Computational Models Spring 2011, Lecture 12

- The classes P and NP (reminder)
- Poly-Time Reductions
- NP completeness
- SAT is NP Complete

- Sipser, Chapter 7, Sections 7.3, 7.4,

Time Classes Definition, Again

Let

$$t : \mathcal{N} \longrightarrow \mathcal{N}$$

be a function.

Definition:

$$\text{DTIME}(t(n)) = \{L \mid L \text{ is a language, decided by an } O(t(n))\text{-time TM}\}$$

Non-Deterministic Time (reminder)

Let N be a non-deterministic TM, and let

$$f : \mathcal{N} \longrightarrow \mathcal{N}$$

We say that N runs in time $f(n)$ if

- For every input x of length n,
- the maximum number of steps that N uses,
- on any branch of its computation tree on x,
- is at most $f(n)$.

NTime Classes Definition (reminder)

Let

$$f : \mathcal{N} \longrightarrow \mathcal{N}$$

be a function.

Definition:

$$\text{NTIME}(f(n)) = \{L \mid L \text{ is a language, decided by an } O(f(n))\text{-time NTM}\}$$

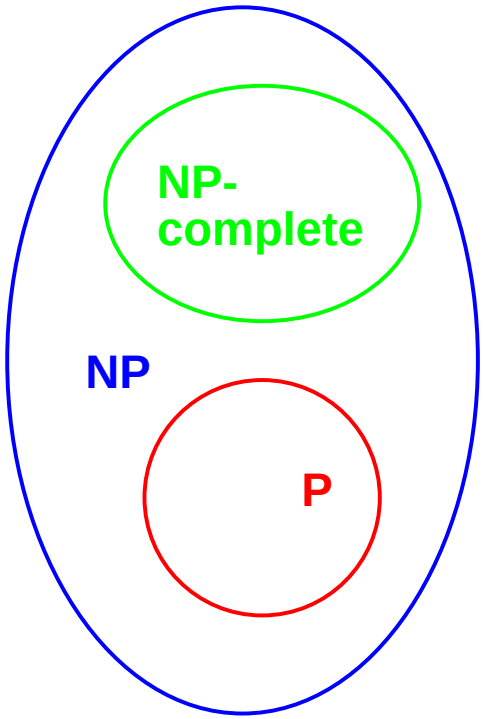
The Class NP (reminder)

Definition: NP is the set of languages decidable in polynomial time on non-deterministic TMs.

$$NP = \bigcup_{c \geq 0} \text{NTIME}(n^c)$$

- The class NP is
- Invariant for all TMs with any number of tapes.
- Insensitive to choice of reasonable non-deterministic computational model.
- Rougly corresponds to problems whose positive solutions cannot be efficiently generated ($\Rightarrow$ intractable), but can be efficiently checked.

NP Completeness



The class of NP-complete languages are

- "hardest" languages in $\mathcal{NP}$
- "least likely" to be in $\mathcal{P}$
- If any NP-complete $\mathcal{A} \in P$, then $\mathcal{NP} = P$.

Cook–Levin (1971-1973)

Theorem: There is a language $\mathcal{S} \in NP$ such that $\mathcal{S} \in P$ if and only if $\mathcal{P} = NP$.

This theorem establishes the class of NP-complete languages. Such languages, "carry on their backs" the burden of all of $\mathcal{N}P$.

Poly-Time Computable Functions

Definition: A function

$$f : \Sigma^* \longrightarrow \Sigma^*$$

is polynomial-time computable if there is a poly-time deterministic TM that

- starts with input w, and
- halts with $f(w)$ on tape.

Poly-Time Reducibility

Definition: We say that a language $\mathcal{A}$ is polynomial time mapping reducible to $\mathcal{B}$, written

$$A \leq_P B,$$

if there is a poly-time computable function

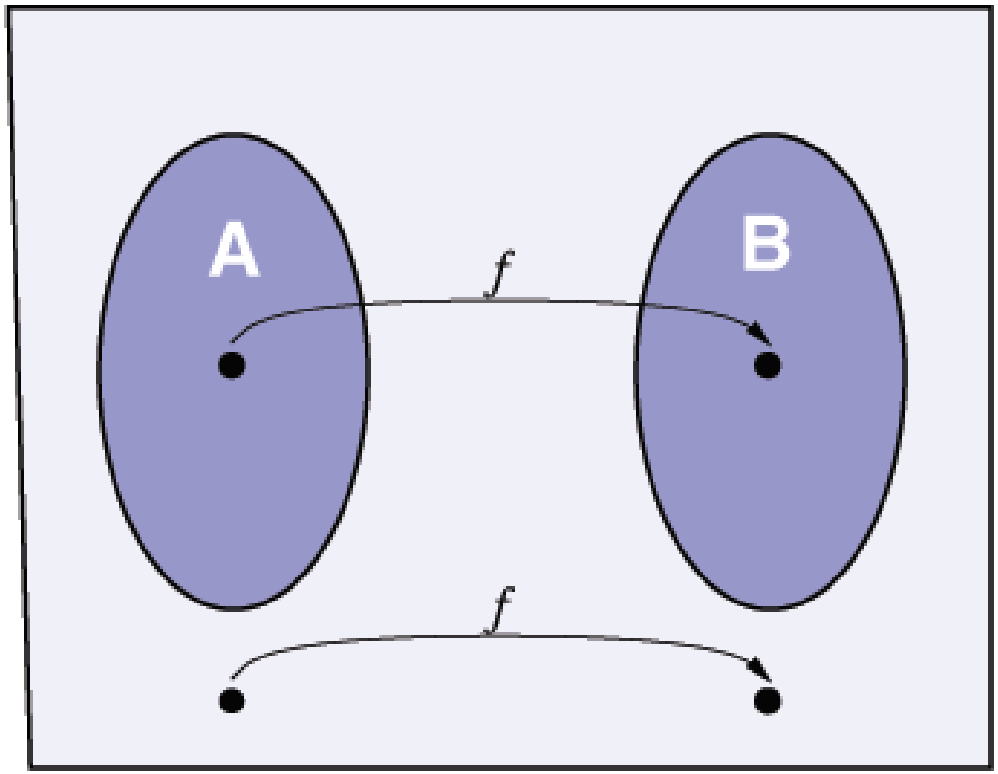
$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

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

Poly-Time Reductions



Convert questions about membership in $\mathcal{A}$ to membership in $\mathcal{B}$, and do it efficiently.

Poly-Time Reductions

Theorem: If $\mathcal{A} \leq_P B$ and $\mathcal{B} \in P$ then $\mathcal{A} \in P$.

Proof: Let

- f the reduction from $\mathcal{A}$ to $\mathcal{B}$, computed by TM M_f.
- On input x of length n, M_f takes at most $c_1 n^{a_1}$ steps.
- M be the poly-time decider for $\mathcal{B}$.
- On input y of length m, M takes at most $c_2 m^{a_2}$ steps.

Poly-Time Reductions

Define $\mathcal{N}$: on input x

1. compute $f(x)$
2. run $\mathcal{M}$ on input $f(x)$ and output whatever $\mathcal{M}$ outputs.

