Learning using Local Membership Queries

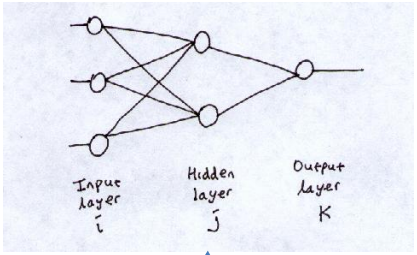
Pranjal Awasthi(CMU)
Vitaly Feldman(IBM Almaden)
Varun Kanade(UC Berkeley)

PAC + MQ Learning [Angluin ‘87]

Distribution Oracle

x » D
(x, f(x))

Learning algorithm



x ↓ ↑ f(x)

f

h: Pr[f ≠ h] · ^{2}

PAC + MQ Learning

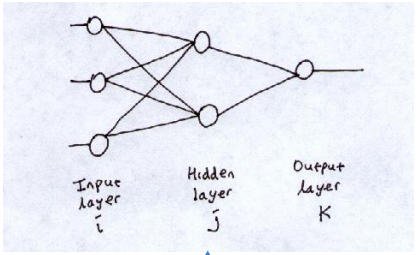
- Can learn decision trees [Kushilevitz-Mansour ‘91, Bshouty ’93]
  - Agnostic learning [Kalai et al. ‘08]
- DNFs under product distributions [Jackson ‘94]
- Learning intersection of 2 halfspaces [Baum ‘91, Gopalan et al. ‘12]

PAC + MQ Learning[Angluin ‘87]

Distribution Oracle

x » D
(x, f(x))

Learning algorithm



x ↓ ↑ f(x)

f

h: Pr[f ≠ h] · ^{2}

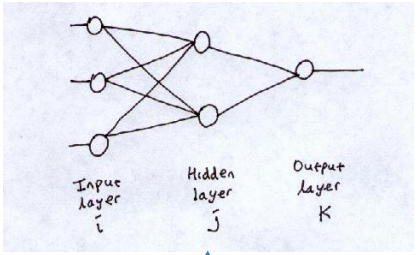
Too Strong

PAC + MQ Learning[Angluin ‘87]

Distribution Oracle

x » D
(x, f(x))

Learning algorithm



h: Pr[f ≠ h] · ^{2}

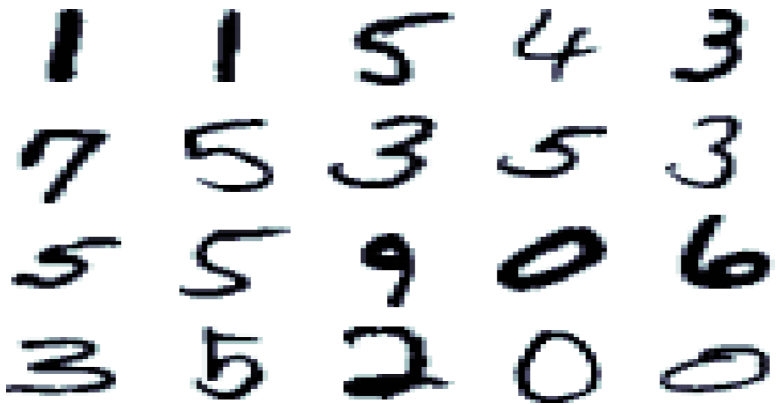
x f(x)

Too Strong

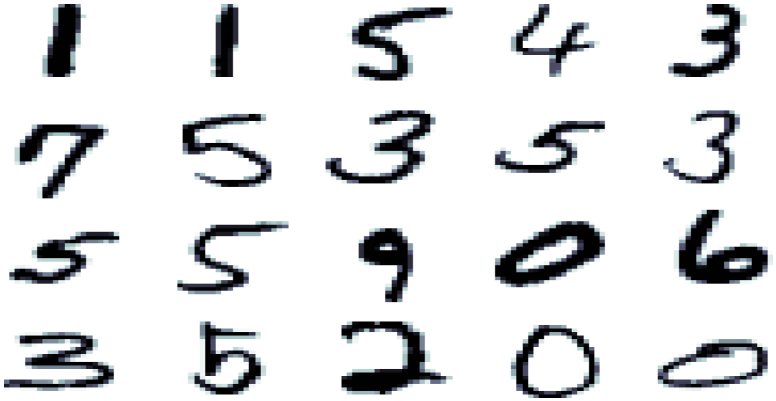
Not every x will make sense to a labeler

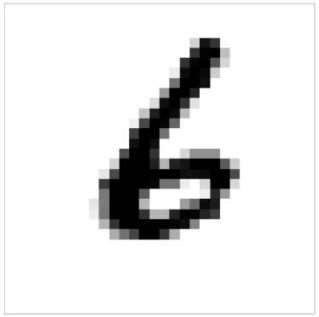
PAC + MQ learning

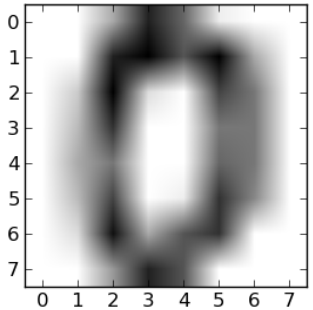
[Lang, Baum'93]:
Query algorithms do not work well when a human oracle is used.

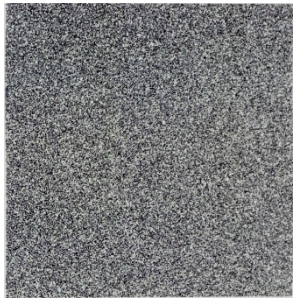


Our model







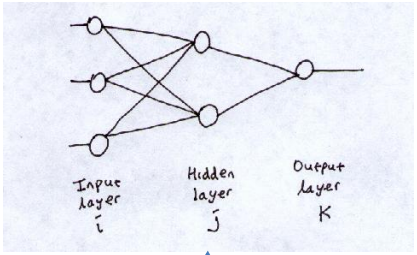


PAC + local-MQ Learning

Distribution Oracle

x » D
(x, f(x))

Learning algorithm



h: Pr[f ≠ h] · ^{2}

Local query

x ↓ f ↑ f(x)

A query x is r-local if there exists a data point x' in the sample such that d(x,x') · r.

For this talk:
x 2 {0,1}^{n}
d = Hamming

Warmup: 1-local MQs

Claim:

- Can learn noisy parities using 1-local MQs.
  - Compute Pr[f(x) changes | x_i is flipped]
- Can learn k-juntas under uniform distribution.

Our Results(1)

Theorem:
Efficient O(log(n))-local algorithm for learning sparse polynomials over $\{0,1\}^n$ under log-Lipschitz distributions.

Includes log-depth decision trees.

Our Results(2)

Theorem:
Efficient O(log(n))-local algorithm for learning poly size decision trees under constant biased product distributions.

Our Results(3)

Theorem:
O(log(n))-local algorithm for learning poly size DNFs under uniform distribution.

