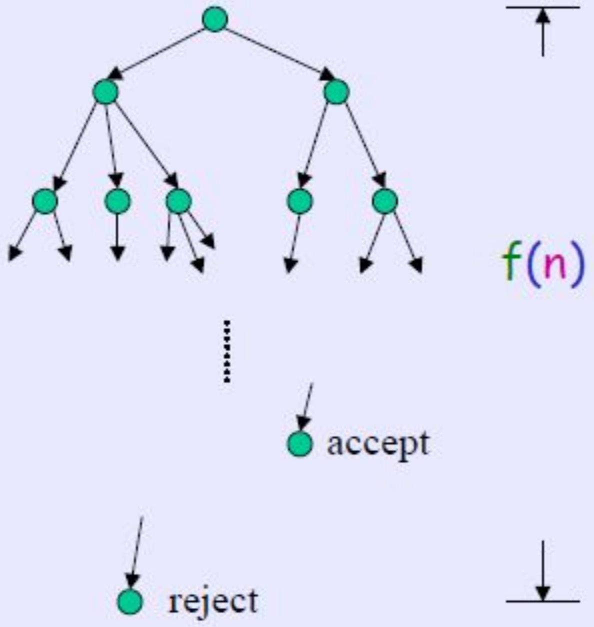
Theorem: Let $t(n)$ be a function, $t(n) \geq n$.
Then every $t(n)$-time single-tape NTM decider has an equivalent $2^{O(t(n))}$-time single-tape DTM

Proof: Let M be a NTM that runs in $t(n)$ time. We construct a DTM D that simulates M by searching M's computation tree. We now analyze D's simulation.