Analysis:

- On input x of length n, computing $y = f(x)$ takes at most $c_1 n^{a_1}$ steps.
- On input y of length $m = c_1 n^{a_1}$, M takes at most $c_2 m^{a_2} = c_2 (c_1 n^{a_1})^{a_2} = (c_2 c_1^{a_2}) n^{a_1 \cdot a_2}$ steps.
- Summing both stages, we got a polynomial in n.
- Correctness is clear, so $\mathcal{A} \in P$. ♣

Satisfiability

- A boolean variable assumes values
  - true (written 1), and false (written 0).
- Boolean operations:
  - and: $\wedge$
  - or: $\vee$
  - not: $\neg$
- Examples:

$$\begin{aligned} 0 \wedge 1 &= 0 \\ 0 \vee 1 &= 1 \\ \overline{0} &= 1 \end{aligned}$$

Satisfiability

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

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

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

Satisfiability

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

is satisfiable by

$$\begin{aligned} x &= 0 \\ y &= 1 \\ z &= 0 \end{aligned}$$

This assignment satisfies ϕ.

Satisfiability

Define

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

Satisfiability

It is useful to consider a special version:

- 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

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

Satisfiability

Definition: A Boolean formula is in 3CNF form if it is a CNF formula, and all clauses have three literals.

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

Define

$$\text{3SAT} = \{\langle \phi \rangle \mid \phi \text{ is satisfiable 3CNF formula}\}$$

Clearly, if ϕ is a satisfiable 3CNF formula, then for any satisfying assignment of ϕ, every clause must contain at least one literal assigned 1.

Reductions

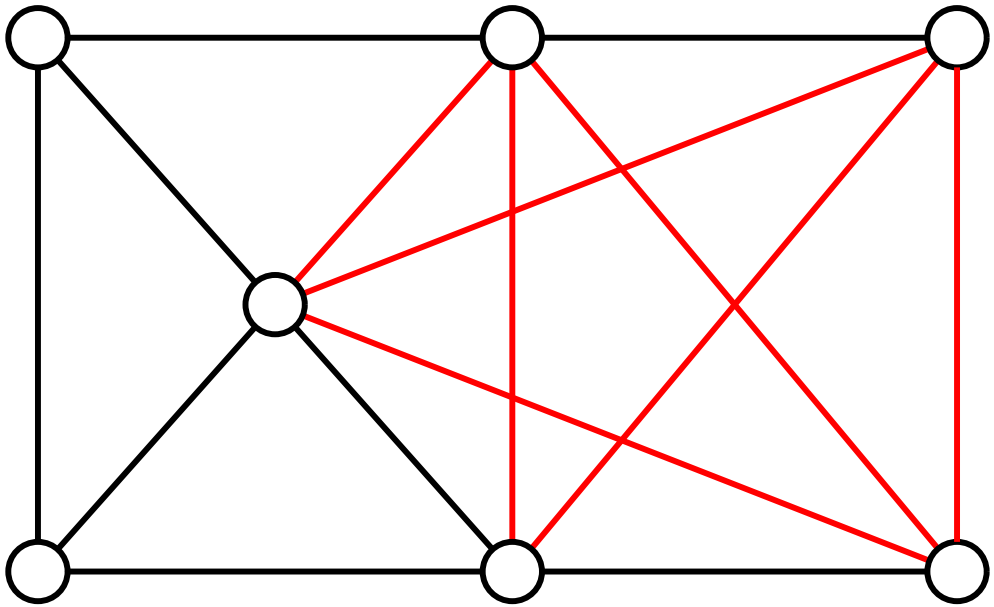
Claim: There is a poly time reduction from 3SAT to CLIQUE. In other words,

$$\text{3SAT} \leq_P \text{CLIQUE} .$$

We'll construct a poly time reduction f that maps 3CNF formulae ϕ to graphs and numbers, $\langle G, k \rangle$.

The function f will have the property that ϕ is satisfiable if and only if G has a clique of size k.

Examples: Clique



Reminder: A clique in a graph is a subgraph where every two nodes are connected by an edge.
A k-clique is a clique of size k. For example, the graph above has a 5-clique.

3SAT $\leq_P$ CLIQUE

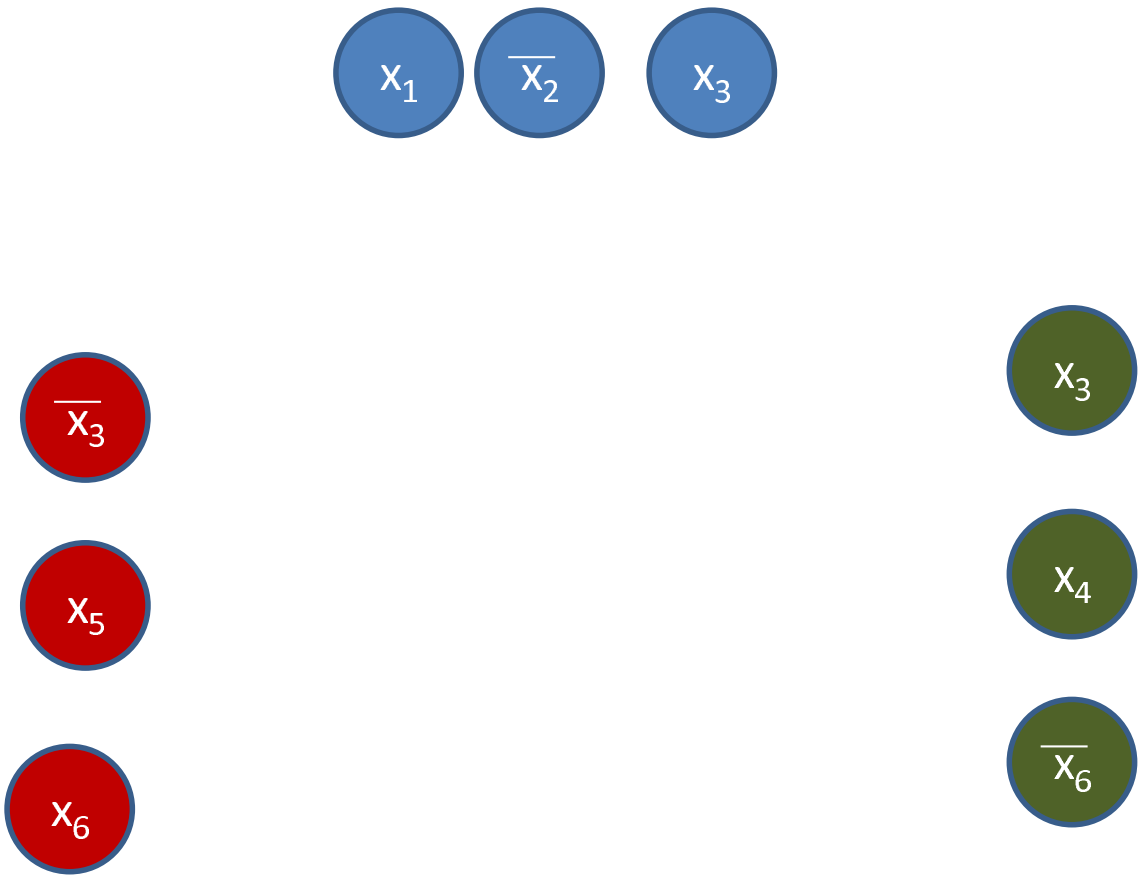
Let ϕ be a 3CNF formula with k clauses.

