Objectives

- Polynomial Time Reducibility
- Prove Cook-Levin Theorem

Polynomial Time Reducibility

- Previously, we learnt that if a problem *A* can be 'mapped' in finite steps into another problem B, we conclude that
  1. "if B is decidable, *A* is decidable"
  2. "if B is recognizable, *A* is recognizable"
- This is called mapping reducibility
- Suppose that we restrict the mapping reducibility to be done in polynomial time. What can we conclude?

Polynomial Time Reducibility (2)

We define (this slide + next slide):

Definition: A function $f:\Sigma^* \rightarrow \Sigma^*$ is a polynomial-time computable function if some polynomial-time TM M exists that halts with just f(w) on its tape, when started with input w

Polynomial Time Reducibility (3)

Definition: Language A is polynomial-time mapping reducible, or simply **polynomial-time reducible**, to language B, written as $A \leq_P B$, if a polynomial-time computable function f exists, where for each w,

$$w \in A \Leftrightarrow f(w) \in B$$

The function f is called a polynomial-time reduction of A to B

Definition of NP-Complete

Definition: Language B is NP-complete if

1. B is in NP, and
2. every language A in NP is polynomial-time reducible to B

What is so special about NP-complete?

Question: What will happen if an NP-complete language can be decided in polynomial time?

Properties of NP-Complete

Answer: Every language in NP can be decided in polynomial time (why??)

- Naturally, a NP-complete language is the "most difficult" language in NP

- In other words, we have…

Theorem: Suppose B is NP-complete. Then, B is in P if and only if P = NP

Cook-Levin Theorem

Recall that Cook-Levin Theorem is the following:

Theorem: SAT is P if and only if P = NP

We have not given its proof yet. To prove this, it is equivalent if we prove:

Theorem: SAT is NP-complete

Proof of Cook-Levin

- To prove SAT is NP-complete, we need to do two things:
  1. Show SAT is in NP
  2. Show every other language in NP is polynomial time reducible to SAT

Proof of 1: Simple

Can you give a DTM verifier proof?
Can you give an NTM decider proof?

Proof of 2: Harder…

Proof of Part 2 (Idea)

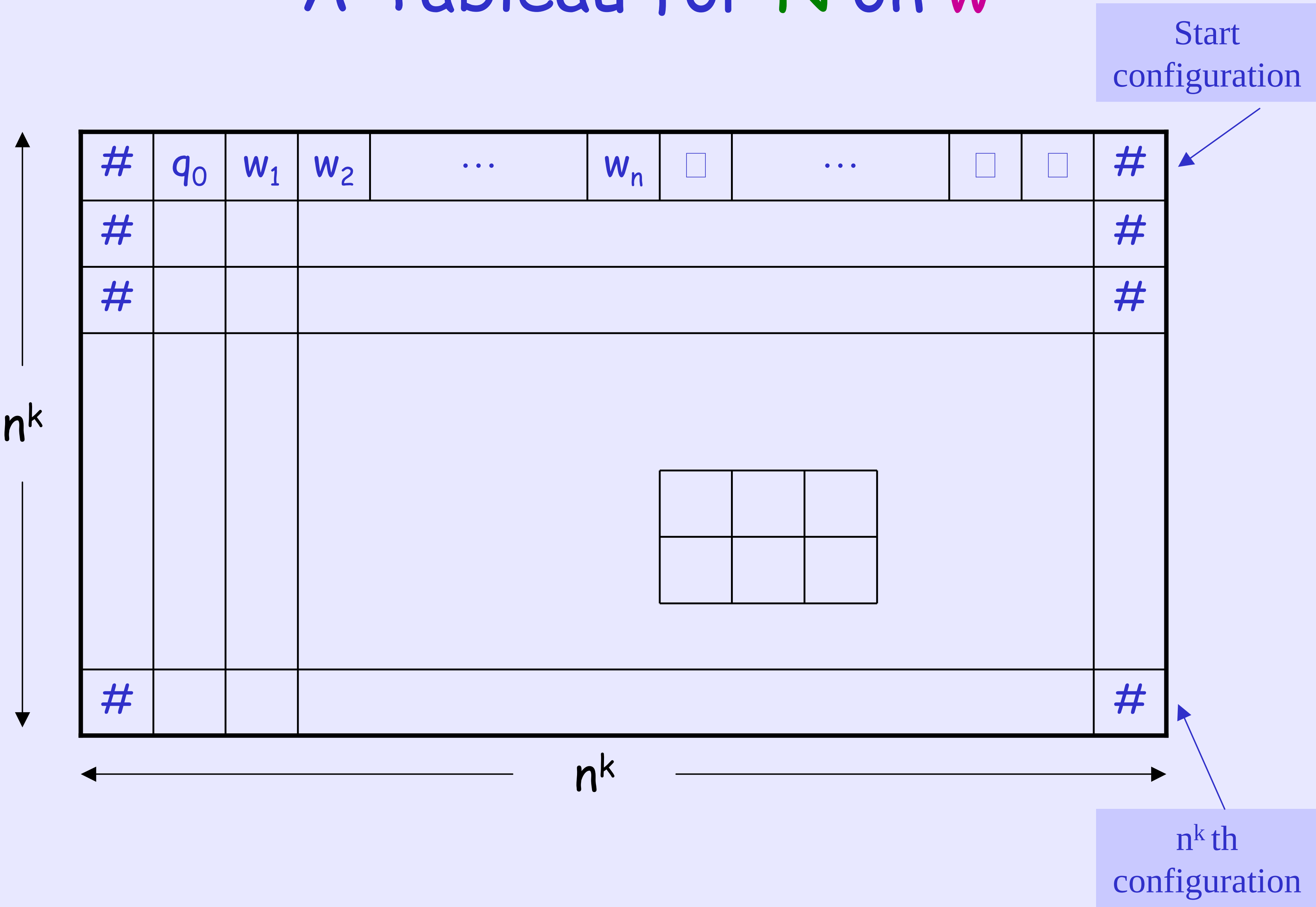
- Idea: We construct a polynomial-time reduction for each A in NP to SAT
- First, let N be an NTM that decides A
- The reduction of A takes a string w and gives a Boolean formula F such that

