CS310 : Automata Theory 2019

Lecture 40: Efficiency in computation
Classifying problems by their complexity

Instructor: S. Akshay

IITB, India

15-04-2019

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis
2. Undecidability
   (i) A proof technique by diagonalization
   (ii) Via reductions
   (iii) Rice's theorem

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis
2. Undecidability
   (i) A proof technique by diagonalization
   (ii) Via reductions
   (iii) Rice's theorem
3. Applications: showing (un)decidability of other problems
   (i) A string matching problem: Post's Correspondance Problem
   (ii) A problem for compilers: Unambiguity of Context-free languages
   (iii) Between TM and PDA: Linear Bounded Automata

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis
2. Undecidability
   (i) A proof technique by diagonalization
   (ii) Via reductions
   (iii) Rice's theorem
3. Applications: showing (un)decidability of other problems
   (i) A string matching problem: Post's Correspondance Problem
   (ii) A problem for compilers: Unambiguity of Context-free languages
   (iii) Between TM and PDA: Linear Bounded Automata
4. Efficiency in computation: run-time complexity.
   (i) Running time complexity: polynomial and exponential time
   (ii) Nondeterministic polynomial time, and the P vs NP problem.
   (iii) NP-completeness, the Cook-Levin Theorem.

The P vs NP problem

- P: class of problems solvable in polynomial time
- NP: class of problems verifiable in polynomial time

The P vs NP problem

- P: class of problems solvable in polynomial time
- NP: class of problems verifiable in polynomial time
  $=$ class of problems solvable in polynomial time in a non-determistic TM.
- EXP: class of problems solvable in exponential time.
- $NEXP$: class of problems solvable in exponential time by a non-det TM.
- $Co-\mathcal{C}$: class of problems whose complement is solvable in $\mathcal{C}$.

Polynomial time reductions

Ptime computable functions

$f : \Sigma^* \to \Sigma^*$ is Ptime computable if there is a polytime TM M, which started on any input w halts with just $f(w)$ on its tape.

Polynomial time reductions

Ptime computable functions

$f : \Sigma^* \to \Sigma^*$ is Ptime computable if there is a polytime TM M, which started on any input w halts with just $f(w)$ on its tape.

Polynomial time reduction

Language A polynomial time reducible to B, denoted $A \leq_P B$ if there is a Ptime computable function f s.t.

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

Such a function is called a Ptime reduction of A to B.

Polynomial time reductions

Ptime computable functions

$f : \Sigma^* \to \Sigma^*$ is Ptime computable if there is a polytime TM M, which started on any input w halts with just $f(w)$ on its tape.

Polynomial time reduction

Language A polynomial time reducible to B, denoted $A \leq_P B$ if there is a Ptime computable function f s.t.

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

Such a function is called a Ptime reduction of A to B.

Theorem

If $A \leq_p B$ and $B \in P$, then $A \in P$

Polynomial time reductions

Ptime computable functions

$f : \Sigma^* \to \Sigma^*$ is Ptime computable if there is a polytime TM M, which started on any input w halts with just $f(w)$ on its tape.

Polynomial time reduction

Language A polynomial time reducible to B, denoted $A \leq_P B$ if there is a Ptime computable function f s.t.

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

Such a function is called a Ptime reduction of A to B.

Theorem

If $A \leq_p B$ and $B \in P$, then $A \in P$

(Note: if there is a "halting TM" reduction from A to B, then A undecidable implied B undecidable!)

Exercise (H.W)

- Show that $3SAT$ is polytime reducible to CLIQUE.
- Show that $3SAT$ is polytime reducible to SUBSETSUM.

NP-completeness

Definition

A language B is NP-complete if two conditions hold:

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

NP-completeness

Definition

A language B is NP-complete if two conditions hold:

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

If only condition 2 holds B is said to be NP-hard.

NP-completeness

Definition

A language B is NP-complete if two conditions hold:

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

If only condition 2 holds B is said to be NP-hard.

Exercises:

- If B is NP-complete, and $B \in P$, then $P = NP$.
- If B is NP-complete, and $B \leq_P C$, then C is

NP-completeness

Definition

A language B is NP-complete if two conditions hold:

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

If only condition 2 holds B is said to be NP-hard.