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

We define a graph $\mathcal{G}$ as follows:

- nodes in $\mathcal{G}$ 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.

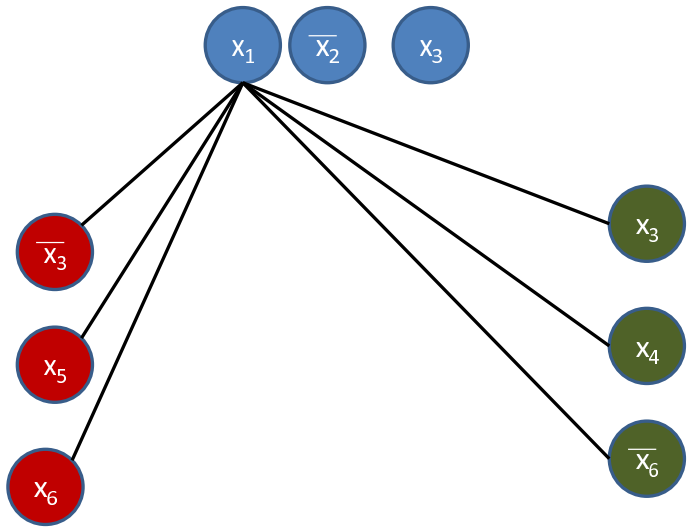
3SAT $\leq_P$ CLIQUE



3SAT vs. CLIQUE

Now add edges between all vertex pairs, except

- within same triple
- between contradictory literals



3SAT $\leq_P$ CLIQUE

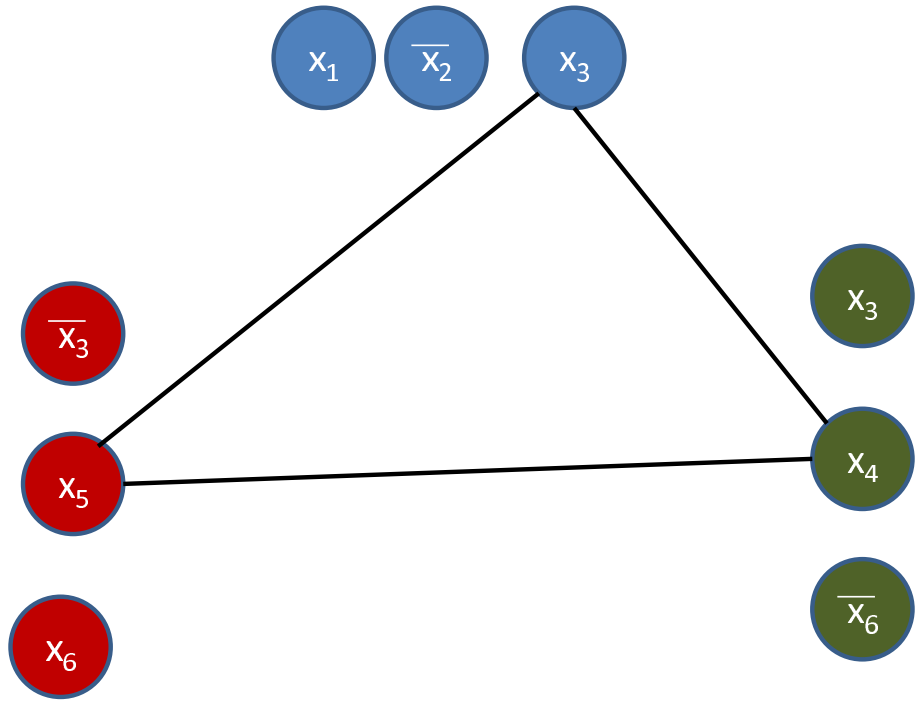
Claim: If ϕ is satisfiable, $\mathcal{G}$ has a k-clique.

Suppose ϕ is satisfiable.

- at least one literal is true in every clause
- in every tuple, select one true literal
- they can be joined by edges (why?)
- yielding a k-clique

3SAT $\leq_P$ CLIQUE

Claim: If ϕ is satisfiable, $\mathcal{G}$ has a k-clique.



3SAT $\leq_P$ CLIQUE

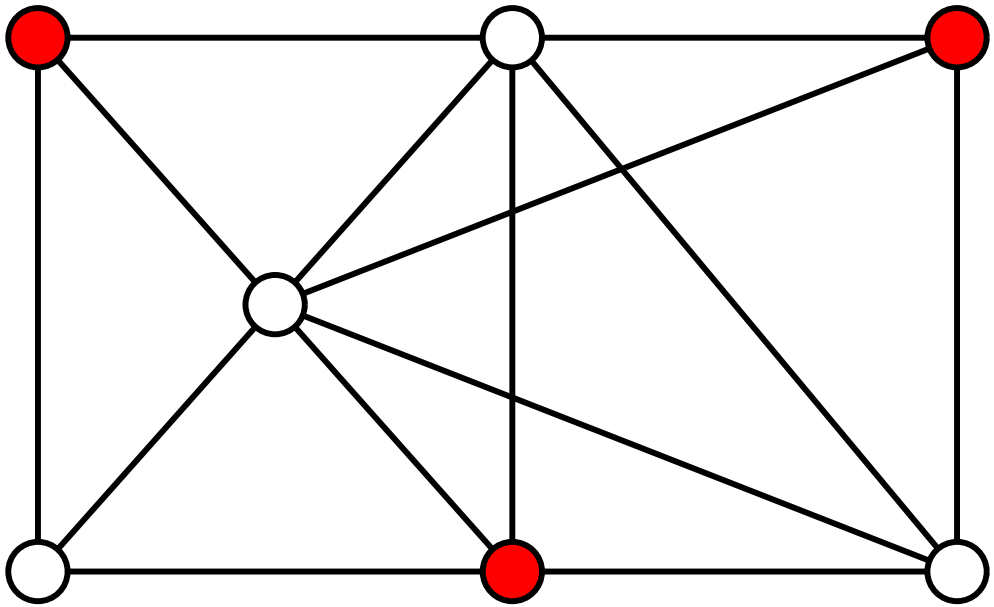
Claim: If $\mathcal{G}$ has a k-clique, ϕ is satisfiable.

- No two of the cliques nodes are in the same triple. (why?)
- Have k vertexes and k clauses, so
- each triple has exactly one clique node.
- Assign 1 to each node in clique
- no contradictions (why?)

3SAT $\leq_P$ CLIQUE

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

Independent Set



An independent in a graph is a set of vertexes, no two of which are linked by an edge.

The independent set problem asks whether there exists an independent set of size k.

