Computational Models — Lecture 12[1]

Handout Mode

Iftach Haitner.

Tel Aviv University.

January 16, 2017

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

Talk Outline

- Reminder – deterministic and nondeterministic time classes
- The class $\mathcal{NP}$
- Verifiability
- Additional NP languages
- The class *co*-$\mathcal{NP}$
- $\mathcal{P}$ Verses $\mathcal{NP}$
- NP-completeness
- Satisfiability
- Cook-Levin theorem

- Sipser's book, 7.4–7.5

Reminder — Deterministic time

Definition 1 (deterministic Time)

Let M be a deterministic TM, and let $t : \mathbb{N} \mapsto \mathbb{N}$. We say that M runs in time $t(n)$, if For every input x of length n, the number of steps that $M(x)$ uses is at most $t(n)$.

Question 2

What is a “step"?

Definition 3 (DTIME)

For $t \colon \mathbb{N} \mapsto \mathbb{N}$, let $\mathrm{DTIME}(t(n)) =$
$\{\mathrm{L} \subseteq \Sigma^* : \ \mathrm{L}$ is decided by an $O(t(n))$-time single tape TM$\}$

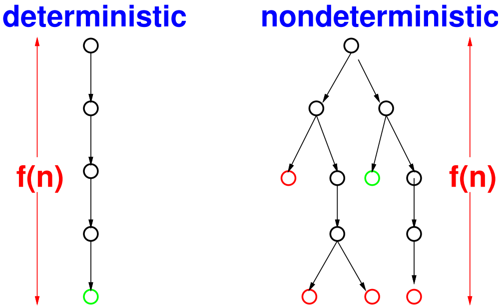
Note that $t(n)$ running time, is also required for strings not in L.

Non-deterministic time

Definition 4 (nondeterministic time)

A non-deterministic TM N runs in time $f(n)$, where $f\colon \mathbb{N} \mapsto \mathbb{N}$, 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 $f(n)$.

Notice that also non-accepting branches must reject within $f(n)$ many steps.



TAKE NOTE: the depth of the tree, not the size of the tree!!!

Part I

The Class NP

The Class $\mathcal{NP}$

Definition 5 (NTIME)

For $t\colon \mathbb{N} \mapsto \mathbb{N}$, let $\mathrm{NTIME}(t(n)) =$
$\{\mathrm{L} \subseteq \Sigma^*\colon$ L is decided by an $O(t(n))$-time single tape NTM$\}$

$\mathcal{NP}$ is the set of languages decidable in polynomial time on non-deterministic TMs.

Definition 6 ($\mathcal{NP}$)

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

The class *NP* is important because:

- Insensitive to choice of reasonable non-deterministic computational model.
- Roughly corresponds to problems whose positive solutions are efficiently verified.

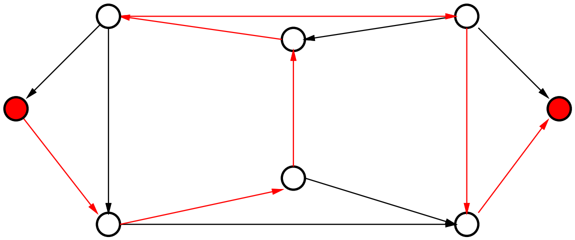
Are language in $\mathcal{NP}$ decidable in polynomial time?

We don't know!

but what we do know is this:

Large class of fundamental languages in $\mathcal{NP}$ that are "the hardest"
(i.e., if ONE is efficiently solvable then ALL are efficiently solvable)

Hamiltonian path



A Hamiltonian path in a directed G visits each node exactly once.

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

Question 7

How hard is it to decide HAMPATH?

Easy to obtain exponential time algorithm:

- Generate each potential path
- Check whether it is Hamiltonian

HAMPATH $\in \mathcal{NP}$

Here is an NTM that decides HAMPATH in polynomial time.

