CSC373: Algorithm Design, Analysis and Complexity
Fall 2017

Allan Borodin

October 25, 2017

Week 7 : Annoucements

- Term test 1 has been graded and hopefully Assignment 1 will be available this week on Markus.
- Term test 1, Question 1 wording issue. We are resolving this by gradding the term test out of 30 (instead of 45). Students were able to obtain bonus marks for partial or good answers (and even for "I do not know " answers).
- **Next assignment:** The questions for Assignment 2 have been posted. Due date: Wednesday, November 15 at 10AM. We had to delay the due date because of reading week. Part of todays lecture will be devoted to clarifying any questions about the assignment. Solutions will be posted by 11AM on November 15.
- The second term test is scheduled for Thursday, Nov 16 at 5PM.
- Some comments regarding questions posed on Piazza. For example, here is a question posed on Piazza: "What exactly is an encoding?"

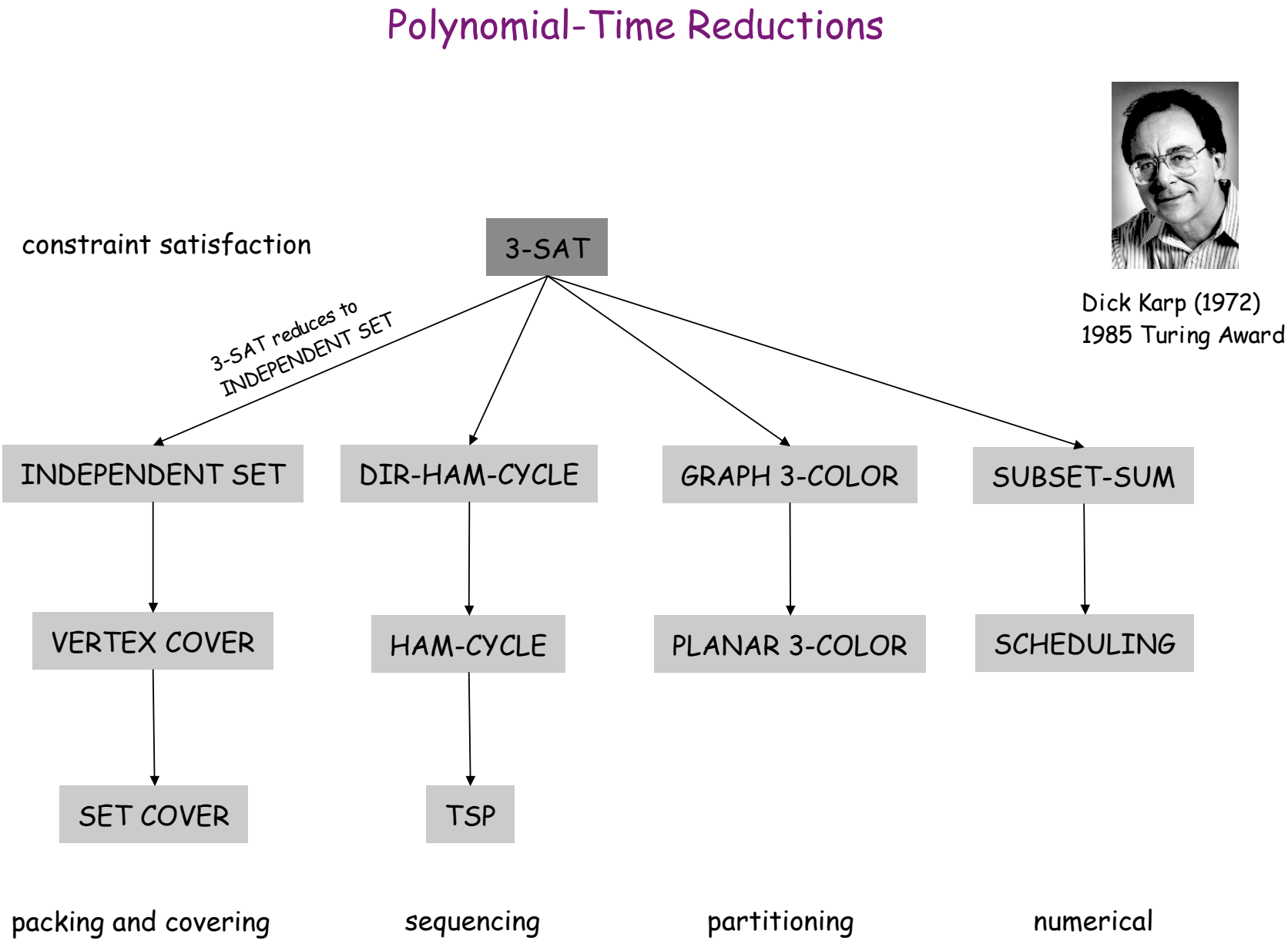
More announcements

- We are merging Greg's and Sasa's tutorial sections into one tutorial so as to save TA hours (given the low attendance). Both Greg's section in BA2139 and Sasa's sectionn in BA2145 will now meet in BA2145.
- Answer to question posed in last class: If $P \neq NP$, then there are languages in $L \in NP \setminus P$ such that L is not NP-complete. I was pretty sure of this being the result (and that there would be a rich substructure between P and NP) but I forgot the reference. See Richard Ladners' 1975 paper "On the Structure of Polynomial Time Reducibility".

