CS310 : Automata Theory 2019

Lecture 41: Efficiency in computation
Classifying problems by their complexity

Instructor: S. Akshay

IITB, India

16-04-2019

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.

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

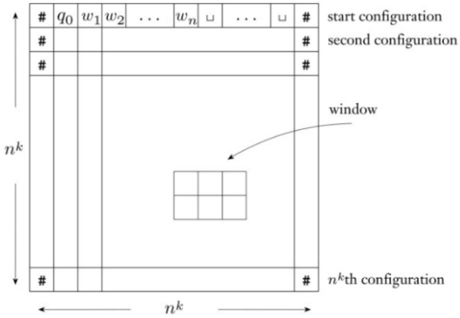
SAT is NP-complete.

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

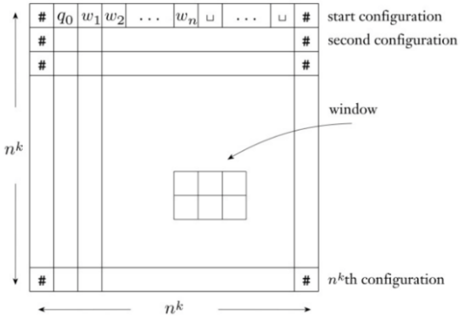
Sketch: Step 1 - language to tableau

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.)



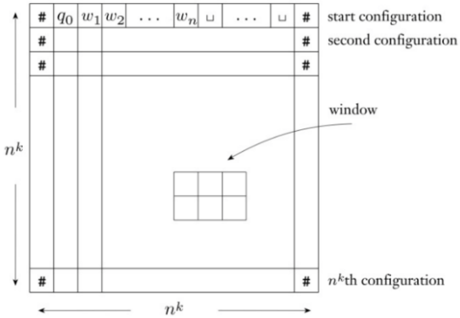
Cook-Levin Theorem (Contd.)



Sketch: Step 2 - language to tableau to SAT

Given N and w, produce formula φ:

Cook-Levin Theorem (Contd.)

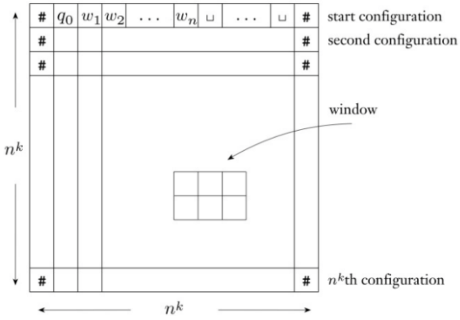


Sketch: Step 2 - language to tableau to SAT

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.)

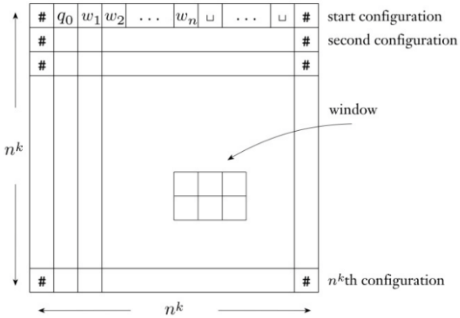


Sketch: Step 2 - language to tableau to SAT

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.)

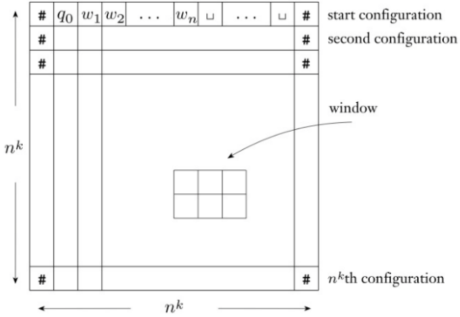


Sketch: Step 2 - language to tableau to SAT

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:
   $\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \dots$
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)

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

1. enough to ensure that each 2×3 window of cells is legal, i.e., does not violate the transition function of N.

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

1. enough to ensure that each 2×3 window of cells is legal, i.e., does not violate the transition function of N.
2. legal windows are "like" the dominos that we used in PCP!

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

1. enough to ensure that each 2×3 window of cells is legal, i.e., does not violate the transition function of N.
2. legal windows are "like" the dominos that we used in PCP!
3. Ex: Give 3 examples of legal windows and 3 illegal windows, given $\delta(q, a) = \{q, b, R\}$, $\delta(q, b) = \{(q', c, L), (q', a, R)\}$

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

1. enough to ensure that each 2×3 window of cells is legal, i.e., does not violate the transition function of N.
2. legal windows are "like" the dominos that we used in PCP!
3. Claim: If top row is start, every window is legal, then each row is config that follows prev one.

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

1. enough to ensure that each 2×3 window of cells is legal, i.e., does not violate the transition function of N.
2. legal windows are "like" the dominos that we used in PCP!
3. Claim: If top row is start, every window is legal, then each row is config that follows prev one.

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

Cook-Levin Theorem (Contd.)