Run time is $n^{O(\log\log(n))}$

Learning sparse polynomials

Theorem:

Efficient O(log(n))-local algorithm for learning sparse polynomials over $\{0,1\}^n$ under log-Lipschitz distributions.

Learning sparse polynomials

- $X = \{0,1\}^n$
- D is ® log-Lipschitz
  - 8 x,x' s.t. d(x,x') = 1, D(x)/D(x') · ®
  - ® = 1, uniform
  - Includes constant biased product distributions, smooth distributions of [Kalai et al. '09].

Learning sparse polynomials

- $X = \{0,1\}^n$
- Target function: $f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$
- Bounded coefficients
- $|\mathrm{supp}(f)| = \mathrm{poly}(n)$
- Goal: Output h s.t. $\|f - h\|_2 \leq \epsilon$

Learning sparse polynomials

- Low degree approximation
- Identifying candidate low degree monomials
- L_2 regression over candidate monomials.

Learning sparse polynomials

- Low degree approximation
- Identifying candidate low degree monomials
- L_2 regression over candidate monomials.

Low degree approximation

- $f(x) = \sum_S c_S \prod_{i \in S} x_i$
- $f_d(x) = \sum_{S : |S| \le d} c_S \prod_{i \in S} x_i ; \quad d = O(\log(n/\varepsilon^2))$
- **Claim:** $\|f - f_d\|_2^2 \le \varepsilon^2$

Proof:

- For any S, $\left(\frac{1}{1+\delta}\right)^{|S|} \cdot \Pr_D\left[\bigwedge_{i \in S} x_i = 1\right] \le \left(\frac{\delta}{1+\delta}\right)^{|S|}$

Learning sparse polynomials

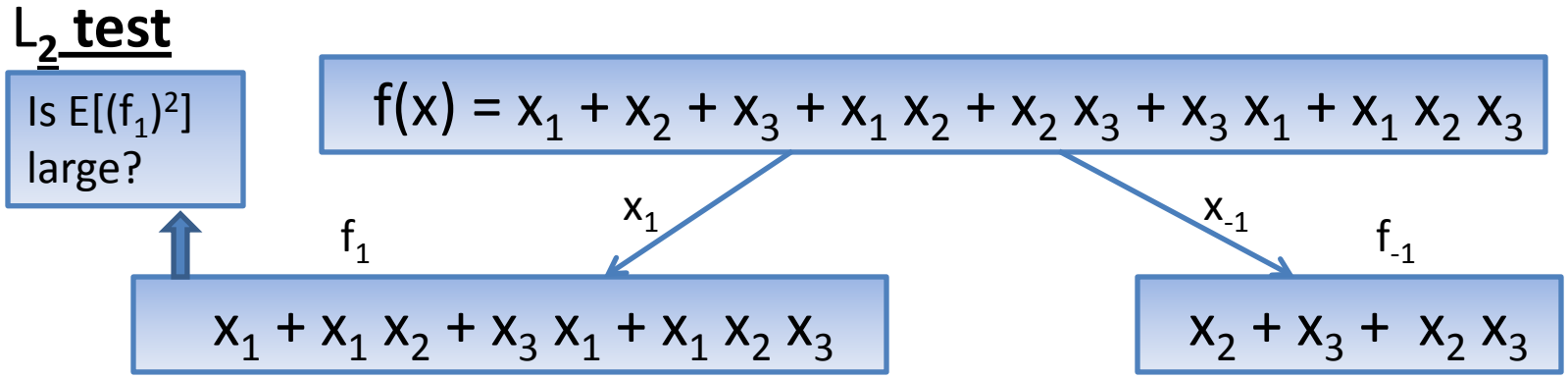
- Low degree approximation
- Identifying candidate low degree monomials
- L_2 regression over candidate monomials.

Kushilevitz-Mansour Algorithm

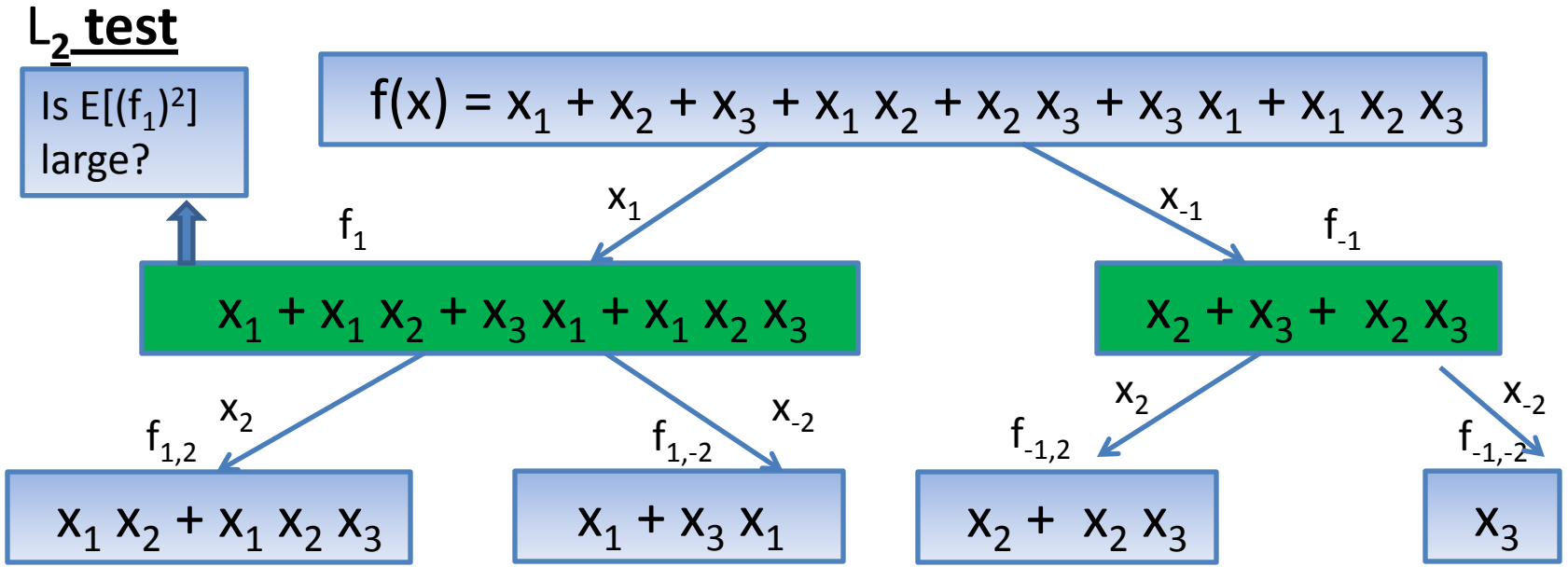
Theorem[KM '91]:

For any $f: \{0,1\}^n \rightarrow \{0,1\}$, can output all μ heavy monomials in time $\text{poly}(n, 1/\mu, L_1(f))$