Todays agenda

- Reviewing Karp tree of reductions
- Easy transformations: Independent Set $\leq_p$ Vertex Cover and Vertex Cover $\leq_p$ Set Cover
- Review transformation of 3SAT to Subset Sum
- Some comments on the "art" of provng *NP*-completeness.
- A not so easy transformation: 3SAT $\leq_p$ 3-Color
- Turing machines and proving that 3SAT is *NP*-complete.

A tree of reductions/transformations



Dick Karp (1972)
1985 Turing Award

Independent Set $\leq_p$ Vertex Cover

Let $G = (V, E)$ be a graph.

Definitions:

1. A subset $S \subseteq V$ is an independent set in G if for all $u, v \in S, (u, v) \notin E$.
2. A subset $S \subseteq V$ is a vertex cover if for all $e = (u, v) \in E$, at least one of u, v is in S.
3. Independent Set $= \{(G, k) : G$ has independent set of size at least $k\}$
4. Vertex Cover $= \{(G, \ell) : G$ has a vertex cover of size at most $\ell\}$

Easy claim: S is an independent set in G iff $V \setminus S$ is a vertex cover in G.

Therefore, the required transformation is (G, k) is transformed to $(G, |V| - k)$.

Vertex Cover $\leq_p$ Set Cover

Definition:
Let $U = \{u_1, u_2, \ldots, u_n\}$ be a universe of elements of size n. Let $\mathcal{C} = \{S_1, S_2, \ldots, S_m\}$ with $S_i \subseteq U$ be a collection of subsets of U.

A subcollection $\mathcal{C}' \subseteq \mathcal{C}$ is a set cover if for all $u_j \in U$, there is at least one $S_j \in \mathcal{C}' : u_i \in S_j$.

Set Cover $= \{\mathcal{C}$ has a set cover of size at most $\ell\}$

We can transform Vertex Cover into an instance (i.e. a special case) of Set Cover by letting the sets be vertices whose elements are the adjacent edges. That is $S_i = \{e \in E :$ vertex v_i is one of the end points of edge $e\}$

Vertex Cover and Set Cover are well known optimization problems where in the weighted versions of these problems the vertcies (resp. the sets) have weights and the goal is to minimize the total weight of the covering.

3SAT reduces to Subset Sum

Subset Sum $= \{(a_1, \ldots, a_n, t) : \exists S \subseteq \{1, \ldots, n\} \sum_{i \in S} = t\}$
This is a somewhat more difficult transformation.

Theorem

3SAT $\leq_p$ Subset Sum

- Given an instance F of 3SAT, we construct an instance of Subset Sum that has solution iff F is satisfiable.
- In the following array (next slide), rows represent integers represented in decimal. For each propositional variable we have a column specifying that each variable has just one truth assignment (i.e., *true* $= 1$) and for each clause we have a column saying that the clause is satisfiable. The "dummy rows" makes it possible to sum each column to 4 if and only if there is at least one literal set to true. Note that the decimal representation nsures that addition in each column will not carry over to the next column.

3SAT reduces to Subset Sum continued

The figure illustrates how a specific 3CNF formula is transformed into a set of integers and a target (bottow row).

$$C_1 = \bar{x} \vee y \vee z$$
$$C_2 = x \vee \bar{y} \vee z$$
$$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$$

	x	y	z	C_1	C_2	C_3
x	1	0	0	0	1	0
¬ x	1	0	0	1	0	1
y	0	1	0	1	0	0
¬ y	0	1	0	0	1	1
z	0	0	1	1	1	0
¬ z	0	0	1	0	0	1
dummies to get clause columns to sum to 4	0	0	0	1	0	0
	0	0	0	2	0	0
	0	0	0	0	1	0
	0	0	0	0	2	0
	0	0	0	0	0	1
	0	0	0	0	0	2
W	1	1	1	4	4	4

Wnat has to be proven?

Some consequences of Subset Sum completeness

- SubsetSum $\leq_p$ Knapsack where

  $$\text{Knapsack} = \left\{(\langle s_1, v_1\rangle, \ldots, \langle s_n, v_n\rangle; B, V) \mid \exists S : \sum_{i\in S} s_i \leq B, \sum_{i\in S} v_i \geq V\right\}$$

- SubsetSum $\leq_p$ Partition where

  $$\text{Partition} = \{a_1, \ldots, a_n \mid \exists S, \sum_{i\in S} a_i = \tfrac{1}{2}\sum_{i=1}^{n} a_i\}.$$

As suggested in Karp's trree of reductions, many scheduling problems can be shown to be NP-hard by a reduction from Subset Sum and, more specifically, sometimes from Partition.

Reviewing how to show some L is NP complete.