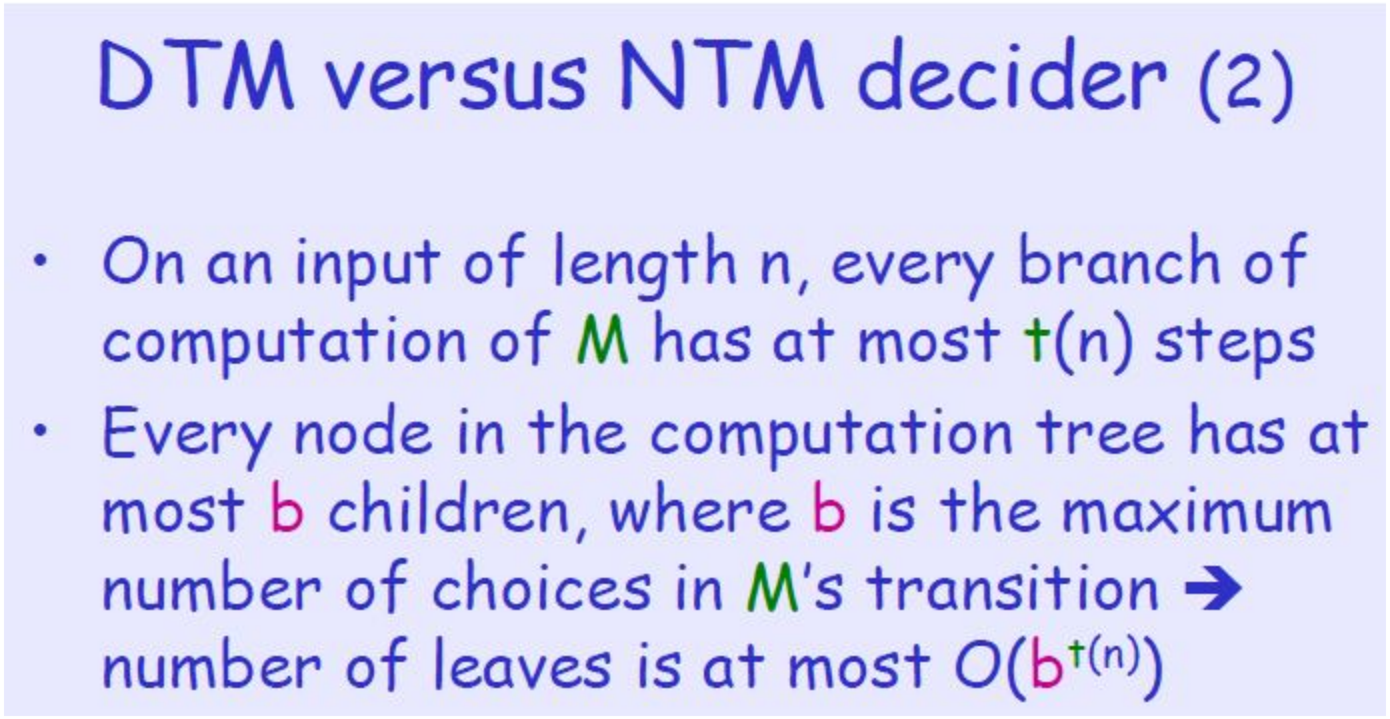
Comparison of Running Times



Deterministic time



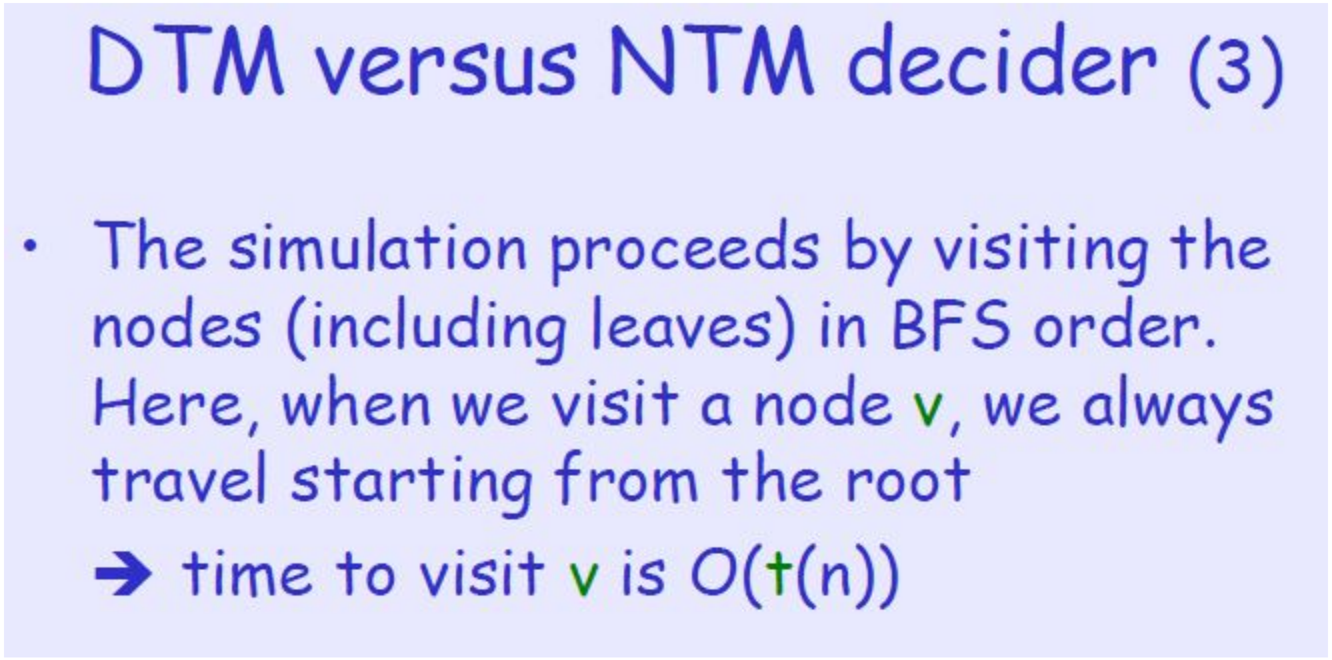
Non-deterministic time



DTM versus NTM decider (2)

- On an input of length n, every branch of computation of M has at most t(n) steps
- Every node in the computation tree has at most b children, where b is the maximum number of choices in M's transition ➔ number of leaves is at most $O(b^{t(n)})$

- Total number of nodes in the tree $\leq$ 2 times number of leaves
- Hence total number of nodes = $O\left(b^{t(n)}\right)$



DTM versus NTM decider (3)

- The simulation proceeds by visiting the nodes (including leaves) in BFS order. Here, when we visit a node v, we always travel starting from the root

  ➔ time to visit v is O(t(n))

- Therefore, running time $= O\left(t(n)\ b^{t(n)}\right) = 2^{O\left(t(n)\right)}$.
- This is a 3 tape DTM, to simulate this over a single tape DTM, we need to square the upperbound, so
- Running time over single-tape DTM

$$= \left(2^{O\left(t(n)\right)}\right)^2 = 2^{O\left(2t(n)\right)}$$
$$= 2^{O\left(t(n)\right)}.$$

Time Complexity Class

Definition: Let $t: N \to R^+$ be a function. We define the time complexity class, TIME(t(n)), to be the collection of all languages that are decidable by an O(t(n))-time Turing machine

the language $A = \{0^k1^k \mid k \geq 0\}$ is in TIME(n^2)

The Class P

Definition: **P** is the class of languages that are decidable in polynomial time on a single-tape DTM. In other words,

$$\bigcup_{k=1} \text{TIME}(n^k)$$

- **P** is invariant for all computation models that are polynomially equivalent to the single-tape DTM, and
- **P** roughly corresponds to the class of problems that are realistically solvable