Algorithm 8 (N)

On input $\langle G = (V, E), s, t \rangle$,

1. Guess a list of numbers $p_1, \ldots, p_m$, where $m = |V|$ and $1 \leq p_i \leq m$.
2. `Accept` if all the following hold (otherwise `Reject`):
   - No repetitions in list
   - $p_1 = s$ and $p_m = t$.
   - $(p_i, p_{i+1}) \in E$ for every $1 \leq i \leq m - 1$

- How does a TM guess a string?

Claim 9

N runs in polynomial time

Verifiability of HAMPATH

This problem has one very interesting feature: polynomial verifiability:

> *We don't know a fast way to find a Hamiltonian path, but we can check whether a given path is Hamiltonian in polynomial time.*

Verifying correctness of a path is much easier than determining whether one exists

Section 1

Verifiability

Verifiability

Definition 10 (verifier)

A *deterministic* algorithm V is a verifier for a language L, if

- $x \in L \implies \exists c \in \{0,1\}^*$ s.t. $V(x,c) = 1$.
- $x \notin L \implies \nexists c \in \Sigma^*$ s.t. $V(x,c) = 1$.

- The verifier uses the additional information c to verify $x \in L$.
- If V accepts (x, c) (i.e., outputs 1), the string c is called a certificate (also known as, proof or witness) for x.
- A polynomial verifier runs in polynomial time in $|x|$ (i.e., in the length of its left-hand-side input parameter).
- A language L is polynomially verifiable, if it has a polynomial verifier.
- A certificate for $\langle G, s, t\rangle \in$ HAMPATH is simply the Hamiltonian path from s to t.

  Easy to verify in time polynomial in $|\langle G, s, t\rangle|$ whether given path is Hamiltonian.
- Not all languages are known to be polynomially verifiable.

NP and Verifiability

Theorem 11

A language is in $\mathcal{NP}$ iff it has a polynomial time verifier.

Proof's idea:

- The NTM emulates the verifier by guessing the certificate.
- Verifier emulates NTM by using accepting branch as certificate.

Verifiability $\implies \mathcal{NP}$

Claim 12

If L has a poly-time verifier, then it is decided by some polynomial-time NTM.

Proof: Let V be poly-time verifier for L of running time $p(n)$ for some $p \in$ poly.

Algorithm 13 (N)

On input $x \in \{0,1\}^n$:

1. Guess a string c of length $p(n)$.
2. Emulate V on $\langle x, c\rangle$
3. `Accept` if V accepts; Otherwise `Reject`.

♣

- Why is it suffices to guess a string of length $p(n)$?

$\mathcal{NP} \implies$ Verifiability

Claim 14

If L is decided by a polynomial-time NTM N, then L has a poly-time verifier.

Proof: Assume for simplicity that at each step of N, the number of possible non-deterministic moves is at most two.

Algorithm 15 (V)

On input (x, c):

1. Emulate $N(x)$, treating each symbol of c as a description of the non-deterministic choice in each step of N.
2. `Accept` if this branch accepts; Otherwise `Reject`.

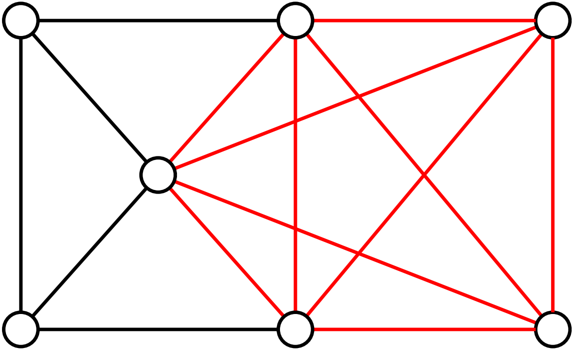
♣

Without the simplifying assumption?

Section 2

A few more NP languages

CLIQUE



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.

Question 16

What is the largest k-clique in the figure?

CLIQUE cont.

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

Theorem 17

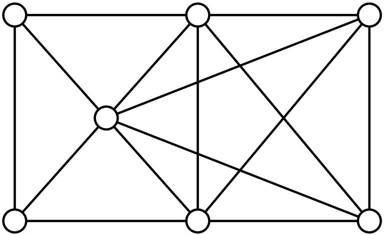
CLIQUE $\in \mathcal{NP}$

Proof's idea: The clique is the certificate.

