Computational Models — Lecture 13[1]

Handout Mode

Ronitt Rubinfeld and Iftach Haitner.

Tel Aviv University.

June 6/8, 2016

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Reminder - poly-time reductions, NP-completeness and SAT
- 3SAT $\leq_P$ CLIQUE
- SAT is NP-Complete
- SAT $\leq_P$ 3SAT

- Sipser, 7.4–7.5

Section 1

Reminder

Reminder – Poly-time Reductions

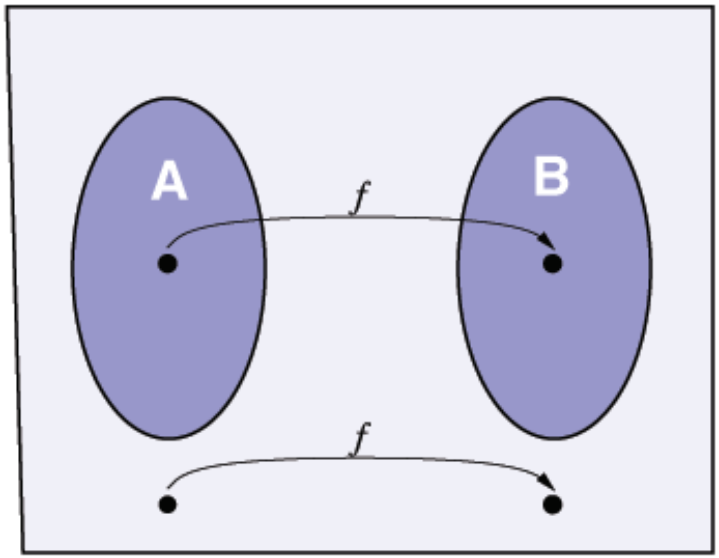
Definition 1

A language A is poly-time mapping reducible to B, denoted $A \leq_P B$, if exists poly-time computable function f such that

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

for every $w \in \Sigma^*$.
The function f is called a polynomial-time reduction from A to B.

The mapping f efficiently converts questions about membership in A to membership in B.



Reminder – NP Completeness

Definition 2 (NP-complete)

A language B is NP-complete, if

- B $\in \mathcal{NP}$, and
- Every A $\in \mathcal{NP}$ is poly-time reducible to B

We let $\mathcal{NPC}$ denote the class of all NP-complete languages.

$A_{NP} = \{\langle M, x, 1^n\rangle : M \text{ is a TM} \wedge \exists c \in \Sigma^* \text{ s.t.} M(x, c) \text{ accepts within } n \text{ steps}\}$.

Theorem 3

$A_{NP} \in \mathcal{NPC}$.

Theorem 4

Assume that B $\in \mathcal{NP}$, A $\in \mathcal{NPC}$ *and* A $\leq_P$ B, *then* B $\in \mathcal{NPC}$.

What does a proof of NP-completeness look like?

Checklist:

- Show that L is in $\mathcal{NP}$
- Show that L is $\mathcal{NP} - hard$: reduce from a known NP-complete language L' via f
  - show that f is poly-time computable
  - show that if $w \in L'$ then $f(w) \in L$
  - show that if $w \notin L'$ then $f(w) \notin L$

Boolean Formulas and SAT

A Boolean formula is an expression involving Boolean variables and operations.

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Definition 5 (satisfiable formula)

A formula is satisfiable, if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

The formula $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ is satisfiable by the assignment

$$x = 0$$
$$y = 1$$
$$z = 0$$

The language of satisfied formulas:

$$\text{SAT} = \{\langle\phi\rangle : \ \phi \text{ is a satisfiable Boolean formula}\}$$

Conjunctive Normal Form

It is useful to consider a special version of SAT

- A literal is a variable or negated variable: x or $\overline{x}$.
- A clause is several literals joined by $\vee$s: $(x_1 \vee \overline{x_2} \vee \overline{x_3})$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses, connected with $\wedge$s.

  For example: $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$

Definition 6

A Boolean formula is in k-CNF form, if it is a CNF formula, and all clauses have k literals.

- Example of 3CNF: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

CSAT and 3SAT

- SAT $= \{\langle\phi\rangle : \ \phi$ is a satisfiable Boolean formula$\}$
- CSAT $= \{\langle\phi\rangle : \ \phi$ is a satisfiable CNF formula$\}$
- 3SAT $= \{\langle\phi\rangle \ : \ \phi$ is satisfiable 3CNF formula$\}$.