Further points to notice

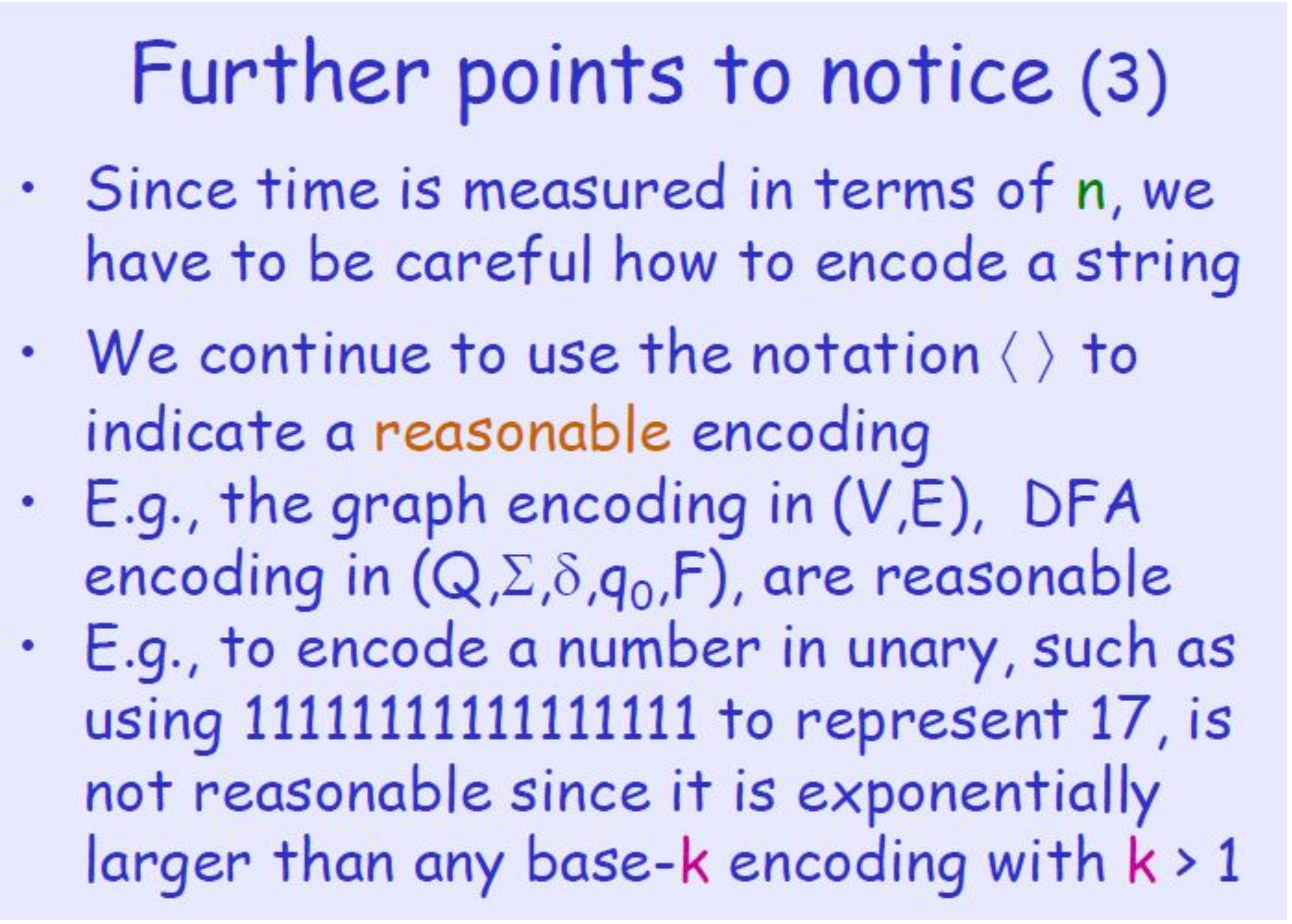
- When we describe an algorithm, we usually describe it with stages, just like a step in the TM, except that each stage may actually consist of many TM steps
- Such a description allows an easier (and clearer) way to analyze the running time of the algorithm

Further points to notice (2)

- So, when we analyze an algorithm to show that it runs in poly-time, we usually do:
  1. Give a polynomial upper bound on the number of stages that the algorithm uses when its input is of length n
  2. Ensure that each stage can be implemented in polynomial time on a reasonable deterministic model
- When the two tasks are done, we can say the algorithm runs in poly-time (why??)

Further points to notice (3)

- Since time is measured in terms of n, we have to be careful how to encode a string
- We continue to use the notation $\langle\ \rangle$ to indicate a reasonable encoding
- E.g., the graph encoding in (V,E), DFA encoding in (Q,Σ,δ,q_0,F), are reasonable
- E.g., to encode a number in unary, such as using 11111111111111111 to represent 17, is not reasonable since it is exponentially larger than any base-k encoding with $k > 1$



Further points to notice (3)

- Since time is measured in terms of n, we have to be careful how to encode a string
- We continue to use the notation $\langle\ \rangle$ to indicate a reasonable encoding
- E.g., the graph encoding in (V,E), DFA encoding in (Q,Σ,δ,q_0,F), are reasonable
- E.g., to encode a number in unary, such as using 11111111111111111 to represent 17, is not reasonable since it is exponentially larger than any base-k encoding with $k > 1$

- **That which works in polynomial time with a binary encoded number, may take exponential time with unary encoded number.**

Examples of Languages in P

Let PATH be the language

$\{ \langle G, s, t \rangle \mid G$ is a graph with path from s to $t\}$

Theorem: PATH is in P.

How to prove??

… Find a decider for PATH that runs in polynomial time

$PATH = \{\langle G, s, t\rangle |\ G$ is a directed graph that has a directed path from s to $t\}$.