Algorithm 18 (V)

On input $(\langle G, k \rangle, c)$
`Accept` if c is a k-clique subgraph of G;
Otherwise `Reject`.

Independent set



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

A k-IS is an independent set of size k.

Question 19

What is the largest k-IS in the figure?

Independent set cont.

$$\text{IND-SET} = \{\langle G, k\rangle :\ G \text{ contains an independent set of size } k\}$$

Theorem 20

IND-SET $\in \mathcal{NP}$

Proof's idea: The independent set is the certificate.

Algorithm 21 (V)

On input $(\langle G, k\rangle, c)$
`Accept` if c is a k-IS of G (no edges between nodes in c, and $|c| = k$);
Otherwise `Reject`.

Section 3

co-$\mathcal{NP}$

The class *co*-$\mathcal{NP}$

$\overline{\text{CLIQUE}} = \{\langle G, k\rangle \colon G$ is an undirected graph with no k-clique$\}$ seems not to be member of $\mathcal{NP}$.

It seems harder to efficiently verify that something does not exist than to efficiently verify that something does exist.

Definition 22 (*co*-$\mathcal{NP}$)

co-$\mathcal{NP} = \{\mathsf{L} \colon \overline{\mathsf{L}} \in \mathcal{NP}\}$.

But.. we are not sure...So far, no one knows if *co*-$\mathcal{NP}$ is distinct from $\mathcal{NP}$.

Claim 23

$\mathcal{P} \subseteq$ *co*-$\mathcal{NP}$.

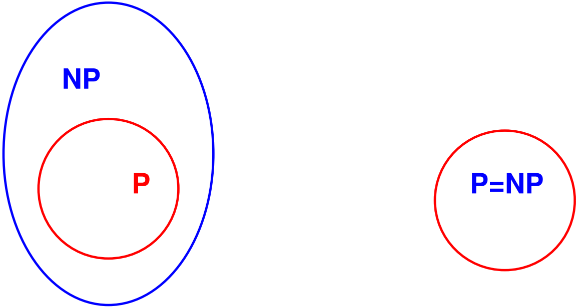
Proof? $\mathsf{L} \in \mathcal{P} \implies \overline{\mathsf{L}} \in \mathcal{P} \implies \overline{\mathsf{L}} \in \mathcal{NP} \implies \mathsf{L} \in$ *co*-$\mathcal{NP}$.

Is Primality in $\mathcal{NP}$? *co*-$\mathcal{NP}$?
How would you prove that a number is prime without trying all divisors?
Actually it is in P! (not obvious at all)

Section 4

P vs. NP

$\mathcal{P}$ vs. $\mathcal{NP}$



The question $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ is one of the great unsolved mysteries in contemporary mathematics.

$\mathcal{P}$ vs. $\mathcal{NP}$

- Most computer scientists believe the two classes are not equal
- Most bogus proofs show them equal (?)
- One of 7 Clay Millenium Prize problems ($1,000,000\$$!)
- “Computer Science’s greatest intellectual export” (Papadimitriou 2007)

$\mathcal{P}$ Vs. $\mathcal{NP}$, cont.

If $\mathcal{P}$ differs from $\mathcal{NP}$, then the distinction between $\mathcal{P}$ and $\mathcal{NP} \setminus \mathcal{P}$ is meaningful and important.

- languages in $\mathcal{P}$ are tractable
- languages in $\mathcal{NP} \setminus \mathcal{P}$ are intractable

Until we can prove that $\mathcal{P} \neq \mathcal{NP}$, there is no hope of proving that a specific language lies in $\mathcal{NP} \setminus \mathcal{P}$.