- Are these all NP-complete?
  - Will soon show the Cook-Levin theorem: SAT is NP-complete!
  - Can slightly modify proof to show CSAT $\in \mathcal{NPC}$! (take home exercise :-))
  - Hence, to prove that 3SAT $\in \mathcal{NPC}$, it suffices to prove that CSAT $\leq_P$ 3SAT.

Section 2

CSAT vs. 3SAT

CSAT $\leq_P$ 3SAT

- The reduction maps CNF formulae to 3CNF ones "clause by clause".
- A clause with $d \leq 3$ literals is mapped to equivalent clause with 3 literals.

  $(x_1 \vee x_2) \longmapsto (x_1 \vee x_2 \vee x_2)$
- A clause with $d > 3$ literals is mapped to $d - 2$ clauses, built on the original literals together with $(d - 3)$ new variables.

  $\phi = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_8) \longmapsto$
  $\phi_3 = (x_1 \vee \overline{x_2} \vee y_1) \wedge (\overline{y_1} \vee \overline{x_3} \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee x_8)$

The reduction works!

Claim 7

ϕ has a satisfying assignment iff ϕ_3 does.

Proof's idea: (for case $d > 3$)
$\Leftarrow$ Given a satisfying assignment for ϕ_3, use the settings of the x_i's to get satisfying assignment for ϕ. Why does it work? For each set of clauses that came from a clause in ϕ, note that each y_i sets exactly one clause to true. There are $k-2$ clauses but only $k-3$ y_i's. So at least one clause needs to be satisfied by one of the x_j or $\overline{x_j}$ literals. Set this literal to true in ϕ.

$\Rightarrow$ An assignment satisfying ϕ, makes at least one literal per clause happy. In the "ϕ_3 clause" of this literal the new variables are under no constraints. Use the $d-2$ new variables to set the remaining $d-2$ clauses to "true".

CSAT $\leq_P$ 3SAT, cont.

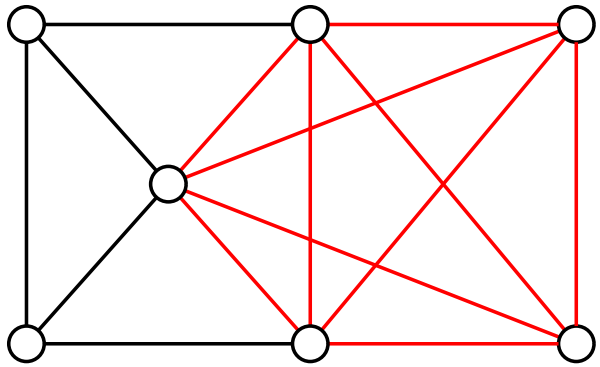
$\phi = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_8) \longmapsto$
$\phi_3 = (x_1 \vee \overline{x_2} \vee y_1) \wedge (\overline{y_1} \vee \overline{x_3} \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee x_8)$

- Doing the above mapping to each clause of a CNF formula, we get a 3CNF that is satisfied iff the original one is.
- Since this mapping is polynomial time (?), we get CSAT $\leq_P$ 3SAT. ♣.

Section 3

Clique

3SAT $\leq_P$ CLIQUE



CLIQUE $= \{\langle G, k\rangle : G$ is an undirected graph with a k-clique$\}$

Claim 8

3SAT $\leq_P$ CLIQUE.

Since CLIQUE $\in \mathcal{NP}$, it follows that CLIQUE $\in \mathcal{NPC}$.

Proof's idea: We'll construct a poly-time reduction f that maps 3CNF formulae ϕ to pairs $\langle G, k\rangle$ of graphs and numbers.

The function f will have the property that ϕ is satisfiable, iff G has a clique of size k.

Proving 3SAT $\leq_P$ CLIQUE

On input ϕ, a 3CNF formula, the mapping reduction is defined as follows:

let k be the number of clauses in ϕ.

We construct a graph $G = G(\phi)$, see below, and output (G, k).

The graph G is defined as follows:

- Nodes in G are organized into triples $t_1, \ldots, t_k$.
- Each triple corresponds to a clause of ϕ
- Each node in a triple corresponds to a literal in corresponding clause.