Independent Set

Define

INDEPENDENT-SET $=$
$\{\langle G, k\rangle | G$ contains an independent set of size $k\}$

Claim: INDEPENDENT-SET is polynomial time reducible to CLIQUE,

$$\text{INDEPENDENT-SET} \leq_P \text{CLIQUE}$$

and vice-versa,

$$\text{CLIQUE} \leq_P \text{INDEPENDENT-SET}$$

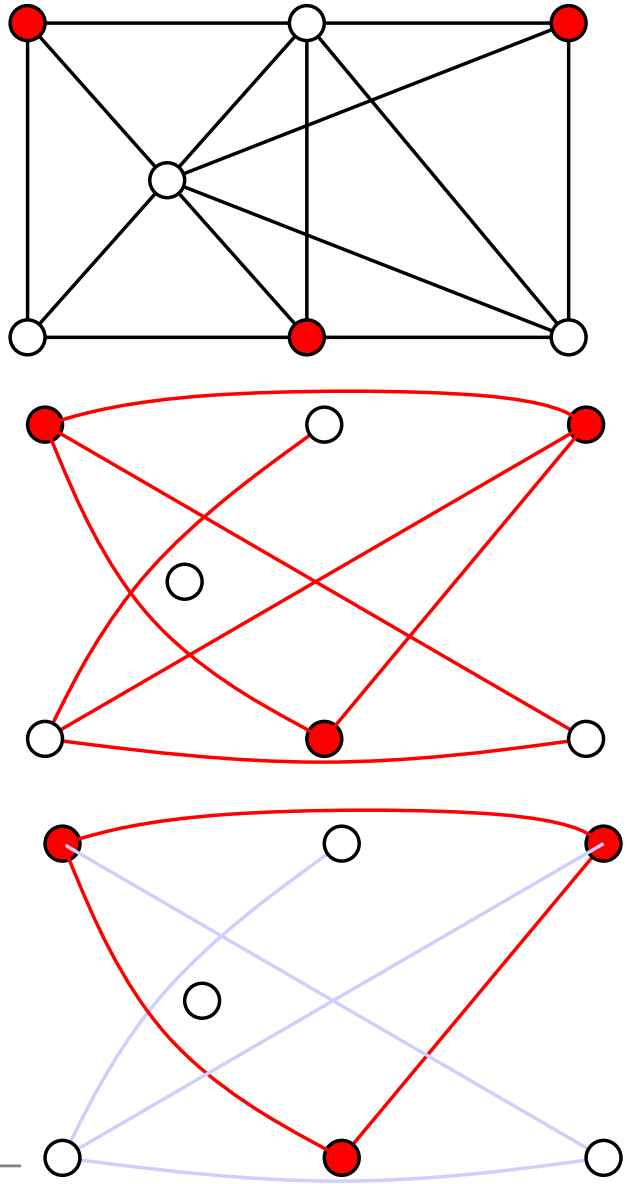
Independent Set

Definition: The complement of a graph $G = (V, E)$ is a graph $G^c = (V, E^c)$, where

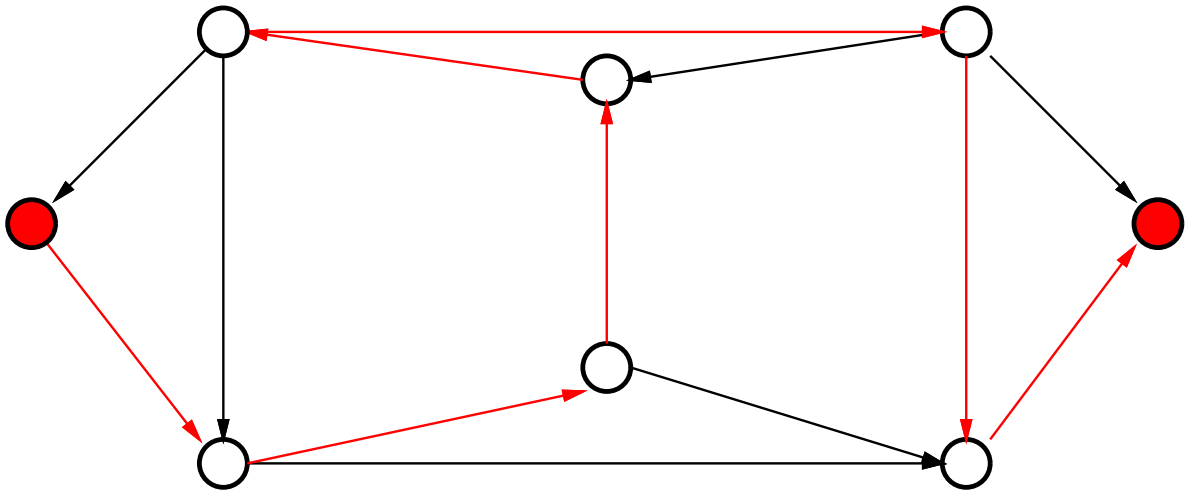
$E^c = \{(v_1, v_2) | v_1, v_2 \in V$ and $(v_1, v_2) \notin E\}$.

Claim: If U is an independent set in G, then U is a clique in G^c.

Independent Set



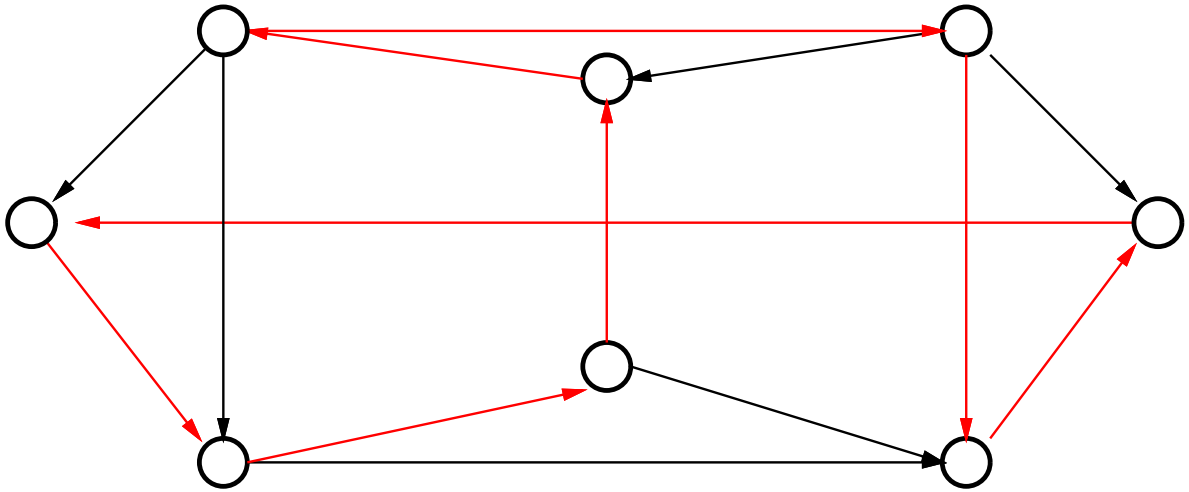
Hamiltonian Path



A Hamiltonian path in a (directed or undirected) graph, G, visits each note once.

