A Simple Algorithm for Worst Case Optimal Join and Sampling

Florent Capelli, Oliver Irwin, Sylvain Salvati
CRIL, Université d'Artois

DATA Lab @ Northeastern

11 April 2025

Joining relations

A simple algorithm for joins

$Q :\!-\; R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

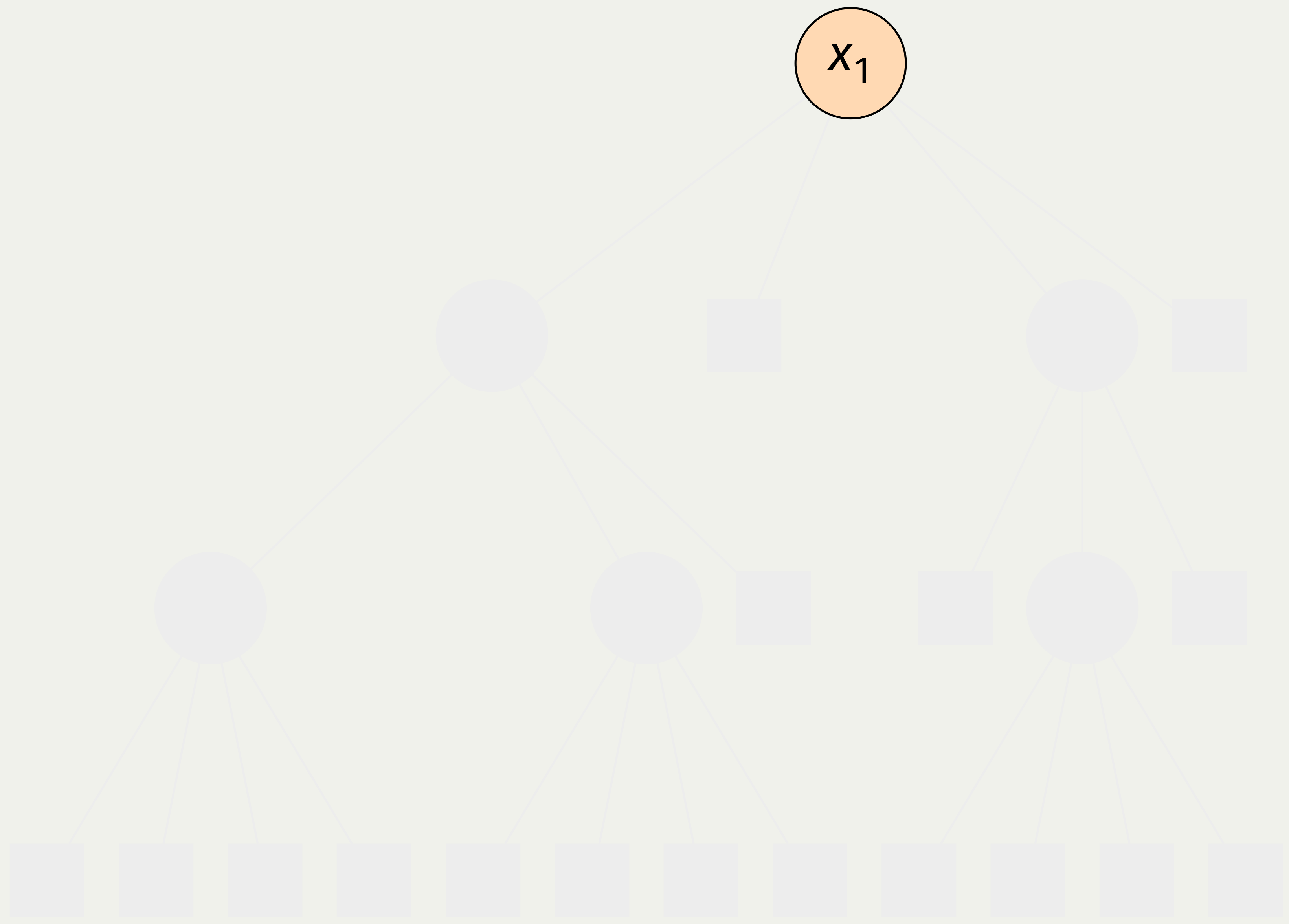
A simple algorithm for joins

$$Q :- R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



:::