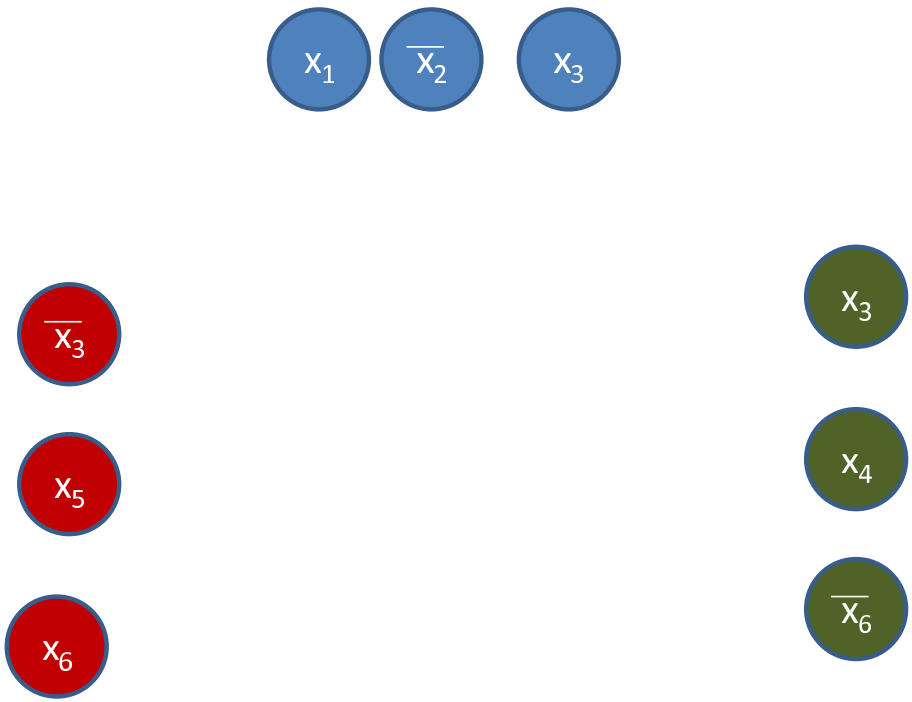
Ongoing example:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