Step 3 - Encoding NTM as SAT

φ_{move} encodes that each row corresponds to config that "legally" follows preceding row acc to N

1. enough to ensure that each 2×3 window of cells is legal, i.e., does not violate the transition function of N.
2. legal windows are "like" the dominos that we used in PCP!
3. Claim: If top row is start, every window is legal, then each row is config that follows prev one.

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

Encode this as a disjunct $\bigvee_{a_1 \ldots a_6 \text{is legal}} (x_{i,j-1,a_1}) \wedge x_{i,j,a_2} \wedge \ldots)$

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.
- Size of formula is $O(n^{2k})$ and can be constructed in polynomial time (from w).

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.
- Size of formula is $O(n^{2k})$ and can be constructed in polynomial time (from w).
  1. No. of cells is $(n^k)^2 = n^{2k}$, no. of cells in top row is n^k.

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.
- Size of formula is $O(n^{2k})$ and can be constructed in polynomial time (from w).
  1. No. of cells is $(n^k)^2 = n^{2k}$, no. of cells in top row is n^k.
  2. φ_{start} is $O(n^k)$ (why?)

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.
- Size of formula is $O(n^{2k})$ and can be constructed in polynomial time (from w).
  1. No. of cells is $(n^k)^2 = n^{2k}$, no. of cells in top row is n^k.
  2. φ_{start} is $O(n^k)$ (why?)
  3. All others are fixed size for each cell $= O(n^{2k})$

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.
- Size of formula is $O(n^{2k})$ and can be constructed in polynomial time (from w).
  1. No. of cells is $(n^k)^2 = n^{2k}$, no. of cells in top row is n^k.
  2. φ_{start} is $O(n^k)$ (why?)
  3. All others are fixed size for each cell $= O(n^{2k})$
  4. Can be constructed in polynomial time from N, w, i.e., polytime reduction

Cook-Levin Theorem (Completed!)

Completing the proof

- N has an acc computation on w iff φ is SAT.
- Size of formula is $O(n^{2k})$ and can be constructed in polynomial time (from w).
  1. No. of cells is $(n^k)^2 = n^{2k}$, no. of cells in top row is n^k.
  2. φ_{start} is $O(n^k)$ (why?)
  3. All others are fixed size for each cell $= O(n^{2k})$
  4. Can be constructed in polynomial time from N, w, i.e., polytime reduction

Thus, we have a PTime reduction from any NP problem to SAT, i.e., SAT is NP-hard.

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)
2. CLIQUE is NP-complete.

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)
2. CLIQUE is NP-complete.
3. SUBSETSUM is NP-complete.

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)
2. CLIQUE is NP-complete.
3. SUBSETSUM is NP-complete.
4. Vertex cover is NP-complete.

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)
2. CLIQUE is NP-complete.
3. SUBSETSUM is NP-complete.
4. Vertex cover is NP-complete.
5. A whole growing book of NP-complete problems! (Garey-Johnson)

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)
2. CLIQUE is NP-complete.
3. SUBSETSUM is NP-complete.
4. Vertex cover is NP-complete.
5. A whole growing book of NP-complete problems! (Garey-Johnson)

So, why are so many natural problems either in P or NP-complete?

NP-completeness examples

1. $3SAT$ is NP-complete. (why?)
2. CLIQUE is NP-complete.
3. SUBSETSUM is NP-complete.
4. Vertex cover is NP-complete.
5. A whole growing book of NP-complete problems! (Garey-Johnson)

So, why are so many natural problems either in P or NP-complete?

Well, there are some natural problems that are not known to be in P, but that are in NP and not known to be NP-hard.

Getting around NP-hardness

Many problems are *NP*-complete. So what do people do?

An algorithmician's answer:

1. Approximate!

Getting around NP-hardness

Many problems are NP-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.

Getting around NP-hardness

Many problems are *NP*-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.
3. Parametrize.

Getting around NP-hardness

Many problems are *NP*-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.
3. Parametrize.
4. Quantize!

Getting around NP-hardness

Many problems are NP-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.
3. Parametrize.
4. Quantize!
5. average/amortized/smoothed complexity.

Getting around NP-hardness

Many problems are NP-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.
3. Parametrize.
4. Quantize!
5. average/amortized/smoothed complexity.

A practitioner's answer

1. What hardness? SAT is easy (almost always)!

Getting around NP-hardness

Many problems are *NP*-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.
3. Parametrize.
4. Quantize!
5. average/amortized/smoothed complexity.

A practitioner's answer

1. What hardness? SAT is easy (almost always)!
2. The advent of SAT-solvers. Can solve SAT queries on millions of variables in seconds!

Getting around NP-hardness

Many problems are *NP*-complete. So what do people do?

An algorithmician's answer:

1. Approximate!
2. Randomize.
3. Parametrize.
4. Quantize!
5. average/amortized/smoothed complexity.

A practitioner's answer

1. What hardness? SAT is easy (almost always)!
2. The advent of SAT-solvers. Can solve SAT queries on millions of variables in seconds!
3. Goal nowadays: develop efficient encoding into SAT!