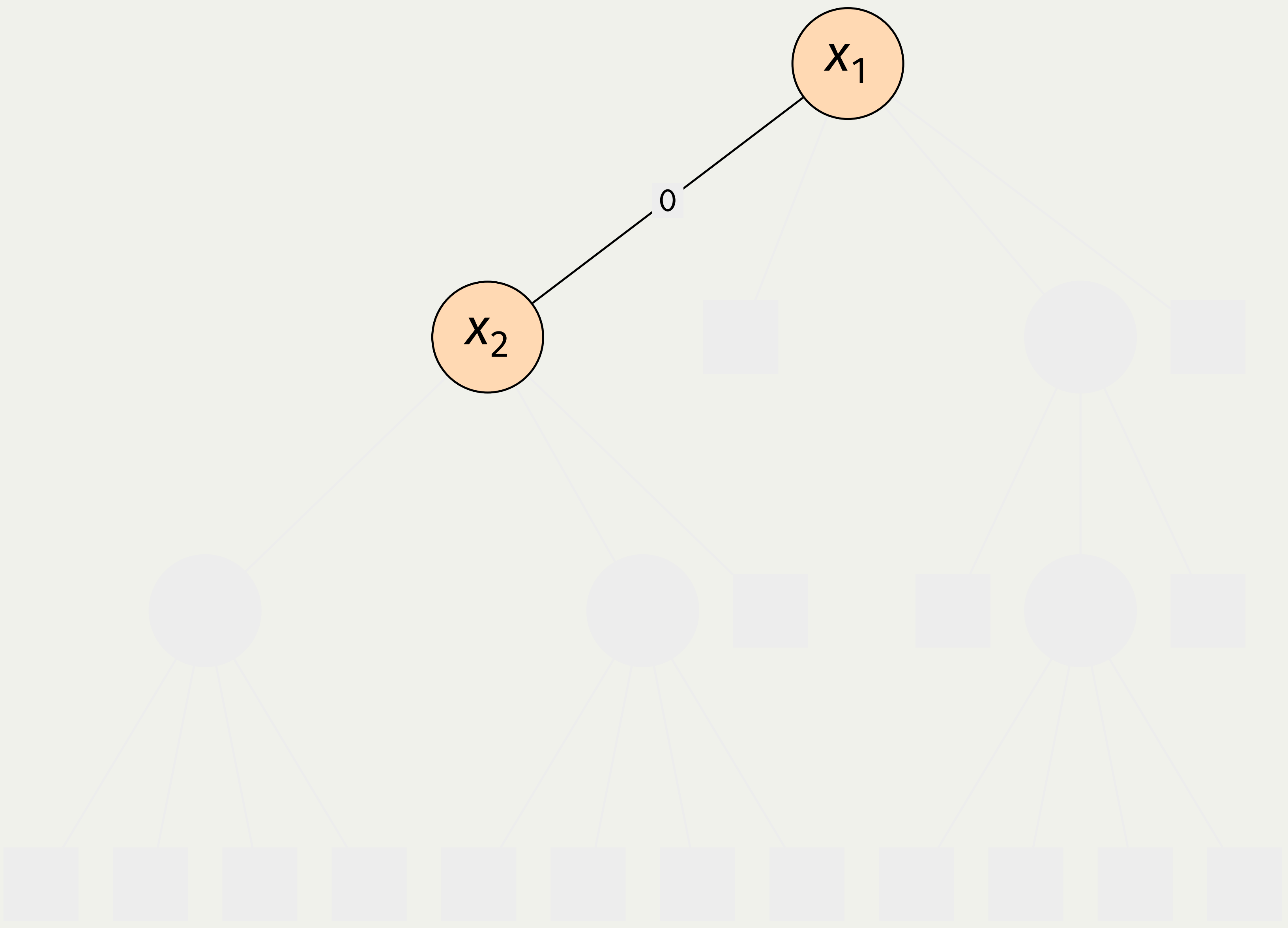
A simple algorithm for joins

$Q :- R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



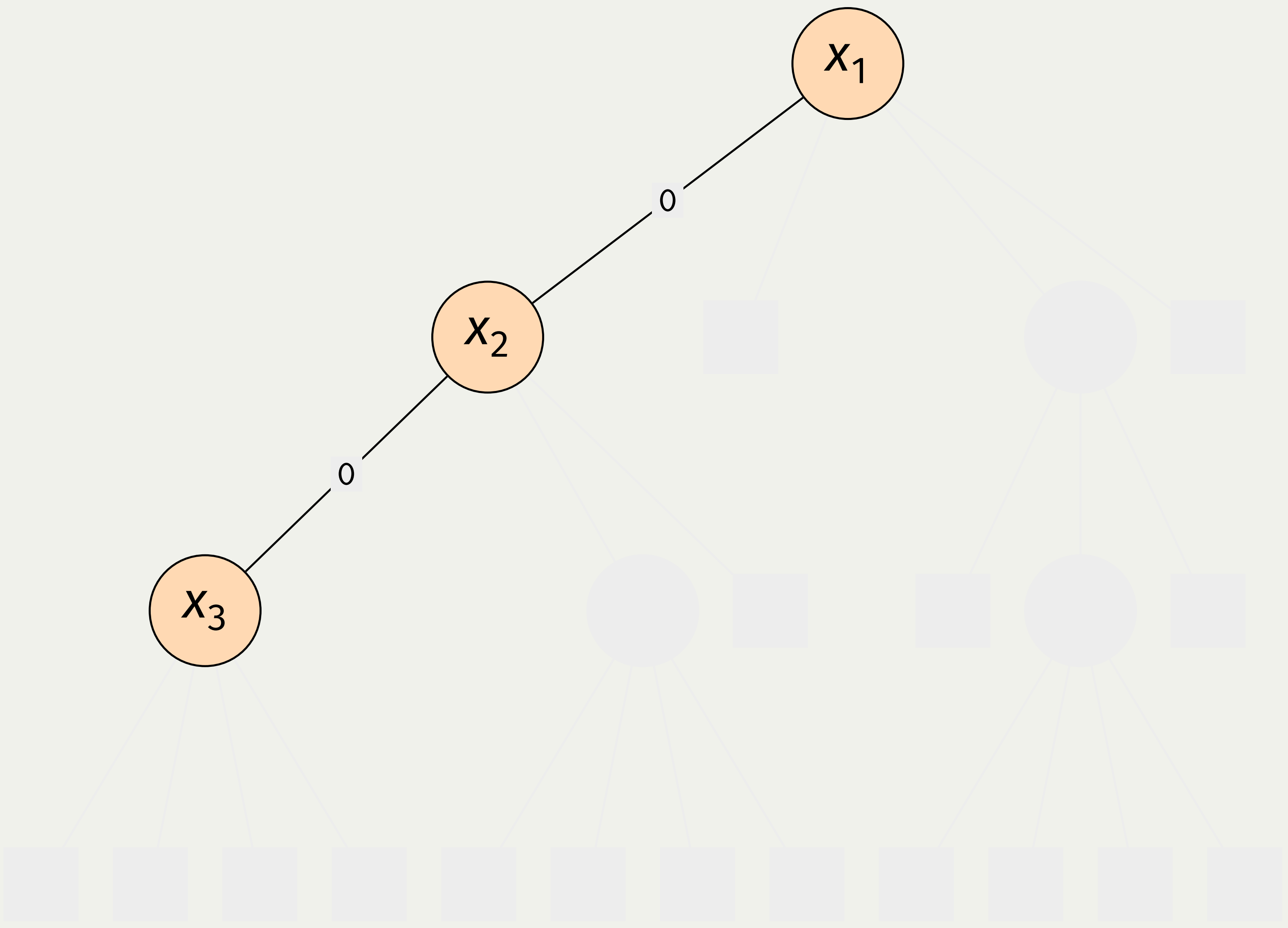
A simple algorithm for joins

$Q \coloneq R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



:::