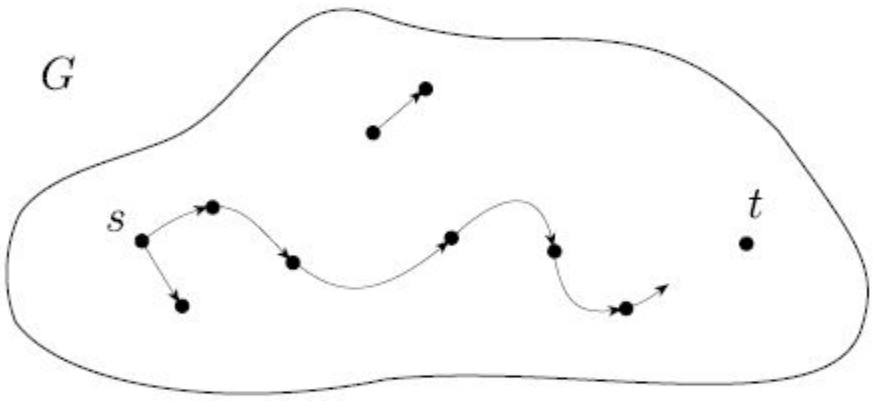


FIGURE 7.13
The *PATH* problem: Is there a path from s to t?

PATH is in P

Proof: A polynomial time decider M for PATH operates as follows:

```
M = "On input ⟨G,s,t⟩,
  1. Mark node s
  2. Repeat until no new nodes are marked
     i. Scan all edges of G to find an edge
        that has exactly one marked node.
        Mark the other node
  3. If t is marked, accept. Else, reject."
```

PATH is in P (2)

What is the running time for M?

- Let m be the number of nodes in G
- Stages 1 and 3 each involves O(1) scan of the input
- Stage 2 has at most m runs, each run checks at most all the edges of G. Thus, each run involves at most $O(m^2)$ scans of the input ➔ Stage 2 involves $O(m^3)$ scans
- Since $m = O(n)$, where n = input length, the total time is polynomial in n

Every CFL is in P

Theorem: Every CFL is in P

How to prove??

... Let's recall an old idea for deciding a particular CFL ...

Every CFL is in P (2)

Proof(?): Let C be the CFL and G be the CFG in Chomsky Normal form that generates C. Define M as follows:

```
M = “On input w = w_1 w_2 … w_n,
    1. Construct all possible derivations in G
       with 2n-1 steps
    2. If any derivation generates w, accept.
       Else, reject.”
```

Quick Quiz: Does M run in polynomial time?

Every CFL is in P (2)

Proof(?): Let C be the CFL and G be the CFG in Chomsky Normal form that generates C. Define M as follows:

```
M = "On input w = w_1 w_2 ... w_n,
    1. Construct all possible derivations in G
       with 2n-1 steps
    2. If any derivation generates w, accept.
       Else, reject."
```

Quick Quiz: Does M run in polynomial time?

- **If number of productions is p, then each step can use (in the worst case) any of these p productions. So, total number of derivations (each of size $2n-1$) is $O\left(p^{2n-1}\right)$.**
- **Obviously, this is not polynomial in n.**