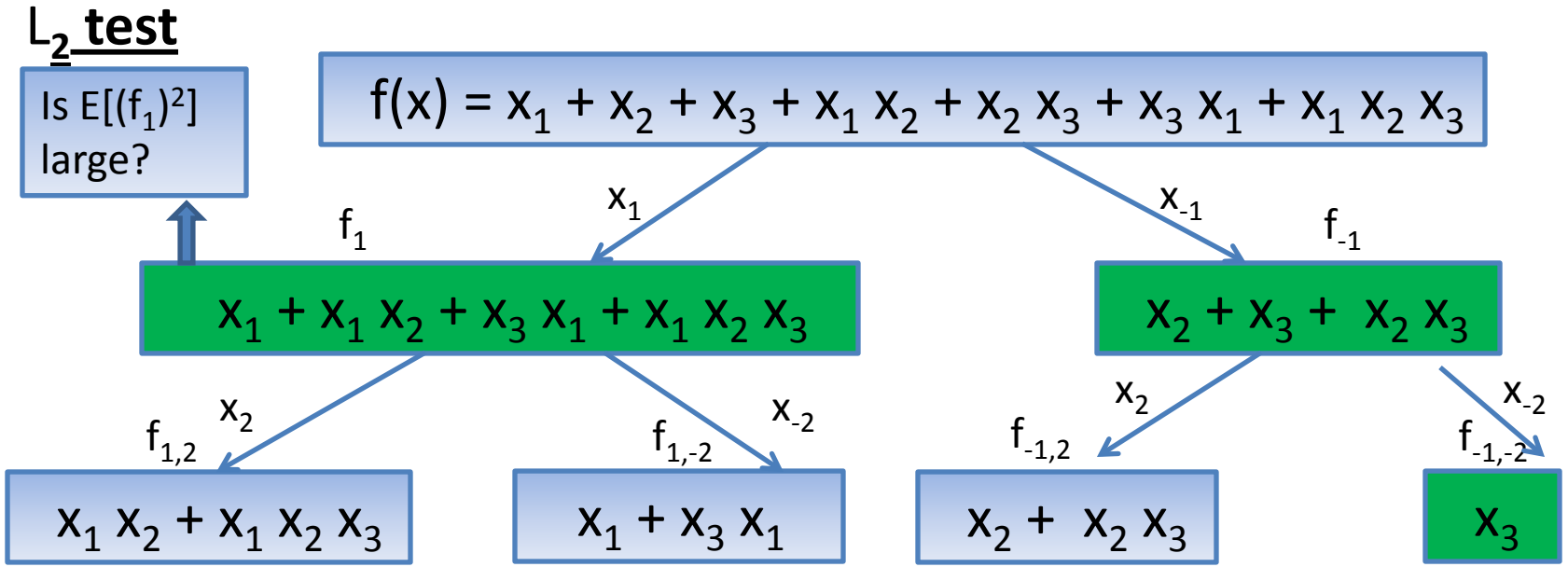
Kushilevitz-Mansour Algorithm



Kushilevitz-Mansour Algorithm

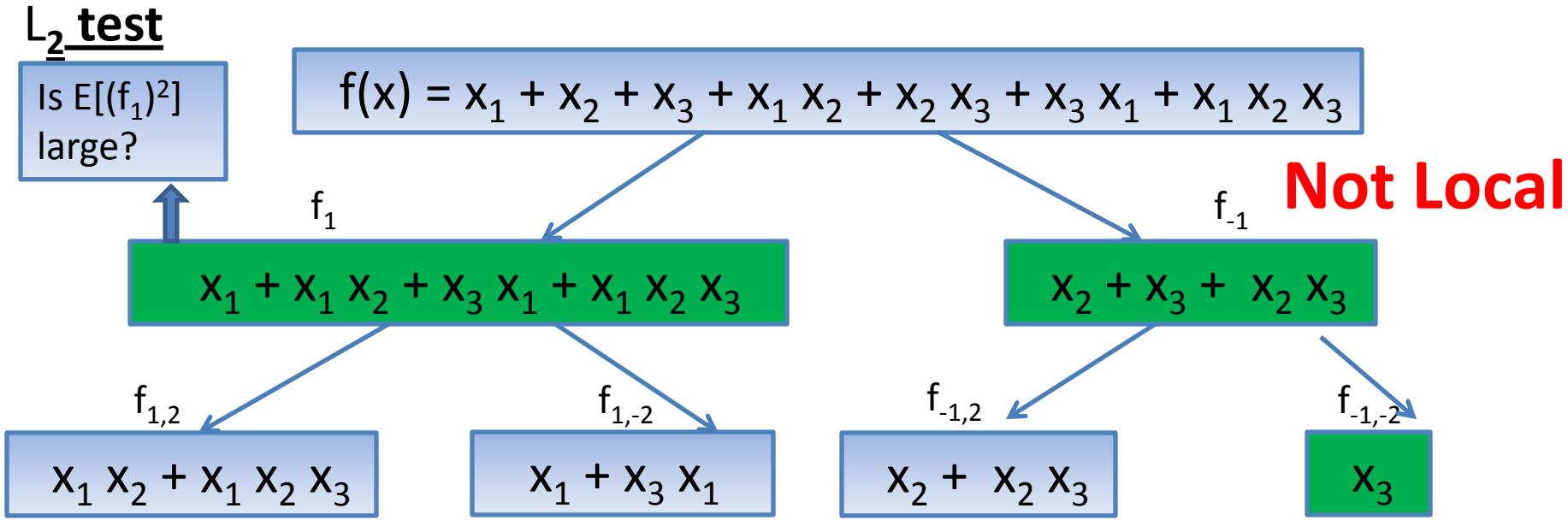


Kushilevitz-Mansour Algorithm



Leaves represent important monomials.

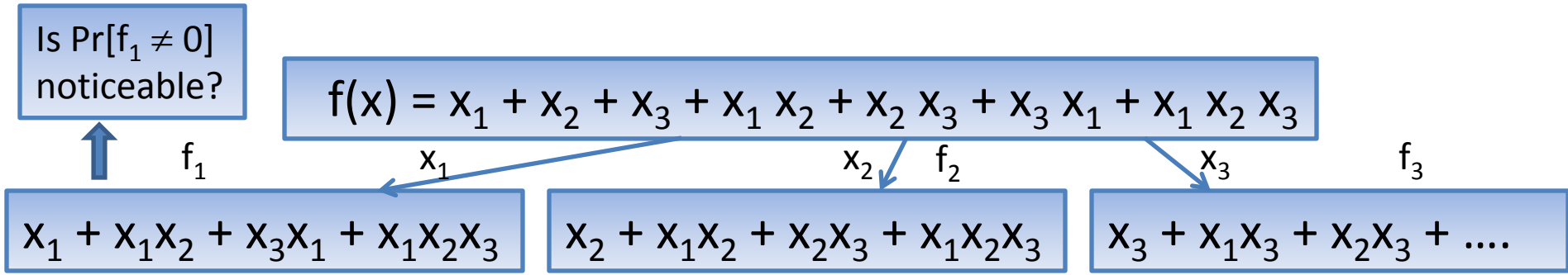
Kushilevitz-Mansour Algorithm



At each step "active" nodes partition the polynomial.