$$\text{HAMPATH} = \{\langle G, s, t\rangle | G \text{ has a Hamiltonian path from } s \text{ to } t\}$$

Hamiltonian Circuit



Hamiltonian Circuit

- visits each node once.
- ends up where it started

Hamiltonian Circuit

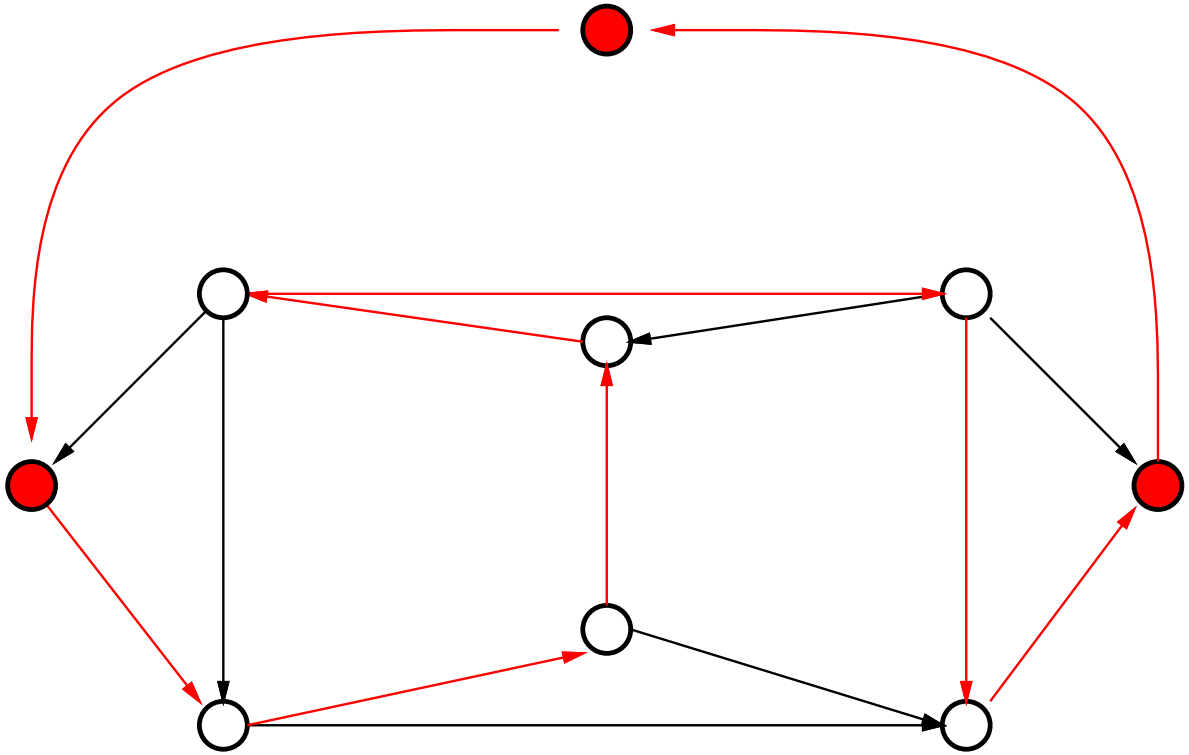
HAMCIRCUIT $= \{\langle G \rangle \mid G$ has Hamiltonian circuit$\}$

Theorem: HAMPATH is polynomial-time reducible to HAMCIRCUIT,

$$\text{HAMPATH} \leq_P \text{HAMCIRCUIT} \ .$$

Reduction

Theorem: HAMPATH is polynomial-time reducible to HAMCIRCUIT.



Hey, is the new vertex really needed? Why not just add an edge from t to s?

Reduction

Theorem: HAMCIRCUIT is polynomial-time reducible to HAMPATH.

Proof: Left as an easy (recommended) exercise.

Definition

A language $\mathcal{B}$ is NP-complete if it satisfies

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

Compare

A language $\mathcal{B}$ is RE-complete if it satisfies

- $\mathcal{B} \in RE$, and
- Every $\mathcal{A}$ in RE is mapping reducible to $\mathcal{B}$

Theorem

Theorem: If $\mathcal{B}$ is NP-complete and $\mathcal{B} \in P$, then $\mathcal{P} = \mathcal{NP}$.

To show $\mathcal{P} = \mathcal{NP}$ (and make an instant fortune, see www.claymath.org/millennium/P_vs_NP/), suffices to find a polynomial-time algorithm for any NP-complete problem.

Theorem

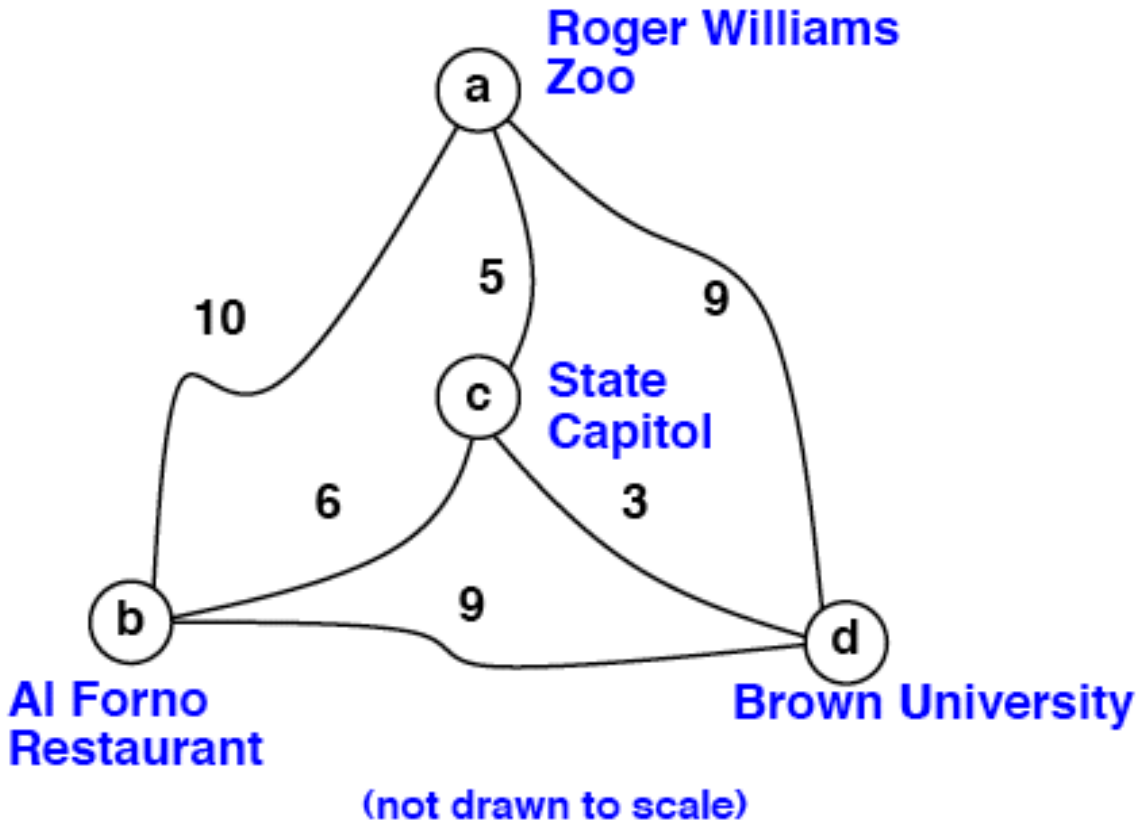
Theorem: If $\mathcal{B}$ is NP-complete, $\mathcal{C} \in NP$, and $\mathcal{B} \leq_P \mathcal{C}$, then $\mathcal{C}$ is NP-complete.

