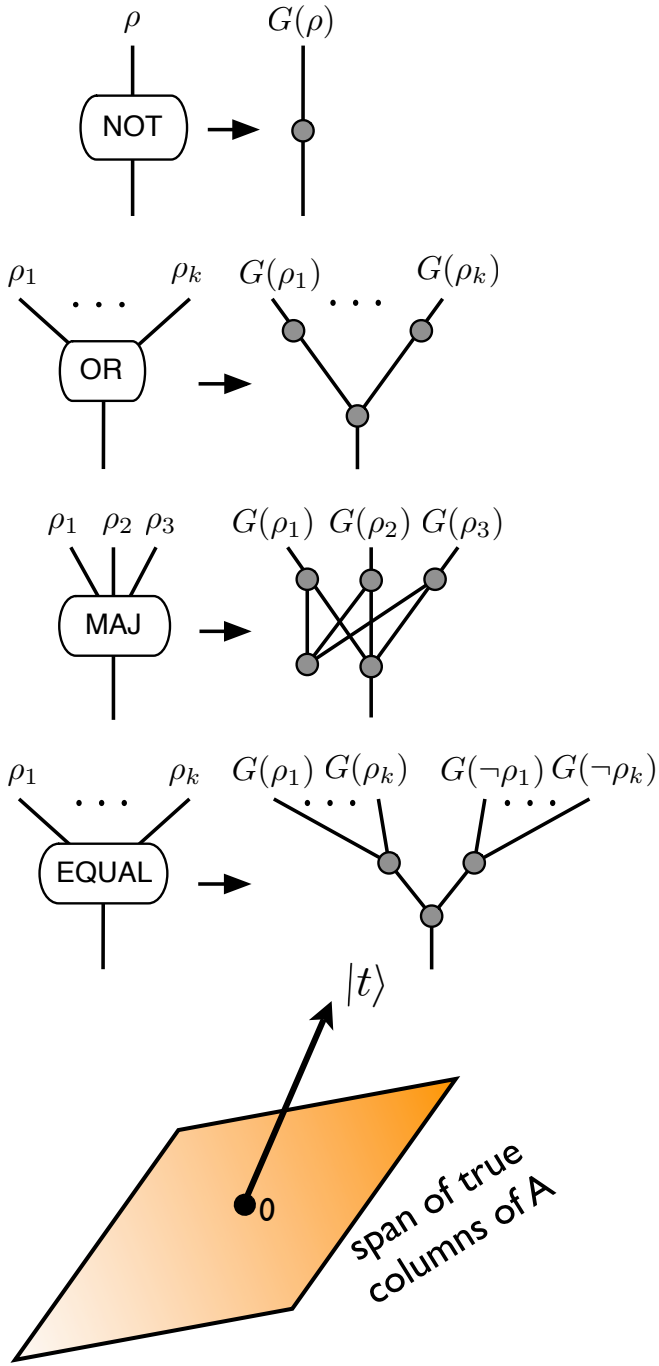
Span-program-based quantum algorithm for formula evaluation

Ben Reichardt
Caltech

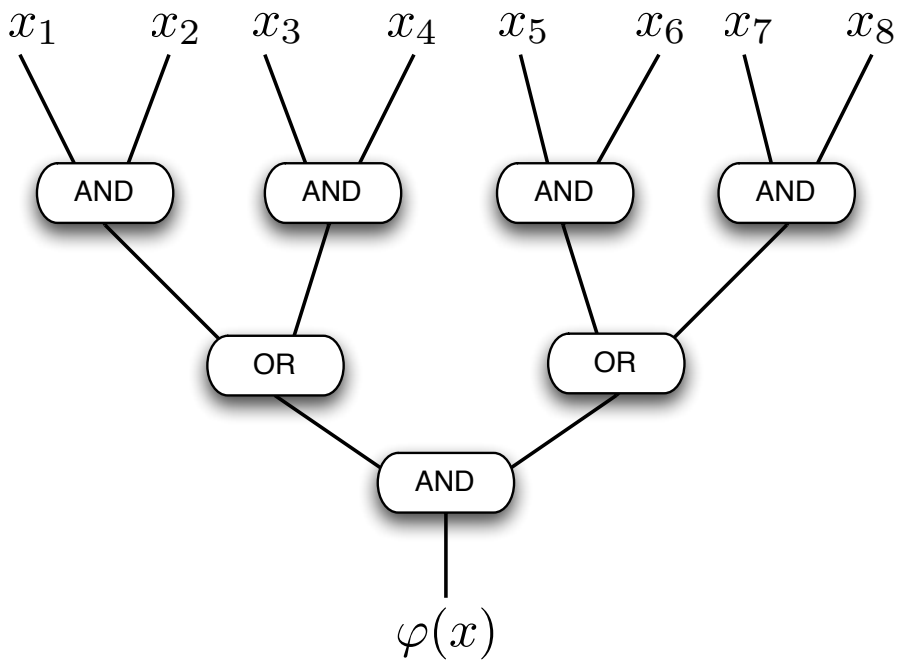
Robert Špalek
Google

[quant-ph/0710.2630]



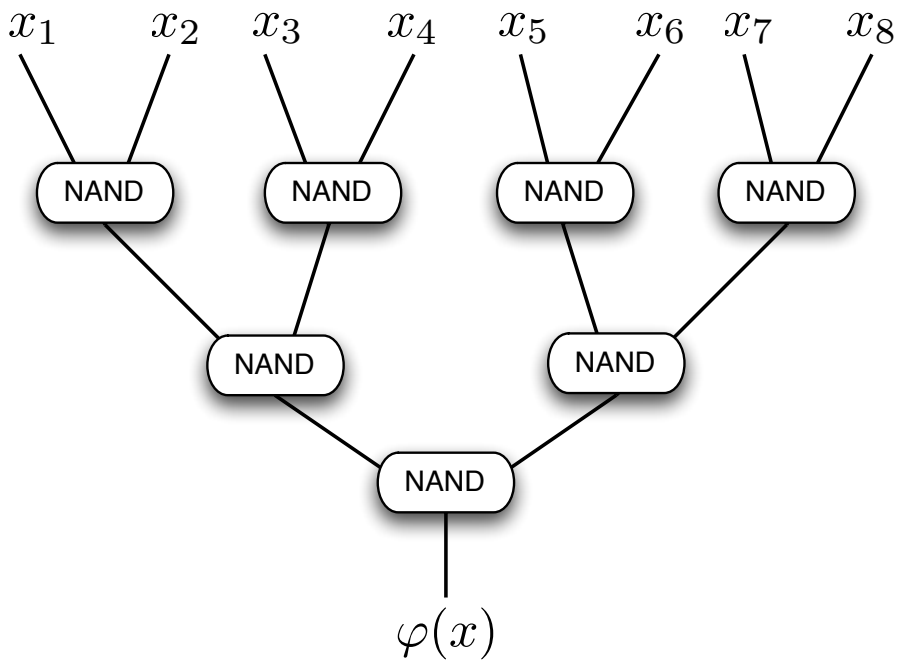
Farhi, Goldstone, Gutmann '07 algorithm

- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.



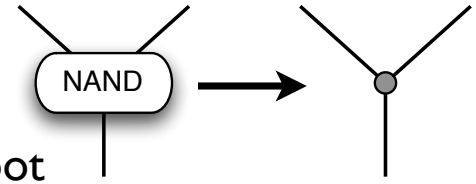
Farhi, Goldstone, Gutmann '07 algorithm

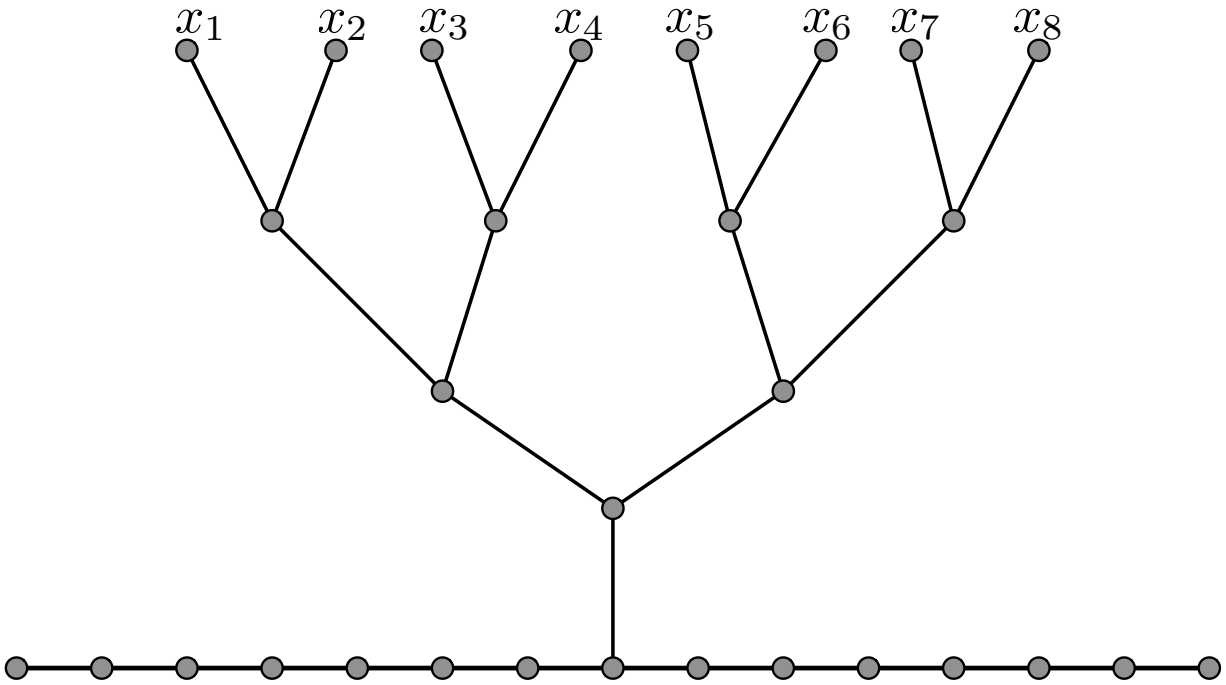
- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.



Farhi, Goldstone, Gutmann '07 algorithm

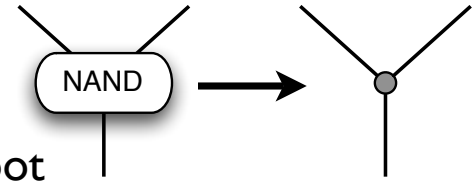
- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.
- Convert formula to a tree:
- Attach an infinite line to the root

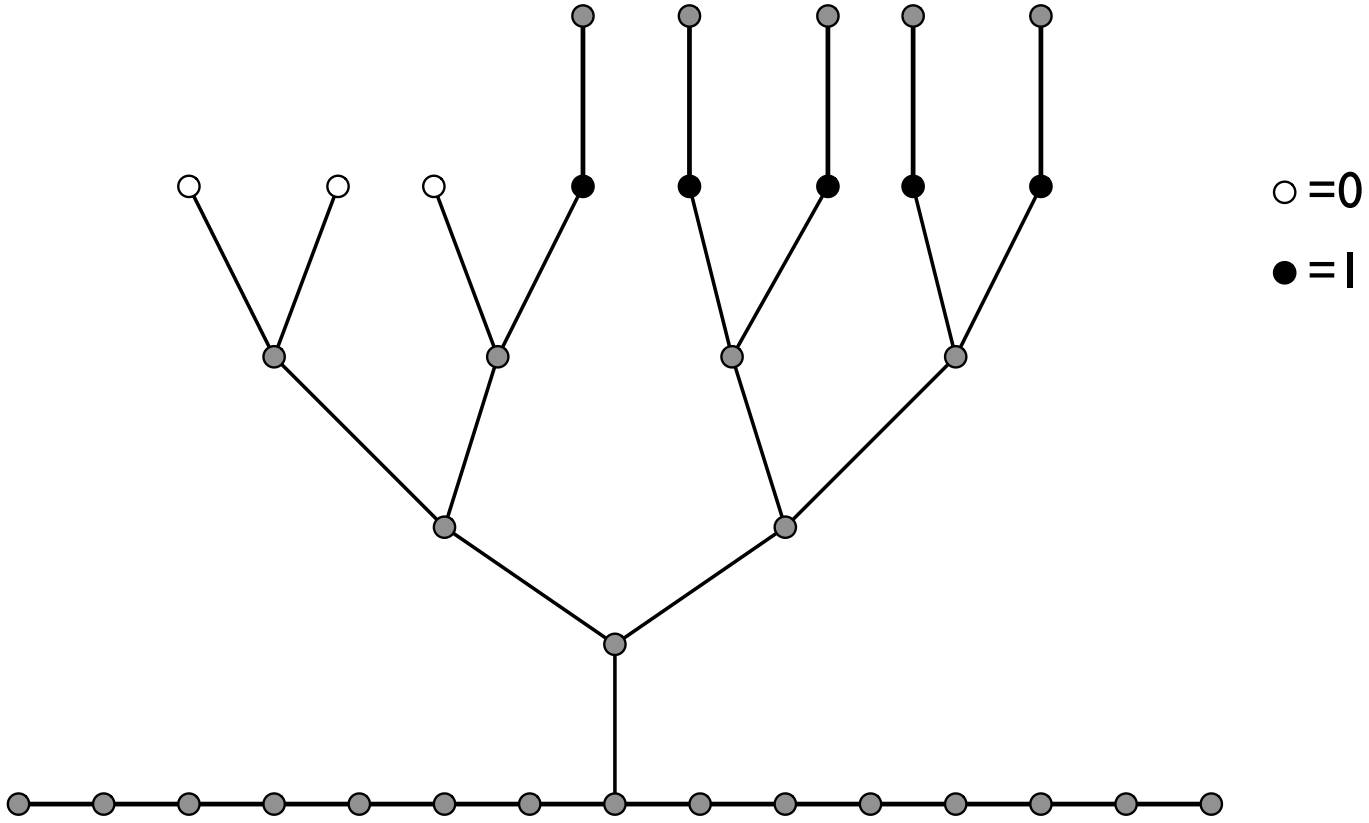




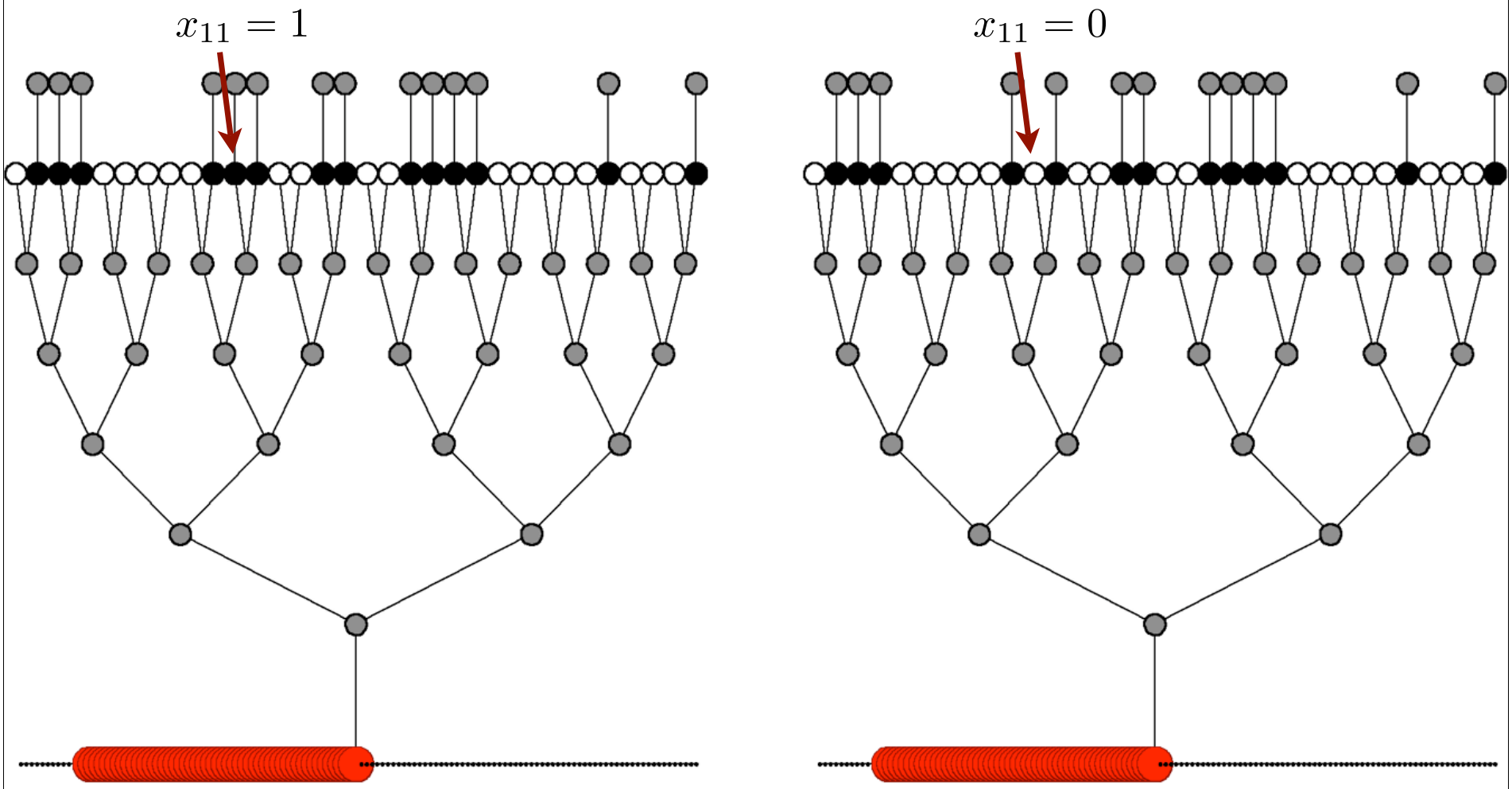
Farhi, Goldstone, Gutmann '07 algorithm

- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.
- Convert formula to a tree:
- Attach an infinite line to the root
- Add edges above leaf nodes evaluating to one…