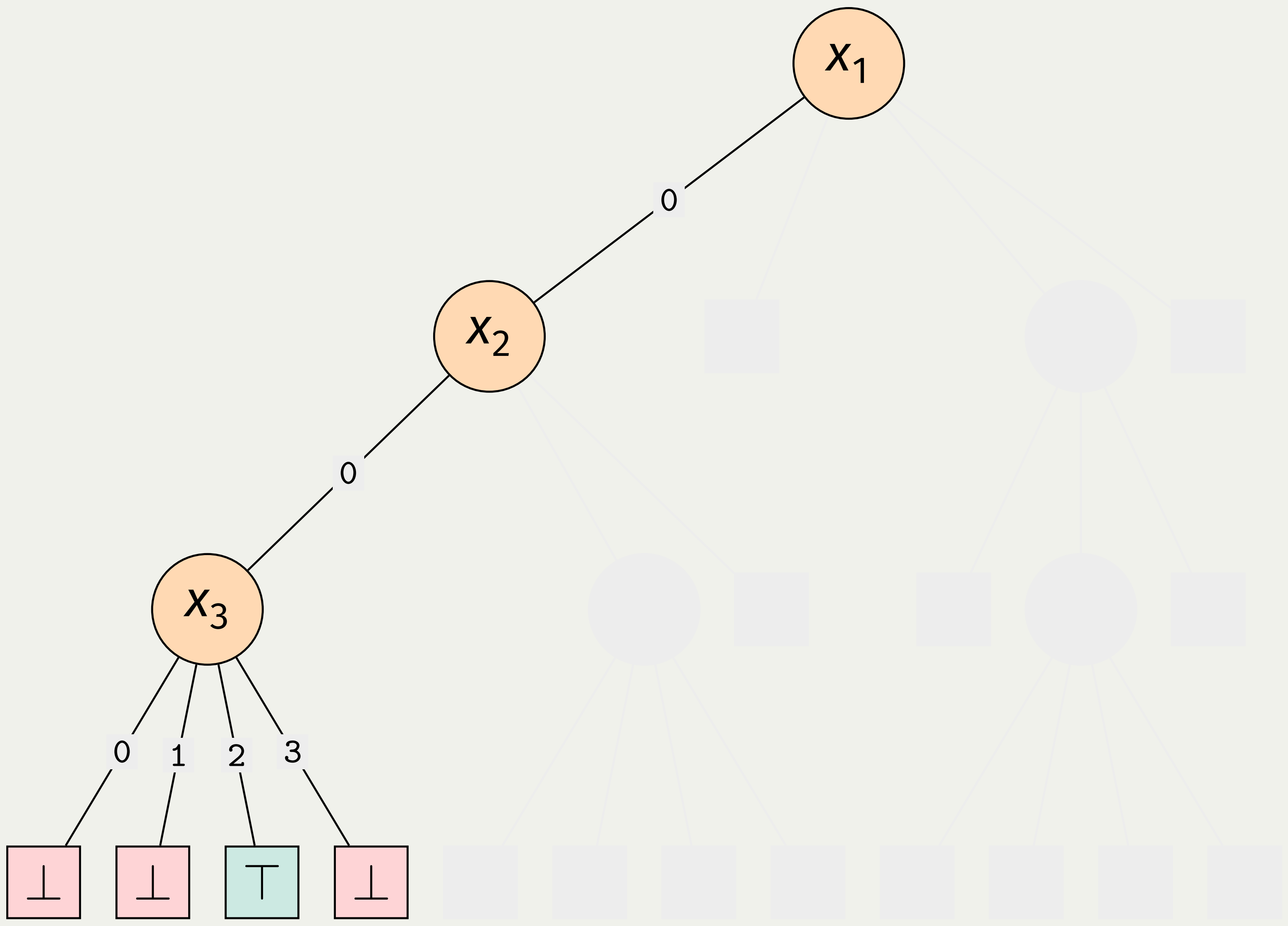
A simple algorithm for joins

$Q :- R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



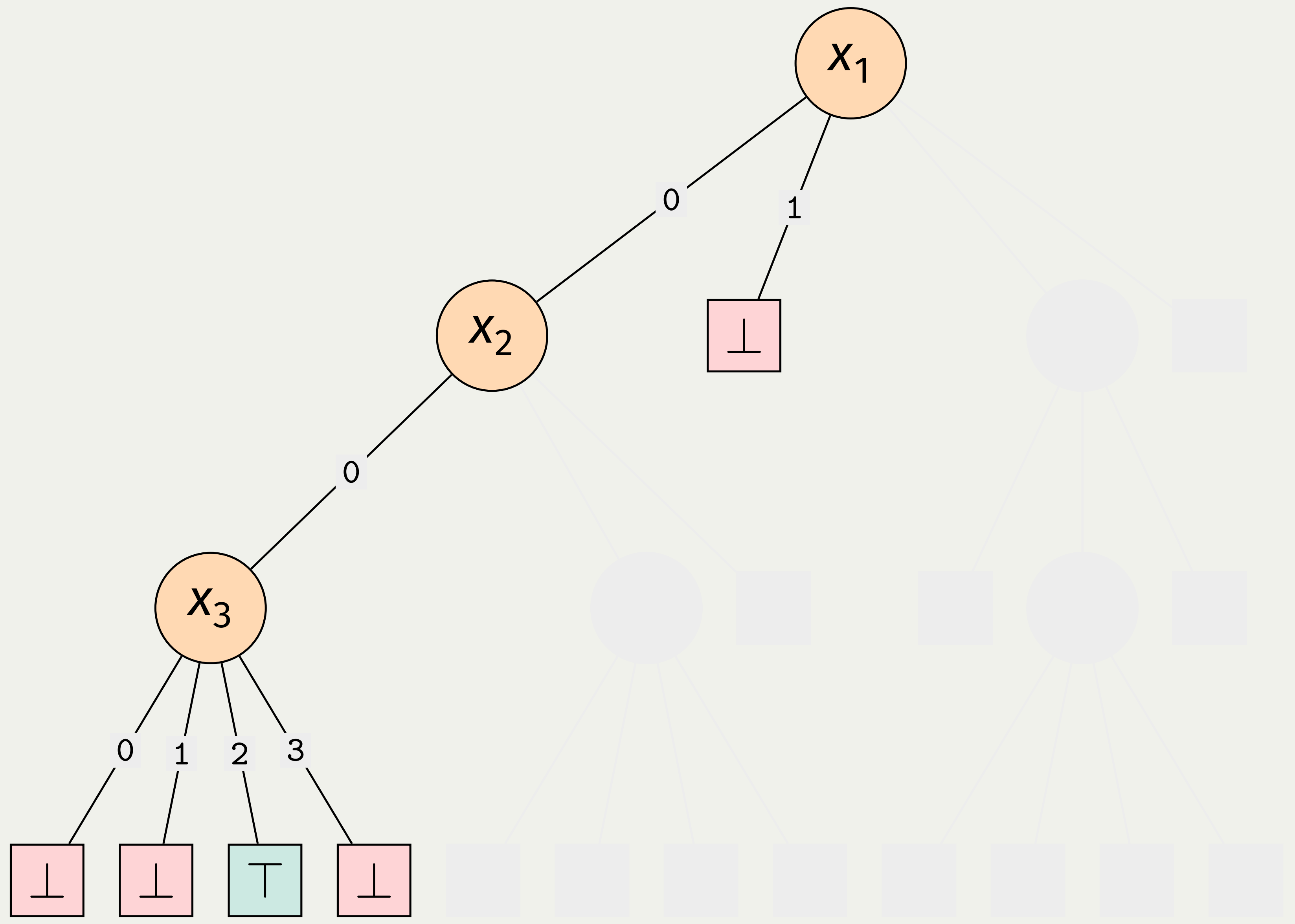
A simple algorithm for joins

$Q :- R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