Exercises:

- If B is NP-complete, and $B \in P$, then $P = NP$.
- If B is NP-complete, and $B \leq_P C$, then C is NP-hard.

NP-completeness

Definition

A language B is NP-complete if two conditions hold:

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

If only condition 2 holds B is said to be NP-hard.

Exercises:

- If B is NP-complete, and $B \in P$, then $P = NP$.
- If B is NP-complete, and $B \leq_P C$, then C is NP-hard.

The Cook-Levin Theorem

SAT is NP-complete.

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

- $SAT \in NP$.

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

- $SAT \in NP$.
- Take any language $A \in NP$ and show it is polytime reducible to SAT.

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

- $SAT \in NP$.
- Take any language $A \in NP$ and show it is polytime reducible to SAT.
- Reduction via computation histories!

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

- $SAT \in NP$.
- Take any language $A \in NP$ and show it is polytime reducible to SAT.
- Reduction via computation histories!

Sketch: Step 1 - language to tableau

1. Spse N is NTM decides A in n^k time.

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

- $SAT \in NP$.
- Take any language $A \in NP$ and show it is polytime reducible to SAT.
- Reduction via computation histories!

Sketch: Step 1 - language to tableau

1. Spse N is NTM decides A in n^k time.
2. Write $n^k \times n^k$ tableau for each computation of N on w, rows are config.

Cook-Levin Theorem

The Cook-Levin Theorem

SAT is NP-complete.

- $SAT \in NP$.
- Take any language $A \in NP$ and show it is polytime reducible to SAT.
- Reduction via computation histories!

Sketch: Step 1 - language to tableau

1. Spse N is NTM decides A in n^k time.
2. Write $n^k \times n^k$ tableau for each computation of N on w, rows are config.
3. Every accepting tableau (some config is acc) is an accepting computation branch of N on w.

Cook-Levin Theorem

The Cook-Levin Theorem

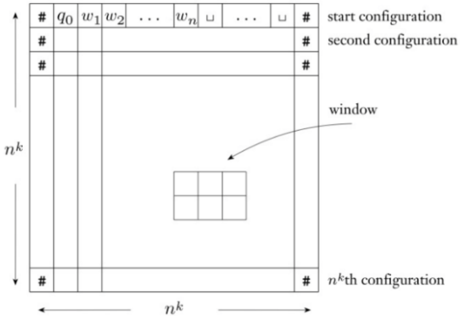
SAT is *NP*-complete.

- $SAT \in NP$.
- Take any language $A \in NP$ and show it is polytime reducible to SAT.
- Reduction via computation histories!

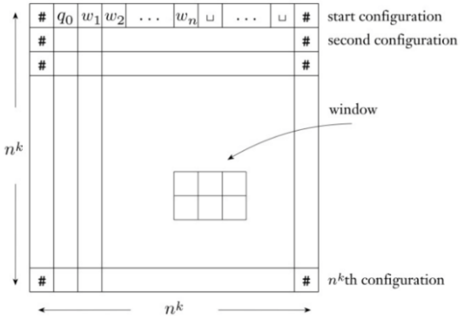
Sketch: Step 1 - language to tableau

1. Spse N is NTM decides A in n^k time.
2. Write $n^k \times n^k$ tableau for each computation of N on w, rows are config.
3. Every accepting tableau (some config is acc) is an accepting computation branch of N on w.
4. Thus, N acc w iff there exists an accepting tableau for N on w.

Cook-Levin Theorem (Contd.)



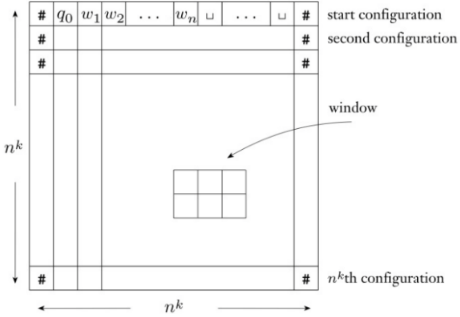
Cook-Levin Theorem (Contd.)



Sketch: Step 2 - language to tableau to SAT

Given N and w, produce formula φ:

Cook-Levin Theorem (Contd.)