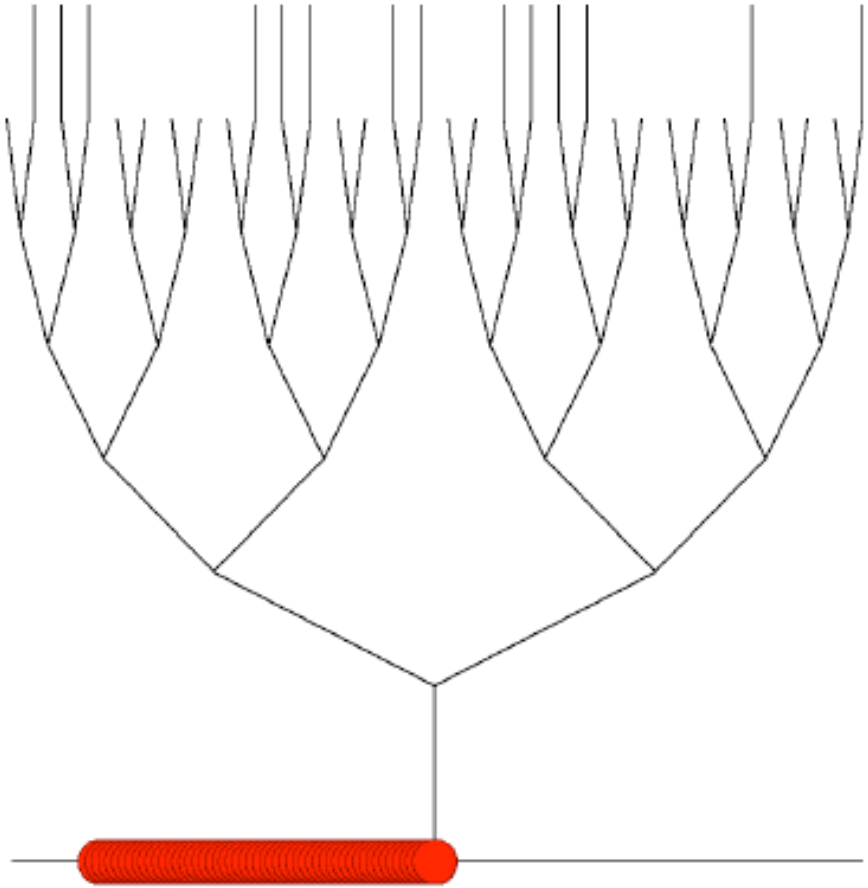


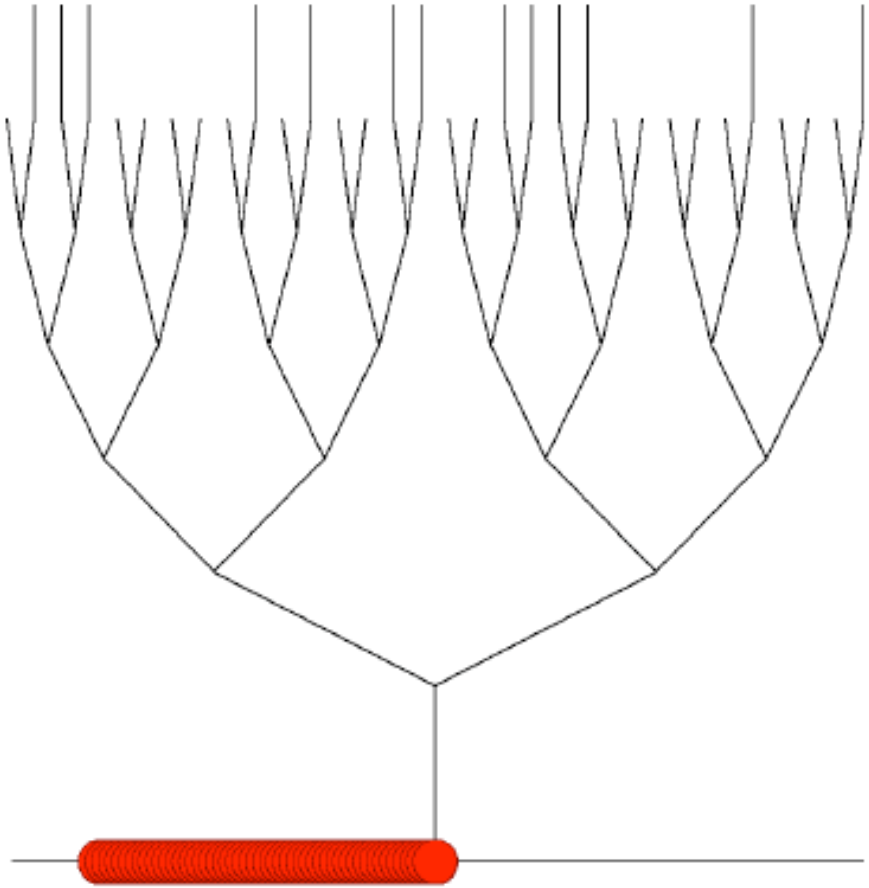


Continuous-time quantum walk [FGG '07]

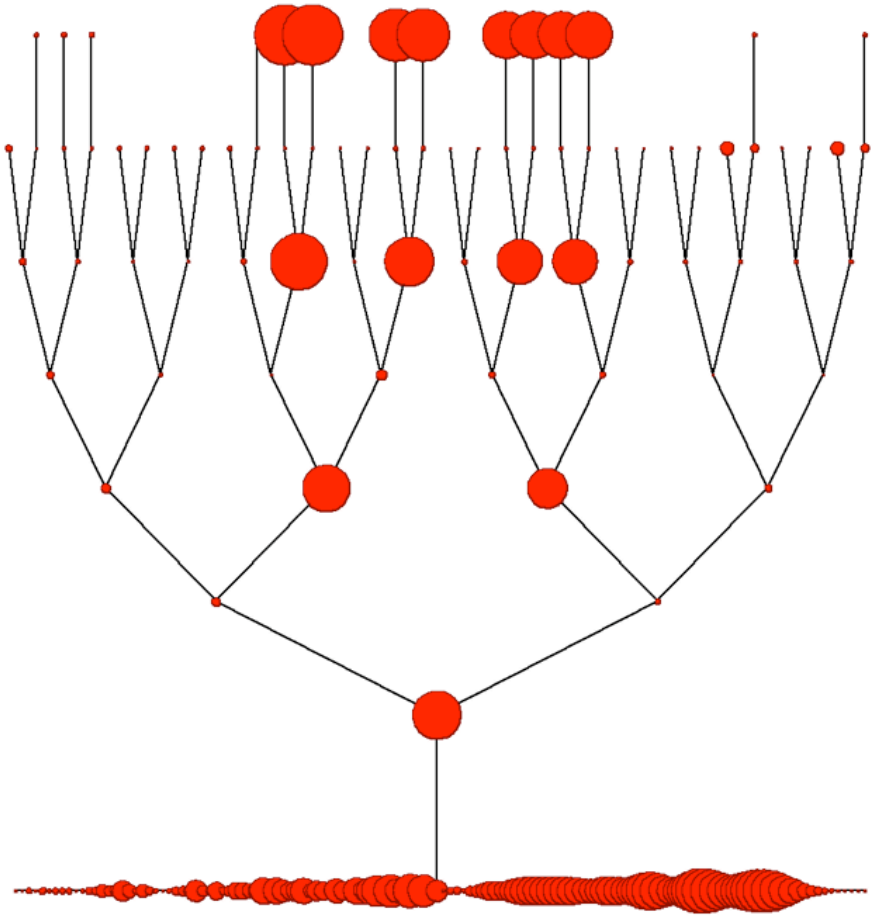


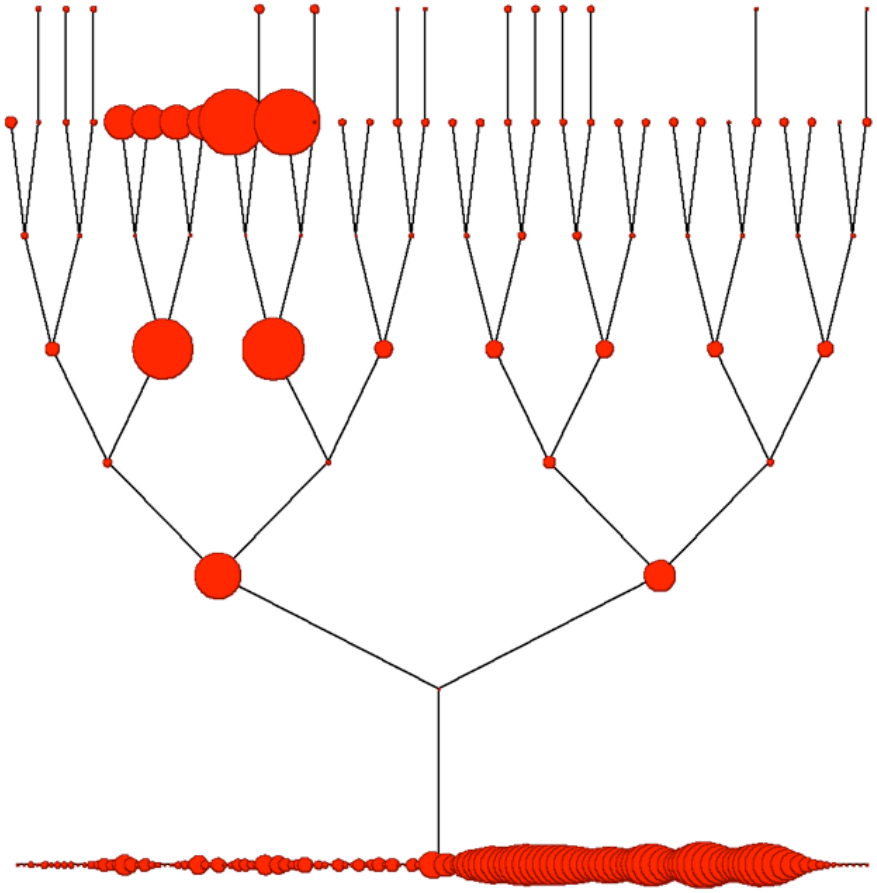
FGG quantum walk $|\psi_t\rangle = e^{iA_G t}|\psi_0\rangle$