- We must show $L \in NP$. To do so, we provide a polynomial time verification predicate $R(x, y)$ and polynomial length certificate y for every $x \in L$; that is, $L = \{x | \exists y, R(x, y) \text{ and } |y| \leq q(|x|)\}$.
- We must show that L is NP hard (say with respect to polynomial time tranformations); that is, for some known NP complete L', there is a polynomial time transducer function h such that $x \in L'$ iff $h(x) \in L$. This then establishes that $L' \leq_p L$.
- Warning The reduction/transformation $L' \leq_p L$ must be in the correct direction and h must be defined for every input x; that is, one must also show that if $h(x) \in L$ then $x \in L'$ as well as showing that if $x \in L'$ then $h(x) \in L$.

Some transformations are easy, some not

- Tranformations are (as we have been arguing) algorithms computing a function and hence like any algorithmic problem, sometimes there are easy solutions and sometimes not.
- In showing NP-completeness it certainly helps to choose the right known NP-complete problem to use for the transformation.
- In the Karp tree, there are some transformations that are particularly easy such as :
  - IndependentSet $\leq_p$ VertexCover
  - VertexCover $\leq_p$ SetCover
- A transforrmation of moderate difficulty is 3SAT $\leq_p$ 3-COLOR
- I am using Kevin Wayne's slides to illustrate the transformation. See slides for "Poly-time reductions" in

  `http://www.cs.princeton.edu/courses/archive/spring05/cos423/lectures.php`

3CNF $\leq_p$ 3-COLOR: Outline of Transformation

Claim. 3-SAT $\leq_P$ 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable.

Construction.

i. For each literal, create a node.
ii. Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
iii. Connect each literal to its negation.
iv. For each clause, add gadget of 6 nodes and 13 edges.

↑
to be described next

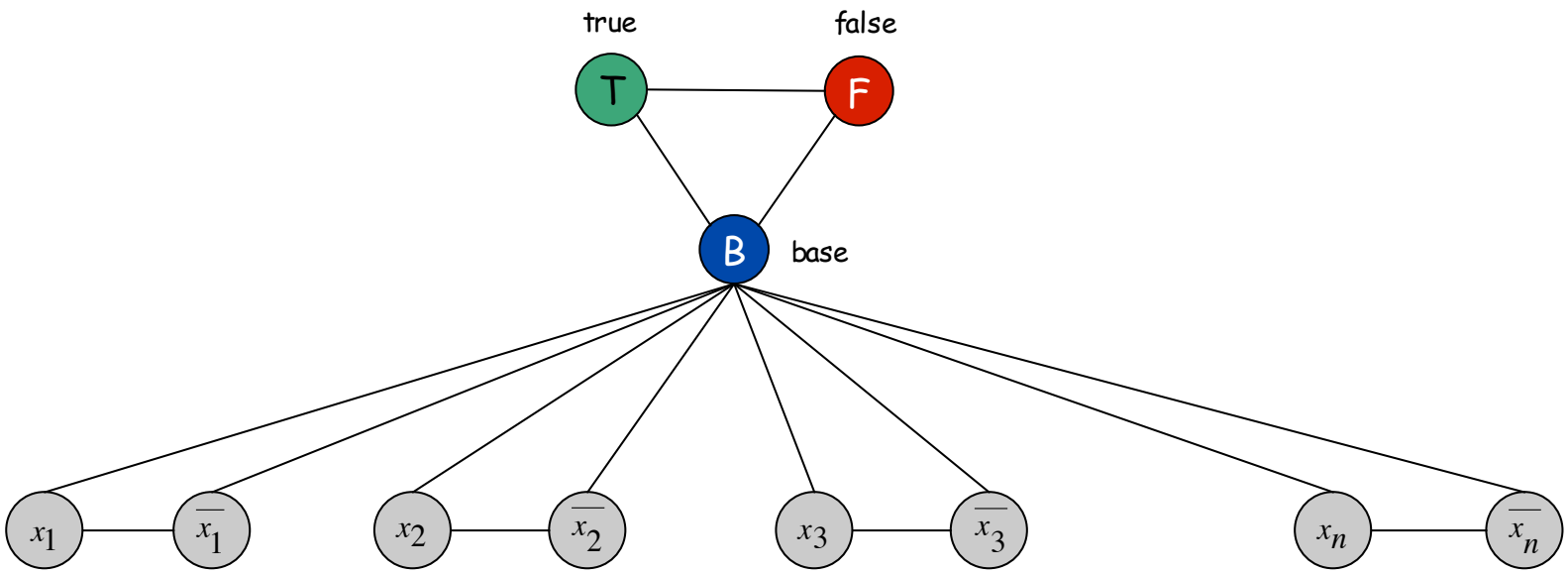
- If ϕ is a 3CNF formula in n variables and m clauses, then $h(\phi) = G_\phi$ will have $2n + 6m + 3$ nodes and $3n + 13m + 3$ edges.

3CNF $\leq_p$ 3-COLOR: Consistent literals

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.