- We know that $\mathcal{C} \in NP$,
- must show that every $\mathcal{A}$ in NP is poly-time reducible to $\mathcal{C}$.
- Because $\mathcal{B}$ is NP-complete,
- every language in NP is poly-time reducible to $\mathcal{B}$.
- $\mathcal{B}$ is poly-time reducible to $\mathcal{C}$
- Can compose poly-time reductions (why?), so
- $\mathcal{A}$ is poly-time reducible to $\mathcal{C}$. ♣

Strategy

- Once we have one “structured” NP-complete problem, we can generate more by poly-time reduction.
- Getting the first one requires some work.
- This is what Steve Cook (then in Berkeley, now in Toronto) and Leonid Levin (then in Moscow, now in Boston) did in the early seventies.

Traveling Salesman



Parameters:

- set of cities C
- set of inter-city distances D
- goal k

Traveling Salesman

Define TRAVELING-SALESMAN $= \{\langle C, D, k\rangle \mid (C, D)$ has a TS tour of total distance $\leq k\}$

Remark: Can consider two versions – undirected and directed.

Recall

HAMCIRCUIT $= \{\langle G\rangle \mid G$ has Hamiltonian circuit$\}$

Theorem: HAMCIRCUIT is polynomial-time reducible to TRAVELING-SALESMAN,

$$\text{HAMCIRCUIT} \leq_P \text{TRAVELING-SALESMAN}\ .$$

HAMCIRCUIT$\leq_P$ TSP

The reduction: Given a directed graph $G = (V, E)$ we construct a directed traveling salesman instance.

- The cities are identical to the nodes of the original graph, $C = V$.
- The distance of going from v_1 to v_2 is 1 if $(v_1, v_2) \in E$, and 2 otherwise.
- The bound on the total distance of a tour is $k = |V|$.

HAMCIRCUIT$\leq_P$ TSP

- Validity of Reduction
  - $\Longrightarrow$ Suppose G has a Hamiltonian circuit. The distance assigned by the reduction to all edges in this circuit is 1. Thus in (C, D) there is a traveling salesman tour of total distance $|V| = k$, namely $(C, D, k) \in$ TRAVELING-SALESMAN.
  - $\Longleftarrow$ Suppose (C, D) has a traveling salesman tour of total distance $|V| = k$. Tour cannot contain any edge of distance 2. Therefore it gives a Hamiltonian circuit in G.
- Efficiency: Reduction in quadratic time (filling up distances for all edges of the complete graph). ♣

3SAT (reminder)

Definition: A Boolean formula is in 3CNF form if it is a CNF formula, and all terms have three literals.

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

Define

$$\text{3SAT} = \{\langle \phi \rangle \mid \phi \text{ is satisfiable 3CNF formula}\}$$

Clearly, if ϕ is a satisfiable 3CNF formula, then for any satisfying assignment of ϕ, every clause must contain at least one literal assigned 1.

The Language SAT

Definition: A Boolean formula is in conjunctive normal form (CNF) if it consists of terms, 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: SAT $= \{\langle \phi \rangle \mid \phi$ is satisfiable CNF formula$\}$

Strategy

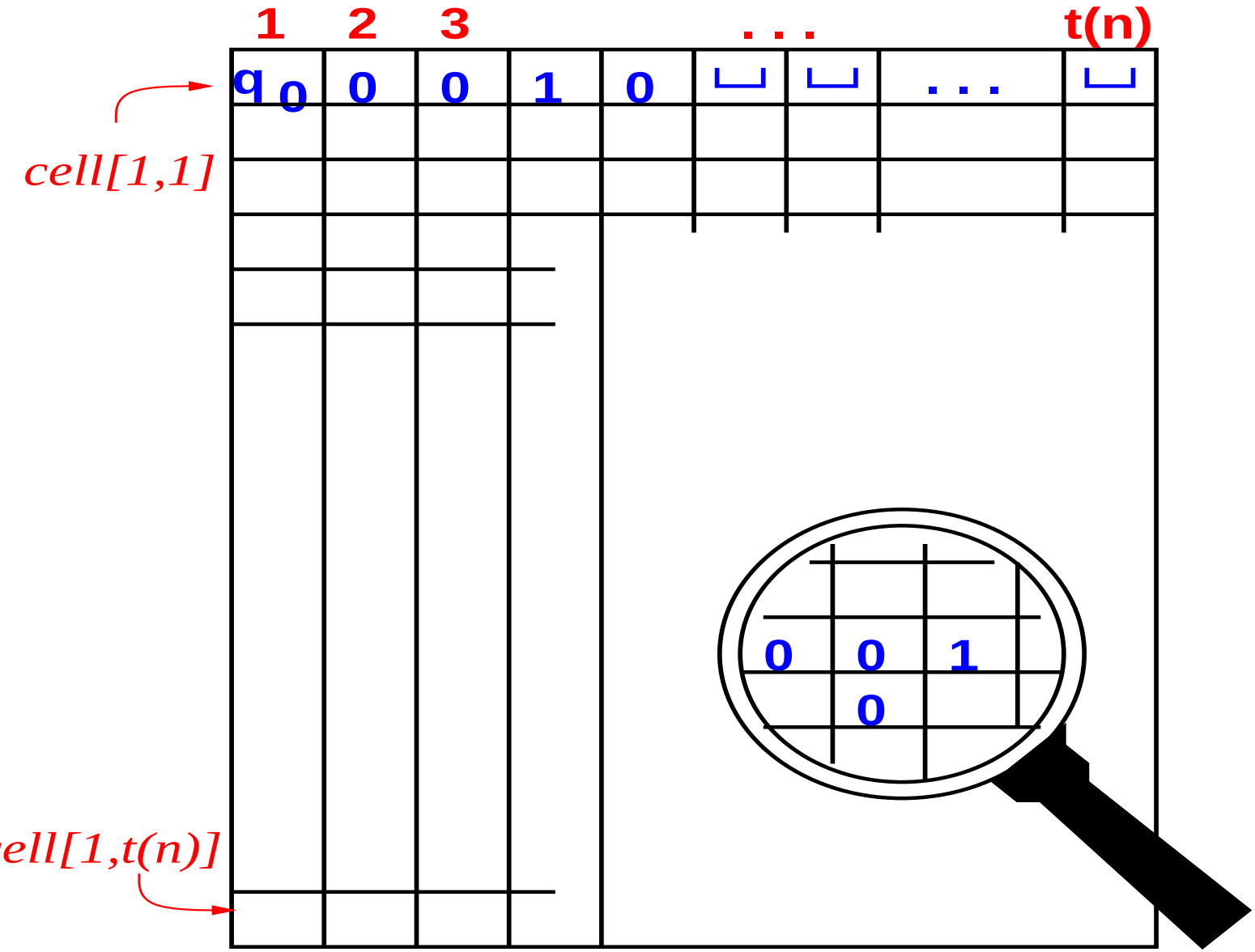
- Once we have one structured NP-complete problem, we can generate more by poly-time reductions.
- Getting the first one requires some work.