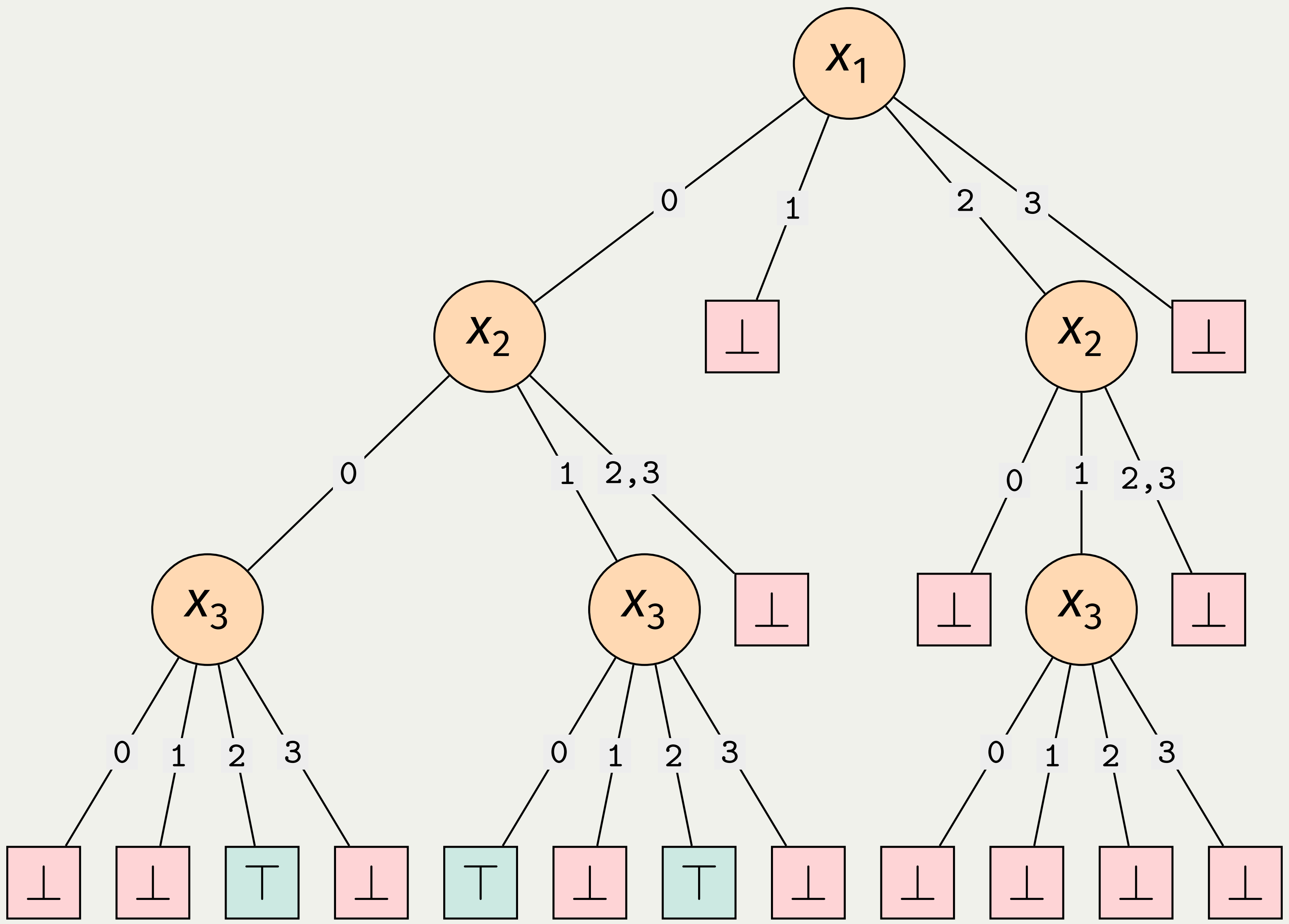
A simple algorithm for joins

$Q :- R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$

R	x_1	x_2
	0	0
	0	1
	2	1

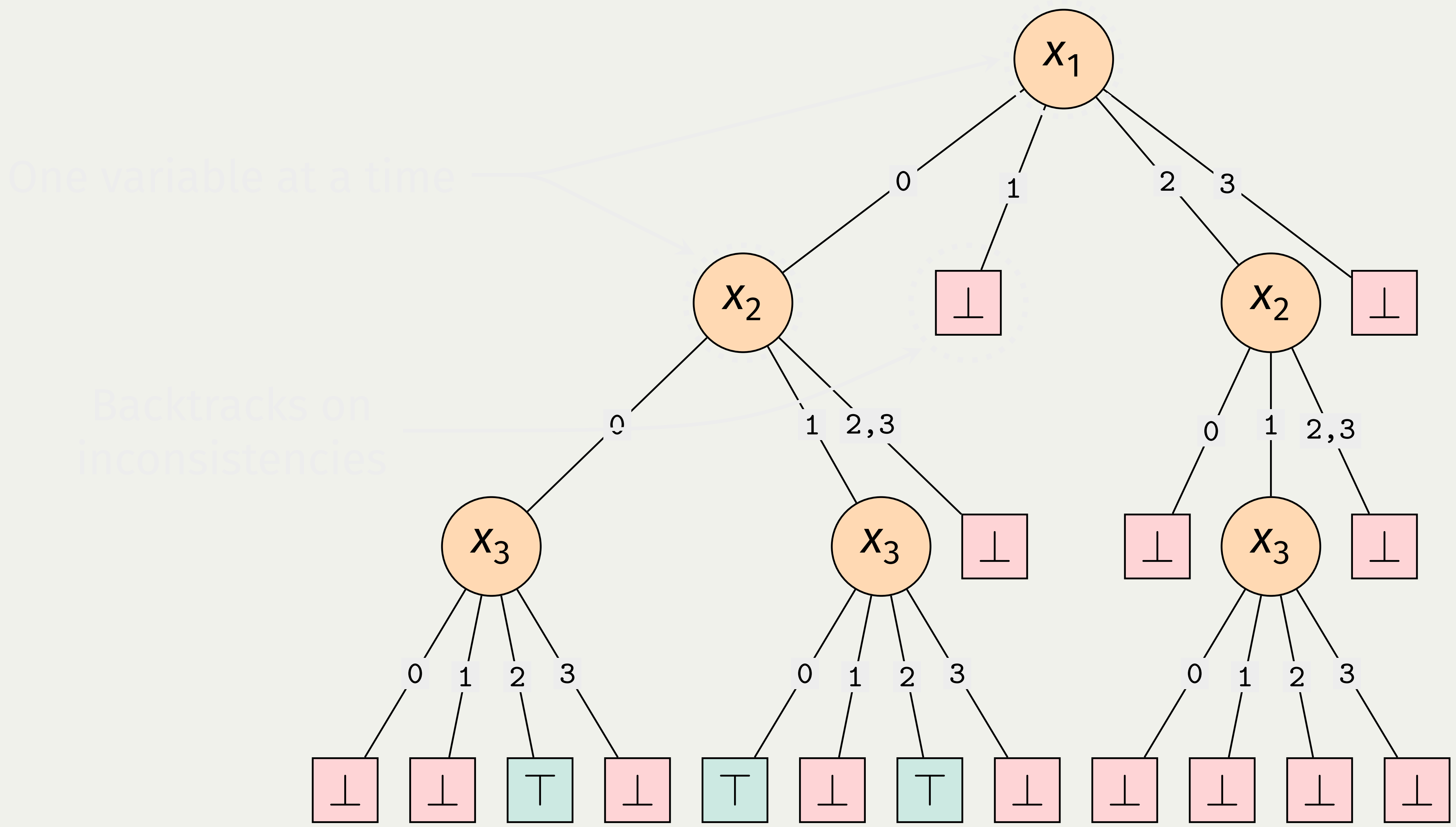
S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