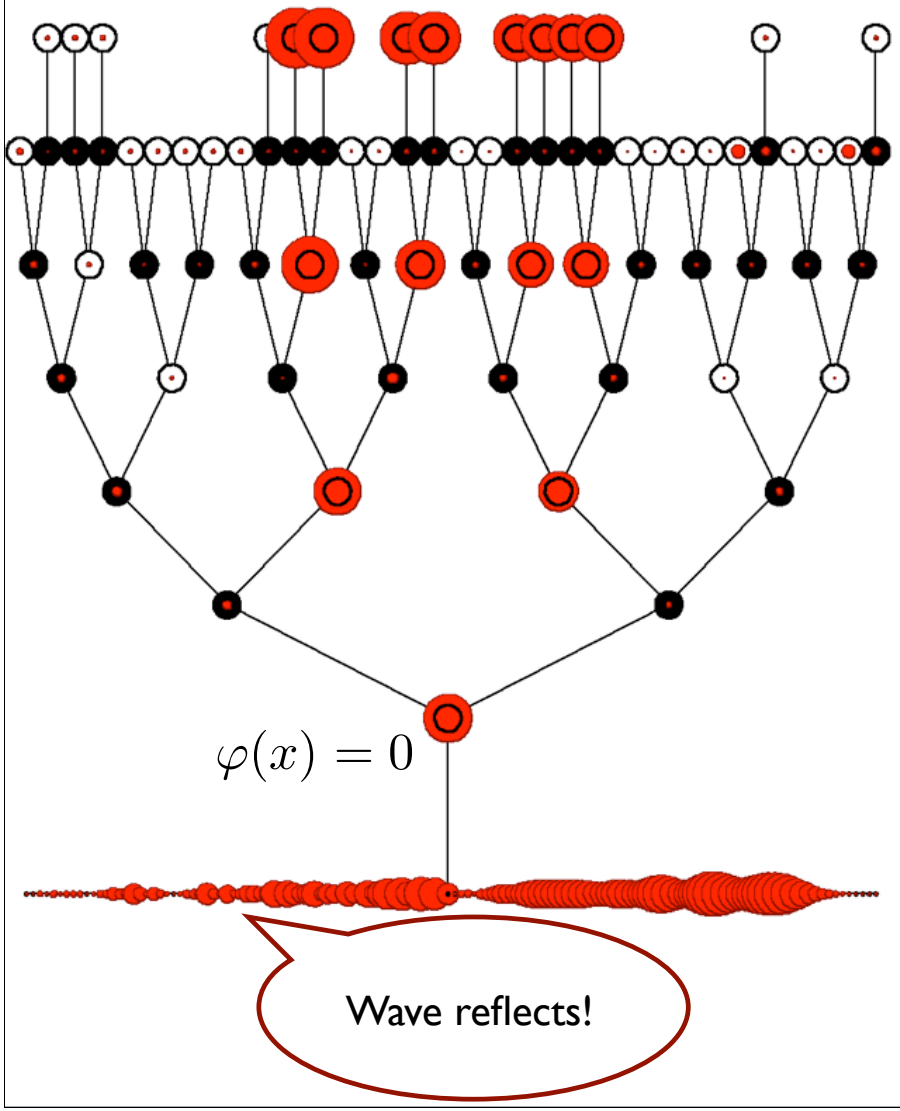


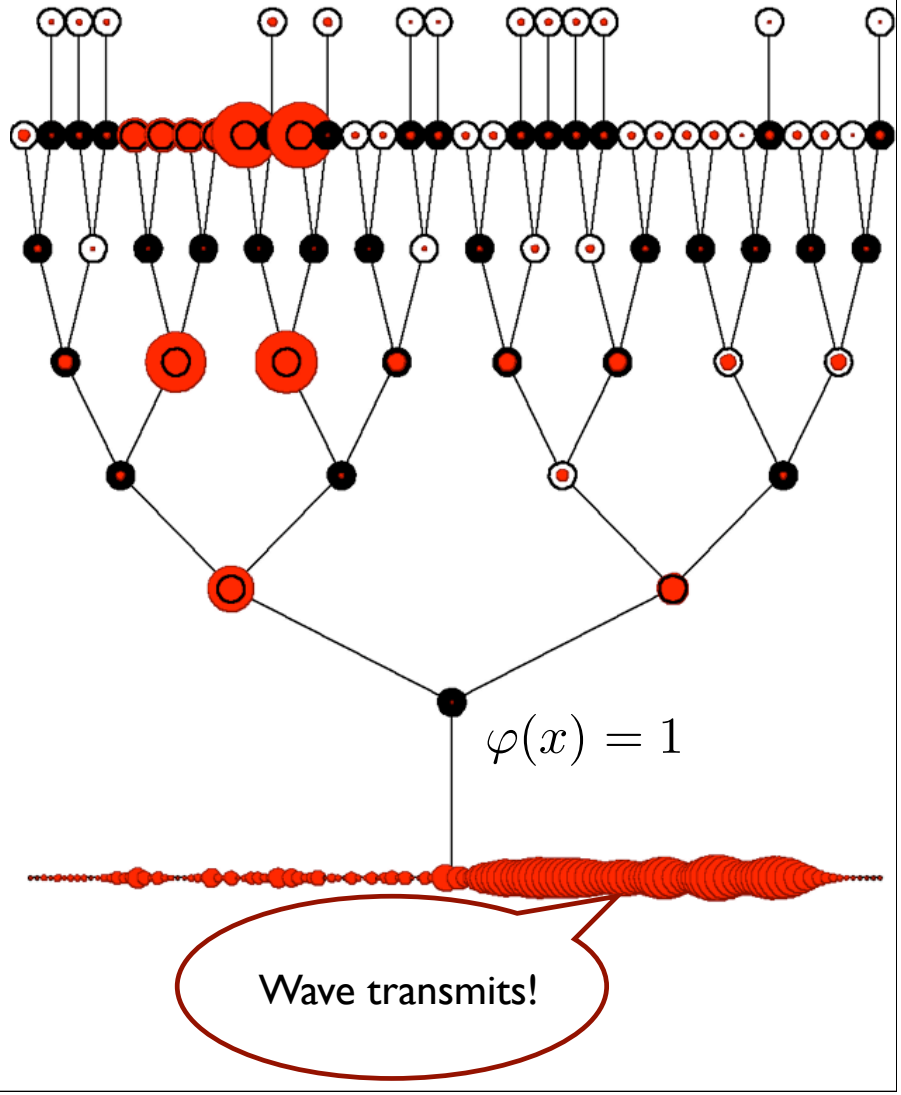
FGG quantum walk $|\psi_t\rangle = e^{iA_G t}|\psi_0\rangle$





FGG quantum walk $|\psi_t\rangle = e^{iA_G t}|\psi_0\rangle$





Farhi, Goldstone, Gutmann '07 algorithm

- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.

Questions:

1. Why does it work?

2. How does it connect to what we know already?

3. How does it generalize?

4. What kinds of problems can we hope to solve with this technique?

Farhi, Goldstone, Gutmann '07 algorithm

- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.

Questions:	Answers:
1. Why does it work?	
2. How does it connect to what we know already?	"span programs" [Karchwer/Wig. '93]
3. How does it generalize?	formula evaluation problem over extended gate sets
4. What kinds of problems can we hope to solve with this technique?	

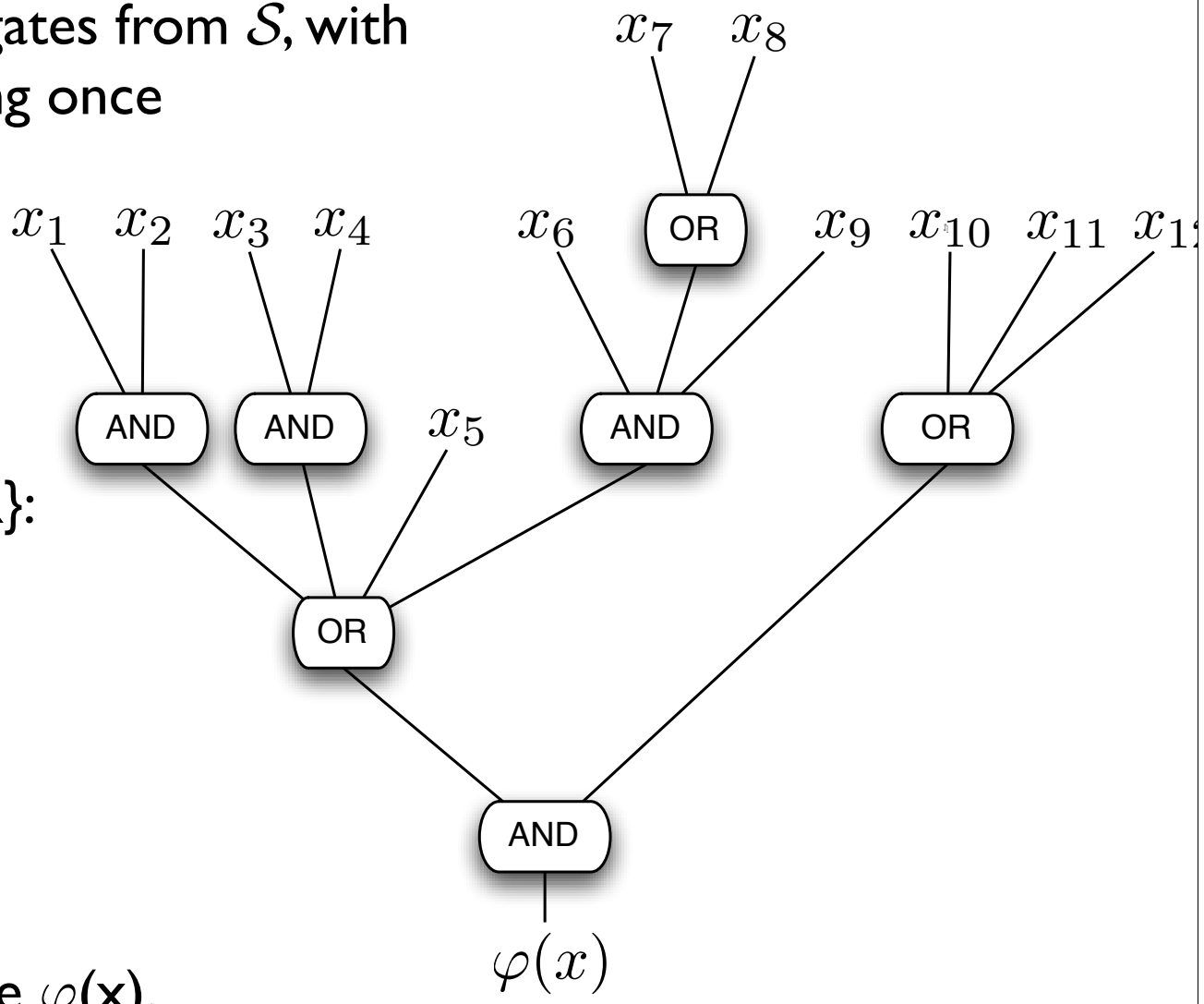
Farhi, Goldstone, Gutmann '07 algorithm

- **Theorem** ([FGG '07, CCJY '07]): A balanced binary NAND formula can be evaluated in time $N^{1/2+o(1)}$.

Questions:	Answers:
1. Why does it work?	
2. How does it connect to what we know already?	"span programs" [Karchwer/Wig. '93]
3. How does it generalize?	formula evaluation problem over extended gate sets
4. What kinds of problems can we hope to solve with this technique?	
5. No, really, *WHY* does it work?	???

Def: Read-once formula φ on gate set $\mathcal{S}$
= Tree of nested gates from $\mathcal{S}$, with each input appearing once

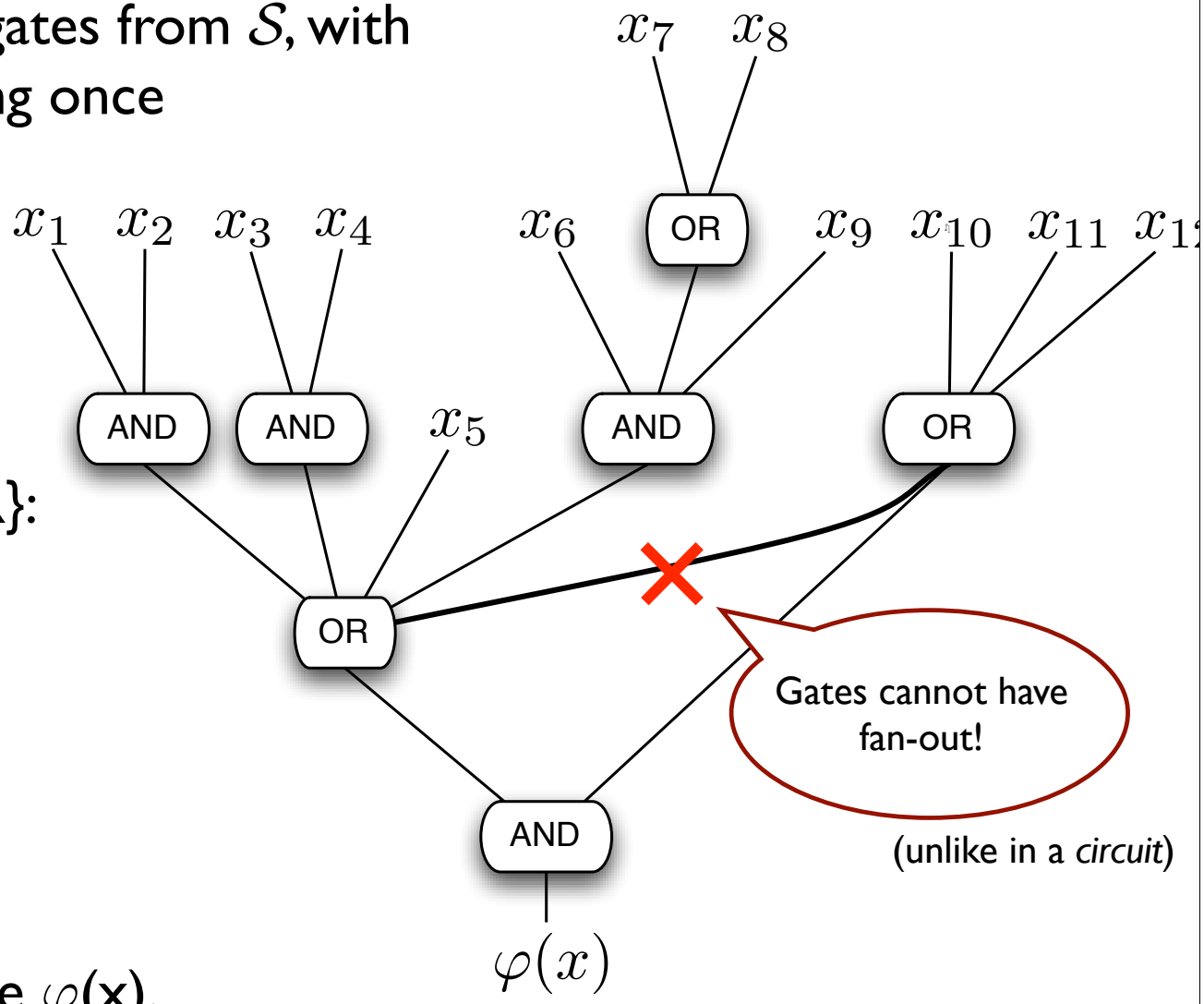
Ex: $\mathcal{S}$ = {AND, OR}:



Problem: Evaluate φ(x).

Def: Read-once formula φ on gate set $\mathcal{S}$
= Tree of nested gates from $\mathcal{S}$, with each input appearing once

Ex: $\mathcal{S}$ = {AND, OR}:

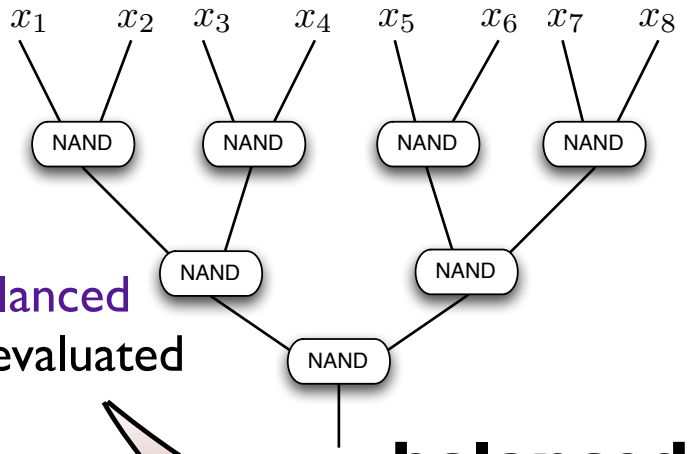


Problem: Evaluate φ(x).

[FGG '07] algorithm

- **Theorem** ([FGG '07, CCJY '07]): A balanced binary AND-OR formula can be evaluated in time $N^{1/2+o(1)}$.

  Analysis by scattering theory.



balanced, more gates

unbalanced AND-OR

[ACRŠZ '07] algorithm

- **Theorem**:
  - An "approximately balanced" AND-OR formula can be evaluated with $O(\sqrt{N})$ queries (optimal for read-once!).
  - A general AND-OR formula can be evaluated with $N^{1/2+o(1)}$ queries.

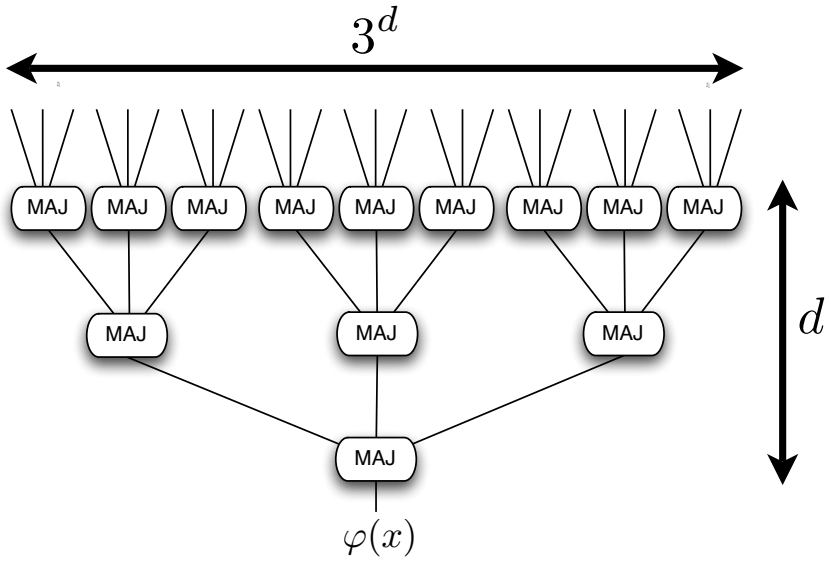
Running time is $N^{1/2+o(1)}$ in each case, after preprocessing.

[RŠ '07] algorithm

- **Theorem:** A balanced ("adversary-bound-balanced") formula φ over a gate set including all three-bit gates (and more…) can be evaluated in $O(ADV(\varphi))$ queries (optimal!).

  (Some gates, e.g., AND, OR, PARITY, can be unbalanced—but not most!)