Nodes of G

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$

Add a node per literal

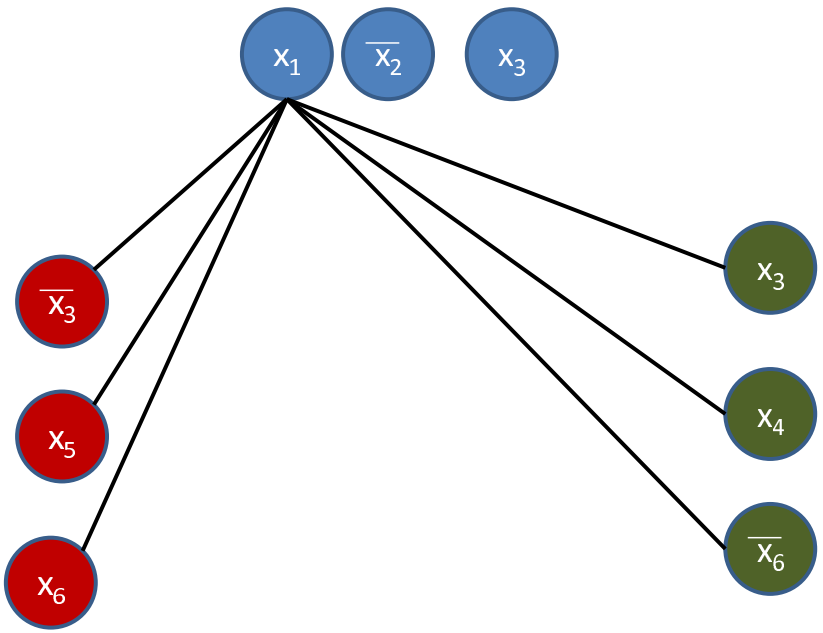


Edges of G

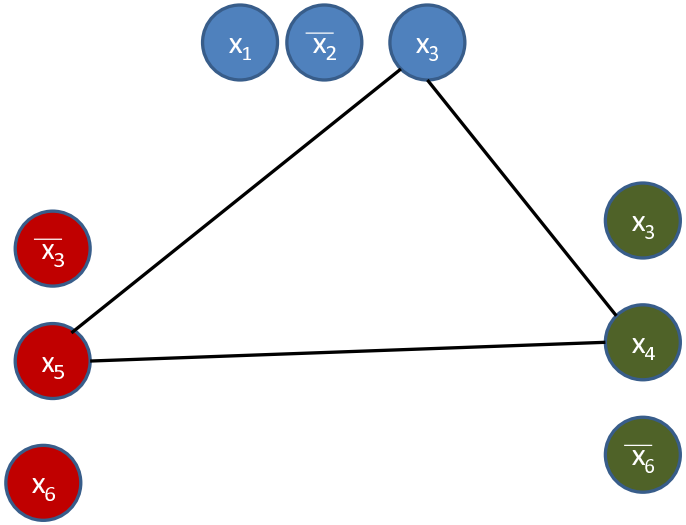
Add edges between all vertex pairs, except

- within same triple
- between contradictory literals (e.g., x_3 and $\overline{x_3}$)

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$



$\phi \in 3SAT \implies \langle G, k \rangle \in CLIQUE$

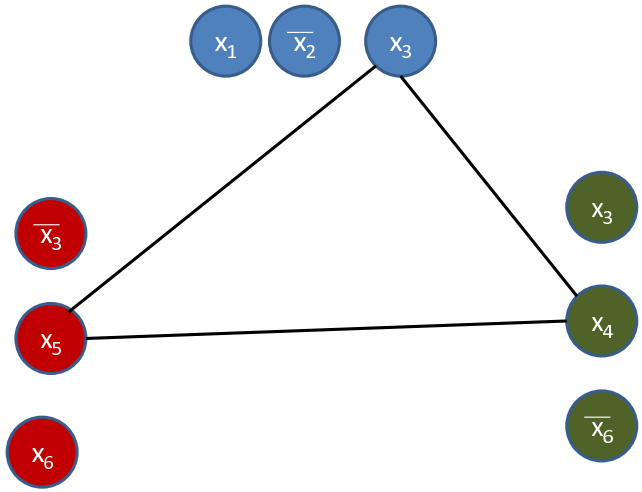


Proof: Suppose ϕ is satisfiable by an assignment ψ.
With respect to ψ:

- At least one literal is assigned to 1 in every clause (?)
- Select a 1-literal in every tuple;
  These literals can be joined by edges (?)
  Yielding a k-clique in G. ♣

♣

$\langle G, k \rangle \in \text{CLIQUE} \implies \phi \in \text{3SAT}$



Proof: Suppose G has a k-clique.

- No two of the cliques nodes are in the same triple. (?)
- G has $3k$ vertexes and k clauses, so each triple has exactly one clique node.
- Assign 1 to each node in clique;
  Assignment has no contradictions (?)
  Yielding a satisfying assignment to ϕ.

Recap

- We've constructed a poly-time computable function f.
- We saw that f has the property that $\phi \in$ 3SAT iff $f(\phi) \in$ CLIQUE.
- Therefore, f is a poly-time reduction from 3SAT to CLIQUE

  $\Longrightarrow$ 3SAT $\leq_P$ CLIQUE. ♣

Section 4

SAT is NP-Complete

SAT $\in \mathcal{NPC}$

Theorem 9 (Cook-Levin, early 70s)

SAT $\in \mathcal{NPC}$.

Proof's idea:

- Encode all information about whole history of computation as a (large but polynomial) set of clauses, where settings of variables give you all information about the history.
  - e.g., variables for each (tape cell, time step) telling you whether the tape head is there, and what is written
- Each step of computation changes only very local information (only variables near the tape head). All other information must stay the same.
  - if tape head not on my tape cell at time i, then I must stay the same at time $i + 1$
  - if tape head is on my tape cell, then my contents change according to δ
- Verifying that a computation history is "legal" involves only local checks (a lot of them, but only polynomial)

SAT $\in \mathcal{NPC}$

Theorem 10 (Cook-Levin, early 70s)