Our Algorithm

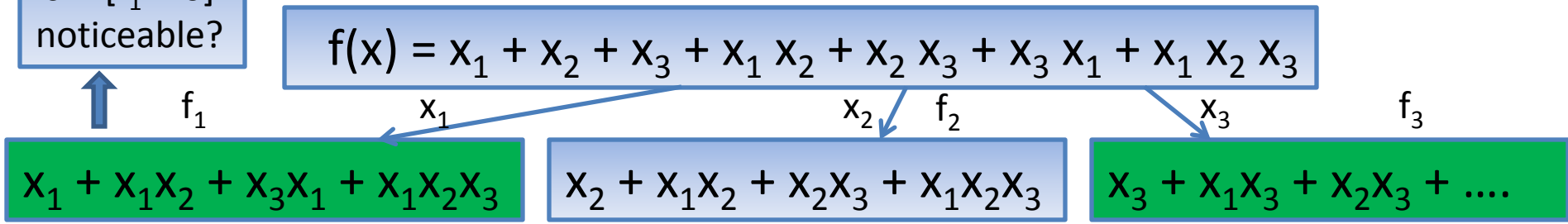
Non-zero-test



Our Algorithm

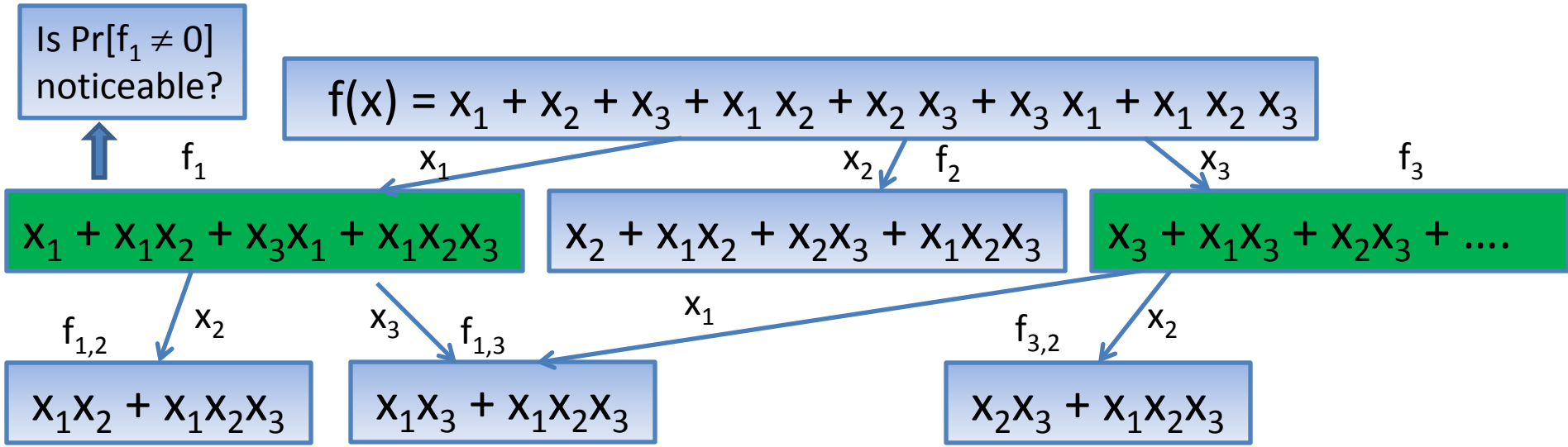
Non-zero-test

Is $\Pr[f_1 \neq 0]$ noticeable?