The Generation Problem for CFGs

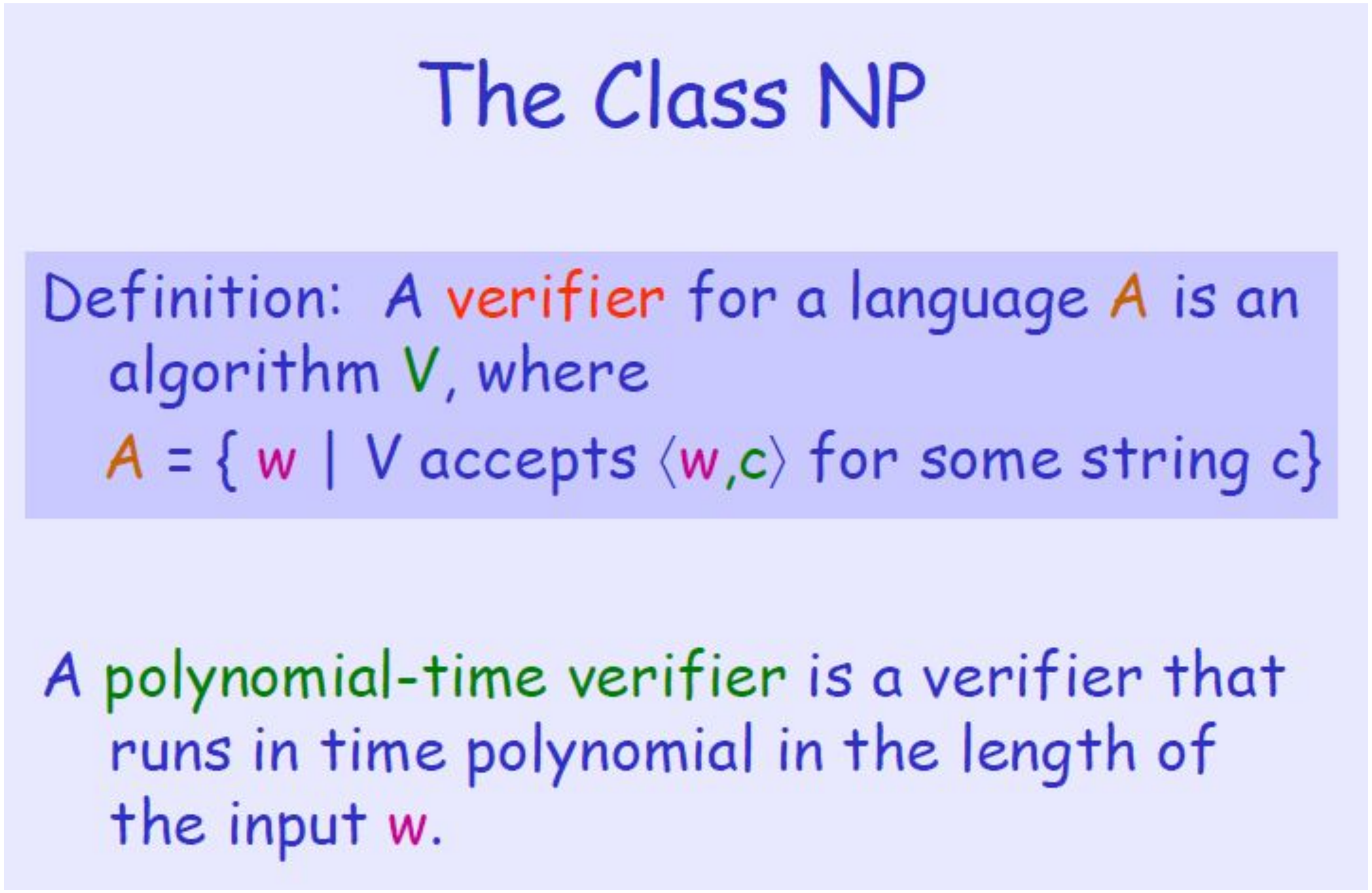
Theorem 4.7

$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ is a decidable language.

Proof

- Convert G to Chomsky Normal Form and use the CYK algorithm.
- $C =$ "On input $\langle G, w\rangle$ where G is a CFG
    1. Convert G to an equivalent grammar in CNF
    2. Run CYK algorithm on w of length n
    3. If $S \in V_{i,n}$ *accept*; otherwise *reject*."

But, CYK algorithm runs in $O(n^3)$ time.



The Class NP

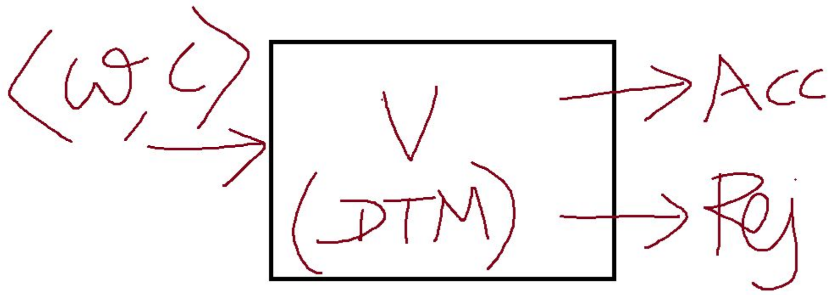
Definition: A verifier for a language A is an algorithm V, where
$A = \{ w \mid V \text{ accepts } \langle w,c \rangle \text{ for some string } c\}$

A polynomial-time verifier is a verifier that runs in time polynomial in the length of the input w.

- **c is called a certificate or proof.**
- **If there is a polynomial-time verifier then the length of c is a polynomial in the length of w. {in poly-time, the verifier can see only this much!}**

V is a DTM

- It is important to understand that V is indeed a DTM that takes <w, c> as input and decides



- Further, V runs in $O\left(n^k\right)$ where $|w| = n$.
- V is called a poly-time verifier.