Recursive 3-bit majority tree



- Best quantum lower bound is
  $\Omega\big(\mathrm{ADV}(\varphi) = 2^d\big)$ [LLS'05]

- Expand majority into {AND, OR} gates:
  $\mathrm{MAJ}_3(x_1, x_2, x_3)$
  $= (x_1 \wedge x_2) \vee (x_3 \wedge (x_1 \vee x_2))$

∴ {AND, OR} formula size is ≤ 5^d

∴ $O(\sqrt{5^d}) = O(2.24^d)$-query algorithm
[FGG, ACRSZ '07]

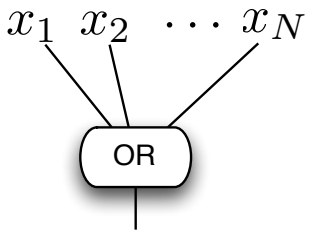
[RŠ '07] algorithm

- **Theorem:** A balanced ("adversary-bound-balanced") formula φ over a gate set including all three-bit gates (and more…) can be evaluated in O(ADV(φ)) queries (optimal!).

  (Some gates, e.g., AND, OR, PARITY, can be unbalanced—but not most!)

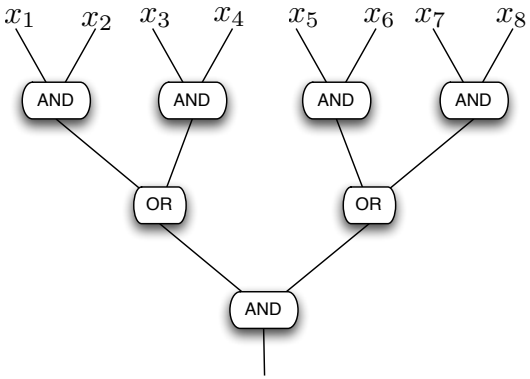
- New: $O(2^d)$-query quantum algorithm

Classical complexity of formula evaluation



Θ(N)

Balanced AND-OR



$\Theta(N^{0.753\ldots})$ [Snir '85, Saks & Wigderson '86, Santha '95]

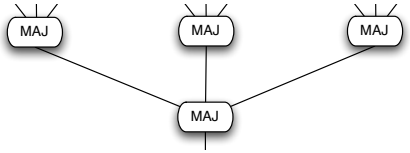
(fan-in two, tight bounds also for "well-balanced" formulas)

General read-once AND-OR

$R(f) = \Omega(N^{0.51})$ [Heiman, W '91]

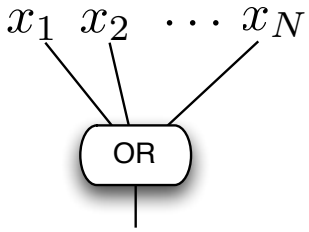
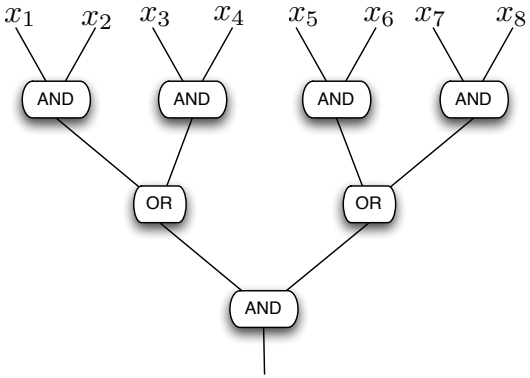
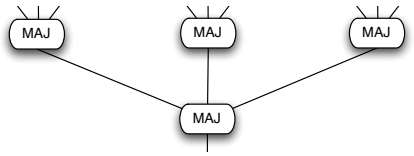
Conjecture: $R(f) = \Omega(D(f)^{0.753\ldots})$ [SW '86]

Balanced MAJ_3

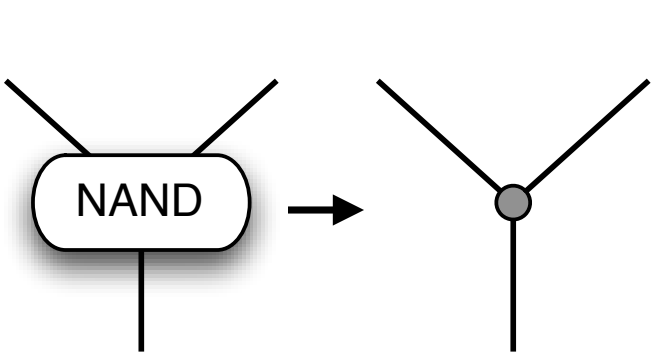


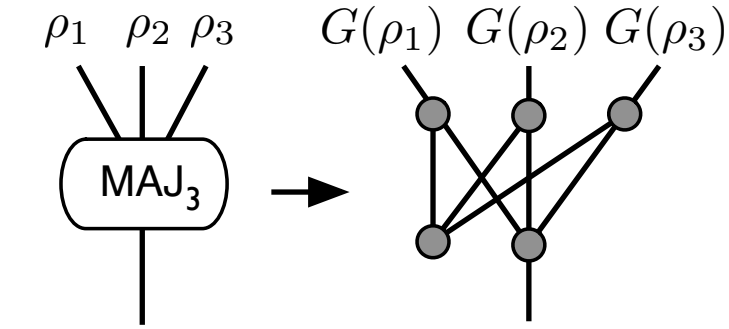
$\Omega((7/3)^{depth}) = R_2(f) = O((2.6537\ldots)^{depth})$

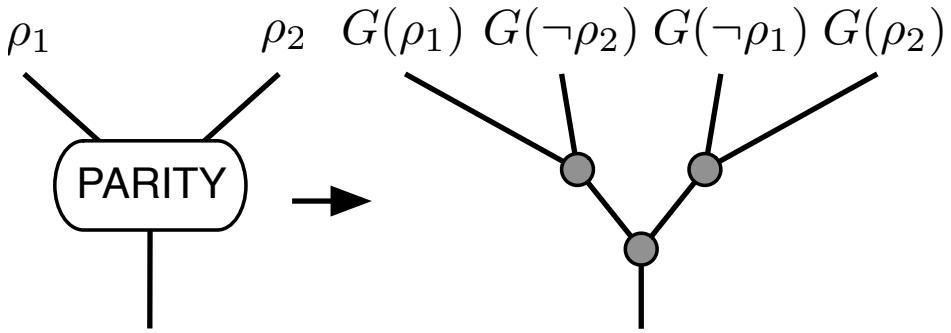
[Jayram, Kumar, Sivakumar '03]

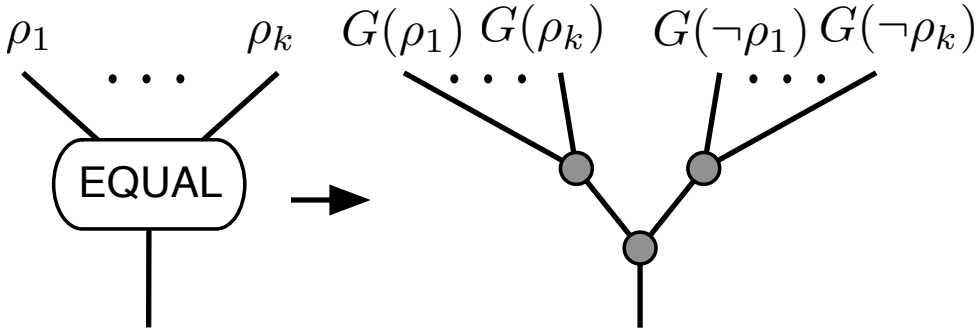
	Classical	Quantum
	$\Theta(N)$	$\Theta(\sqrt{N})$ [Grover '96]
Balanced AND-OR 	$\Theta(N^{0.753\ldots})$ (fan-in two) [S'85, SW'86, S'95]	This morning: $\Theta(\sqrt{N})$ [FGG, ACRŠZ '07]
General read-once AND-OR ⋮	$\Omega(N^{0.51})$ [HW'91] Conj.: $\Omega(D(f)^{0.753\ldots})$ [SW '86]	$\Omega(\sqrt{N})$, $\sqrt{N} \cdot 2^{O(\sqrt{(\log N)})}$ [BS '04] [ACRŠZ '07]
Balanced MAJ_3 	$\Omega((7/3)^d)$, $O((2.6537\ldots)^d)$ [JKS '03]	This afternoon: $\Theta(2^d = N^{\log_3 2})$ and much more…

More substitution rules

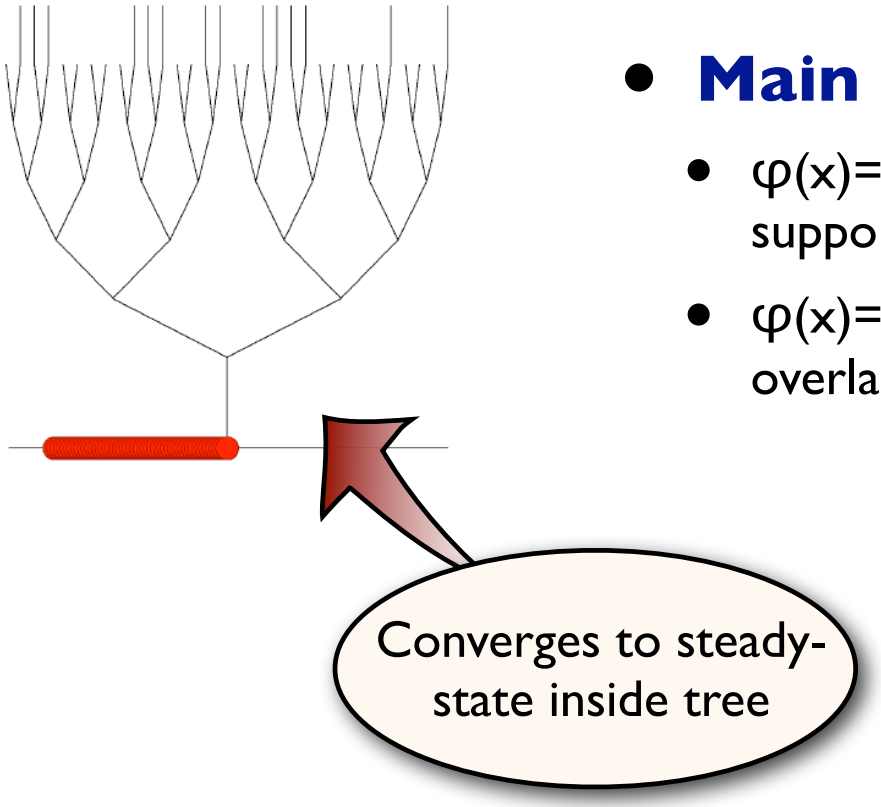




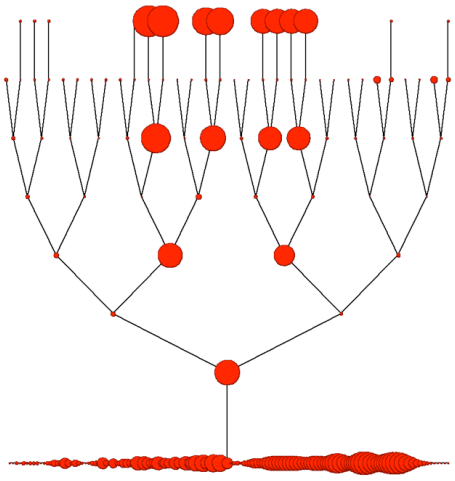




(with appropriate edge weights)



- **Main Theorem:**
  - $\varphi(x)=1 \Rightarrow A_G$ has $\lambda=0$ eigenstate with $\Omega(1)$ support on the root.
  - $\varphi(x)=0 \Rightarrow A_G$ has no eigenvectors overlapping the root with $|\lambda|<1/\sqrt{N}$.



- **Main Theorem:**
  - $\varphi(x)=1 \Rightarrow A_G$ has $\lambda=0$ eigenstate with $\Omega(1)$ support on the root.
  - $\varphi(x)=0 \Rightarrow A_G$ has no eigenvectors overlapping the root with $|\lambda|<2/\sqrt{N}$.

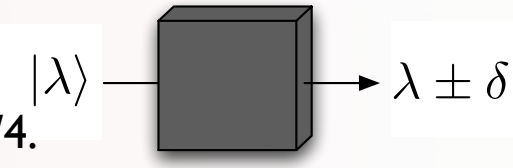
Faster Algorithm: ($2^{\sqrt{\log N}}$ times faster than FGG scattering)

- Start at the root
- Apply phase estimation to the quantum walk with precision $1/\sqrt{N}$
- If measured phase is 0, output "$\varphi(x)=1$." Otherwise, output "$\varphi(x)=0$."

Running time is $\sqrt{N}$

Recall:

Precision-δ phase estimation on a unitary U, starting at an e-state, returns the e-value to precision δ, except w/ prob. 1/4. It uses $O(1/\delta)$ calls to c-U.



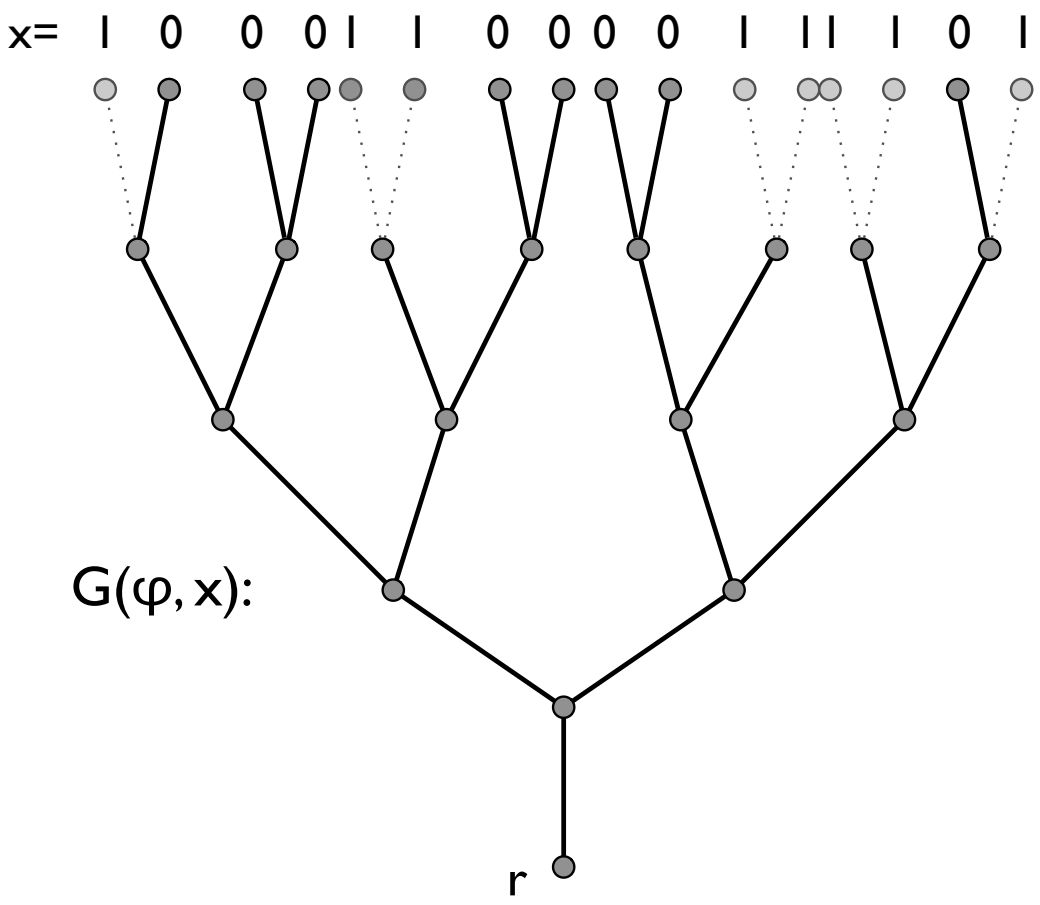
- **Theorem:** $\varphi(x)=1 \iff \exists$ a $\lambda=0$ eigenstate of $A_{G(\varphi,x)}$ supported on root r.