  N accepts w $\Leftrightarrow$ F is satisfiable
- In particular, we choose (a long and strange) F such that its satisfying assignment corresponds to the (accepting) computation for N to accept w

Proof of Part 2 (Details)

- Let N be an NTM that decides A.
- Let n^k be the running time of N on input of length n, with some constant k.
- We define a tableau for N on input w to be an n^k by n^k table that represents a branch of computation of N on w
  - Each row stores a configuration in the branch of computation
- For instance, (see next slide)

A Tableau for N on w



More on Tableau

- For convenience, we assume each configuration starts and ends with #
- The 1st row is the starting configuration, and each row follows from the previous row legally
- A tableau is accepting if any row of the tableau is an accepting configuration
    - Thus, every accepting tableau corresponds to an accepting computation

Proof of Cook-Levin (cont)

- So, deciding whether N accepts w is equivalent to deciding whether an accepting tableau for N on w exists
- Our task now is to find a formula F that can check if an accepting tableau exists …
- Let us try a formula F that contains a variable $x_{i,j,s}$ for each cell (i, j) in the tableau, and each s in C = $Q \cup \Gamma \cup \{\#\}$,
  - Later, we hope $x_{i,j,s} = 1 \Leftrightarrow$ cell (i,j) stores symbol s

Defining the Formula F

- Let us be more ambitious: we hope that when F is satisfiable, the satisfying assignment of F can tell us a valid and accepting tableau

- So, we want to ensure that the satisfying assignment (when F is satisfiable) guarantees:
  1. Each cell is occupied by exact 1 symbol
  2. The tableau has accepting configuration
  3. Each row is correct

Proof of Cook-Levin (cont)

- In particular, we will use sub-formula to represent the above three cases, so that these sub-formula is satisfiable if the corresponding three cases are correct

- The final F is obtained by "And"-ing all these formula, so that if F is satisfiable, all three cases must be correct

Each Cell has only 1 symbol

- The sub-formula $f_{i,j,1}$ ensures cell (i,j) contains at least one symbol:

$$f_{i,j,1} = \bigvee_{s \in C} \; x_{i,j,s}$$

- The sub-formula $f_{i,j,2}$ ensures cell (i,j) contains at most one symbol:

$$f_{i,j,2} = \bigwedge_{s,t \in C,\, s \neq t} ((\neg x_{i,j,s}) \vee (\neg x_{i,j,t}))$$