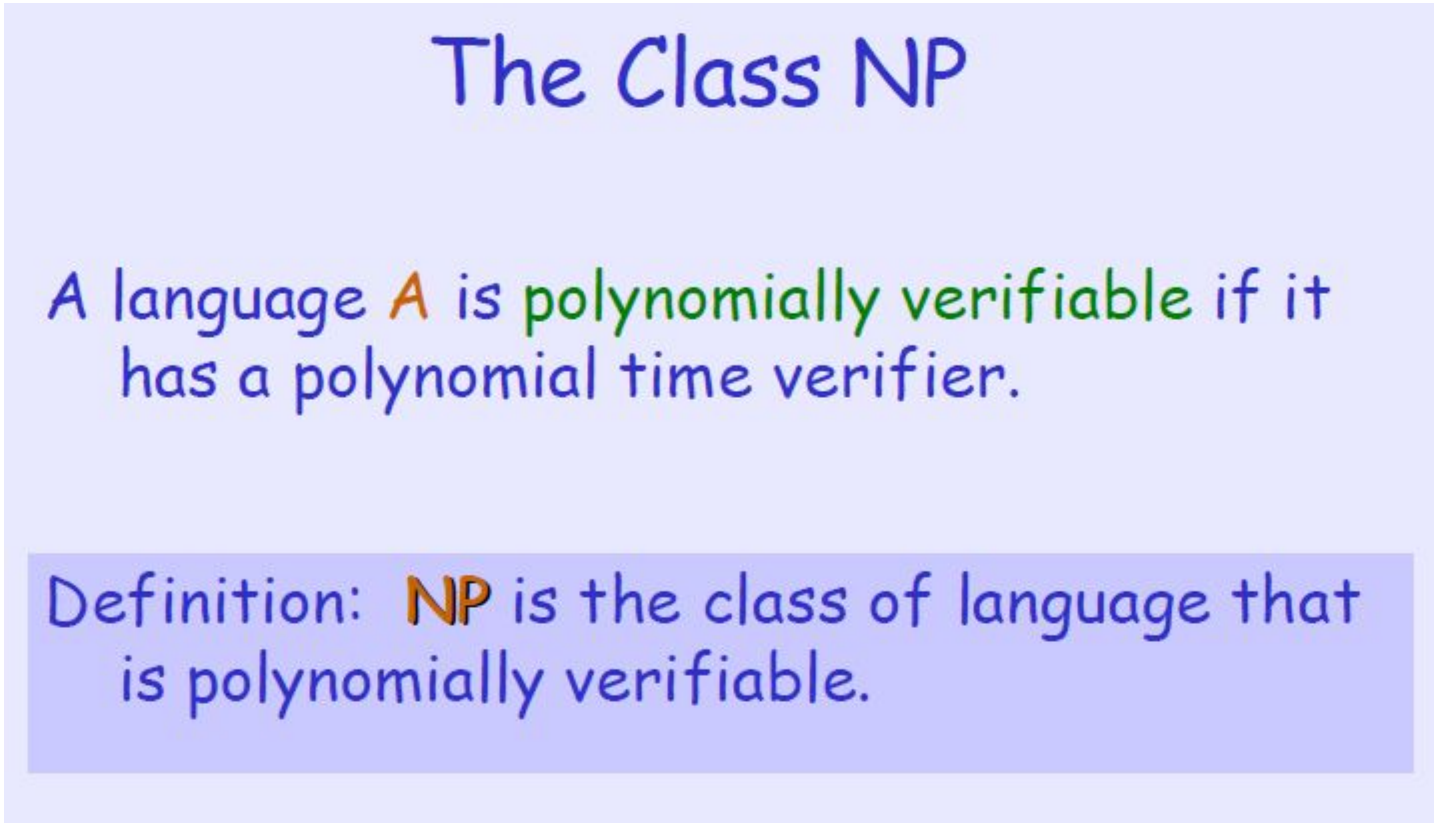
c is also a string over Γ

- c is also a string over the tape alphabet Γ, just like w is a string over Γ.

The Class NP

A language A is polynomially verifiable if it has a polynomial time verifier.

Definition: NP is the class of language that is polynomially verifiable.