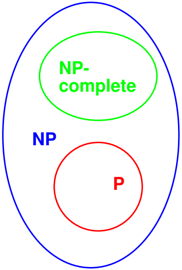
Nevertheless, we can prove statements of the form

If $\mathsf{A} \in \mathcal{NP} \setminus \mathcal{P}$*, then* $\mathsf{B} \in \mathcal{NP} \setminus \mathcal{P}$*.*

Section 5

NP Completeness

NP Completeness



The class of NP-complete languages are

- "hardest" languages in $\mathcal{NP}$
- If any NP-complete $\mathsf{L} \in P$, then $\mathcal{NP} = \mathcal{P}$.

Question 24

Are there NP-complete languages?

Polynomial-time reducibility

Definition 25 (poly-time computable functions)

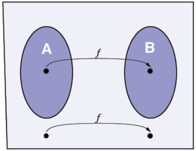
A function $f : \Sigma^* \mapsto \Sigma^*$ is polynomial-time computable, if there is a poly-time deterministic TM that

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

Definition 26 (poly-time reduction)

A polynomial-time computable $f : \Sigma^* \mapsto \Sigma^*$ is a poly-time reduction from language A to B, if $x \in A \iff f(x) \in B$ for every $x \in \Sigma^*$.
Is such a reduction from A to B exists, we say that A is poly- time mapping reducible to B, denoted $A \leq_P B$.

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



Example: CLIQUE $\leq_P$ IND-SET

Proof:

Definition 27

The complement of a graph $G = (V, E)$ is a graph $G^c = (V, E^c)$, where $E^c = \{(v_1, v_2) \colon v_1, v_2 \in V \text{ and } (v_1, v_2) \notin E\}$.

The reduction $f(G, k)$ from CLIQUE to IND-SET simply computes the complement of the graph and outputs (G^c, k).

f satisfies:

- U is a clique in G $\iff$ U is a independent set in G^c.
- computable in polynomial time!

♣

Remark 28

Same reduction shows that IND-SET $\leq_P$ CLIQUE

Reductions to $\mathcal{P}$

Theorem 29

If $\mathsf{A} \leq_P \mathsf{B}$ *and* $\mathsf{B} \in \mathcal{P}$ *then* $\mathsf{A} \in \mathcal{P}$*.*

Proof:

- Let f the reduction from A to B, computed by TM M_f.
  On input x, the TM M_f makes at most $c_f \cdot |x|^{a_f}$ steps.
- Let M_B be the poly-time decider for B.
  On input y, the TM M_B makes at most $c_\mathsf{B} \cdot |y|^{a_\mathsf{B}}$ steps.

Algorithm 30 (Decider M_A for A)

On input x, return $\mathsf{M}_\mathsf{B}(f(x))$

- M_A decides A
- Since $|f(x)| \leq c_f|x|^{a_f}$, running time of $\mathsf{M}_\mathsf{B}(x)$, is at most
  $c_\mathsf{B} \cdot (c_f \cdot |x|^{a_f})^{a_\mathsf{B}} = (c_\mathsf{B} \cdot c_f^{a_\mathsf{B}}) \cdot |x|^{a_f \cdot a_\mathsf{B}} \in \mathrm{poly}(|x|)$

  Hence, $\mathsf{A} \in \mathcal{P}$

What $A \leq_P B$ tells us about B?

Question 31

Assume that $\{0^n1^n \colon n \geq 0\} \leq_P L$. Does it yield that $L \in \mathcal{P}$?

Answer: No. (Reduction in the wrong direction!)

Let $L = H_{TM,\varepsilon}$ and define $f(x) = \begin{cases} M_{stop}, & x \in \{0^n1^n \colon n \in \mathbb{N}\} \\ M_{no-stop}, & \text{otherwise.} \end{cases}$.

$A \leq_P B$ does tell us that B is "at least as hard" as A.

NP completeness, formal definition

Definition 32 ($\mathcal{NP}$-complete)

A language B is NP-complete, if

- B $\in \mathcal{NP}$, and
- Every A $\in \mathcal{NP}$ is poly-time reducible to B (i.e., A $\leq_P$ B)

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

Compare to

Definition 33 (RE-complete)

A language B is RE-complete, if

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

Why NP completeness?

Theorem 34

If B $\in \mathcal{NPC}$ *and* B $\in \mathcal{P}$, *then* $\mathcal{P} = \mathcal{NP}$.