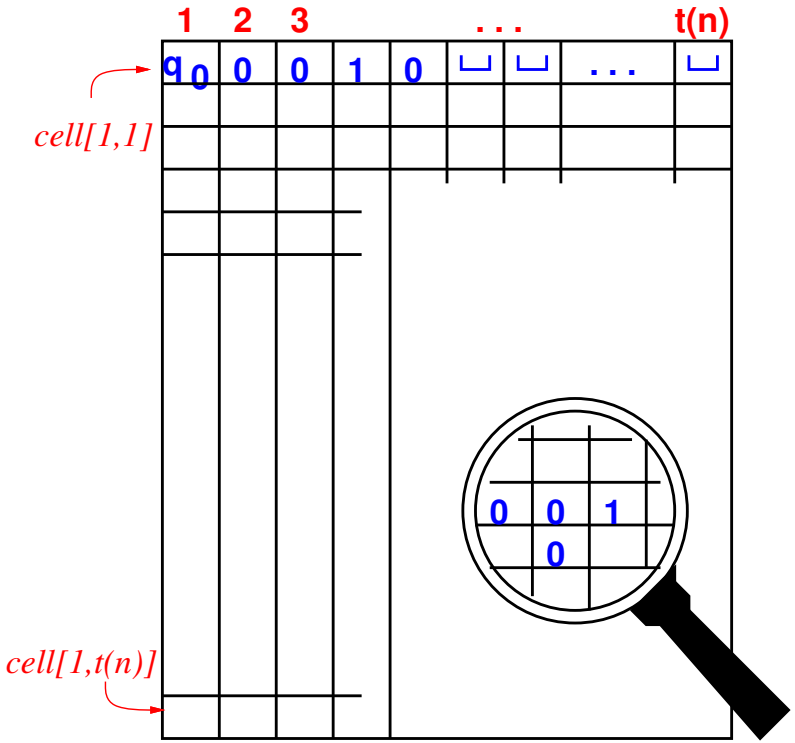
SAT $\in \mathcal{NPC}$.

Proof's idea:

- Suppose L $\in \mathcal{NP}$ and let $N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ be an n^c-time NTM that accepts L.
- Given the string $w \in \{0,1\}^*$, construct in time $O(|w|^{2c})$ a formula $\phi_{N,w}$ such that: $\phi_{N,w} \in$ SAT iff N accepts w.
- Hence, the mapping $w \mapsto \phi_{N,w}$ is a poly-time reduction from L to SAT, establishing L $\leq_P$ SAT.
- In the following fix L, N and $w \in \{0,1\}^n$.
- We assume wlg. that $\delta(q_r, \cdot) = \emptyset$.

The Configuration-History Tableau

Consider the n^c-by-n^c Tableau that describes a possible accepting computation history of N on input w.



- First row represents initial configuration of N on input w.
- i'th row represents the i-th configuration in a possible computation of N on input w.

Some interesting variables telling us the contents of the tape at any particular time

How do we encode the contents of the tape cell j at time i?

- How about "let $x_{i,j} = s$"? Problem: NOT BOOLEAN!
- WHAT WE DO: Let $x_{i,j,s} = 1$ iff j^{th} cell contains s at time i
- What if the tape head is there too? Let's use an extra cell: state of tape head inserted to the left of the cell that it is sitting on.

Note that for each i, j, exactly one $x_{i,j,s}$ should be set to 1. Not very compressed, but still polynomial!

The formula $\phi_{N,w}$

- Let $S = Q \cup \Gamma$ (the alphabet of the configuration history).
- $\phi_{N,w}$ uses Boolean variables $\{x_{i,j,s}\}_{i,j \in [n^c], s \in S}$.

$$\phi_{N,w} = \phi_{\text{Cell}(N)} \wedge \phi_{\text{Start}(w)} \wedge \phi_{\text{Move}(N)} \wedge \phi_{\text{Accept}(N)}$$

- Given an assignment z for $\phi_{N,w}$, let $T(z)$ be the $n^c \times n^c$ Tableau, defined by setting the j-th cell in i'th configuration to s, if $x_{i,j,s} = 1$ in z.

  ($T(z)$ is undefined, if $x_{i,j,s'} = x_{i,j,s} = 1$ for some $s \neq s' \in S$, or $x_{i,j,s} = 0$ for all $s \in S$).

- $T(z)$ will represents a (possible) accepting execution of $N(w)$, iff z is an a satisfying assignment for $\phi_{N,w}$.

The formula $\phi_{\text{Cell}(N)}$

$\phi_{\text{Cell}(N)}$ guarantees that the variables encode legal configurations:

- Each cell (i,j) has at least one letter: $\bigvee_{s \in S} x_{i,j,s}$.
- No cell (i,j) has two or more letters $\bigwedge_{s \neq s' \in S} \overline{x_{i,j,s} \wedge x_{i,j,s'}}$.