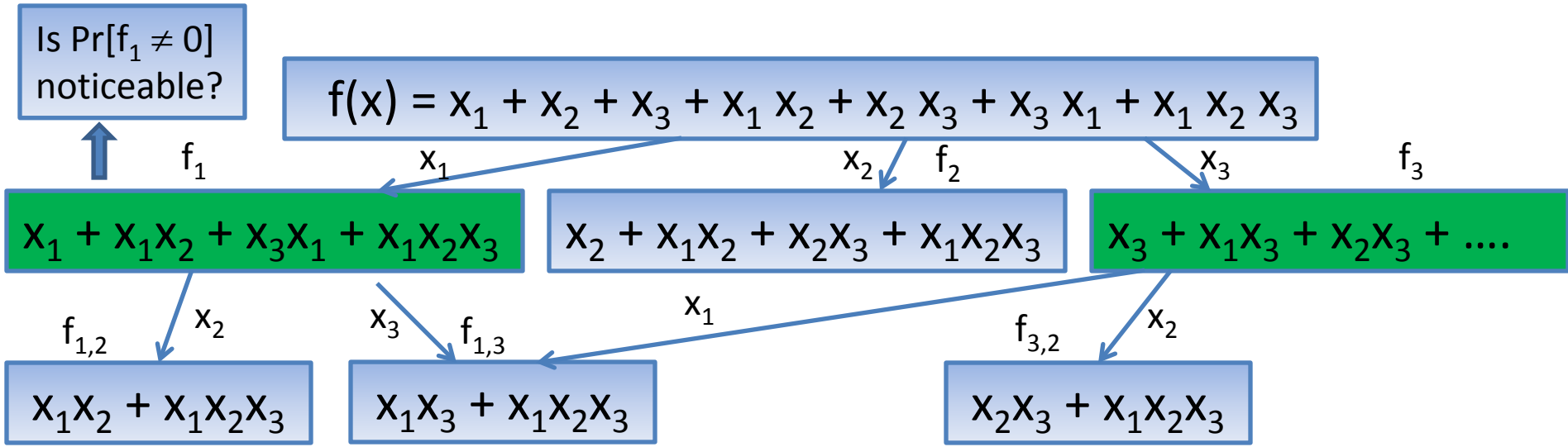
Our Algorithm

Non-zero-test



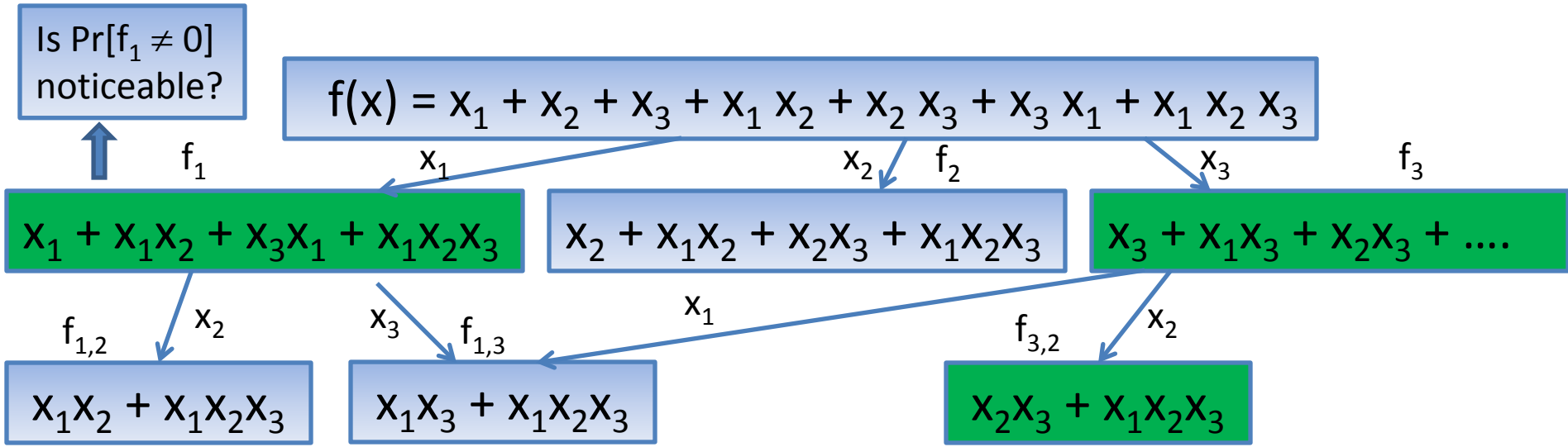
Our Algorithm

Non-zero-test



Our Algorithm

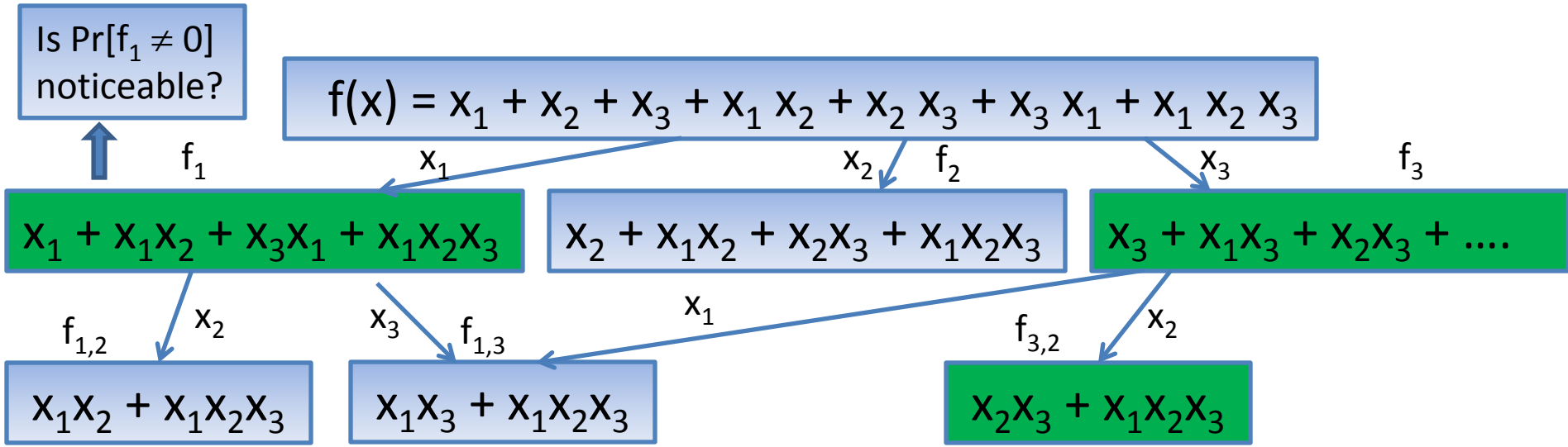
Non-zero-test



Till depth d = log ($n/^2$)

Our Algorithm

Non-zero-test



If f_S passes the test, add S

Non-zero-test(x_1)

$$f(x_1; x_2; ¢¢¢; x_n) = x_1 f_1(x_2; x_3; ¢¢¢; x_n) + f_{i\ 1}(x_2; x_3; ¢¢¢; x_n)$$

The test passes if $Pr_D [f_1 \neq 0]$ is noticeable.

Claim:
Non-zero-test can be implemented using local MQs.

Non-zero-test(x_1)

$$f(x_1; x_2; ¢¢¢; x_n) = x_1 f_1(x_2; x_3; ¢¢¢; x_n) + f_{i\ 1}(x_2; x_3; ¢¢¢; x_n)$$

$$f(1; x_2; ¢¢¢; x_n) = f_1(x_2; x_3; ¢¢¢; x_n) + f_{i\ 1}(x_2; x_3; ¢¢¢; x_n)$$

$$f(0; x_2; ¢¢¢; x_n) = 0\ ¢ f_1(x_2; x_3; ¢¢¢; x_n) + f_{i\ 1}(x_2; x_3; ¢¢¢; x_n)$$

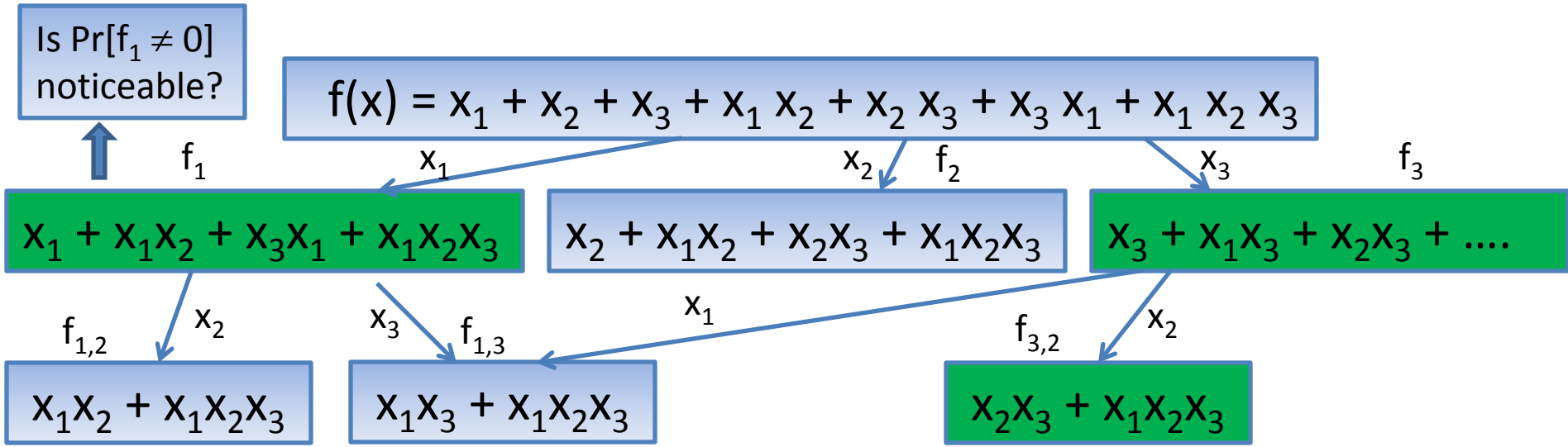
$$f_1(x_2; ¢¢¢; x_n) = \cdot_{x_1}(2x_1 ¡\ 1) f(x_1; x_2; x_3; ¢¢¢; x_n)$$

Non-zero-test(S)

- $f(x) = \left(\sum_{x_i \in S} x_i\right) f_S(x) + f_{\setminus S}(x)$
- $f_S(x) = \prod_{x_i : i \in S} \sum_{i \in S} (2x_i - 1) f(x)$
- Needs $|S|$-local queries.