Proof: Immediately follows by Thm 29. ♣

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.

Question 35

Is $\mathcal{NPC}$ empty?

$\mathcal{NPC}$ Is not empty

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

Theorem 36

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

Proof:

- Clearly $\mathrm{A_{NP}} \in \mathcal{NP}$.
- Let $\mathrm{L} \in \mathcal{NP}$, let V be a verifier for L and let $p \in$ poly be a bound on the running time of V (i.e., $\mathrm{V}(x, \cdot)$ halts within $p(|x|)$ steps, for every $x \in \Sigma^*$).
- Define $f(x) = \langle \mathrm{V}, x, 1^{p(|x|)} \rangle$.
- f is poly-time computable
- $x \in \mathrm{L} \iff f(x) \in \mathrm{A_{NP}}$.

♣

Finding additional NP-complete languages

Theorem 37

Assume that

1. $\mathsf{B} \in \mathcal{NP}$
2. $\mathsf{A} \in \mathcal{NPC}$ *and* $\mathsf{A} \leq_P \mathsf{B}$

then $\mathsf{B} \in \mathcal{NPC}$.

Proof: Home exercise . . . ♣

We would like to find $\mathsf{L} \in \mathcal{NPC}$ that is “natural" and “easy" to reduce to.

Section 6

Satisfiability

Boolean variables

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

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

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 38 (satisfiable formula)

A formula is satisfiable, if some Boolean assignment to its variables, makes the formula evaluate to 1.

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

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

The language of satisfied formulas:

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

SAT $\in \mathcal{NPC}$

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

Theorem 39 (Cook-Levin (early 70s))

SAT $\in \mathcal{NPC}$.

- The "most important" $\mathcal{NP}$-complete language.
- It is easy to see that SAT $\in \mathcal{NP}$

Section 7

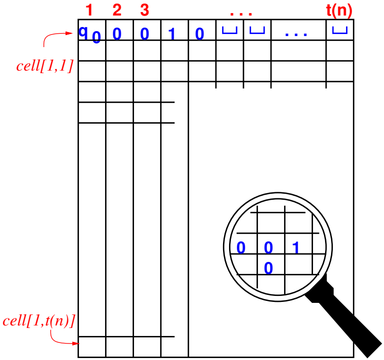
Proving SAT $\in \mathcal{NPC}$

The proof, high level

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

The configuration-history Tableau

Consider the $t(n)$-by-$t(n)$ 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.

The formula $\phi_{N,w}$

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

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

- Given an assignment z for $\phi_{N,w}$, let $T(z)$ be the $t(n) \times t(n)$ 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 $\phi_{\mathsf{Cell}(N)}$

$\phi_{\mathsf{Cell}(N)}$ 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:

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

Claim 40

If an assignment z satisfies $\phi_{\mathsf{Cell}(N)}$, 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., $q_0 w$).

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

Claim 41

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

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

$\phi_{\mathsf{Move}(N)}$ 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 $\delta(q_1, a) = \{(q_1, b, \mathrm{R})\}$ and $\delta(q_1, b) = \{(q_2, c, \mathrm{L}), (q_2, a, \mathrm{R})\}$.
- (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

- There is a constant number of legal rectangles (determined by δ).
- Denote this set by $C = C(\delta)$.

$\phi_{\mathsf{Move}(N)}$ – 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'

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

a	q	b
a	b'	q'

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

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.

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

- For each entry $(i,j) \in [t(n)] \times [t(n)]$ and $c \in C$, let $\phi_{\mathrm{Move},i,j,c}$ 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 $c = \begin{array}{|c|c|c|}\hline \mathrm{a} & q_1 & \mathrm{b} \\ \hline q_2 & \mathrm{a} & \mathrm{d} \\ \hline \end{array}$,

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

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

Claim 42

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

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

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., $uq_a v$):

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

Claim 43

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 $\phi_{N,w} \in \text{SAT}$. $\clubsuit$.
- For complete details, consult Sipser chapter 7.4.