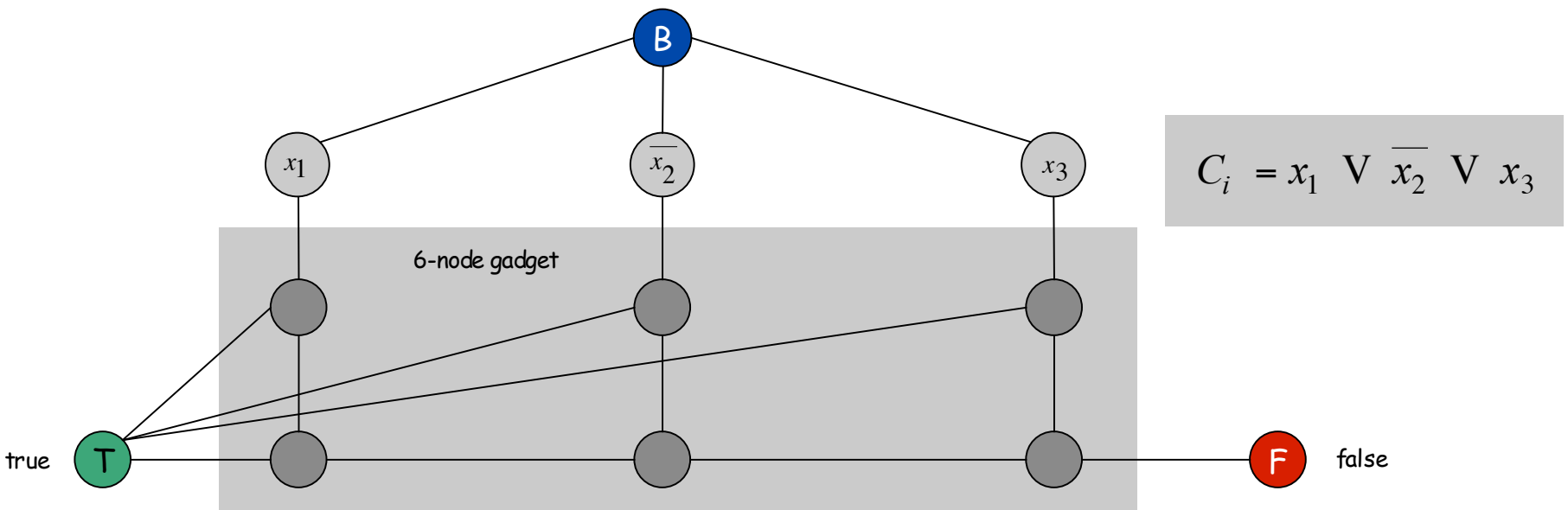


3CNF $\leq_p$ 3-COLOR: The clause gadget

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.

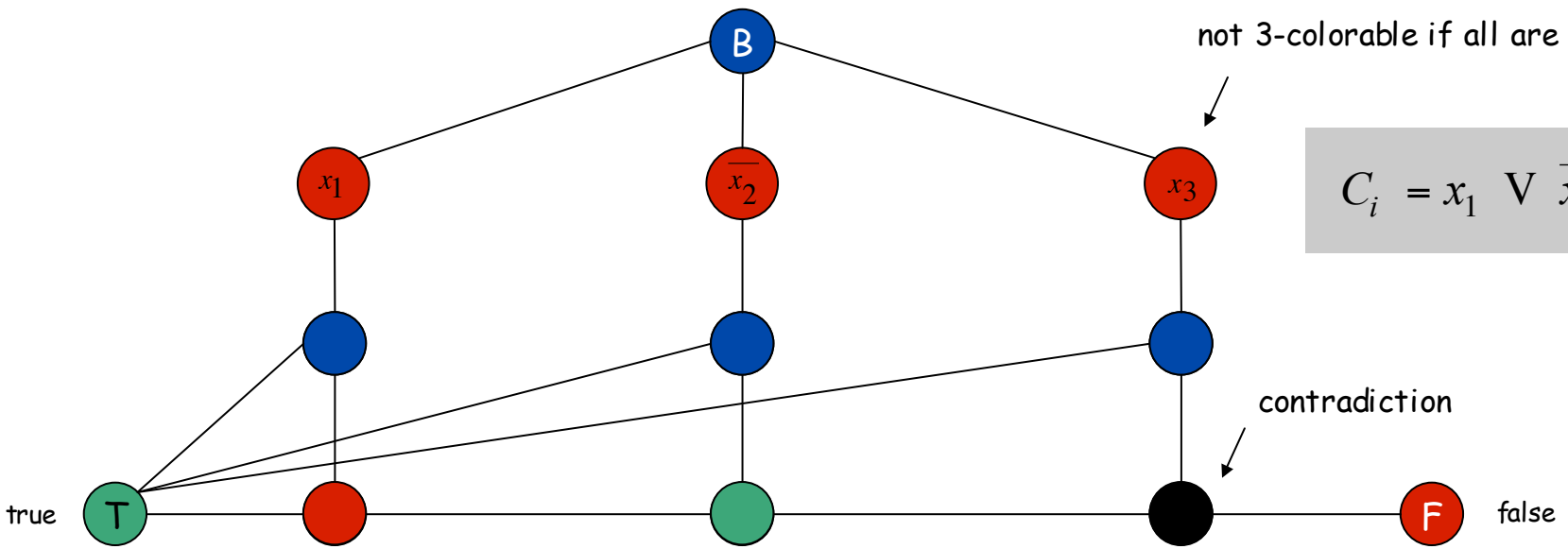


G_ϕ is 3-colourable $\Rightarrow$ ϕ satisfiable

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.