Our Algorithm

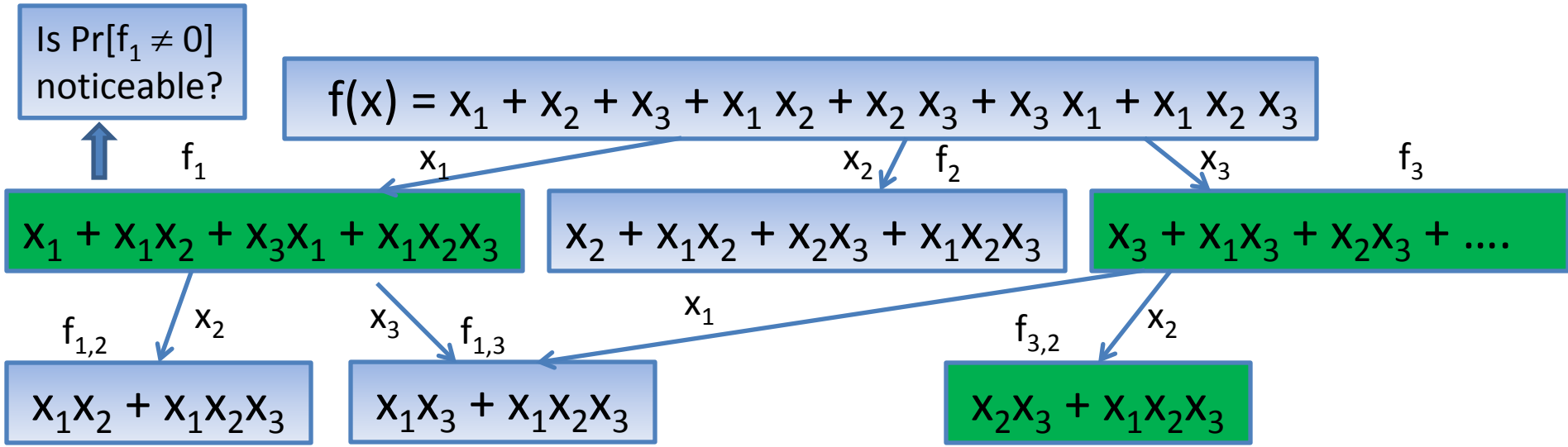
Non-zero-test



Till depth d = log ($n/^2$)

Our Algorithm

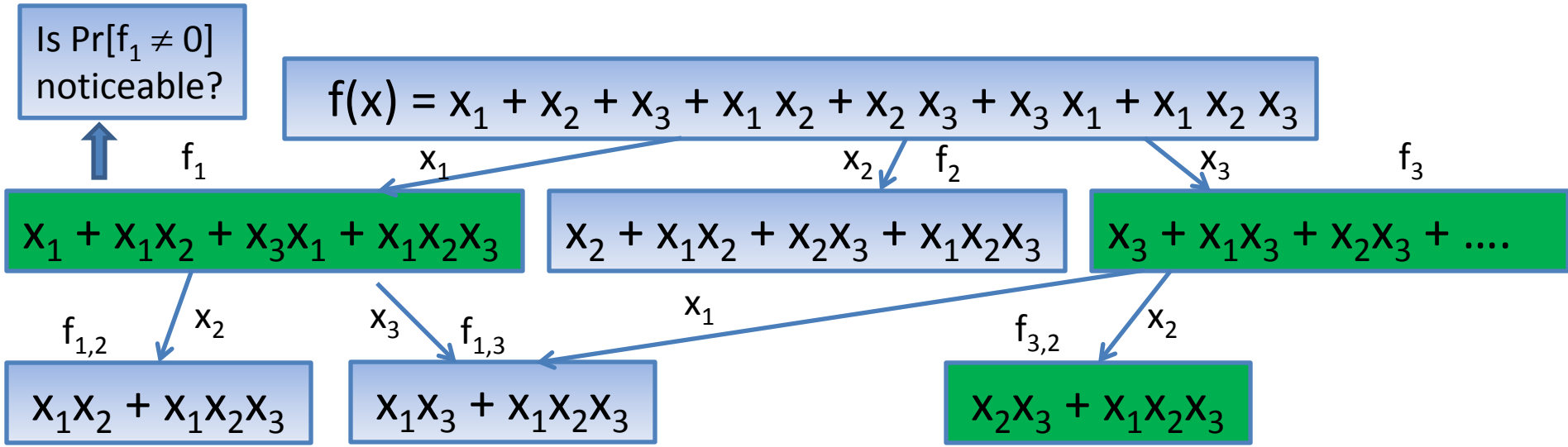
Non-zero-test



Fit a polynomial p(x) over added terms which minimizes $E((p(x) - f(x))^2)$.

Our Algorithm

Non-zero-test



It Works!

Conclusions

log(n)-local algorithms for

- Sparse polynomials under ® log-Lipschitz distributions.
- Decision trees under product distributions.
- DNFs under uniform distribution in time $n^{O(\log\log(n))}$.

Open Problems

- Algorithms for poly size decision trees under log-Lipschitz distributions?
- Polynomial time algorithm for DNFs under uniform distribution?
- Local-MQ agnostic learning algorithms?
  - O(1)-local queries won't help.

THANK YOU

Correctness

- $f(x) = \sum_S c_S \prod_{i \in S} x_i$
- $f_d(x) = \sum_{S : |S| \le d} c_S \prod_{i \in S} x_i; \quad d = O(\log(n/\epsilon^2))$

Claim:

$\forall S \in \mathrm{supp}(f_d); \Pr[f_S(x) \neq 0] \ge \exp(-d) = 1/\mathrm{poly}(n)$

Correctness

- $f(x) = \sum_{S} c_S \prod_{i \in S} x_i$
- $f_d(x) = \sum_{S : |S| \le d} c_S \prod_{i \in S} x_i$; $d = O(\log(n/\epsilon^2))$