Proof: $\lambda=0$ eigenstate:

Want $|\alpha\rangle = \sum_v \alpha_v |v\rangle$

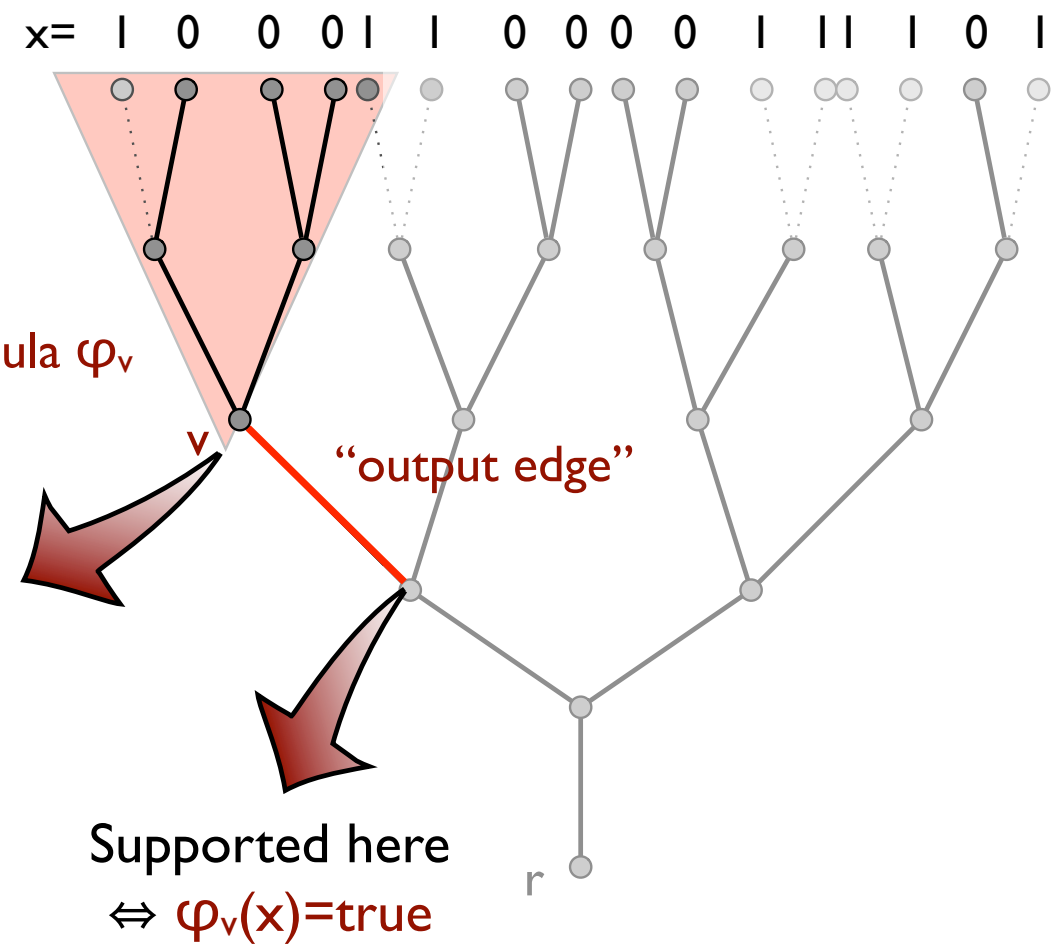
such that $\forall$ v,

$$\langle v|A_G|\alpha\rangle = \sum_{w\sim v} \alpha_w = 0$$



- **Theorem:** $\varphi(x)=1 \iff \exists$ a $\lambda=0$ eigenstate of $A_{G(\varphi,x)}$ supported on root r.

Induction Claim: Each edge gives a "dual-rail" encoding for the evaluation of the subformula above that edge…



The $\lambda=0$ eigenstate of $G(\varphi_v,x)$ is:

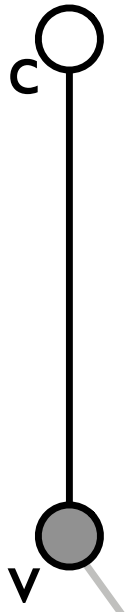
Supported here
$\Leftrightarrow \varphi_v(x)$=false

Supported here
$\Leftrightarrow \varphi_v(x)$=true

- **Proof of the Induction Claim:**

Base case: v an input

x_i=0:

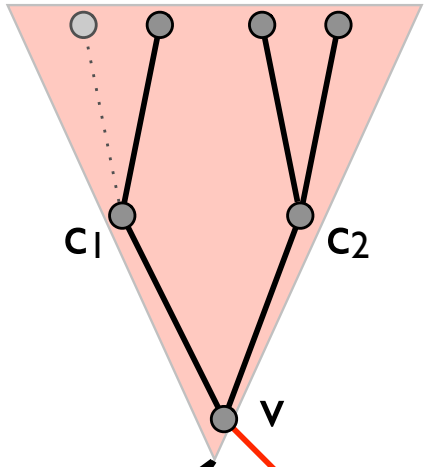


x_i=1:

λ=0 eigenvector constraint at c is α_v=0. ✓

α_v is not constrained since v and c are not connected.✓

Induction Claim: Each edge (p,v) gives a "dual-rail" encoding…



The $\lambda=0$ eigenstate of $G(\varphi_v,x)$ is:

Supported on v
$\Leftrightarrow \varphi_v(x)=\text{false}$

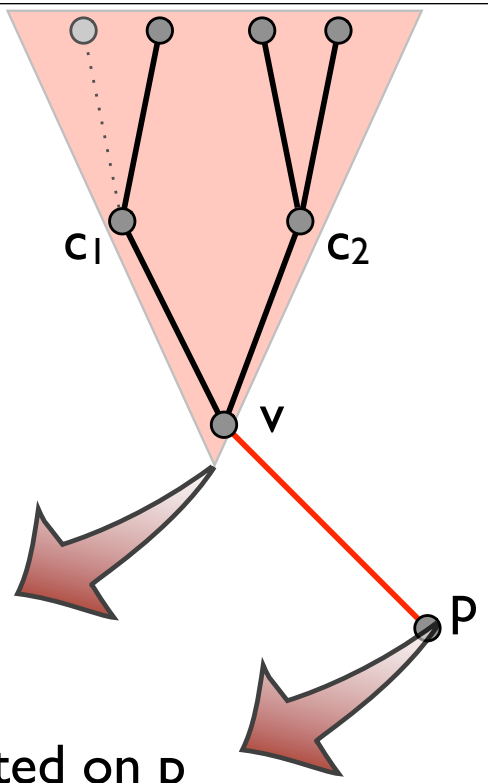
Supported on p
$\Leftrightarrow \varphi_v(x)=\text{true}$

Proof (induction step):

- $\alpha_v \neq 0 \Leftrightarrow$ both input subtrees φ_{c_1} and φ_{c_2} are true $\Leftrightarrow \text{NAND}(\varphi_{c_1},\varphi_{c_2}) = 0$
- Since $\alpha_p+\alpha_{c_1}+\alpha_{c_2} = 0$, α_p can be $\neq 0 \Leftrightarrow$ either $\alpha_{c_i} \neq 0 \Leftrightarrow \text{NAND}(\varphi_{c_1},\varphi_{c_2}) = 1$

☑

Induction Claim: Each edge (p,v) gives a "dual-rail" encoding…



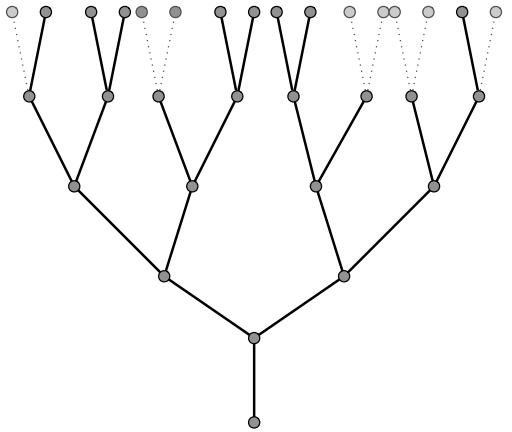
The $\lambda=0$ eigenstate of $G(\varphi_v,x)$ is:

Supported on v
$\Leftrightarrow \varphi_v(x)=$false

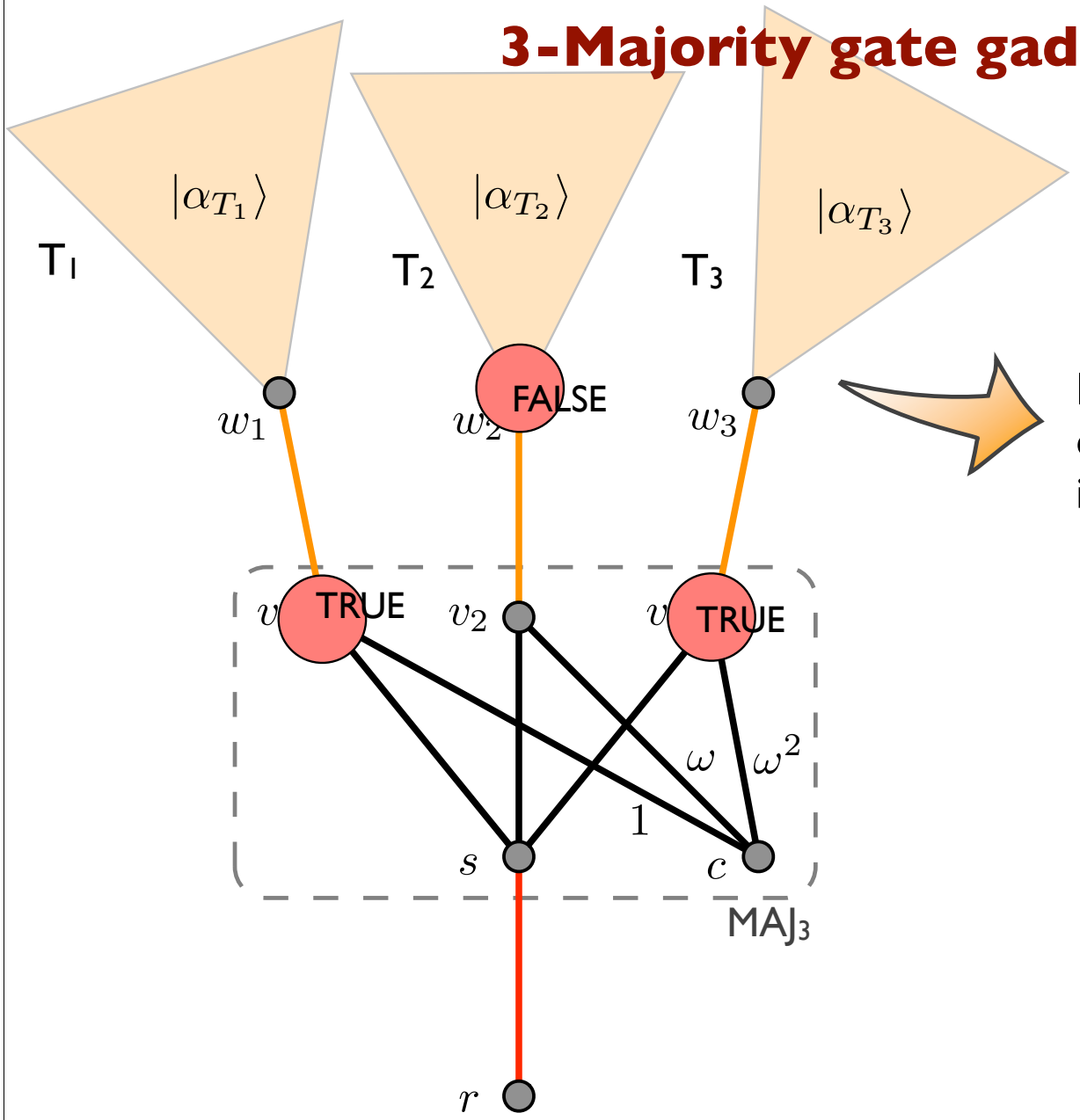
Supported on p
$\Leftrightarrow \varphi_v(x)=$true □

- **Theorem:** $\varphi(x)=1 \iff \exists$ a $\lambda=0$ eigenstate of $A_{G(\varphi,x)}$ supported on root r. □

- **Main Theorem:**
  - $\varphi(x)=1 \Rightarrow A_G$ has eigenvalue-0 e.v. with $\Omega(1)$ support on the root.
  - $\varphi(x)=0 \Rightarrow A_G$ has no eigenvectors overlapping the root with $|\lambda|<1/\sqrt{N}$.