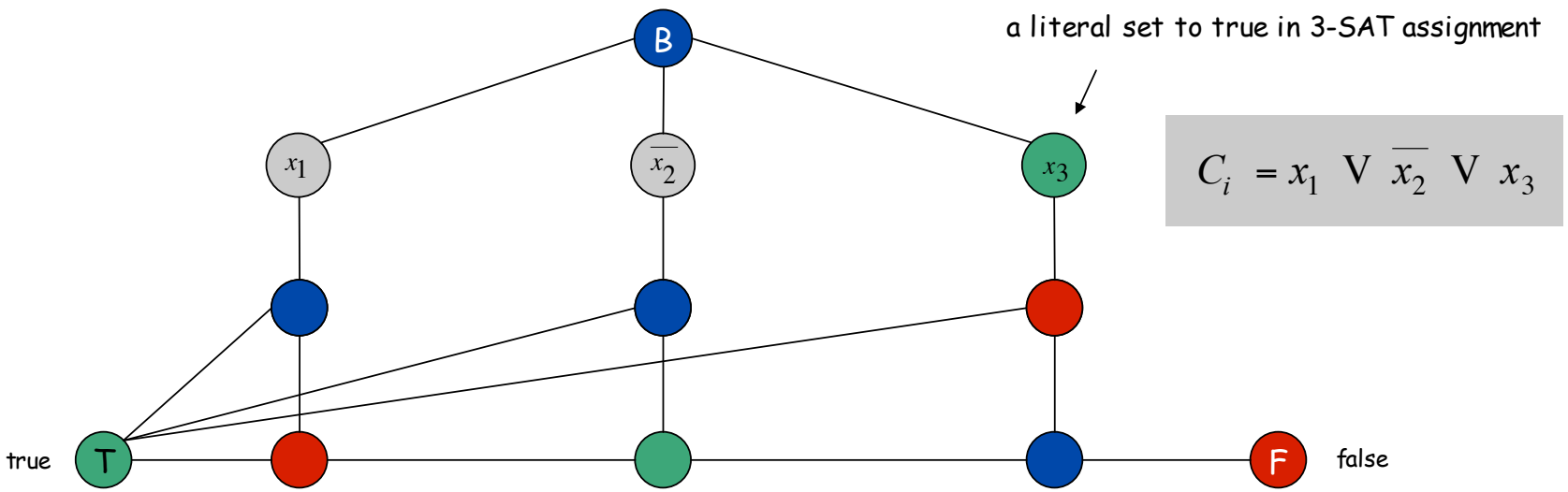


ϕ satisfiable $\Rightarrow$ G_ϕ is 3-colourable

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Leftarrow$ Suppose 3-SAT formula Φ is satisfiable.

- Color all true literals T.
- Color node below green node F, and node below that B.
- Color remaining middle row nodes B.
- Color remaining bottom nodes T or F as forced. ▪



- Note we are choosing precisely one green node in each clause to force colouring.

Brief introduction to Turing machines

- We are using the classical one tape TM. This is the simplest variant to formalize which will enable the proof for the NP completeness of SAT. In the proof, we are assuming (without loss of generality) that all time bounds $T(n)$ are computable in polynomial time.
- Claim Any reasonable (classical) computing model algorithm running in time $T(n)$, can be simulated by a TM in time $T(n)^k$ for some k. Hence we can use the TM model in the definition of P and NP.
- Since we are only considering decision problems we will view TMs that are defined for decision problems and hence do not need an output other than a reject and accept state.
- Following standard notation, formally, a specific TM is defined by a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$
- We briefly explain (using the board) the model and notation. Note that Q, Σ, Γ are all finite sets.

Satisfiability is NP-Complete

- SAT = { < ϕ > | ϕ is a satisfiable Boolean formula }
- Theorem: SAT is NP-complete.
- Lemma 1: SAT $\in$ NP.
- Lemma 2: SAT is NP-hard.
- Proof of Lemma 1:
  - Recall: L $\in$ NP if and only if ($\exists$ V, poly-time verifier) ($\exists$ p, poly)
    x $\in$ L iff ($\exists$ c, |c| $\leq$ p(|x|)) [V(x, c) accepts]
  - So, to show SAT $\in$ NP, it's enough to show ($\exists$ V) ($\exists$ p)
    ϕ $\in$ SAT iff ($\exists$ c, |c| $\leq$ p(|x|)) [V(ϕ, c) accepts]
  - We know: ϕ $\in$ SAT iff there is an assignment to the variables such that ϕ with this assignment evaluates to 1.
  - So, let certificate c be the assignment.
  - Let verifier V take a formula ϕ and an assignment c and accept exactly if ϕ with c evaluates to true.
  - Evaluate ϕ bottom-up, takes poly time.