Together:

$$\phi_{\text{Cell}(N)} = \bigwedge_{i,j} \left[\left(\bigvee_{s \in S} x_{i,j,s} \right) \wedge \left(\bigwedge_{s \neq s' \in S} \overline{x_{i,j,s} \wedge x_{i,j,s'}} \right) \right]$$

Claim 11

If an assignment z satisfies $\phi_{\text{Cell}(N)}$, then $T(z)$ is defined.

The formula $\phi_{\text{Start}(w)}$

$\phi_{\text{Start}(w)}$ guarantees that the first row encodes the initial configuration (i.e., $q_0 w$).

$$\phi_{\text{start}(w)} = x_{1,1,q_0} \wedge x_{1,2,w_1} \wedge x_{1,3,w_2} \wedge \ldots \wedge x_{1,n+1,w_n} \wedge x_{1,n+2,\sqcup} \wedge \ldots \wedge x_{1,n^c,\sqcup}$$

Claim 12

If z satisfies $\phi_{\text{Cell}(N)} \wedge \phi_{\text{Start}(w)}$, then the first line of $T(z)$ is $q_0 w \underbrace{\sqcup \ldots \sqcup}_{n^c - n - 1}$.

The formula $\phi_{\text{Move}(N)}$

$\phi_{\text{Move}(N)}$ is the "heart" of $\phi_{N,w}$. To construct it, we employ locality of computations.

Observation: Configuration C, with head location h, yields configuration C' (with respect to δ), if the following holds.

- $C'_i = C_i$ for any $i \notin \{h-1, h, h+1\}$
- $C'_{h-1,h,h+1}$ is consistent (with respect to δ) with $C_{h-1,h,h+1}$.

We check that each configuration in $T(z)$ yields the next one, by inducing local "checks" on z.

$\phi_{\text{Move}(N)}$ – Rectangles

A rectangle is a 2×3 configuration sub-table.

Assume that
$\delta(q_1, a) = \{(q_1, b, \text{R})\}$ and $\delta(q_1, b) = \{(q_2, c, \text{L}), (q_2, a, \text{R})\}$.

- (some) Legal 2×3 rectangles:

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

a	b	a
a	b	q_2

b	b	b
c	b	b

a	a	b
a	a	b

- (some) Illegal 2×3 rectangles:

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

The set of legal moves are a constant, determined by δ!

$\phi_{\text{Move}(N)}$ – Characterizing legal rectangles

The formula “verifies" that all 2×3 rectangles in the Tableau are in the list C:

1.

a	b	c
$*$	b	$*$

2.

a	q	b
q’	a	b’

$(\text{L}, q', b') \in \delta(q, b)$

a	q	b
a	b’	q’

$(\text{R}, q', b') \in \delta(q, b)$

3.

q	$*$	$*$
$*$	$*$	$*$

$*$	$*$	q
$*$	$*$	$*$

- Some rectangles in C are clearly illegal.
- For rectangles on the left-most and right-most side of Tableau, we use slightly different first type rectangles.

$\phi_{\text{Move}(N)}$– formal definition

- For each entry $(i,j) \in [n^c] \times [n^c]$ and $c \in C$, let $\phi_{\text{Move},i,j,c}$ be the formula taking the value 1 iff the 2×3 table of cells in the Tableau whose upper-left corner is (i,j) is c.

  For instance, for entry $(1,1)$ and $c =$

a	q_1	b
q_2	a	d

  ,

  let $\phi_{\text{Move},1,1,c} = x_{1,1,a} \wedge x_{1,2,q_1} \wedge x_{1,3,b} \wedge x_{2,1,q_2} \wedge x_{2,2,a} \wedge x_{2,3,d}$

- Finally, let $\phi_{\text{Move}(N)} = \bigwedge_{(i,j)} \bigvee_{c \in C} \phi_{\text{Move},i,j,c}$.

Claim 13

If z satisfies $\phi_{\text{Cell}(N)} \wedge \phi_{\text{Start}(w)} \wedge \phi_{\text{Move}(N)}$, then $T(z)$ is a possible configuration history of $N(w)$.

Proof: By induction on the row index. Base case: z satisfies $\phi_{\text{Cell}(N)} \wedge \phi_{\text{Start}(w)}$.

Assume configuration defined in rows $1, \ldots, i$ is possible and head is in cell j. The configuration of rows $1, \ldots, i+1$ is also possible: Cells of indices not in $\{j-1, j, j+1\}$, by first type of rectangles in C. Other cells, by second type rectangles in C. Q: Why de we need the third type of cells?