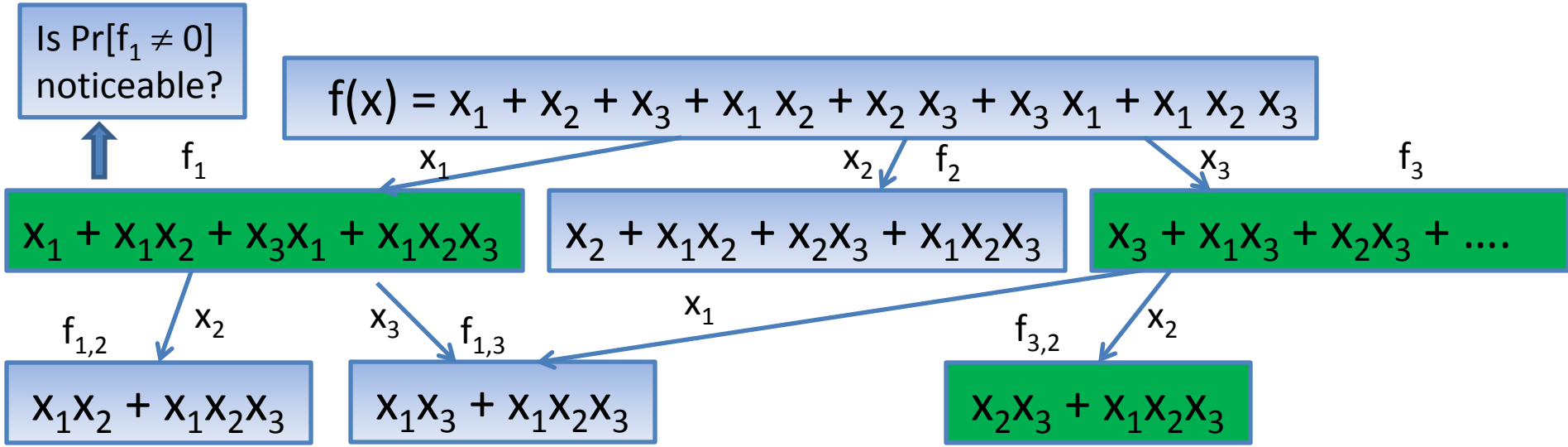
Lemma:

Let f be t-sparse with non-zero constant term and D is

®-log-Lipschitz: $\Pr[f \neq 0] \le (1+®)^{-\log(t)}$

Efficiency

Non-zero-test



Claim:

If S is added then 9 T ¶ S s.t. $c_T \neq 0$ and $|T| \cdot d' = O(d)$

Efficiency

Claim:

If S is added then 9 T ¶ S s.t.

1. $c_T \neq 0$
2. $|T| \cdot d' = O(d)$

terms added · poly(n) $2^{O(d)}$

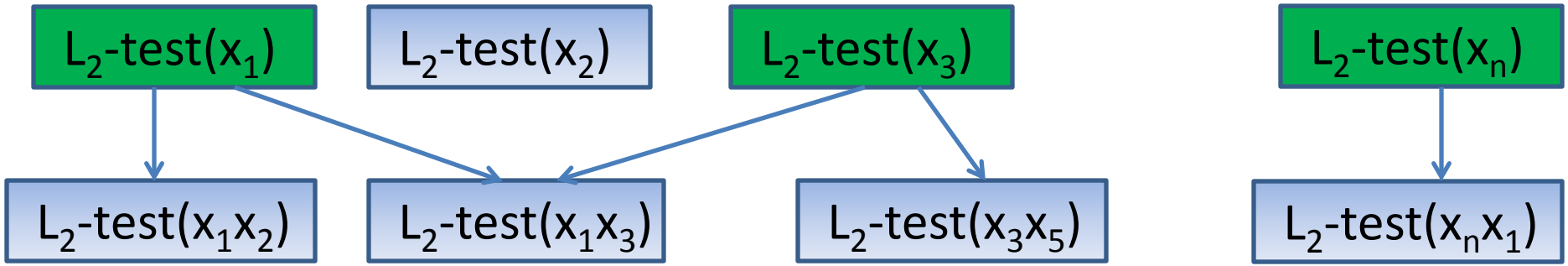
Learning DNFs

Low degree approximation

- $f(x) = \sum_S c_S \hat{A}_S(x)$
- $f_{\leq d}(x) = \sum_{S : |S| \leq d} c_S \hat{A}_S(x); \quad d = O(\log(\frac{n}{\epsilon}))$
- Finding heavy low-degree coefficients is enough.

- Poly-size DNF well approximated by a O(log(n))-term DNF.
- For an s-term DNF access to heavy coefficients up to degree $\log(s/\epsilon)$ is enough [Feldman '12].

The Algorithm

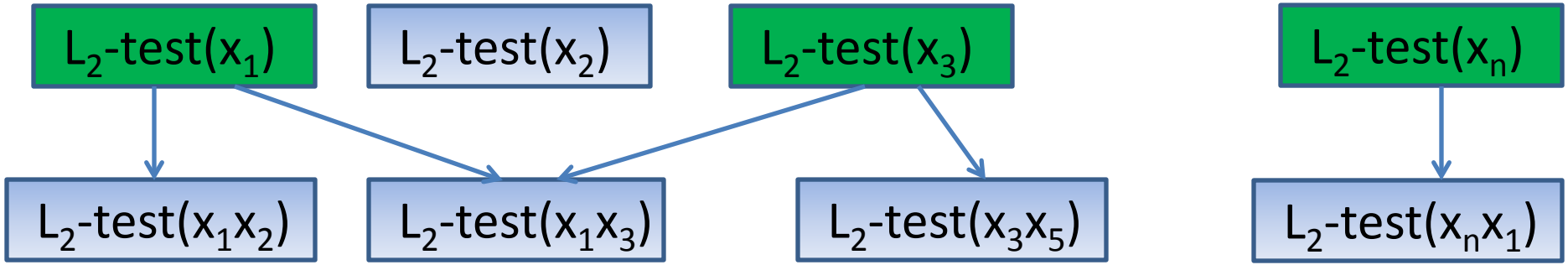


L_2-test(S)

- $f(x) = \hat{A}_S(x) f_S(x) + f_{\neg S}(x)$
- Add S if $E[(f_S(x))^2]$ is large
- Needs $|S|$-local queries.

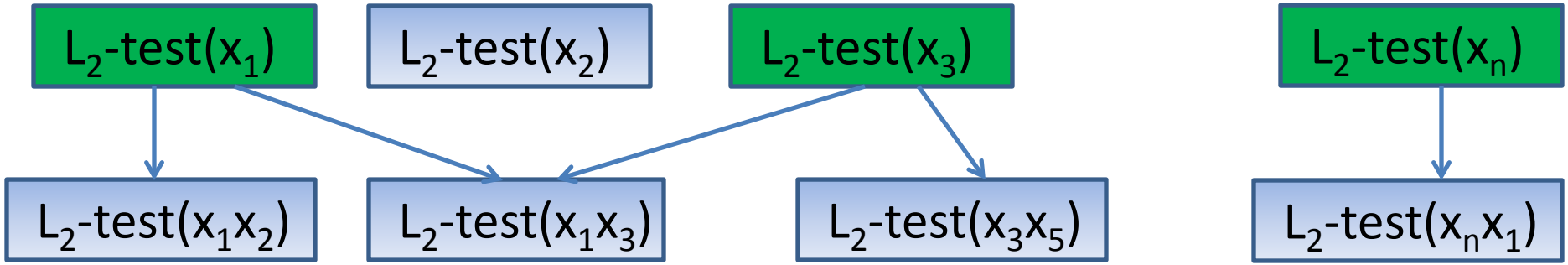
$$\forall S, E[f_S^2(x)] = \sum_{T \subseteq S} c_T^2$$