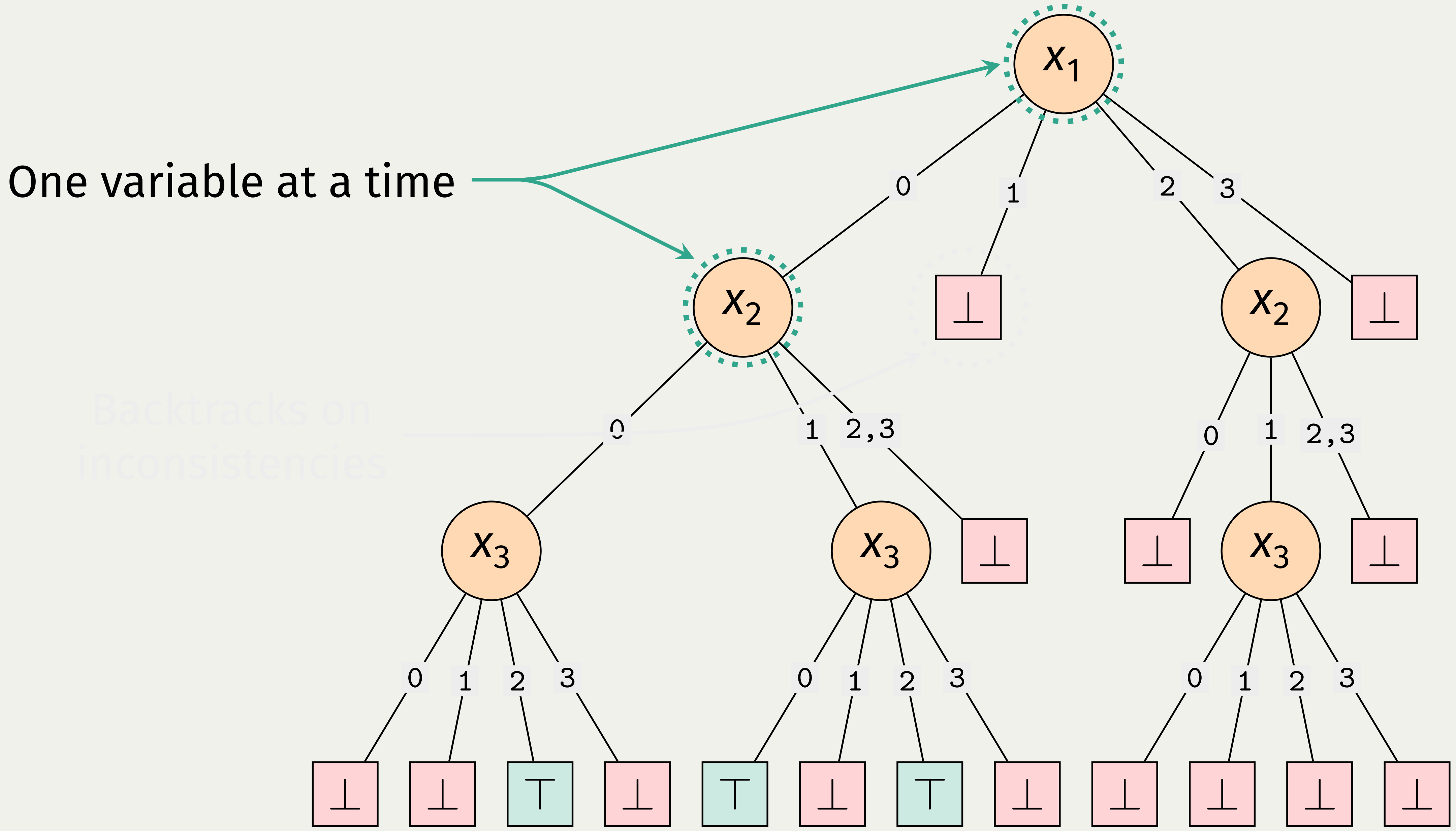


⋮

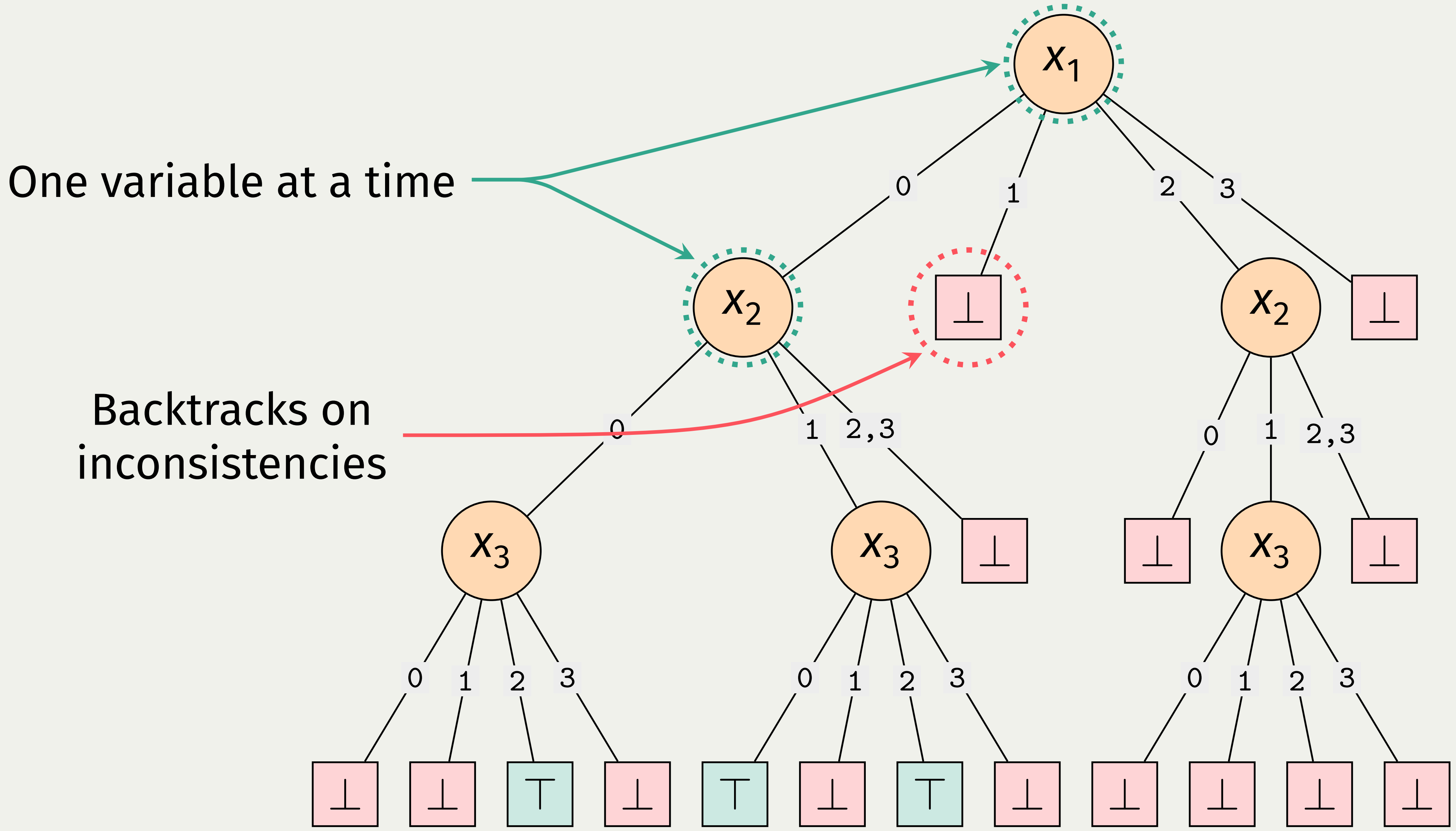
Algorithm overview



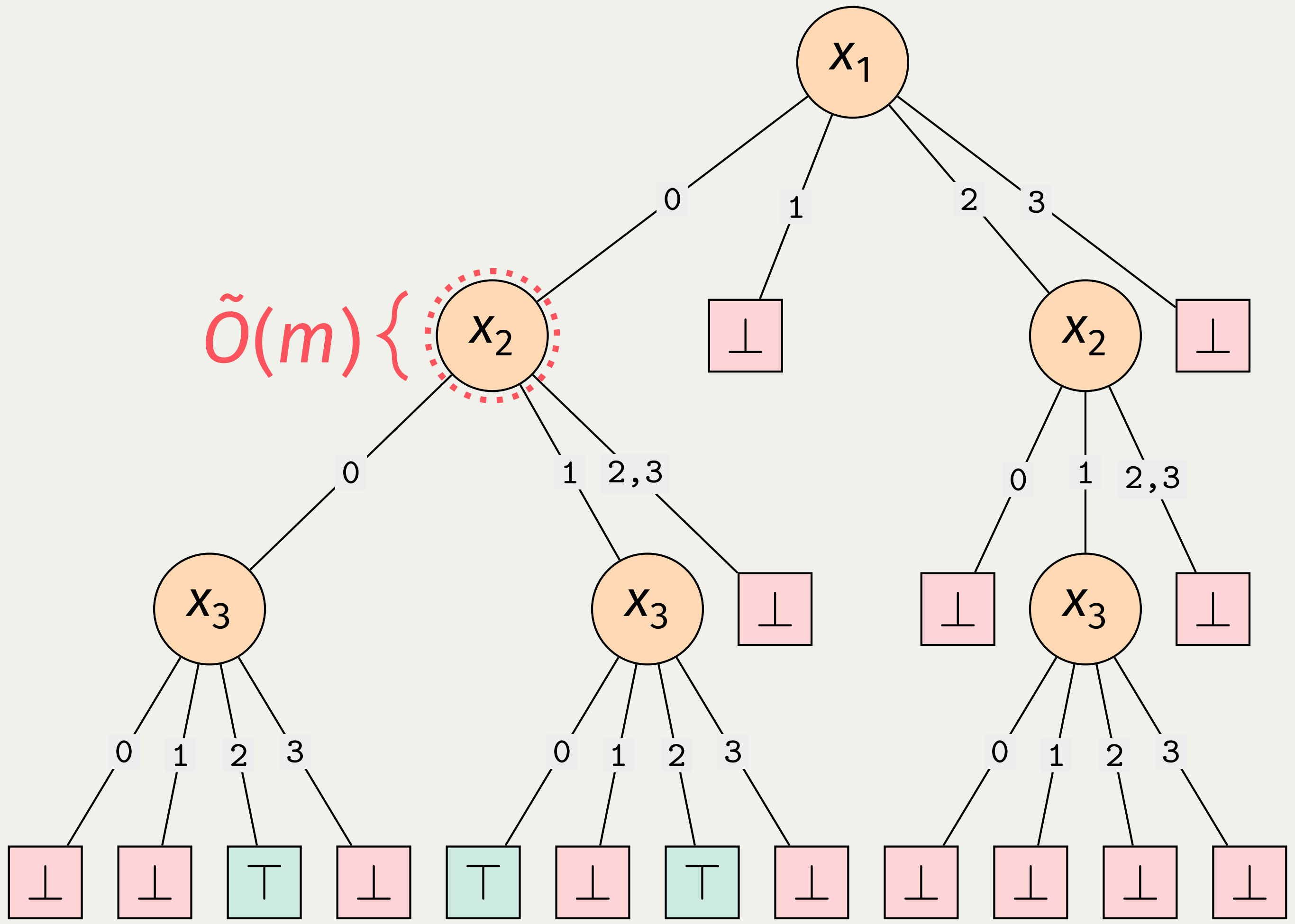
Algorithm overview



Algorithm overview



Complexity analysis



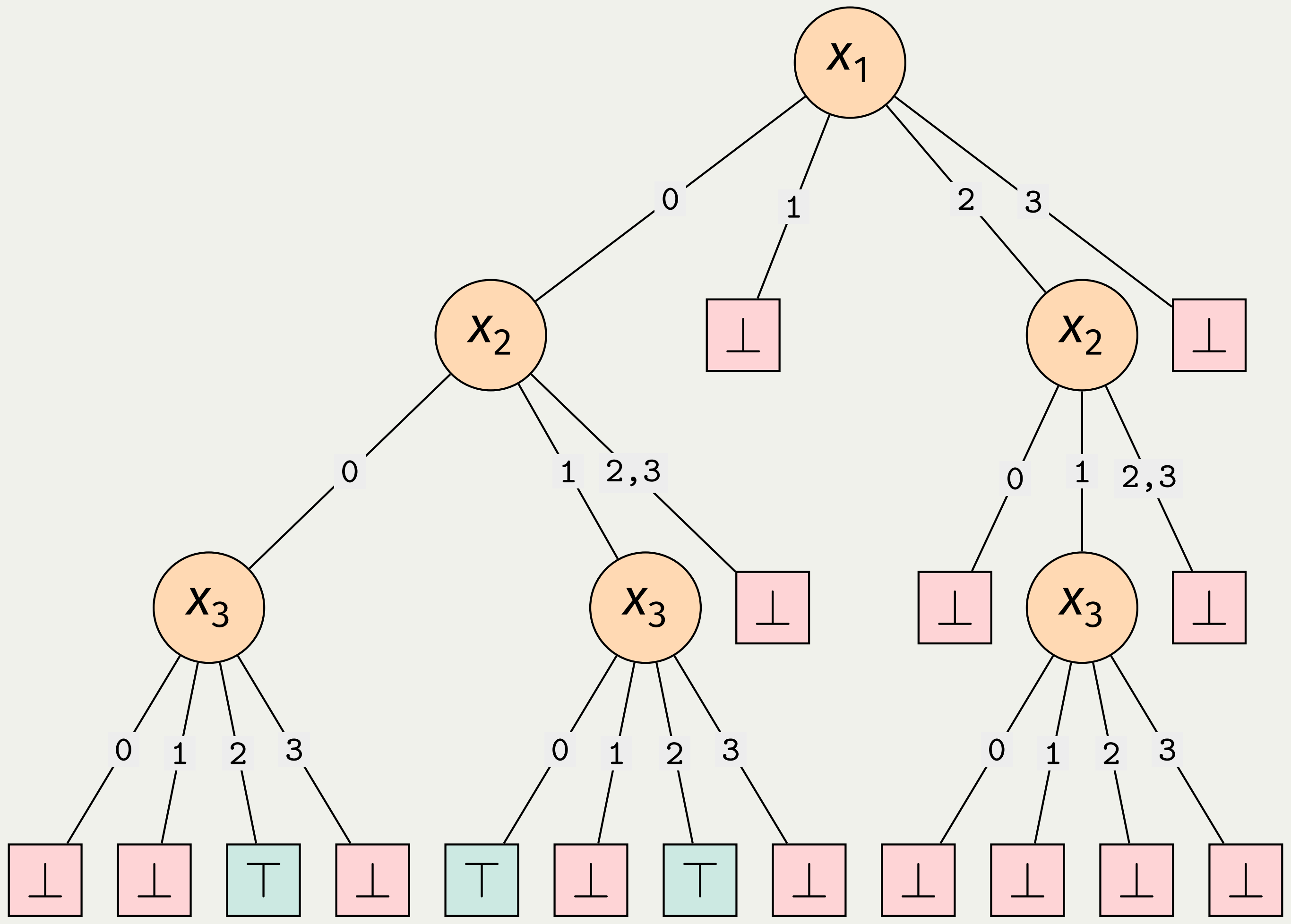
One recursive call:

- branch variable x_i on value $d \in \mathsf{dom}$
- filter/project relations with x_i: $\prod_{x_{i+1}\ldots x_n} \sigma_{x_i=d} R$
- Binary search in $O(\log |R|)$ if R ordered

  ($O(1)$ possible using tries).