Thus, $f_{i,j,1} \wedge f_{i,j,2}$ will ensure cell (i,j) has exactly one symbol, if F is satisfiable

Accepting Configuration

The following sub-formula ensures the tableau has an accepting configuration if F is satisfiable:

$$f_{accept} = \bigvee_{i,j} \; x_{i,j,q_{accept}}$$

Row is Legal

To ensure starting row is correct, we use the following sub-formula:

$$f_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\square} \wedge \ldots \wedge x_{1,n^{k-1},\square} \wedge x_{1,n^k,\#}$$

To ensure the remaining rows are correct, we first define the concept of a window and legal window inside the tableau: (next slide)

Row is Legal (2)

- A window at (i,j) refers to the 2x3 cells of (i,j), (i,j+1), (i,j+2), (i+1,j), (i+1,j+1), and (i+1,j+2)
- A legal window is a window that does not violate the actions specified by the N's transition function, assuming the configuration of each row follows legally from the configuration in the row above

Row is Legal (3)

E.g.,

a	q_1	b
q_2	a	c

This window is legal if there is a transition $\delta(q_1,b) = (q_2,c,L)$

a	q_1	b
a	a	q_2

This window is legal if there is a transition $\delta(q_1,b) = (q_2,a,R)$

a	a	q_1
a	a	b

This window is legal if there is a transition $\delta(q_1,c) = (q_2,b,R)$ for some c and q_2

Row is Legal (4)

E.g.,

#	a	b
#	a	b

This window is also legal

a	b	a
a	b	q_2

This window is legal if there is a transition $\delta(q_1,b) = (q_2,c,L)$ for some q_1, b, and c

a	a	a
b	a	a

This window is legal if there is a transition $\delta(q_1,a) = (q_2,b,L)$ for some q_1 and q_2

Row is Legal (5)

E.g.,

a	b	b
a	a	b

a	q_1	b
q_2	a	q_2

a	q_1	a
q_2	c	b

All these windows cannot be legal, why?

Row is Legal (6)

- Note the the window containing the state symbol in the center top cell guarantees that the corresponding three lower cells are updated consistently with the transition function

- So, if a row stores a configuration c, and if all windows in that row are legal, then the row below it will store a configuration the follows legally from c

Row is Legal (7)

- Based on the legal window concept, the sub-formula f_{move} ensures that each row are following correctly:

$$f_{move} = \bigwedge_{1 \leq i,j \leq n^k-2} (\text{window at } (i,j) \text{ is legal})$$

where "window at (i,j) is legal" is equal to:

$$\bigvee_{a1,a2,\ldots,a6 \text{ is a legal window}} (x_{i,j,a1} \wedge x_{i,j+1,a2} \wedge x_{i,j+2,a3} \wedge x_{i+1,j,a4} \wedge x_{i+1,j+1,a5} \wedge x_{i+1,j+2,a6})$$

Proof of Cook-Levin (cont)

Thus, if

$F = (\bigwedge_{i,j} (f_{i,j,1} \wedge f_{i,j,2})) \wedge f_{accept} \wedge f_{start} \wedge f_{move}$

then F is satisfiable implies that its satisfying assignment represents an accepting tableau → N has an accepting computation on input w → N accepts w

Conversely, if N accepts w, there must be an accepting computation, and F has a satisfying assigment → F is satisfiable

Proof of Cook-Levin (cont)

- In summary, for any w, we have found a Boolean formula F such that
  N accepts w $\Leftrightarrow$ F is satisfiable
- That is, the construction of F gives a reduction from deciding a language in NP to deciding whether a formula is in SAT
- To show SAT is NP-complete, it remains to show that the construction of F is done in polynomial time (in terms of the length of the input w)

Proof of Cook-Levin (cont)

Given w of length n,

- f_{start} can be constructed in $O(n^k)$ time
- sub-formula $\bigwedge_{i,j} (f_{i,j,1} \wedge f_{i,j,2})$, f_{accept}, f_{move}

  can be constructed in $O(n^{2k})$ time (why??)

➔ Time to construct F = polynomial time

- Thus, any language in NP is polynomial-time reducible to SAT and SAT is in NP
  ➔ SAT is NP-complete

Next Time

- More NP-complete problems