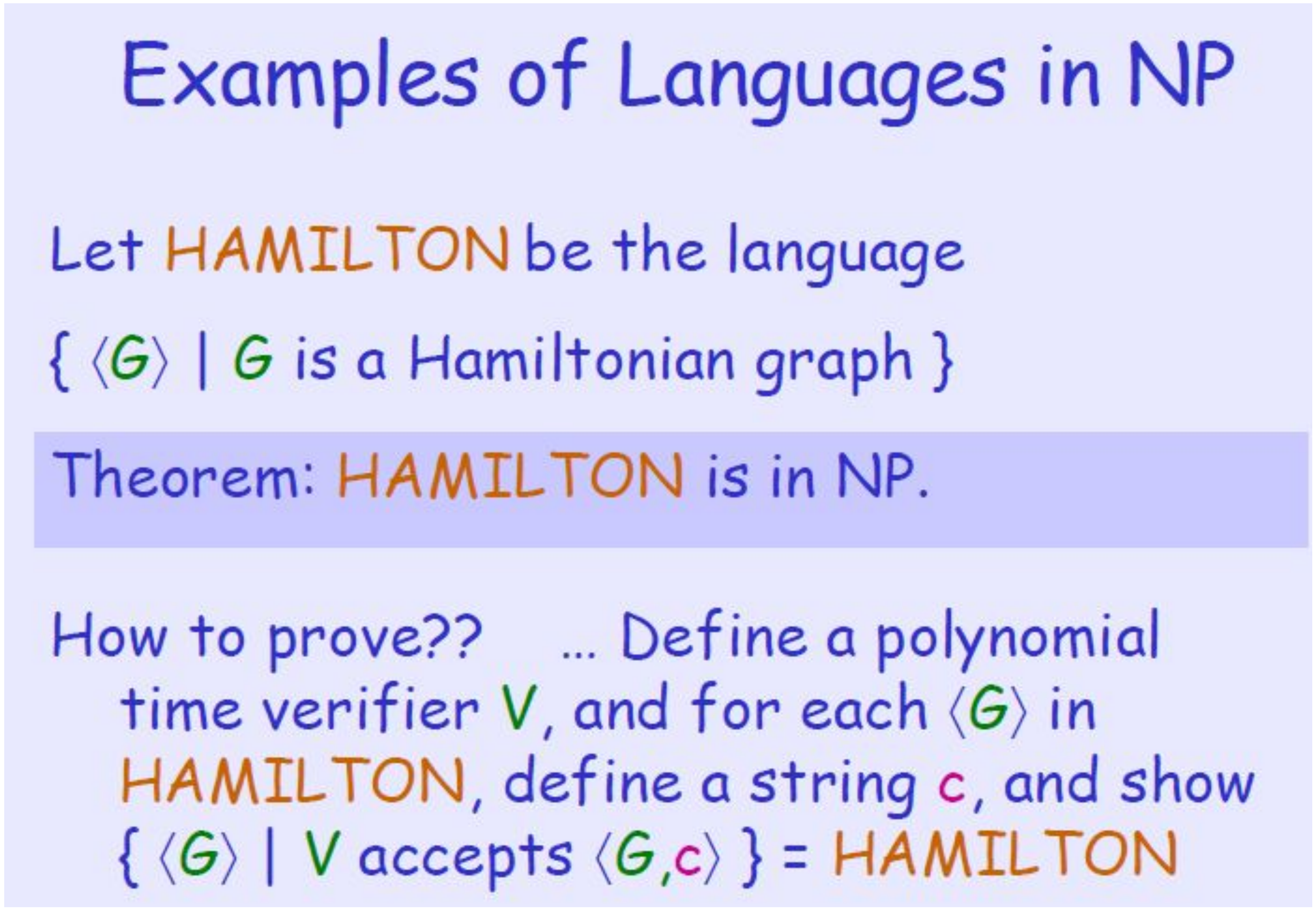
The Class NP

A language A is polynomially verifiable if it has a polynomial time verifier.

Definition: NP is the class of language that is polynomially verifiable.

- Such a V exists.
- One can give the working of V as an **algorithm** and show that it runs in poly-time over the length of w.

HAMILTON is set of graphs where each graph is having a Hamiltonian cycle.



Examples of Languages in NP

Let HAMILTON be the language

$\{ \langle G \rangle \mid G$ is a Hamiltonian graph $\}$

Theorem: HAMILTON is in NP.

How to prove?? … Define a polynomial time verifier V, and for each $\langle G \rangle$ in HAMILTON, define a string c, and show $\{ \langle G \rangle \mid V$ accepts $\langle G,c \rangle \}$ = HAMILTON

- For given graph G, there exists a certificate c whose length is a polynomial of |<G>| such that the V accepts <G, c>.
- Such c exists, which is nothing but the representation of the Hamiltonian cycle, list of nodes in G (obeying to the constraint that successive nodes in the list are connected, and no node is repeated (except the first and last)). {Next slide explains this …}

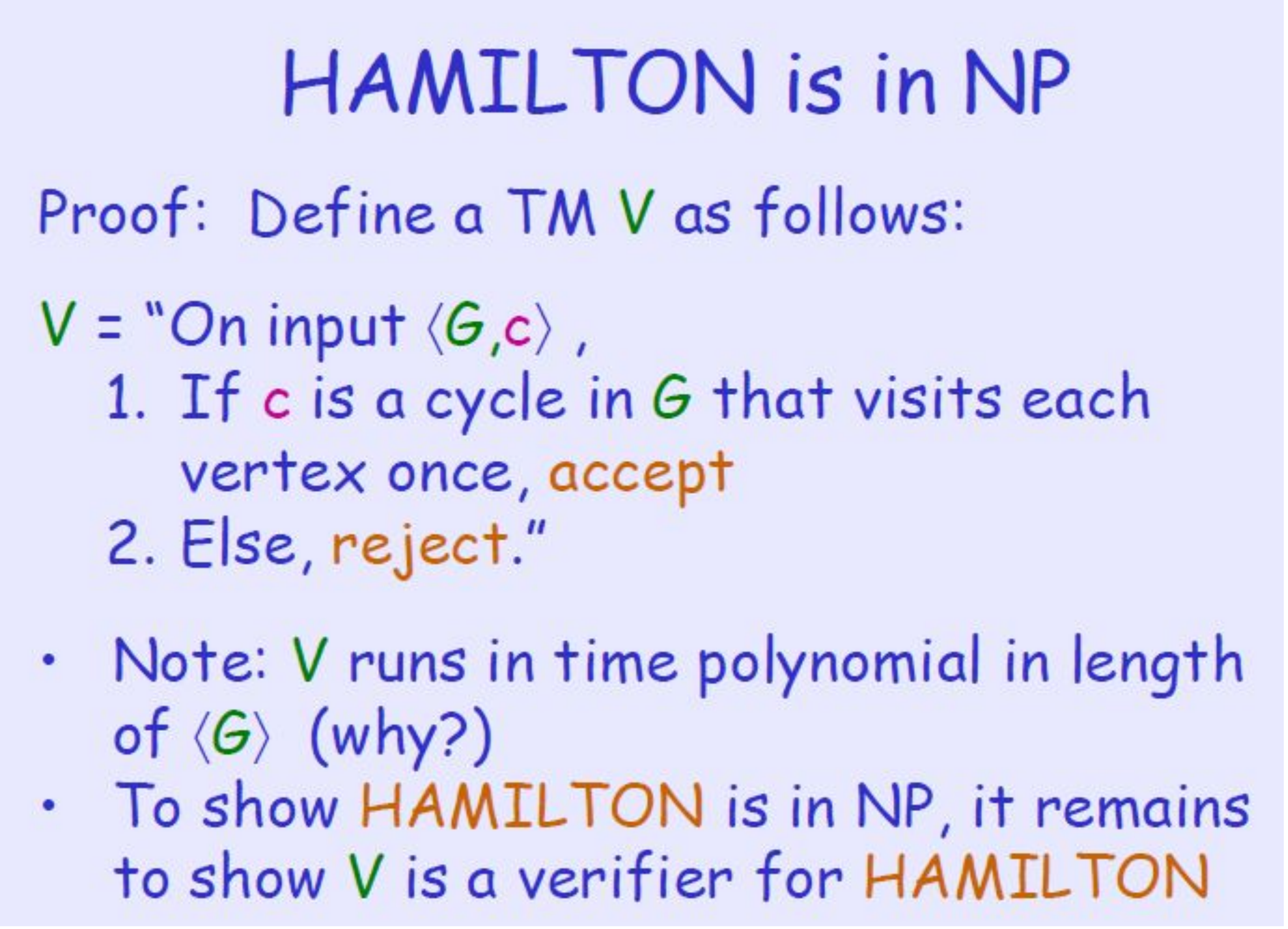
- Note, $|c| = O(|<G>|^k)$ and V runs in poly-time of |<G>|.