The formula $\phi_{\text{Accept}(N)}$

$\phi_{\text{Accept}(N)}$ guarantees that some row encodes an accepting configuration (i.e., uq_av):

$$\phi_{\text{Accept}(N)} = \bigvee_{i,j} x_{i,j,q_a}$$

Claim 14

If z satisfies $\phi_{N,w} = \phi_{\text{Cell}(N)} \wedge \phi_{\text{Start}(w)} \wedge \phi_{\text{Move}(N)} \wedge \phi_{\text{Accept}(N)}$, then $T(z)$ is an accepting configuration history of $N(w)$.

Correctness of reduction

- The transformation $w \mapsto \phi_{N,w}$ is computable in time $O(n^{2c})$.
- An assignment satisfying $\phi_{N,w}$, corresponds to an accepting configuration history of $N(w)$.
- An accepting configuration history of $N(w)$ corresponds to an assignment satisfying $\phi_{N,w}$. (?)

  Therefore, N accepts w iff $\phi_{N,w} \in$ SAT. ♣.

- For complete details, consult Sipser chapter 7.4.

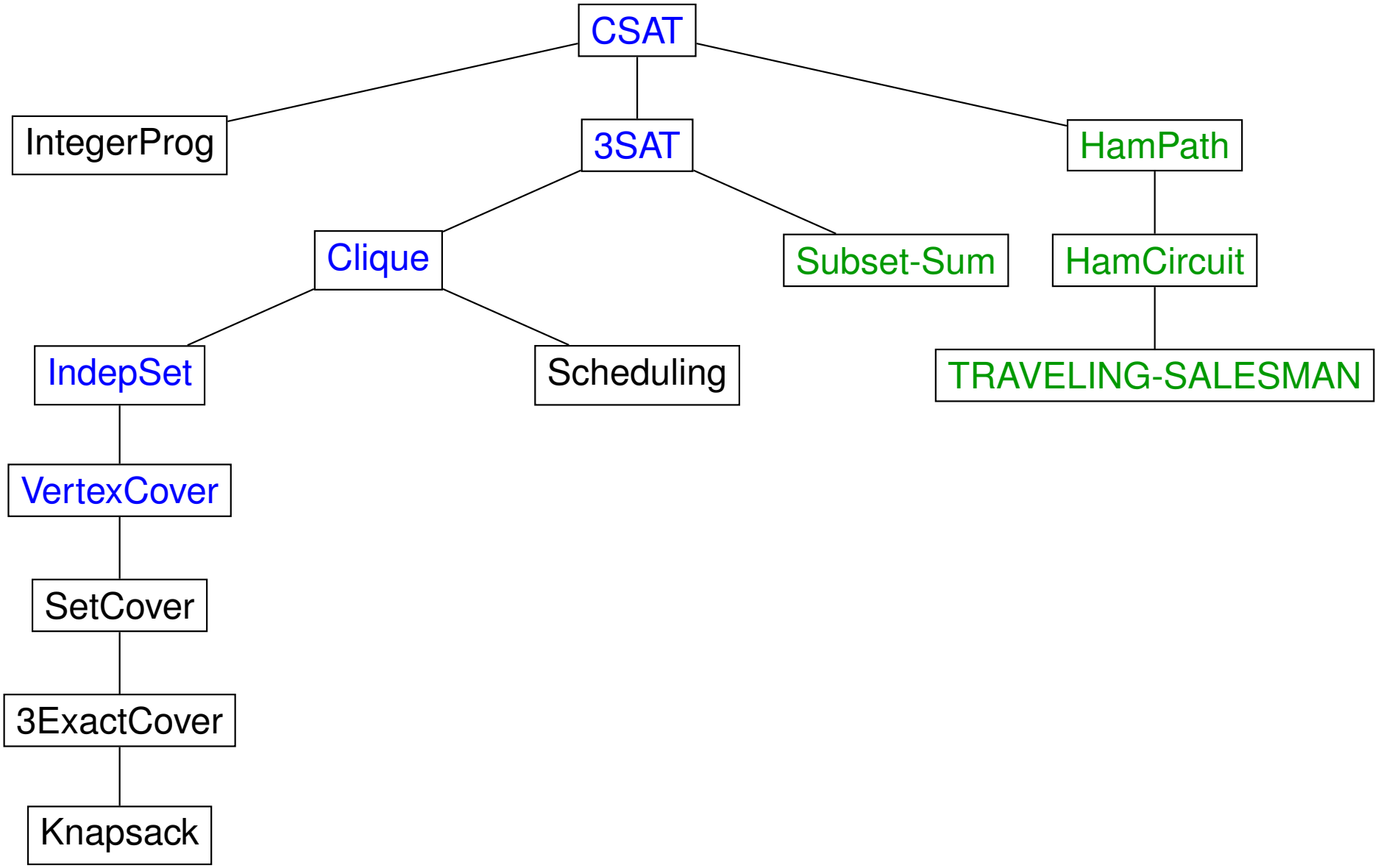
Section 5

Summary so far

Summary so Far

- We have seen that SAT and CSAT are NP-complete.
- CSAT $\leq_P$ 3SAT
- 3SAT $\leq_P$ CLIQUE
- CLIQUE $\leq_P$ IND-SET
- Hence, all of the above languages are NP-complete
- Full list of NP-complete languages contains thousands of known NP-complete problems (from combinatorics, operation research, VLSI design, computational geometry, bioinformatics, . . .).