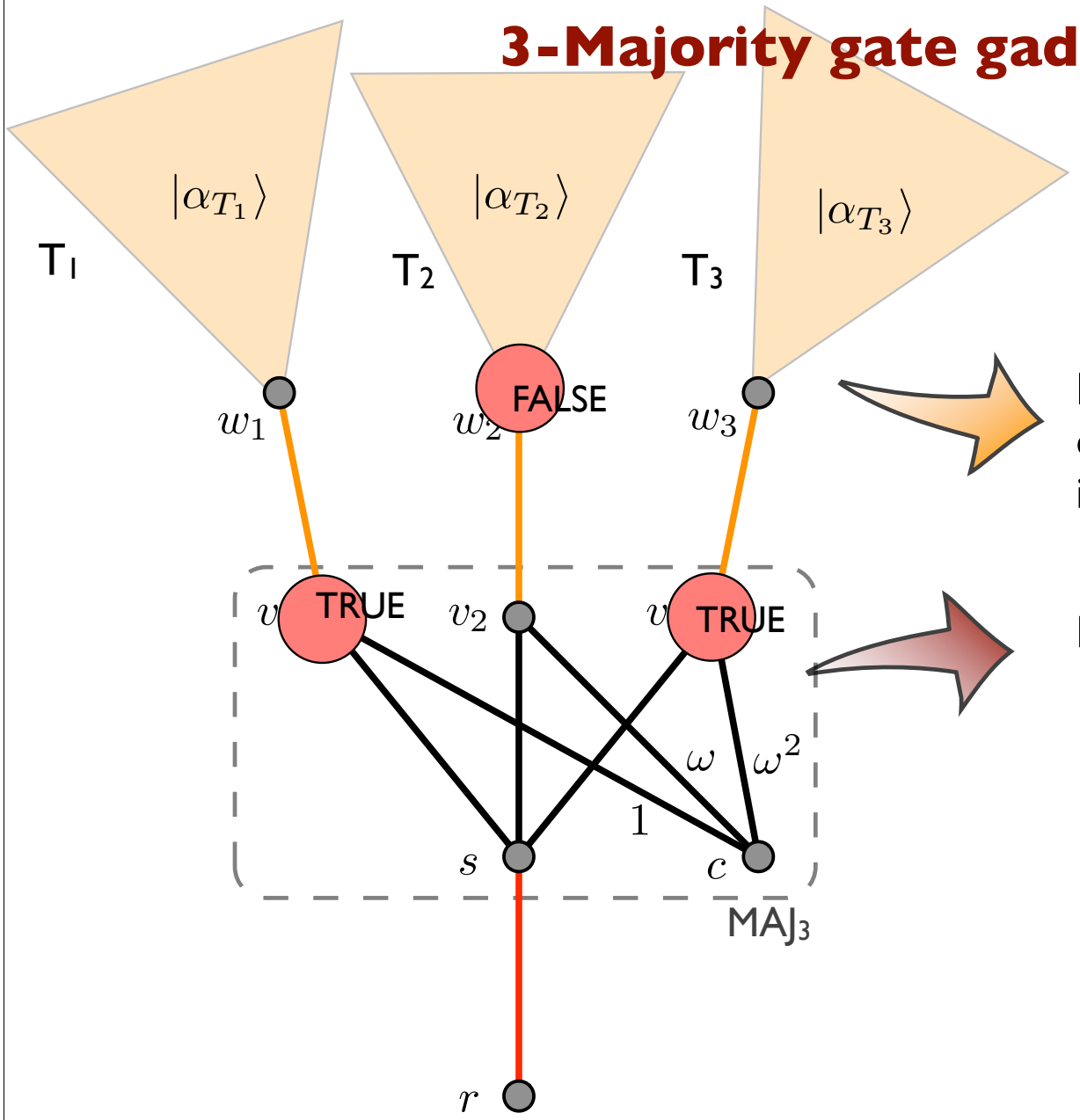


3-Majority gate gadget



Input subformulas with λ=0 eigenstates constructed by induction

3-Majority gate gadget



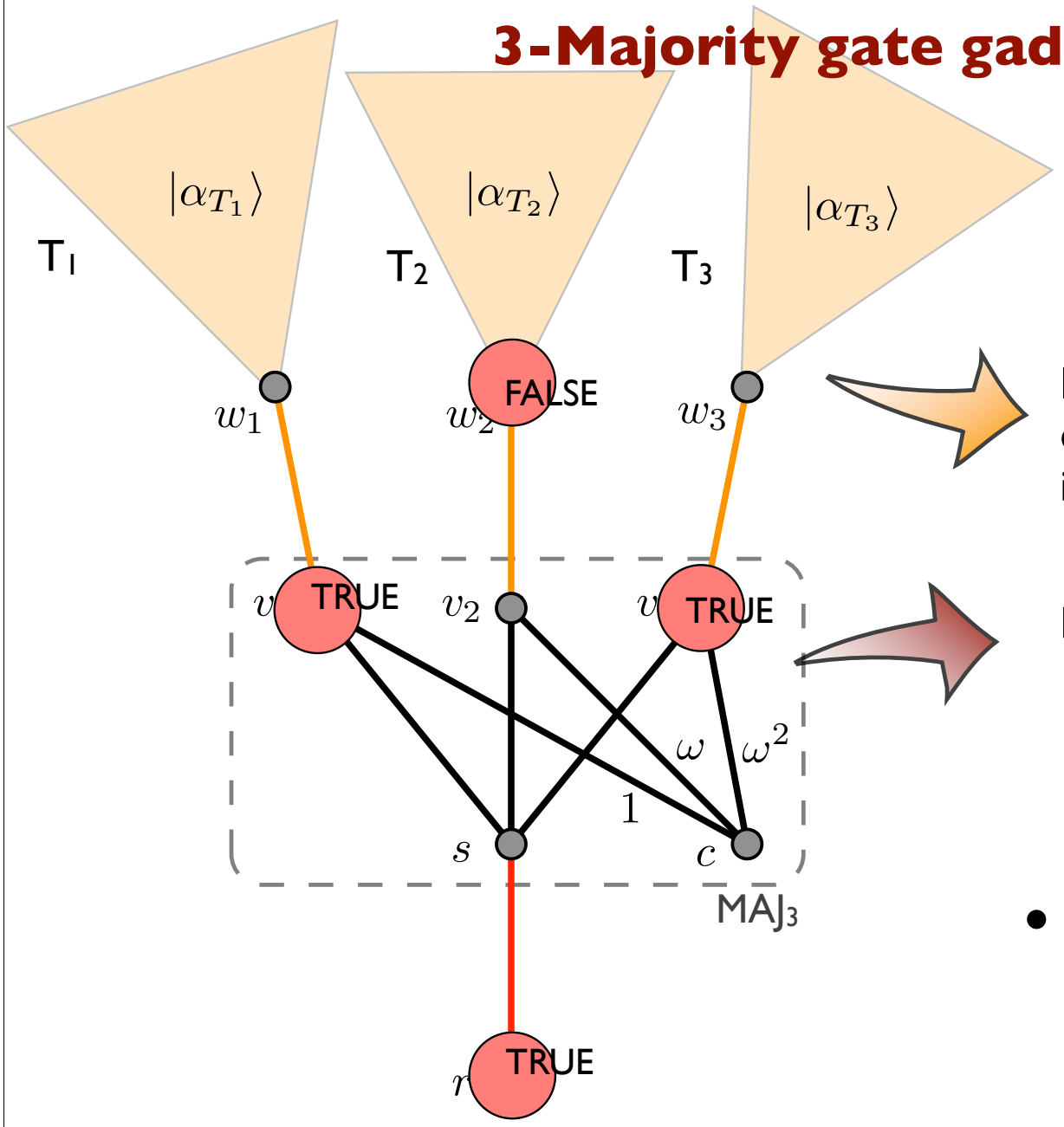
Input subformulas with λ=0 eigenstates constructed by induction

Five new constraints

$$-\alpha_r = \alpha_{v_1} + \alpha_{v_2} + \alpha_{v_3}$$

$$\alpha_{v_1} + \omega\alpha_{v_2} + \omega^2\alpha_{v_3} = 0$$

3-Majority gate gadget



Input subformulas with λ=0 eigenstates constructed by induction

Five new constraints

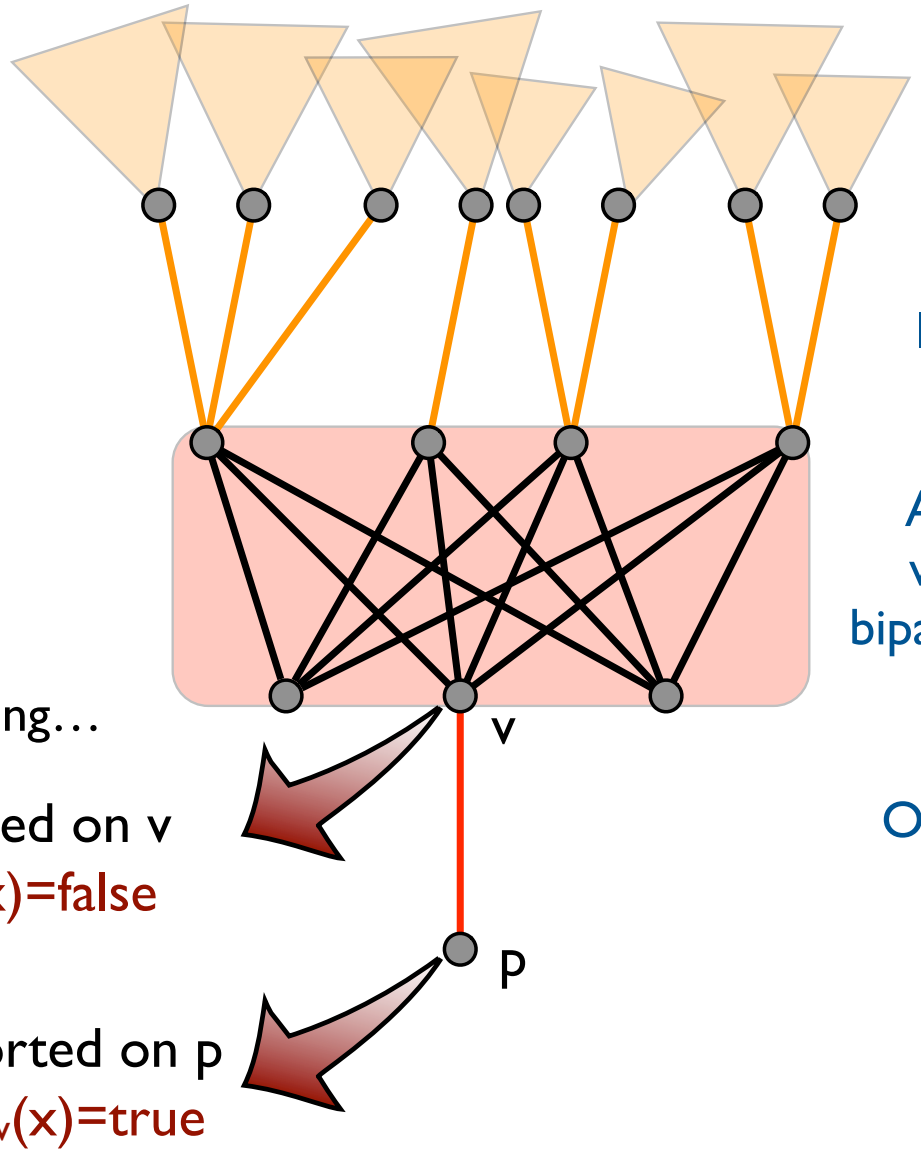
$$-\alpha_r = \alpha_{v_1} + \alpha_{v_2} + \alpha_{v_3}$$

$$\alpha_{v_1} + \omega\alpha_{v_2} + \omega^2\alpha_{v_3} = 0$$

- At least two $\varphi(v_i)$ must be 1 to satisfy second constraint nontrivially.

✓ MAJ3

General graph gadgets



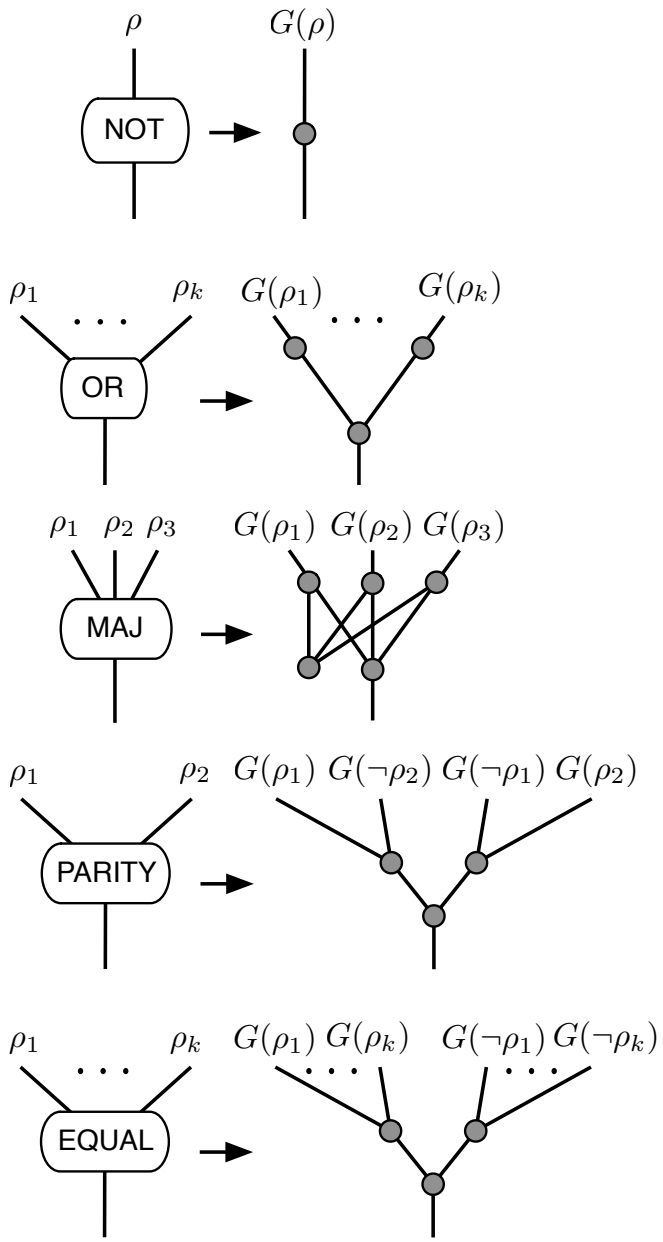
Induction Claim: Each edge (p,v) gives a "dual-rail" encoding…

The $\lambda=0$ eigenstate of $G(\varphi_v,x)$ is:

Supported on v
$\Leftrightarrow \varphi_v(x)=\text{false}$

Supported on p
$\Leftrightarrow \varphi_v(x)=\text{true}$

Substitution rules



Input dependence

- Substitutions define $G(0^N)$; to define graph G(x), delete edges to all leaves evaluating to $x_i=1$.

Main Theorem:

- $\varphi(x)=1 \Rightarrow A_G$ has $\lambda=0$ eigenstate with $\Omega(1)$ support on the root.
- $\varphi(x)=0 \Rightarrow A_G$ has no eigenvectors overlapping the root with $|\lambda| < 1/\Omega(ADV(\varphi))$.

($\therefore$ O(ADV(φ))-query algorithm)

Span program definition

- Substitution rules defining G come from *span programs*. [Karchmer, Wigderson '93]
- **Def:** A span program P is:
  - A target vector t in vector space V over **C**,
  - Input vectors v_j each associated with a literal from $\{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$

Span program P computes f_P: $\{0,1\}^n \to \{0,1\}$,
$f_P(x) = 1 \Leftrightarrow$ t lies in the span of { true v_j }

- **Ex. 1:** P:

$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{matrix} x_1 \\ \begin{pmatrix} 1 \\ a \end{pmatrix} \end{matrix} \quad \begin{matrix} x_2 \\ \begin{pmatrix} 1 \\ b \end{pmatrix} \end{matrix} \quad \begin{matrix} x_3 \\ \begin{pmatrix} 1 \\ c \end{pmatrix} \end{matrix}$$

with a,b,c distinct and nonzero.

➡ $f_P = MAJ_3$

Span program examples

- **Ex. 2:** G a graph with marked source and sink vertices $\mathcal{H} = \mathbf{C}^{|V|}$

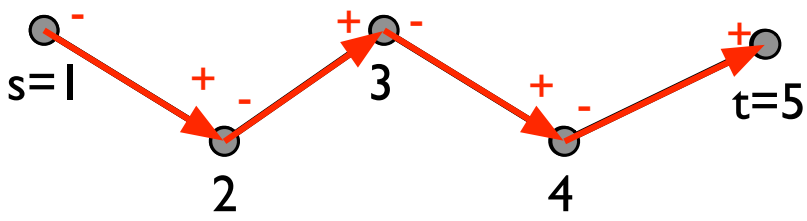
$$|t\rangle = |\text{sink}\rangle - |\text{source}\rangle \qquad |v_{(i,j)}\rangle = |i\rangle - |j\rangle$$

usable if edge (i,j) present

  - f_P = undirected s-t connectivity (USTCON)

E.g.,

t	$e_{(1,2)}$	$e_{(2,3)}$	$e_{(3,4)}$	$e_{(4,5)}$	$e_{(1,3)}$	$e_{(1,4)}$	⋯
-1	-1	0	0	0	-1	-1	
0	1	-1	0	0	0	0	
0	0	1	-1	0	1	0	⋯
0	0	0	1	-1	0	1	
1	0	0	0	1	0	0	



Linear combinations of the edge vectors correspond to flows in the graph.