Satisfiability is NP-Complete

- Lemma 2: SAT is NP-hard.
- Proof of Lemma 2:
  - Need to show that, for any $A \in NP$, $A \leq_p SAT$.
  - Fix $A \in NP$.
  - Construct a poly-time f such that

    $w \in A$ if and only if $f(w) \in SAT$.

    A formula, write it as ϕ_w.
  - By definition, since $A \in NP$, there is a nondeterministic TM M that decides A in polynomial time.
  - Fix polynomial p such that M on input w always halts, on all branches, in time $\leq p(|w|)$; assume $p(|w|) \geq |w|$.
  - $w \in A$ if and only if there is an accepting computation history (CH) of M on w.

Satisfiability is NP-Complete

- Lemma 2: SAT is NP-hard.
- Proof, cont'd:
  - Need $w \in A$ if and only if $f(w)$ $(= \phi_w) \in$ SAT.
  - $w \in A$ if and only if there is an accepting CH of M on w.
  - So we must construct formula ϕ_w to be satisfiable iff there is an accepting CH of M on w.
  - Recall definitions of computation history and accepting computation history from Post Correspondence Problem: # C_0 # C_1 # C_2 …
    - Configurations include tape contents, state, head position.
  - We construct ϕ_w to describe an accepting CH.
  - Let M = (Q, Σ, Γ, δ, q_0, q_{acc}, q_{rej}) as usual.
  - Instead of lining up configs in a row as before, arrange in (p(|w|) + 1) row $\times$ (p(|w|) + 3) column matrix:

Proof that SAT is NP-hard

- ϕ_w will be satisfiable iff there is an accepting CH of M on w.
- Let M = (Q, Σ, Γ, δ, q_0, q_{acc}, q_{rej}).
- Arrange configs in (p(|w|) + 1) $\times$ (p(|w|) + 3) matrix:

#	q_0	w_1	w_2	w_3	…	w_n	--	--	…	--	#
#	…										#
#	…										#
⋮											⋮
#	…										#

- Successive configs, ending with accepting config.
- Assume WLOG that each computation takes exactly p(|w|) steps, so we use p(|w|) + 1 rows.
- p(|w|) + 3 columns: p(|w|) for the interesting portion of the tape, one for head and state, two for endmarkers.

Proof that SAT is NP-hard

- ϕ_w is satisfiable iff there is an accepting CH of M on w.
- Entries in the matrix are represented by Boolean variables:
  - Define $C = Q \cup \Gamma \cup \{ \# \}$, alphabet of possible matrix entries.
  - Variable $x_{i,j,c}$ represents "the entry in position (i, j) is c".
- Define ϕ_w as a formula over these $x_{i,j,c}$ variables, satisfiable if and only if there is an accepting computation history for w (in matrix form).
- Moreover, an assignment of values to the $x_{i,j,c}$ variables that satisfies ϕ_w will correspond to an encoding of an accepting computation.
- Specifically, $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, where:
  - ϕ_{cell} : There is exactly one value in each matrix location.
  - ϕ_{start} : The first row represents the starting configuration.
  - ϕ_{accept} : The last row is an accepting configuration.
  - ϕ_{move} : Successive rows represent allowable moves of M.

ϕ_{cell}

- For each position (i,j), write the conjunction of two formulas:

  $\bigvee_{c \in C} x_{i,j,c}$: Some value appears in position (i,j).

  $\bigwedge_{c, d \in C, c \neq d} (\neg x_{i,j,c} \vee \neg x_{i,j,d})$: Position (i,j) doesn't contain two values.

- ϕ_{cell}: Conjoin formulas for all positions (i,j).

- Easy to construct the entire formula ϕ_{cell} given w input.
- Construct it in polynomial time.
- Sanity check: Length of formula is polynomial in |w|:
  - $O((p(|w|)^2)$ subformulas, one for each (i,j).
  - Length of each subformula depends on C, $O(|C|^2)$.

ϕ_{start}

- The right symbols appear in the first row:

$$\# \quad q_0 \quad w_1 \quad w_2 \quad w_3 \quad \dots \quad w_n \quad -- \quad -- \quad \dots \quad -- \quad \#$$

$$\phi_{start}: \; x_{1,1,\#} \wedge x_{1,2,q0} \wedge x_{1,3,w1} \wedge x_{1,4,w2} \wedge \dots$$
$$\wedge \; x_{1,n+2,wn} \wedge x_{1,n+3,--} \wedge \dots$$
$$\wedge \; x_{1,p(n)+2,--} \wedge x_{1,p(n)+3,\#}$$

ϕ_{accept}

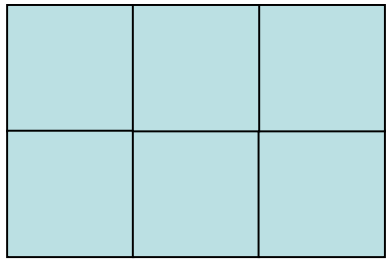
- For each j, $2 \leq j \leq p(|w|) + 2$, write the formula:

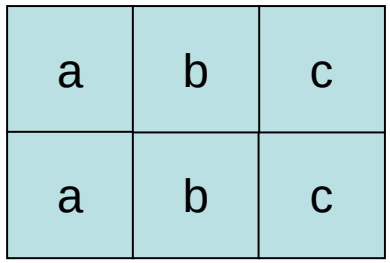
$$x_{p(|w|)+1,j,qacc}$$

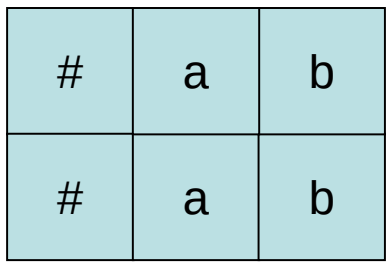
- q_{acc} appears in position j of the last row.
- ϕ_{accept}: Take disjunction (or) of all formulas for all j.
- That is, q_{acc} appears in some position of the last row.

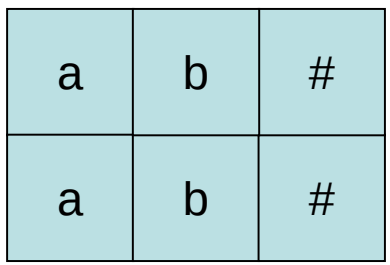
ϕ_{move}

- As for PCP, correct moves depend on correct changes to local portions of configurations.
- It's enough to consider 2×3 rectangles:
- If every 2×3 rectangle is "good", i.e., consistent with the transitions, then the entire matrix represents an accepting CH.
- For each position (i,j), $1 \leq i \leq p(|w|)$, $1 \leq j \leq p(|w|)+1$, write a formula saying that the rectangle with upper left at (i,j) is "good".
- Then conjoin all of these, $O(p(|w|)^2)$ clauses.
- Good tiles for (i,j), for a, b, c in Γ:









ϕ_{move}

- Other good tiles are defined in terms of the nondeterministic transition function δ.
- E.g., if $\delta(q_1, a)$ includes tuple (q_2, b, L), then the following are good:
  - Represents the move directly; for any c:
  - Head moves left out of the rectangle; for any c, d:
  - Head is just to the left of the rectangle; for any c, d:
  - Head at right; for any c, d, e:
  - And more, for #, etc.
- Analogously if $\delta(q_1, a)$ includes (q_2, b, R).
- Since M is nondeterministic, $\delta(q_1, a)$ may contain several moves, so include all the tiles.

c	q_1	a
q_2	c	b

q_1	a	c
d	b	c

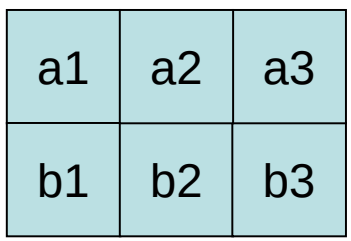
a	c	d
b	c	d

d	c	q_1
d	q_2	c

e	d	c
e	d	q_2

ϕ_{move}

- The good tiles give partial constraints on the computation.
- When taken together, they give enough constraints so that only a correct CH can satisfy them all.
- The part (conjunct) of ϕ_{move} for (i,j) should say that the rectangle with upper left at (i,j) is good:
- It is simply the disjunction (or), over all allowable tiles, of the subformula:



$$x_{i,j,a1} \wedge x_{i,j+1,a2} \wedge x_{i,j+2,a3} \wedge x_{i+1,j,b1} \wedge x_{i+1,j+1,b2} \wedge x_{i+1,j+2,b3}$$

- Thus, ϕ_{move} is the conjunction over all (i,j), of the disjunction over all good tiles, of the formula just above.

ϕ_{move}

- ϕ_{move} is the conjunction over all (i,j), of the disjunction over all good tiles, of the given six-term conjunctive formula.
- Q: How big is the formula ϕ_{move}?
- $O(p(|w|)^2)$ clauses, one for each (i,j) pair.
- Each clause is only constant length, O(1).
  - Because machine M yields only a constant number of good tiles.
  - And there are only 6 terms for each tile.
- Thus, length of ϕ_{move} is polynomial in |w|.
- $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, length also poly in |w|.

ϕ_{move}

- $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, length poly in |w|.
- More importantly, can produce ϕ_w from w in time that is polynomial in |w|.
- $w \in A$ if and only if M has an accepting CH for w if and only if ϕ_w is satisfiable.
- Thus, $A \leq_p SAT$.
- Since A was any language in NP, this proves that SAT is NP-hard.
- Since SAT is in NP and is NP-hard, SAT is NP-complete.