Correctness



- $f(x) = \sum_S c_S \hat{A}_S(x)$
- $f_d(x) = \sum_{S:|S|\le d} c_S \hat{A}_S(x)$
- $\forall S, E[f_S^2(x)] = \sum_{T \supseteq S} c_T^2$

Efficiency



- # nodes tested · $n^{O(\log\log(n))}$

Noise in MQ model

- Each query answered incorrectly w.p. $\acute{}$ [Sakakibara ‘91]
- Persistent noise model [Goldman et al. ‘90, Angluin, Slonim ‘91]
- Unreliable boundary queries [Blum et al. ‘95]

Learning Decision Trees

- $X = \{-1,1\}^n$, distribution D is uniform.
- $f(x) = \sum_S c_S \hat{A}_S(x); |c_S| \cdot \frac{poly(n)}{2^{|S|}}; \|f\|_1 = poly(n)$
- Goal: Output h s.t. $Pr_D[f(x) \neq h(x)] \cdot \epsilon^2$

Learning Decision Trees

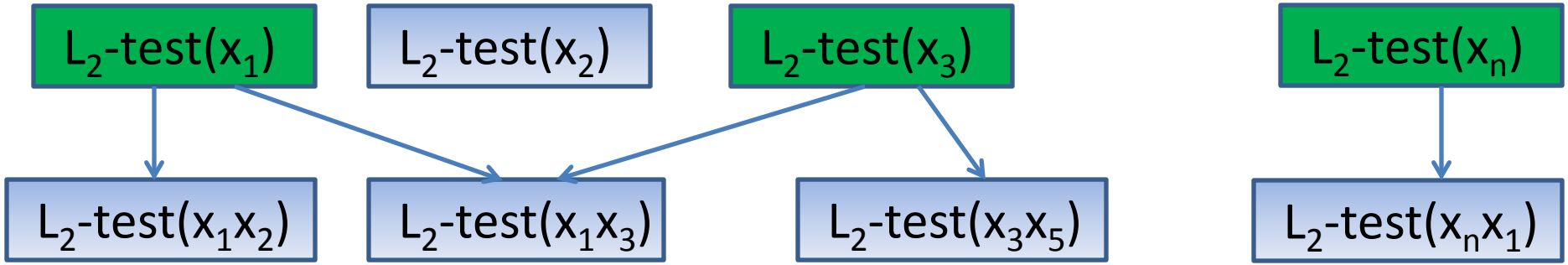
Low degree approximation

- $f(x) = \sum_S c_S \hat{A}_S(x)$
- $f_d(x) = \sum_{S : |S| \le d} c_S \hat{A}_S(x)$
- $\|f - f_d\|_2^2 \le \varepsilon^2; \quad d = O(\log(n/\varepsilon^2))$

Proof

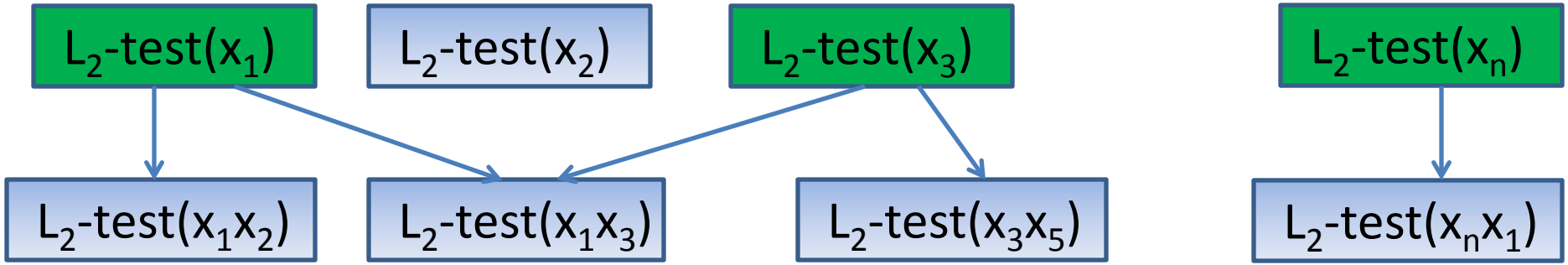
- For any S, $c_S \le \frac{poly(n)}{2^{|S|}}$

Correctness



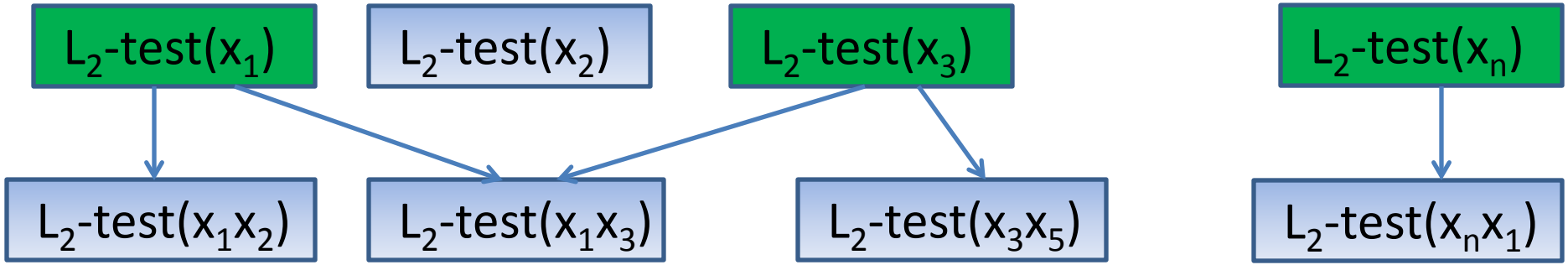
- $f(x) = \sum_S c_S \hat{A}_S(x)$
- $f_d(x) = \sum_{S:|S|\le d} c_S \hat{A}_S(x)$
- $\forall S, E[f_S^2(x)] = \sum_{T \supseteq S} c_T^2$

Efficiency



- $f(x) = \sum_S c_S \hat{A}_S(x)$
- $f_d(x) = \sum_{S:|S|\le d} c_S \hat{A}_S(x)$
- $\forall S, E[f_S^2(x)] = \sum_{T \supseteq S} c_T^2 \le \max_{T \supseteq S} c_T \sum_T |c_T|$

Efficiency



- $f(x) = \sum_S c_S \hat{A}_S(x)$
- $f_d(x) = \sum_{S:|S|\le d} c_S \hat{A}_S(x)$
- If S is added then exists T ¶ S, s.t. $|c_T|$ is large.