Span program examples

- **Ex. 2:** G a graph with marked source and sink vertices $\mathcal{H} = \mathbf{C}^{|V|}$

  $|t\rangle = |\text{sink}\rangle - |\text{source}\rangle \qquad |v_{(i,j)}\rangle = |i\rangle - |j\rangle$

  - f_P = undirected s-t connectivity (USTCON)

- **Ex. 3:** Feasible system of linear equations (FSLE)

  Does M(x) y = b have a solution y?

  poly-size matrix with entries log-degree polynomials in $x_1, \ldots, x_N, \bar{x}_1, \ldots, \bar{x}_N$

  vector of constants

  - Span program is FSLE instance in which each column of M depends linearly on only a single literal x_i or $\bar{x}_i$
  - General FSLE instances can be converted to span programs with poly overhead

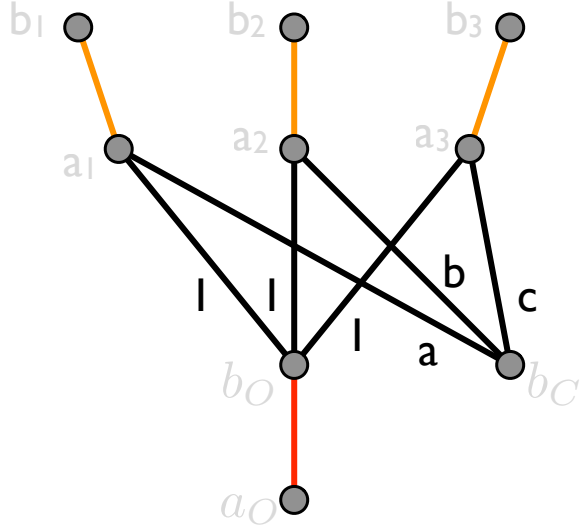
[Allender, Beals, Ogihara '99]

Span program ⇔ Bipartite graph gadget

with t=(1,0,…,0)

E.g., MAJ_3:

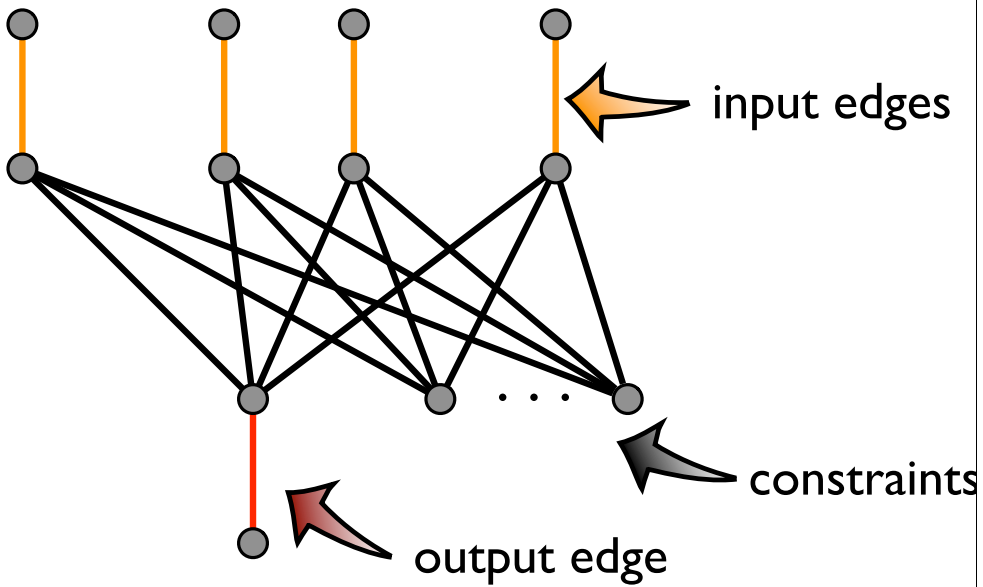
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{matrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \\ a & b & c \end{matrix}$$



In general:

$$t = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

A



Composing span programs

- Given span programs for g, h_1, …, h_k, immediately get s.p. for

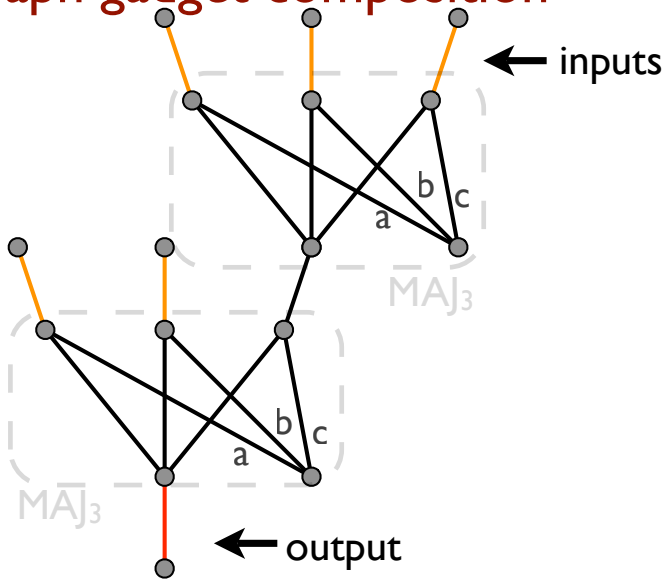
$$f = g \circ (h_1, \ldots, h_k)$$

SP composition

- Ex.: $MAJ_3(x_1, x_2, MAJ_3(x_4,x_5,x_6))$:

t	x_1	x_2	1	x_3	x_4	x_5
1	1	1	1	0	0	0
0	a	b	c	0	0	0
0	0	0	1	1	1	1
0	0	0	0	a	b	c

Graph gadget composition



Composing span programs

- Given span programs for g, h_1, …, h_k, immediately get s.p. for

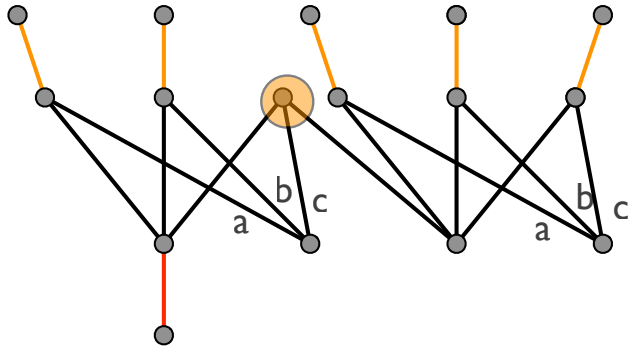
$$f = g \circ (h_1, \ldots, h_k)$$

SP composition

- Ex.: $MAJ_3(x_1, x_2, MAJ_3(x_4,x_5,x_6))$:

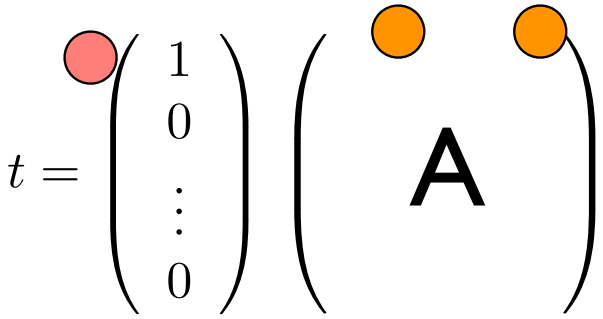
$\underline{\underline{t}}$	x_1	x_2	1	x_3	x_4	x_5
1	1	1	1	0	0	0
0	a	b	c	0	0	0
0	0	0	1	1	1	1
0	0	0	0	a	b	c

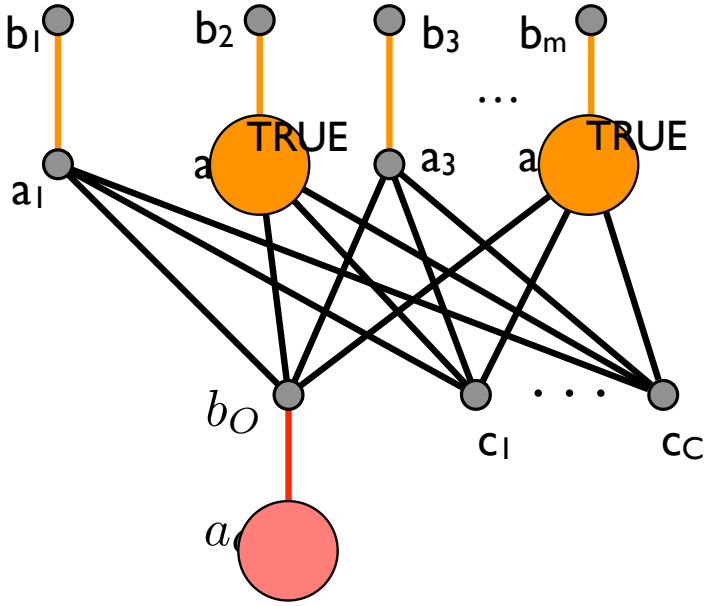
Graph gadget composition



Eigenvalue-zero lemmas

- **Define:** $G_P(x)$ by deleting edges to true input literals
- **Lemma:** $f_P(x)=1 \Leftrightarrow \exists\ \lambda=0$ eigenstate of $A_{G_P(x)}$ supported on a_O.

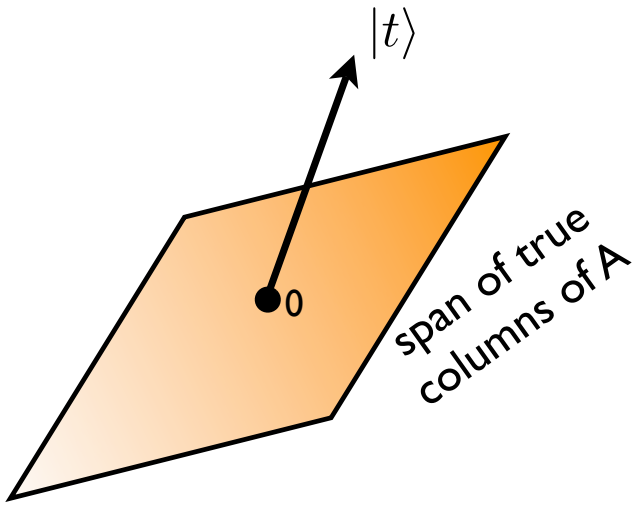


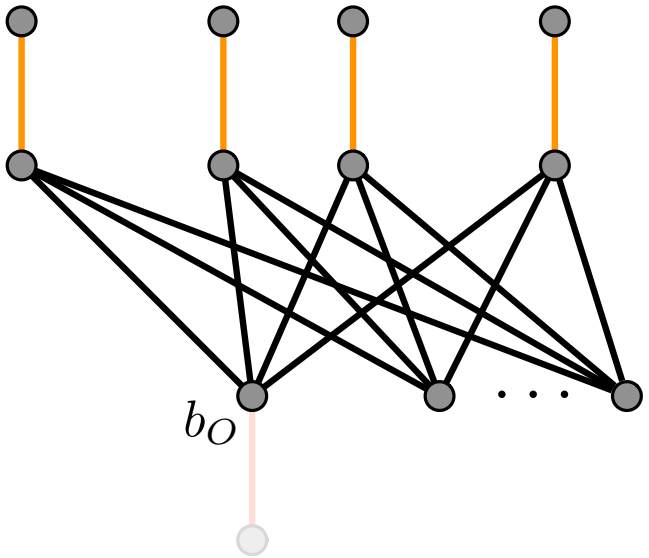


Eigenvalue-zero lemmas

- **Define:** $G_P(x)$ by deleting edges to true input literals
- **Lemma:** $f_P(x)=1 \Leftrightarrow \exists\ \lambda=0$ eigenstate of $A_{G_P(x)}$ supported on a_O.
- **Lemma:** Delete output edge (a_O, b_O). Then
  $f_P(x)=0 \Leftrightarrow \exists\ \lambda=0$ eigenstate supported on b_O.

Proof: $f_P(x)$ is false $\Leftrightarrow |t\rangle$ *not* in span of true columns of A



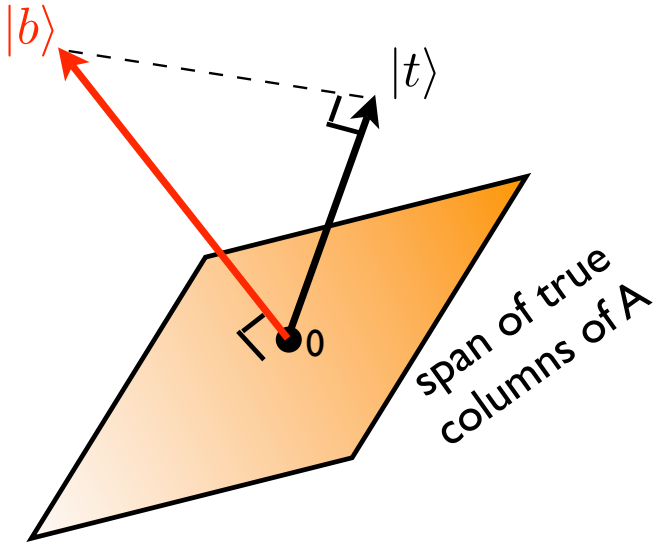


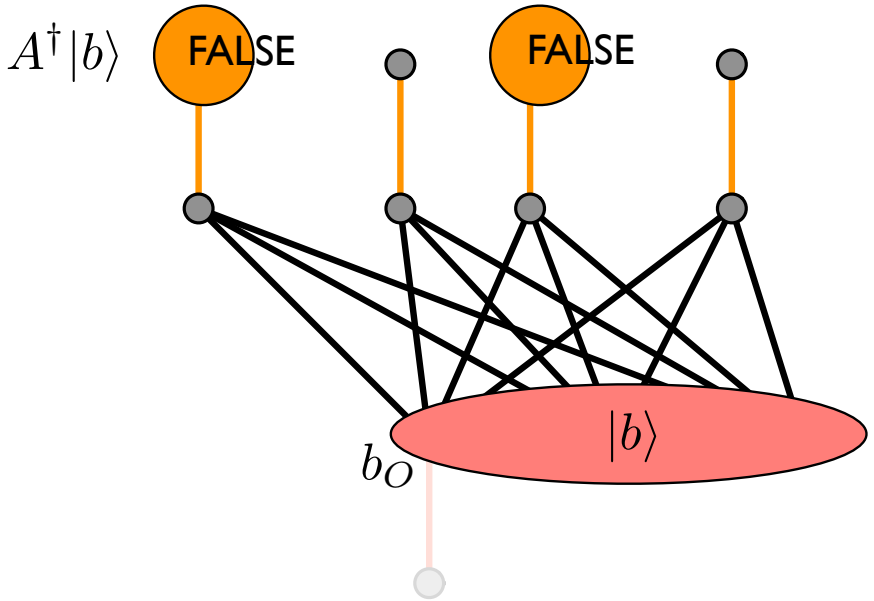
Eigenvalue-zero lemmas

- **Define:** $G_P(x)$ by deleting edges to true input literals
- **Lemma:** $f_P(x)=1 \Leftrightarrow \exists\ \lambda=0$ eigenstate of $A_{G_P(x)}$ supported on a_O.
- **Lemma:** Delete output edge (a_O, b_O). Then
  $f_P(x)=0 \Leftrightarrow \exists\ \lambda=0$ eigenstate supported on b_O.

Proof: $f_P(x)$ is false $\Leftrightarrow |t\rangle$ *not* in span of true columns of A

$\Leftrightarrow \exists\ |b\rangle$ with $\langle b|t\rangle=1$, orthogonal to all true columns of A

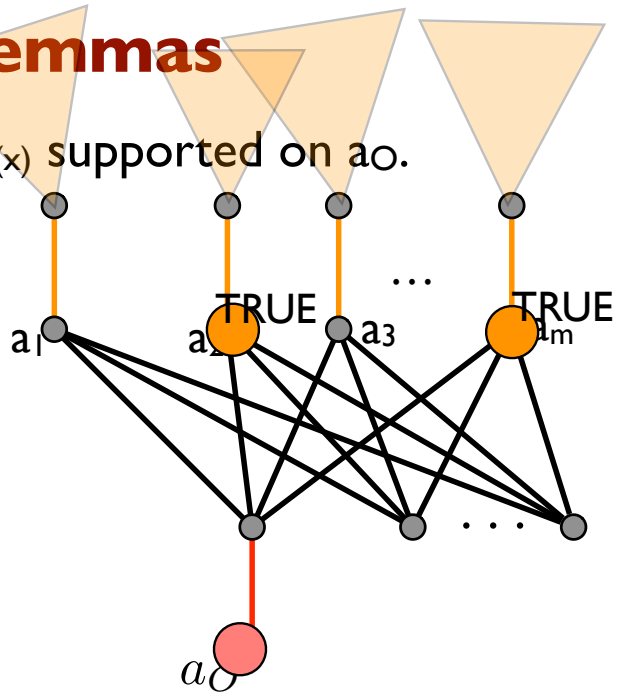




Quantitative Eigenvalue-zero lemmas

- **Lemma:** $f_P(x)=1 \Leftrightarrow \exists\ \lambda=0$ eigenstate of $A_{G_P(x)}$ supported on a_O.

$$t = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} & \\ A & \\ & \end{pmatrix}$$



- Assume that f(x)=1, and that for all true inputs i, we have constructed normalized $\lambda=0$ eigenstates with squared support $\geq \gamma$ on a_i. **Q:** How large can we make $|a_O|^2$ in a normalized $\lambda=0$ eigenstate?

- **Answer:** Fix $a_O=1$ and try to minimize the eigenstate's norm. We want the shortest witness vector:

$$\min_{\substack{|w\rangle: \Pi|w\rangle=|w\rangle \\ A|w\rangle=|t\rangle}} \||w\rangle\|^2 = \|(A\Pi)^- |t\rangle\|^2$$

$:= spc(P,x)$

Π = projection onto true input coords.