HAMILTON is in NP

Proof: Define a TM V as follows:

```
V = "On input ⟨G,c⟩ ,
    1. If c is a cycle in G that visits each
       vertex once, accept
    2. Else, reject."
```

- Note: V runs in time polynomial in length of ⟨G⟩ (why?)
- To show HAMILTON is in NP, it remains to show V is a verifier for HAMILTON



HAMILTON is in NP

Proof: Define a TM V as follows:

```
V = "On input ⟨G,c⟩ ,
   1. If c is a cycle in G that visits each
      vertex once, accept
   2. Else, reject."
```

- Note: V runs in time polynomial in length of ⟨G⟩ (why?)
- To show HAMILTON is in NP, it remains to show V is a verifier for HAMILTON

$c = (n_{i_1}, n_{i_2}, \dots, n_{i_m}, n_{i_1})$. A list of nodes.

$G = (set\ of\ nodes, set\ of\ edges)$.

If number of nodes is m, what is the size of G?? For Adjacency List, it is $O(m^2)$.

So, V can work in $O(m^3)$.

V does not accidentally accept <G, c>

- One has to show the correctness of the V.
- That is V accepts <G, c> if and only if <G> is in HAMILTON.

- This can be easily shown
  1. $\Rightarrow$ (the if part)
  2. $\Leftarrow$ (the only if part)

HAMILTON is in NP (2)

To show V is a verifier, we let $H = \{ \langle G \rangle \mid V$ accepts $\langle G,c \rangle \}$, and show H = HAMILTON

For every $\langle G \rangle$ in H, there is some c that V accepts $\langle G,c \rangle$. This implies $\langle G \rangle$ is a Hamiltonian graph, and $H \subseteq$ HAMILTON

For every $\langle G \rangle$ in HAMILTON, let c be one of the hamilton cycle in the graph. Then, V accepts $\langle G,c \rangle$, and so HAMILTON $\subseteq H$

Similar to Hamiltonian cycle, there is Hamiltonian PATH

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}.$$

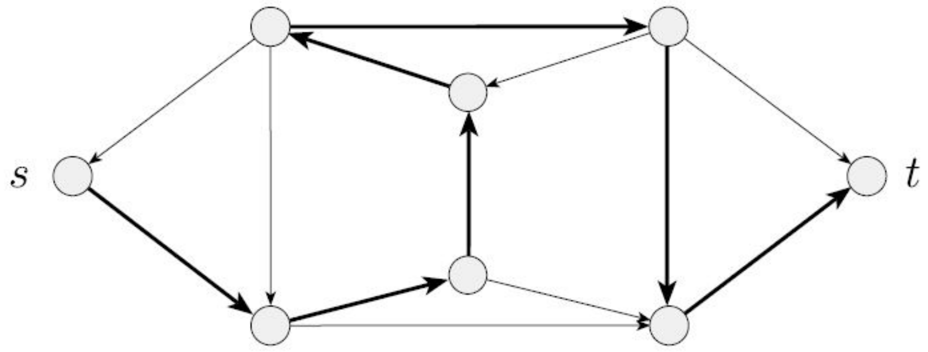


FIGURE 7.17
A Hamiltonian path goes through every node exactly once

Examples of Languages in NP (2)

Let COMPOSITE be the language

$\{ x \mid x$ is a composite number $\}$

Theorem: COMPOSITE is in NP.

How to prove?? ... Define a polynomial time verifier V, and for each x in COMPOSITE, define a string c, and show that $\{ x \mid V$ accepts $\langle x,c \rangle \}$ = COMPOSITE

COMPOSITE is in NP

Proof: Define a TM V as follows:

```
V = "On input ⟨x,c⟩ ,
    1. If c is not 1 or x, and c divides x,
       accept
    2. Else, reject."
```

- Note: V runs in time polynomial in length of ⟨x⟩ (why?)
- To show COMPOSITE is in NP, it remains to show V is a verifier for COMPOSITE

COMPOSITE is in NP (2)

To show V is a verifier, we let $C = \{ x \mid V$ accepts $\langle x,c \rangle \}$, and show $C =$ COMPOSITE

For every x in C, there is some c that V accepts $\langle G,c \rangle$. This implies x is a composite number, and $C \subseteq$ COMPOSITE

For every x in COMPOSITE, let c be one of the divisor of x with $1 < c < x$. Then, V accepts $\langle x,c \rangle$, and so COMPOSITE $\subseteq C$

Next …

- **NP** actually means **N**on-deterministically **P**olynomial.
- This is from NTM…
- Next, we show that for a language in NP there is a NTM that decides in poly-time.