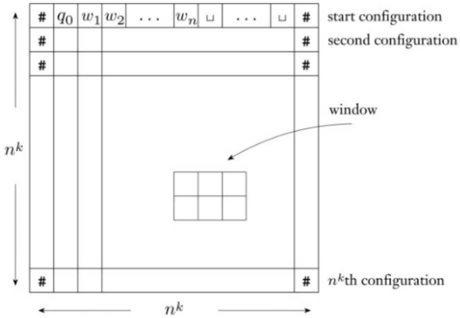


Sketch: Step 2 - language to tableau to SAT

Given N and w, produce formula φ:

1. Variable $x_{i,j,s}$ for each $1 \leq i, j \leq n^k$, $s \in C = Q \cup \Gamma \cup \{\#\}$.

Cook-Levin Theorem (Contd.)

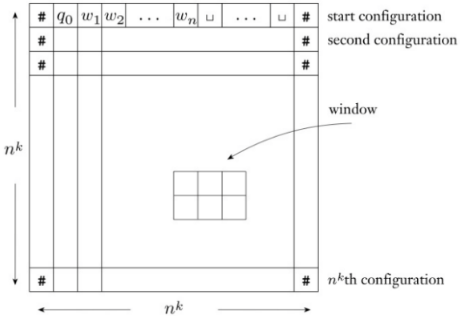


Sketch: Step 2 - language to tableau to SAT

Given N and w, produce formula φ:

1. Variable $x_{i,j,s}$ for each $1 \leq i, j \leq n^k$, $s \in C = Q \cup \Gamma \cup \{\#\}$.
2. Idea: $cell[i, j] = s$ iff $x_{i,j,s} = 1$.

Cook-Levin Theorem (Contd.)

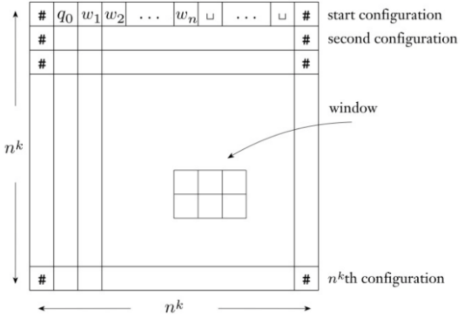


Sketch: Step 2 - language to tableau to SAT

Given N and w, produce formula φ:

1. Variable $x_{i,j,s}$ for each $1 \leq i, j \leq n^k$, $s \in C = Q \cup \Gamma \cup \{\#\}$.
2. Idea: $cell[i,j] = s$ iff $x_{i,j,s} = 1$.
3. Design φ s.t., SAT assignment corresponds to acc tableau for N on w.

Cook-Levin Theorem (Contd.)



Sketch: Step 2 - language to tableau to SAT

Given N and w, produce formula φ:

1. Variable $x_{i,j,s}$ for each $1 \leq i, j \leq n^k$, $s \in C = Q \cup \Gamma \cup \{\#\}$.
2. Idea: $cell[i,j] = s$ iff $x_{i,j,s} = 1$.
3. Design φ s.t., SAT assignment corresponds to acc tableau for N on w.
4. $\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$
2. start encodes that first row is starting config:

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$
2. start encodes that first row is starting config:
   $\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \ldots$

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$
2. start encodes that first row is starting config:
   $\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \ldots$
3. accept says that accepting config should occur somewhere in tableau

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$
2. start encodes that first row is starting config:
   $\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \ldots$
3. accept says that accepting config should occur somewhere in tableau
   $\varphi_{acc} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{acc}}$

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$
2. start encodes that first row is starting config:
   $\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \ldots$
3. accept says that accepting config should occur somewhere in tableau
   $\varphi_{acc} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{acc}}$
4. move encodes that each row correponds to config that "legally" follows the preceding row config acc to N

Cook-Levin Theorem (Contd.)

Sketch: Step 3 - SAT formula

$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{acc}$ where,

1. for each cell one variable is "on", and only one is:
   $\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} [\bigvee_{s \in C} x_{i,j,s} \wedge \bigvee_{s \neq t \in C} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})]$
2. start encodes that first row is starting config:
   $\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \ldots$
3. accept says that accepting config should occur somewhere in tableau
   $\varphi_{acc} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{acc}}$
4. move encodes that each row correponds to config that "legally" follows the preceding row config acc to N
   $\varphi_{move} = \bigwedge_{1, \leq i,j < n^k}$ (the (i,j)-window is legal)