Cook-Levin (early 70s)

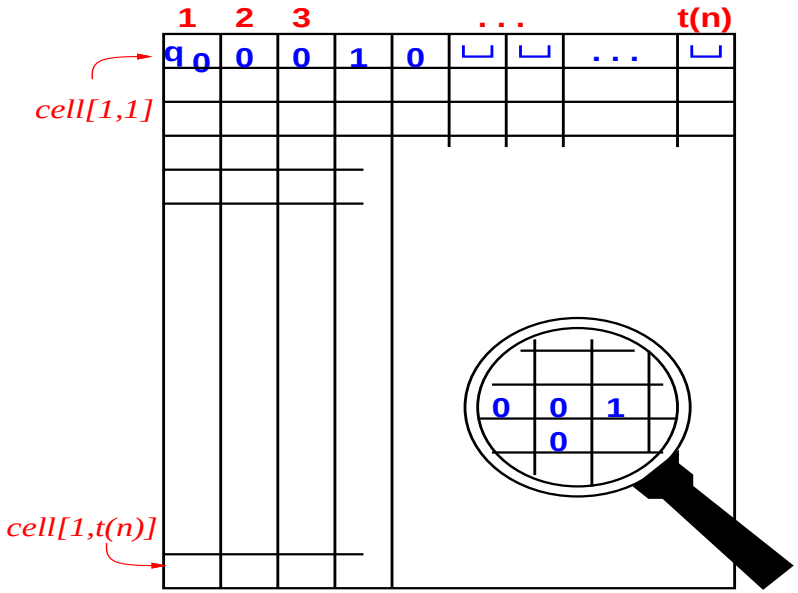
Theorem: SAT is NP complete.

- Must show that every NP problem reduces to SAT in poly-time.
- **Proof Idea**: Suppose $\mathcal{L} \in \mathcal{NP}$, and M is an NTM that accepts $\mathcal{L}$.
- On input w of length n, M runs in time $t(n) = n^c$.
- We consider the n^c-by-n^c tableau that describes the computation of M on input w.
- I.e., computation history

The Tableau



The Tableau



- Row 1 in tableau represents initial configuration of M on input w.
- Row i in tableau represents i-th configuration in a computation of M on input w.

A Formula Simulating the Tableau

- We construct a Boolean CNF formula ϕ_w that "mimics" the tableau.
- Given the string w, it takes $O(n^{2c})$ steps to construct ϕ_w.
- The following property holds:

  $\phi_w \in SAT$ iff M accepts w.
- So the mapping $w \mapsto \phi_w$ is a poly time reduction from $\mathcal{L}$ to SAT, establishing $\mathcal{L} \leq_P SAT$.

- We still got a few small details to take care of...

Details of Formula (Partial List)

- We construct a CNF Boolean formula ϕ_w that "mimics" the tableau: .
- Let $C = Q \cup \Gamma$, the letters in the tableau
- ϕ_w uses Boolean variables:
  - $x_{i,j,s}$ is true iff the j-th cell in i-th configuration contains the $s \in C$.
- The formula ϕ_w consists of four parts:

$$\phi_w \quad = \quad \phi_{\text{cell}(M)} \wedge \phi_{\text{start}(w)} \wedge \phi_{\text{accept}(M)} \wedge \phi_{\text{move}(M)}$$

Details of Formula (cont.)

- $\phi_{\text{cell}(M)}$ guarantees that the variables encode legal configurations.
- Each cell (i, j) has at least one letter $\vee_{s \in C} x_{i,j,s}$.
- No cell (i, j) has two or more letters $\bigwedge_{s,s' \in C, s \neq s'} \overline{x_{i,j,s}} \vee \overline{x_{i,j,s'}}$.
- Together:

$$\phi_{\text{cell}(M)} = \bigwedge_{i,j} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,s' \in C, s \neq s'} \overline{x_{i,j,s}} \vee \overline{x_{i,j,s'}} \right) \right]$$

Details of Formula (cont.)

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

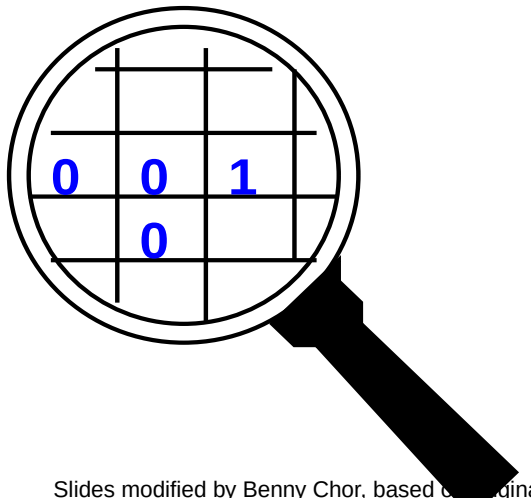
$$\phi_{\text{start}(w)} = x_{1,1,q_0} \wedge x_{1,2,w_1} \wedge x_{1,3,w_2} \wedge \cdots \wedge x_{1,n+1,w_n} \wedge x_{1,n+2,\sqcup} \wedge \cdots \wedge x_{1,n^k,\sqcup}$$

- $\phi_{\text{accept}(M)}$ guarantees that some row encodes an accepting configuration, i.e., $uq_a v$.

$$\phi_{\text{accept}(w)} = \bigvee_{i,j} x_{i,j,q_a}$$

Details of Formula (cont.)

- ϕmove$_{(M)}$ is the "heart" of ϕ_w. To construct it, we employ locality of computations.
- To determine contents of tableau entry (i, j) (cell j in configuration i), only the contents of three tableau entries (from configuration $i-1$), $(i-1, j-1), (i-1, j), (i-1, j+1)$, and M's table, are needed.
- If head not in area, nothing changes. And and if it is, changes are local and are determined using M.



Details of Formula (cont.)