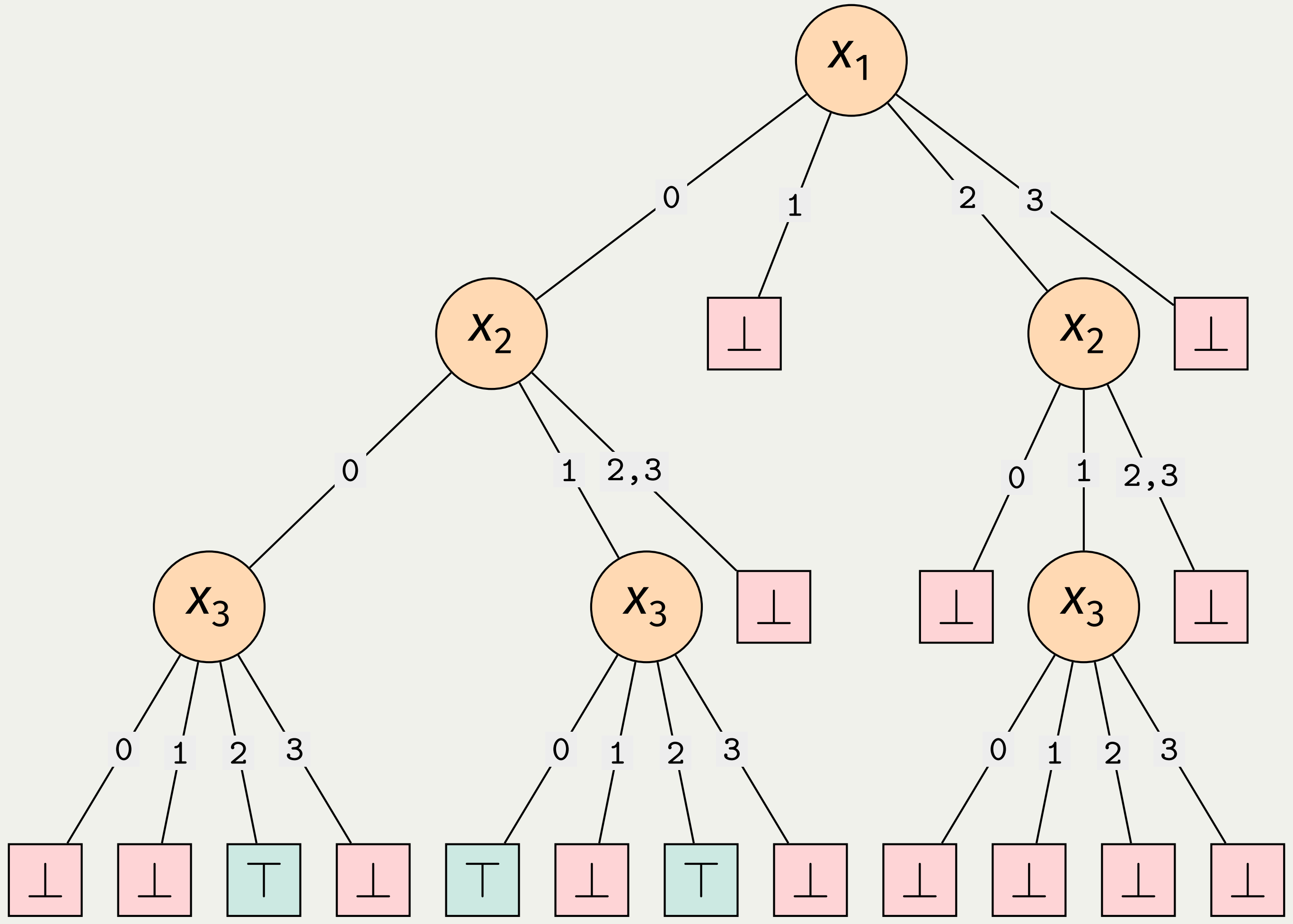
Total complexity: number of recursive calls times $\tilde{O}(m)$ where m is the number of atoms.

Number of calls



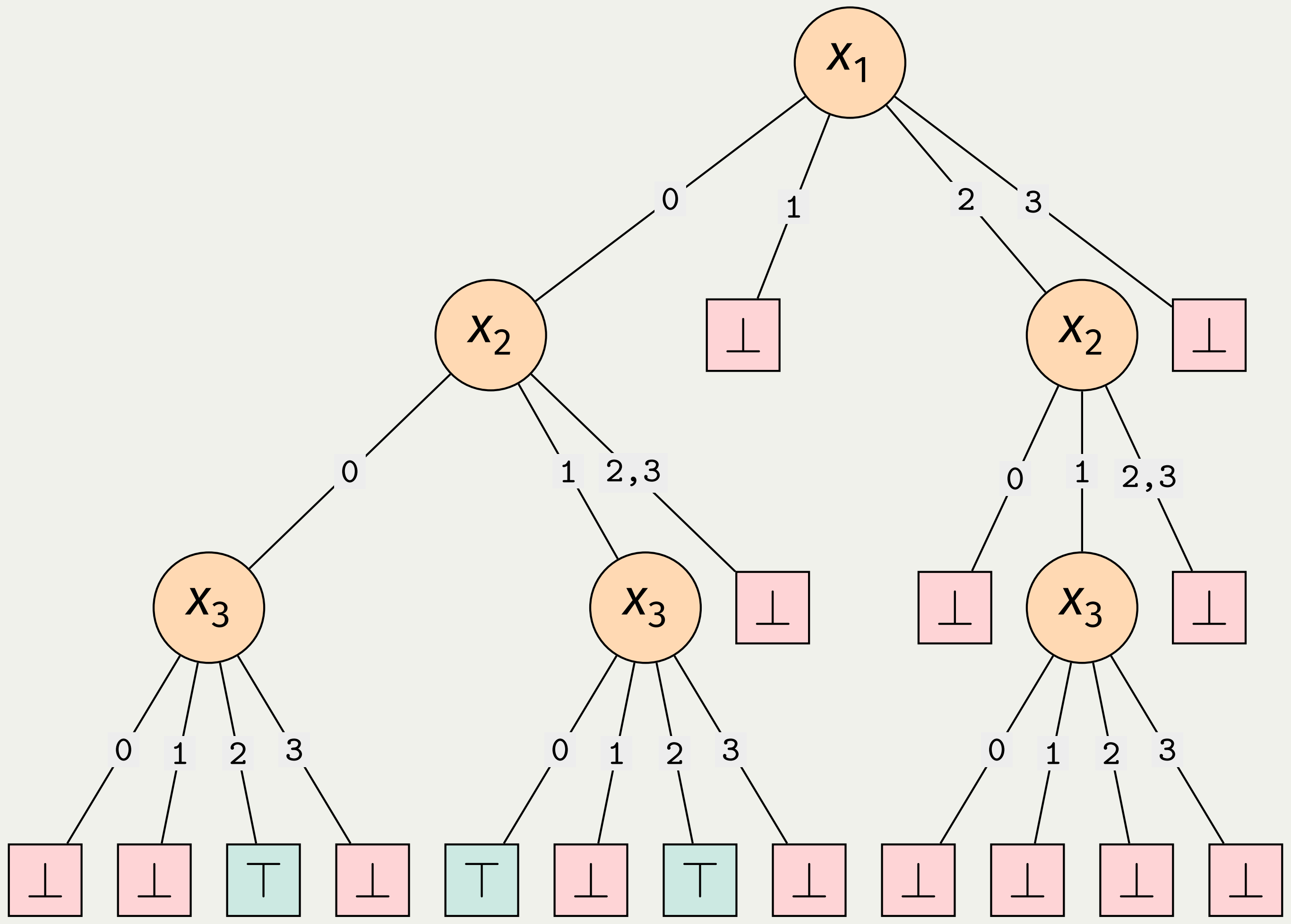
- a call = a node = a partial assignment.
- $\tau := x_1 = d_1, \ldots, x_i = d_i$ current call, not $\bot$:
  - No inconsistency.
  - $R^{\mathbb{D}}[\tau]$ not empty for each $R \in Q$
  - $\tau \in Q_i(\mathbb{D})$ for $Q_i = \bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R$
  - $\leq \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ such nodes!

Number of calls



- a call = a node = a partial assignment.
- $\tau := x_1 = d_1, \dots, x_i = d_i$ current call, not $\bot$:
  - No inconsistency.
  - $R^{\mathbb{D}}[\tau]$ not empty for each $R \in Q$
  - $\tau \in Q_i(\mathbb{D})$ for $Q_i = \bigwedge_{R \in Q} \prod_{x_1 \dots x_i} R$
  - $\leq \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ such nodes!
- $\tau := x_1 = d_1, \dots, x_{i+1} = d_{i+1}$ current call is $\bot$:
  - $x_1 = d_1, \dots, x_i = d_i$ is **not $\bot$**.
  - $\leq |\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ $\bot$-nodes!