Clay Math Institute Millenium Problems: $1,000,000 each

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. $P = NP?$
5. Poincaré Conjecture **(Solved)**[1]
6. Riemann Hypothesis
7. Yang-Mills Theory

[1] Solved by Grigori Perelman 2003: Prize unclaimed

How important is the P vs NP question

Lance Fortnow has an article on P and NP in the September 2009 Communications of the ACM, in which he says

> *"The P versus NP problem has gone from an interesting problem related to logic to perhaps the most fundamental and important mathematical question of our time, whose importance only grows as computers become more powerful and widespread."*

Claim: It is worth well over the $1,000,000

How important is the P vs NP question

Lance Fortnow has an article on P and NP in the September 2009 Communications of the ACM, in which he says

> *"The P versus NP problem has gone from an interesting problem related to logic to perhaps the most fundamental and important mathematical question of our time, whose importance only grows as computers become more powerful and widespread."*

Claim: It is worth well over the $1,000,000

Other long standing and fundamental open problems in complexity theory

For any language L (i.e., decision problem), we define $\bar{L} = \{x : x \notin L\}$.
Similarly, for any class $\mathcal{C}$ of languages the class $co - \mathcal{C} = \{\bar{L} : L \in \mathcal{C}\}$.
An open problem related to the P vs NP issue is the whether or not $NP = co - NP$.

Conjecture: $\bar{SAT} \notin NP$

We have the following fact: $NP = co - NP$ iff $\bar{L} \in NP$ for some (any) NP-complete language L.

As mentioned before It is widely believed that $NP \neq co - NP$
How would you verify that a CNF formula F is *not* satisfiable?

Why we believe integer factoring is not NP-hard

As mentioned, some cryptographic schemes (i.e. for decoding a cryptographically encoded message) rely on the assumption that factoring can *not* be done efficiently (even in some average case sense).

However, it is believed that factoring is not NP-hard. To make this comment precise, consider the following decision problem which can be used to factor an integer: $FACTOR = \{(x, m) | x \text{ has a proper factor} \leq m\}$. It should be clear that $FACTOR$ is in NP.

Claim: $co - FACTOR = \{(x, m) | x \notin FACTOR\}$ is in NP. Hence we do not believe $FACTOR$ can be NP complete.

To see this, we first note that that $PRIME = \{x | x \text{ is a prime number}\}$ is now known to be in P. (It is sufficient to know that $PRIME$ is in NP which was known since the early 1970's.) Then a certificate y for $(x.m)$ being in $co - FACTOR$ is the prime factorization $y = (p_1, e_1, p_2, e_2, \ldots p_r, e_r)$ of x which can be verified by checking that each p_i is a prime and that $x = (p_1^{e_1} \cdot p_2^{e_2} \ldots \cdot p_r^{e_r})$

More complexity theory issues

After sequential time, the most studied complexity measure is *space*. If one maintains the input so that it is "read-only", it is meaningful to consisder sublinear (e.g. $\log n$) space bounds.

Analogous to deteriministic (resp. non-deterministic) time bounded classes $DTIME(T)$ (resp. $NTIME(T)$ we can define space bounded classes.

n contrast to what is widely believed about sequential time bounded classes, we have the following two Theorems for "reasonable space bounds $S(n) \geq \log n$:

- $NSPACE(S) \subseteq DSPACE(S^2)$
- $NSPACE(S) = co - NSPACE(S)$

It is an open problem as to whether or not $NSPACE(S) = DSPACE(S)$.

It is also clear (for the models of computation we consider) that both $DSPACE(S)$ and $NSPACE(S)$ are contained in $\cup_c DTIME(c^{S(n)})$. Why? Hence $DSPACE(\log n) \subseteq P$.

Conjecture: $DSPACE \neq P$

And more complexity issues

Later in the course we will devote a lecture to randomized algorithms. For a number of computational problems, the use of randmization will provide the best known time bounds. And in some computational settings (e,g, cryptography, sublinear time algorithms), randomization is **necessary**.

With regard to complexity theory, analgous to the class P of polynomial time decisions, we have three different classes of decisions problems solvable in *randomized polynomial time*. These classes are ZPP (expected polynomial time, no error), RP (polynomial time, one sided error) and BPP (polynomial time, two sided error).

While it is clear that $RP \cap co - RP = ZPP \subseteq RP \subseteq BPP$, it is not known if any of these inclusions are proper.

Moreover, it is not known if $RP = P$ or $BPP = P$. Lately, some prominent complexity theorists conjecture tht $BPP = P$ in which case, ignoring polynomial factors, randomization can be avoided.

One final complexity comment

Although the *P* vs *NP* issue is not resolved, there are "natural" problems which *provably* require (say) exponential time. (In fact, there are some problems which require "much more" than exponential time.

Without defining things carefully, one such problem is the decision problem for the first order theory of the reals. That is, we want to decide when a fully quantified first order arithmetic formual is true or false when the variables are real numbers.

For example, the statement $\forall x > 0, \exists y[y^2 = x]$ is true for real numbers but false for say the integers or rational numbers. Similarly, $\exists y[y^2 = -1]$ is true for the complex numbers but false for the real numbers.
While it is known that this decision problem is computable (in time $\approx 2^{2^n}$) where n is the length of the 1st order statment, it is also proven that the decision problem cannot be done in time say 2^n.