- Assume that $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$.
- Legal 2×3 configurations:

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

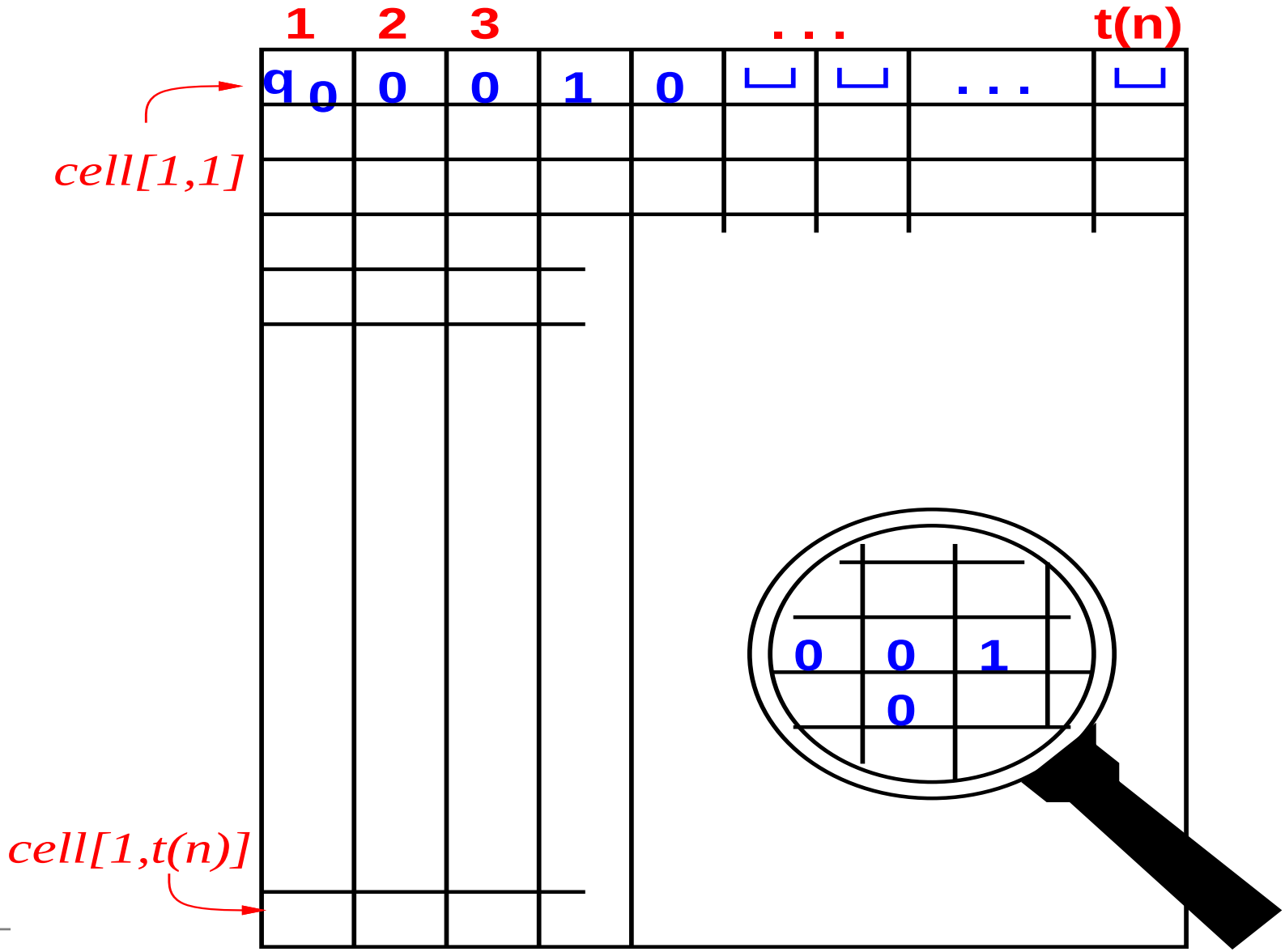
- Illegal configurations:

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

The Tableau in Perspective



Correctness of Reduction

- All four components of ϕ_w can be put in CNF form, so ϕ_w itself ($\wedge$ of the four) is also in CNF.
- The transformation $w \mapsto \phi_w$ is computable in time $O(n^{2c})$.
- An assignment satisfying $\phi_{\text{cell}(M)} \wedge \phi_{\text{start}(w)} \wedge \phi_{\text{move}(M)}$ corresponds to a valid computation of M on w.
- An assignment satisfying, in addition $\phi_{\text{accept}(M)}$, corresponds to an accepting computation of M on w.
- Therefore M accepts w iff $\phi_w \in SAT$.

- For complete details, including conversion to CNF, consult Sipser chapter 7.4.

Strategy

- We have seen that SAT is NP-complete.
- We now reduce SAT to 3SAT.
- And then will reduce 3SAT to a bunch of other problems in NP.
- In class and recitation will give in detail just a few examples.
- Full list contains hundreds or 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.

SAT and 3SAT

Recall

SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable CNF formula$\}$

3SAT $= \{\langle\phi\rangle \mid \phi$ is satisfiable 3CNF formula$\}$

The reduction maps CNF formulae to 3CNF ones "clause by clause". A clause with ℓ literals is mapped to ℓ clauses, built on the original literals together with $\ell - 1$ new ones.

For example:

$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_8)$

$\longmapsto$

$(x_1 \vee y_1) \wedge (\overline{y_1} \vee \overline{x_2} \vee y_2) \wedge (\overline{y_2} \vee \overline{x_3} \vee y_3) \wedge$
$(\overline{y_3} \vee x_4 \vee y_4) \wedge (\overline{y_4} \vee x_8)$

SAT $\leq_P$ 3SAT

Consider mapping $\phi \mapsto \phi_3$, *e.g.* $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_8) \longmapsto$ $(x_1 \vee y_1) \wedge (\overline{y_1} \vee \overline{x_2} \vee y_2) \wedge (\overline{y_2} \vee \overline{x_3} \vee y_3) \wedge (\overline{y_3} \vee x_4 \vee y_4) \wedge (\overline{y_4} \vee x_8)$
Claim: ϕ has a satisfying assignment iff ϕ_3 does.

Proof sketch: $\Leftarrow$ An assignment satisfying ϕ_3 cannot "rely" on new literals alone – at least one original literal must be satisfied.

$\Rightarrow$ An assignment satisfying ϕ makes at least one literal per clause happy. In the "ϕ_3 clause" of this literal the new veriable is under no constraints. This enables propagation to a satisfying assignment that "relies" on new vars alone in rest of ϕ_3 clauses.

This establishes validity of the reduction. Since it is in polynomial time (why?), we get SAT $\leq_P$ 3SAT. ♣.

3SAT and Its Poor Cousin

We now know that SAT $\leq_P$ 3SAT. Since SAT is NP-complete and 3SAT$\in$ NP, this proves that 3SAT is itself NP-complete.

What about the 3SAT $\leq_P$ SAT direction?

We now want to examine what happens if we further reduce the number of literals per clause in CNF formulae.

Definition: A Boolean formula is in 2CNF if it is a CNF formula, and all terms have at most two literals. For example

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

3SAT and Its Poor Cousin

Definition:

$$2SAT = \{\langle \phi \rangle \mid \phi \text{ is satisfiable 2CNF formula}\}$$

- Betting time: Is 2SAT NP-complete? Is it in P? Or maybe we do not know? . . .
- Well, turns out 2SAT is in P. For details, though, you'll have to refer to the algorithms (formerly, efficiency of computations) course.

Chains of Reductions: NPC Problems