- NP-completeness of new and of old problems is still established these days.

Chains of reductions: NPC languages



Why all the fuss about decision problems?

Don't we want to *find* the satisfying assignment?

Given an oracle for the decision problem, show that you can find an assignment in poly time!

Beyond NP-completeness?

Exciting ideas to come in Computational Complexity!!

e.g., you will learn to understand a statement like "Super-Mario Brothers is PSPACE-complete"!

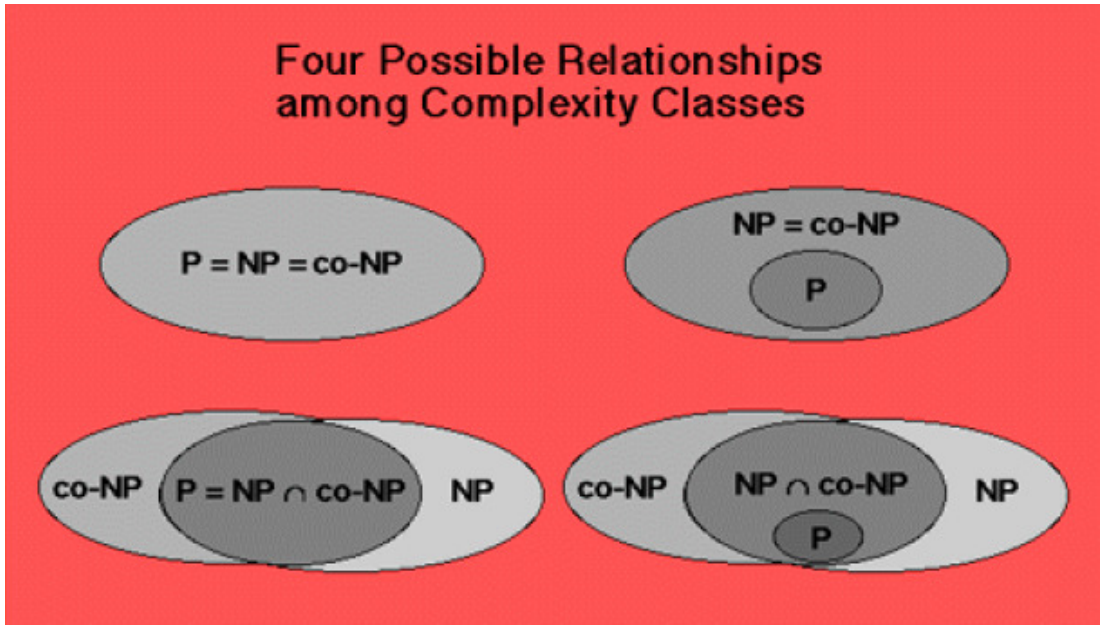
Section 6

Concluding Remarks

Ladies and Gentlemen, Boys and Girls

We have just ended the third part of the course:

Introduction to Computational Complexity (no more).



And with it, naturally, we've ended the whole course.

We hope you have enjoyed your flight, and look forward to meeting you in the future (but not in Moed B :-).

The dreaded exam

- All material covered in class and recitations, from all parts of course.
- Five “open” questions.
- You can bring and use two double sided A4 (normal size) pages marked with your name.
- See website for more info.
- Piece of cake.