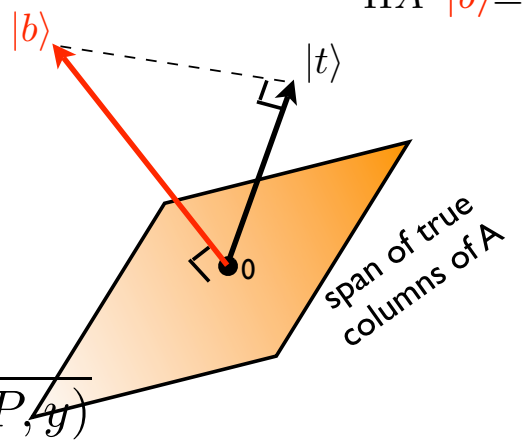
Quantitative Eigenvalue-zero lemmas

- **Assume:** For all inputs i we have constructed normalized λ=0 eigenstates with squared support ≥ γ on a_i or b_i.
- **Lemma:** f(x)=1 ⇒ ∃ unit-normalized λ=0 eigenstate with

$$|a_O|^2 \geq \frac{\gamma}{\mathrm{spc}(P,x)} \qquad \mathrm{spc}(P,x) := \min_{|w\rangle:\ \substack{\Pi|w\rangle=|w\rangle \\ A|w\rangle=|t\rangle}} \||w\rangle\|^2$$

- **Lemma:** f(x)=0 ⇒ ∃ unit-normalized λ=0 eigenstate with

$$|b_O|^2 \geq \frac{\gamma}{\mathrm{spc}(P,x)} \qquad \mathrm{spc}(P,x) := \min_{|b\rangle:\ \substack{\langle t|b\rangle=1 \\ \Pi A^\dagger|b\rangle=0}} \|A^\dagger|b\rangle\|^2$$



- **Def:** Span program complexity of P

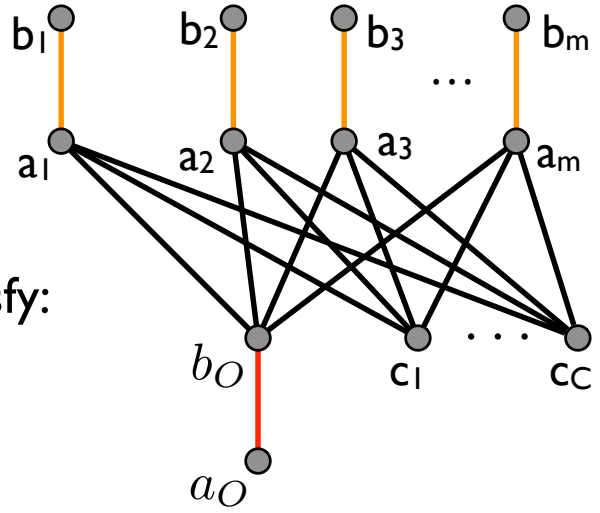
$$\mathrm{spc}(P) = \max_{x,y:f_P(x)\neq f_P(y)} \sqrt{\mathrm{spc}(P,x)\mathrm{spc}(P,y)}$$

$$= \|1 + (\overline{\Pi}A^-A - 1)^-\Pi)A^-|t\rangle\|^2$$

Small λ≠0 analysis

- Construct the eigenvectors starting at the leaves, and working down. Eigenvector equations are

$$\lambda b_C = A_{CJ} a_J$$
$$\lambda b_O = A_{OJ} a_J + a_O$$
$$\lambda a_J = A_{IJ}{}^\dagger b_I + A_{OJ}{}^\dagger b_O + A_{CJ}{}^\dagger b_C$$



- **Induction assumption:** Input ratios $r_i = a_i/b_i$ satisfy:

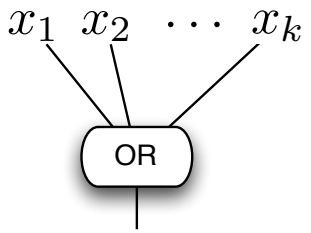
i false ⇒ $r_i \in (0, s_i\lambda)$

i true ⇒ $r_i \in \left(-\infty, \frac{-1}{s_i\lambda}\right)$

- Solve equations for $r_O = a_O/b_O$, apply Woodbury, expand the Taylor series in λ of the matrix inverse (on the range and its Schur complement separately), bound the higher-order terms, QED.
  - The first-order term is the same as the factor spc(P, x) lost in the λ=0 analysis (not so surprisingly)

Span program complexity of OR function

P= $t = (1)$ $\begin{pmatrix} x_1 & x_2 & x_3 & \cdots & x_k \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$

f_P= 

$$\mathrm{spc}(P,x) := \min_{|w\rangle:\ \substack{\Pi|w\rangle=|w\rangle \\ A|w\rangle=|t\rangle}} \||w\rangle\|^2 \quad \text{(if } f_P(x)=1\text{)}$$

$$\min_{|b\rangle:\ \substack{\langle t|b\rangle=1 \\ \Pi A^\dagger|b\rangle=0}} \|A^\dagger|b\rangle\|^2 \quad \text{(if } f_P(x)=0\text{)}$$

- Shortest witness to OR(x) = true is

  $|w\rangle = \frac{1}{|x|} \sum_{i:x_i=1} |i\rangle$, squared length $1/\sqrt{|x|} \le 1$

- Shortest witness to OR(x) = false is

  $|b\rangle = (1)$, squared length of $A^\dagger|b\rangle = (1,1,\ldots,1)$ is k

$\therefore$ span program complexity $\mathrm{spc}(\mathrm{OR}_k) = \sqrt{(1 \cdot k)} = \sqrt{k} = \mathrm{ADV}(\mathrm{OR}_k)$

Span program complexity of MAJ_3 function

P= $t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\begin{matrix} \{x_1\} & \{x_2\} & \{x_3\} \end{matrix}$

$\begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 1 & \omega & \omega^2 \end{pmatrix}$ $f_P = MAJ_3$

$\omega = e^{2\pi i/3}$

- x=110: Shortest witness is $|w\rangle = (1, \omega^2, 0)$, squared length 2
- x=001: Ditto

$\therefore$ span program complexity spc(MAJ_3) = $\sqrt{(2 \cdot 2)} = 2 = ADV(MAJ_3)$

Span program complexity of Threshold$_{2\text{ of }4}$

Plug into *Mathematica*:

```
In[66]:= substitutions = {aw → Sqrt[3/2] / 2, ax → 0, ay → 0, az → 0, bw → -1/(2 Sqrt[6]), bx → 1/Sqrt[3], by → 0, bz → 0,
  cw → -1/(2 Sqrt[6]), cx → -1/(2 Sqrt[3]), cy → -1/2, cz → 0, dw → -1/(2 Sqrt[6]), dx → -1/(2 Sqrt[3]), dy → 1/2, dz → 0};

  pevalSPC[
   {{t, {}, {}, {}, {}, x1, x1, x1, x1, x2, x2, x2, x2, x3, x3, x3, x3, x4, x4, x4, x4},
    {1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0},
    {0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, aw, -ax, -ay, -az, bw, -bx, -by, -bz, cw, -cx, -cy, -cz, dw, -dx, -dy, -dz},
    {0, 0, 0, 0, 0, ax, aw, -az, ay, bx, bw, -bz, by, cx, cw, -cz, cy, dx, dw, -dz, dy},
    {0, 0, 0, 0, 0, ay, az, aw, -ax, by, bz, bw, -bx, cy, cz, cw, -cx, dy, dz, dw, -dx},
    {0, 0, 0, 0, 0, az, -ay, ax, aw, bz, -by, bx, bw, cz, -cy, cx, cw, dz, -dy, dx, dw}} /.
   substitutions]

Function computed is ⌗279        ← Threshold2of4

Out[68]= Sqrt[6]                 ← ADV(Thr2of4)
```

- Span program complexity is easy to compute
- Many more functions are on our web page…
- Coming up with span programs isn't easy, though

$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{matrix} x_1 & x_2 & x_3 & x_4 \\ 1 & 1 & 1 & 1 \\ a & b & c & d \end{matrix}$$

Summary of technical results

- **Def:** Let $\mathcal{S}'$ = { arbitrary two- or three-bit gates, O(1)-fan-in EQUAL gates}
  Let $\mathcal{S}$ = { O(1)-size {AND, OR, NOT, PARITY} formulas on inputs that are themselves possibly elements of $\mathcal{S}'$
- E.g., $\mathrm{MAJ}_3(x_1, x_2, x_3) \wedge (x_4 \oplus x_5 \oplus \cdots \oplus (x_{
- (Idea: Gates other than AND, OR, PARITY need to
  AND, OR, PARITY gates can have constant-factor unb

For every gate f in S, we have a span program P with f_P=f and spc(P) = ADV(f)

- **Def:** Read-once formula φ is "adversary-bound-balanced" if for each gate g, the adversary bounds for its input subformulas are all the same.
- **Main Theorem:** Any adversary-balanced formula φ over gate set $\mathcal{S}$ can be evaluated in O(ADV(φ)) queries.

  Time complexity is the same, up to poly-log N factor, in coherent RAM model after preprocessing. [In many (all?) cases, this caveat can be removed.]

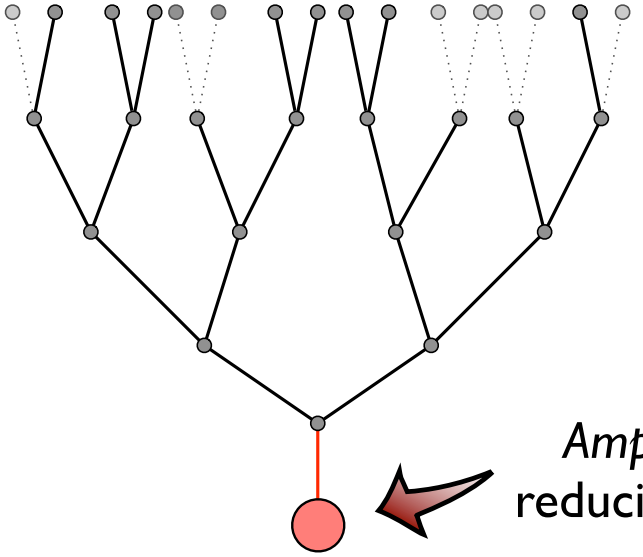
Proof of Main Theorem

- **Theorem:** ADV-balanced φ over $\mathcal{S}$ can be evaluated in O(A(φ)) queries.

- **Proof:** Moving down from the leaves, at every level, we lose a factor of spc(P) = ADV(f_P) in either $|a_O|^2$ or $|b_O|^2$.
  - **Fact:** If $\varphi = g \circ (h_1, \ldots, h_k)$, then

$$A(g) \cdot \min_i A(h_i) \leq A(\varphi) \leq A(g) \cdot \max_i A(h_i)$$ [A '06, HLS '05]

➡ Since φ is ADV-balanced, at the very bottom we have lost a factor of A(φ).

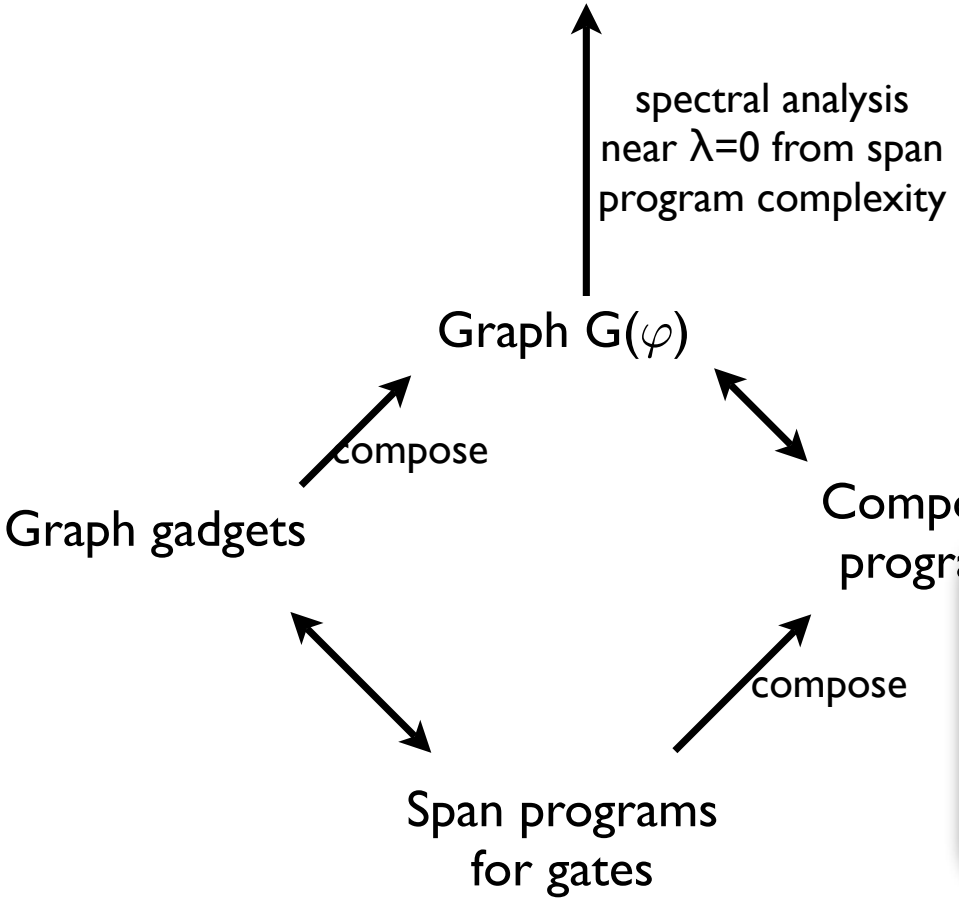


In case φ(x)=0, bound $r_O=a_O/b_O \leq A(\varphi)\ \lambda$ will contradict the last equation…

Amplify $|a_O|^2$ in case φ(x)=1 to Ω(1) by reducing weight of output edge to $1/\sqrt{A(\varphi)}$. □

High-level argument

- Let φ be an adversary-balanced formula over $\mathcal{S}$.
- **Main Theorem:** Formula φ can be evaluated in $O(A(\varphi))$ queries.



Another open problem

Interesting qu algorithms based directly on large span programs?

- E.g., Perfect Matching (Tutte matrix), …

Open ?: Why do span programs work so well?

- **Def:** $ADV^{\pm}$ bound = ADV bound allowing for positive or negative weights in the adversary matrix. $Q_2(f) = \Omega(ADV^{\pm}(f))$ [Høyer, Lee, Špalek QIP'07].

$ADV^{\pm}$ =

Open ?: Why do span programs work so well?

- **Def:** $ADV^{\pm}$ bound = ADV bound allowing for positive or negative weights in the adversary matrix. $Q_2(f) = \Omega(ADV^{\pm}(f))$ [Høyer, Lee, Špalek QIP'07].
- $ADV^{\pm} \geq$ ADV always. Functions with $ADV^{\pm} >$ ADV seem to be "hard" for span programs.
  - For **70** of 92 functions on ≤4 bits with $ADV^{\pm}$ = ADV we have a matching span program (built by piecing together ~20 different base programs) ✓
  - For **0** of 116 functions on ≤4 bits with $ADV^{\pm} >$ ADV do we have a matching span program
- **Theorem:** $spc(P) \geq ADV^{\pm}(f_P)$
  - Proof: Compose f_P on itself d times, our qu. algorithm evaluates f_P^d in time $O(spc(P)^d) \geq \Omega(ADV^{\pm}(f_P^d) \geq ADV^{\pm}(f_P)^d)$. $\therefore spc(P) \geq ADV^{\pm}(f_P)$ □

[Gál, "Characterization of span program size," cc'01]

Extensions to larger gate sets

- Complexity of remaining four-bit gates is open
  - E.g., 22 functions f with $ADV^{\pm}(f) = ADV^{+}(f)$ are good candidates
  - All 116 functions f for which $ADV^{\pm}(f) > ADV^{+}(f)$???
- Unbalanced formulas?
  - Only OR, PARITY, MAJ_3 (partially) accept unbalanced inputs so far
  - For highly unbalanced inputs, formula rebalancing à la [Bshouty/Cleve/Eberly '93] will be required—but effect on adversary bound unstudied