Number of calls



- a call = a node = a partial assignment.
- $\tau := x_1 = d_1, \ldots, x_i = d_i$ current call, not $\perp$:
  - No inconsistency.
  - $R^{\mathbb{D}}[\tau]$ not empty for each $R \in Q$
  - $\tau \in Q_i(\mathbb{D})$ for $Q_i = \bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R$
  - $\leq \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ such nodes!
- $\tau := x_1 = d_1, \ldots, x_{i+1} = d_{i+1}$ current call is $\perp$:
  - $x_1 = d_1, \ldots, x_i = d_i$ is **not** $\perp$.
  - $\leq |\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ $\perp$-nodes!

At most $(|dom| + 1) \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ **calls.**
Complexity: $\tilde{O}(m|\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|)$**.**

Worst-Case Optimality

Worst case value

Consider databases for Q with a bound N on the table size:

$$\mathcal{D}_Q^{\leqslant N} = \{\mathbb{D} \mid \forall R \in Q, |R^{\mathbb{D}}| \leqslant N\}$$

and let:

$$\mathsf{wc}(Q, N) = \mathsf{sup}_{\mathbb{D} \in \mathcal{D}_Q^{\leqslant N}} |Q(\mathbb{D})|$$

$\mathsf{wc}(Q, N)$ is the worst case: the size of the biggest answer set possible with query Q and databases where each table are bounded by N.

Worst case examples

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$
    - $\mathsf{wc}(Q_\square, N) = N^2$

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$
    - $\mathsf{wc}(Q_\square, N) = N^2$
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$
  - $\mathsf{wc}(Q_\square, N) = N^2$
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$
  - Overestimation of the worst case N^2.

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$
  - $\mathsf{wc}(Q_\square, N) = N^2$
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$
  - Overestimation of the worst case N^2.
  - Actually, $\mathsf{wc}(Q_\Delta, N) = N^{1.5}$

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$
- **Similarly**: $Q_k = R_1(x_1) \wedge \cdots \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$
  - $\mathsf{wc}(Q_\square, N) = N^2$
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$
  - Overestimation of the worst case N^2.
  - Actually, $\mathsf{wc}(Q_\Delta, N) = N^{1.5}$

We know how to compute $\rho(Q)$ such that $\mathsf{wc}(Q, N) = \tilde{O}(N^{\rho(Q)})$ (this is known as the AGM-bound but we do not need it yet).

Worst case optimal join (WCOJ) algorithms

A join algorithm is **worst case optimal** (wrt $\mathcal{D}_Q^{\leqslant N}$) if for every Q, $N \in \mathbb{N}$ and $\mathbb{D} \in \mathcal{D}_Q^{\leqslant N}$, it computes $Q(\mathbb{D})$ in time

$$\tilde{O}(f(|Q|) \cdot \mathsf{wc}(Q, N))$$

- For example, it has to compute $Q_\Delta(\mathbb{D})$ in time $N^{1.5}$ where N is the largest relation in $\mathbb{D}$.
- **Naive strategy** $(R(x,y) \bowtie S(y,z)) \bowtie T(x,z)$ may take $N^2 >> N^{1.5}$.

Existing WCOJ Algorithm

Rich literature:

- **NPRR join** (PODS 2012): usual join plans but with relations partitionned into high/low degree tuples.
- **Leapfrog Triejoin**
- **Generic Join**: both branch and bound algorithm as ours but more complex analysis or data structures.
- **PANDA**

Refining our previous analysis

$$
\begin{aligned}
|Q_i(\mathbb{D})| &= |\bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R^{\mathbb{D}}| \\
&= |\bigwedge_{R \in Q} R^{\mathbb{D}'}| \\
&= |Q(\mathbb{D}')|
\end{aligned}
$$

where $R^{\mathbb{D}'} = \prod_{x_1 \ldots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \ldots, x_i}$

Refining our previous analysis

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \dots x_i} R^{\mathbb{D}}| = |\bigwedge_{R \in Q} R^{\mathbb{D}'}| = |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \dots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \dots, x_i}$

Crucial observation:

- $|R^{\mathbb{D}'}| = |\prod_{x_1 \dots x_i} R^{\mathbb{D}}| \leq |R^{\mathbb{D}}| \leq N$
- Hence $\mathbb{D}' \in \mathcal{D}_Q^{\leq N}$.
- $|Q_i(\mathbb{D})| = |Q(\mathbb{D}')| \leq \mathsf{wc}(Q, N)$

Refining our previous analysis

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \dots x_i} R^{\mathbb{D}}| = |\bigwedge_{R \in Q} R^{\mathbb{D}'}| = |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \dots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \dots, x_i}$

Crucial observation:

- $|R^{\mathbb{D}'}| = |\prod_{x_1 \dots x_i} R^{\mathbb{D}}| \leq |R^{\mathbb{D}}| \leq N$
- Hence $\mathbb{D}' \in \mathcal{D}_Q^{\leq N}$.
- $|Q_i(\mathbb{D})| = |Q(\mathbb{D}')| \leq \mathsf{wc}(Q, N)$

The complexity of the branch and bound algorithm is

Refining our previous analysis

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R^{\mathbb{D}}| = |\bigwedge_{R \in Q} R^{\mathbb{D}'}| = |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \ldots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \ldots, x_i}$

Crucial observation:

- $|R^{\mathbb{D}'}| = |\prod_{x_1 \ldots x_i} R^{\mathbb{D}}| \leq |R^{\mathbb{D}}| \leq N$
- Hence $\mathbb{D}' \in \mathcal{D}_Q^{\leq N}$.
- $|Q_i(\mathbb{D})| = |Q(\mathbb{D}')| \leq \mathsf{wc}(Q, N)$

The complexity of the branch and bound algorithm is

$$\tilde{O}(m|\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|)$$

Refining our previous analysis

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R^{\mathbb{D}}|$$
$$= |\bigwedge_{R \in Q} R^{\mathbb{D}'}|$$
$$= |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \ldots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \ldots, x_i}$

Crucial observation:

- $|R^{\mathbb{D}'}| = |\prod_{x_1 \ldots x_i} R^{\mathbb{D}}| \leq |R^{\mathbb{D}}| \leq N$
- Hence $\mathbb{D}' \in \mathcal{D}_Q^{\leq N}$.
- $|Q_i(\mathbb{D})| = |Q(\mathbb{D}')| \leq \mathsf{wc}(Q, N)$

The complexity of the branch and bound algorithm is

$$\tilde{O}(m|\mathsf{dom}| \cdot n\mathsf{wc}(Q, N))$$

Refining our previous analysis

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \dots x_i} R^{\mathbb{D}}| = |\bigwedge_{R \in Q} R^{\mathbb{D}'}| = |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \dots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \dots, x_i}$

Crucial observation:

- $|R^{\mathbb{D}'}| = |\prod_{x_1 \dots x_i} R^{\mathbb{D}}| \leq |R^{\mathbb{D}}| \leq N$
- Hence $\mathbb{D}' \in \mathcal{D}_Q^{\leq N}$.
- $|Q_i(\mathbb{D})| = |Q(\mathbb{D}')| \leq \mathsf{wc}(Q, N)$

The complexity of the branch and bound algorithm is

$$\tilde{O}(mn \cdot |\mathsf{dom}| \cdot \mathsf{wc}(Q, N))$$

Make the domain binary!

R	x	y
	1	2
	2	1
	3	0

$\rightsquigarrow$

$\tilde{R}^b$	x^2	x^1	x^0	y^2	y^1	y^0
	0	0	1	0	1	0
	0	1	0	0	0	1
	0	1	1	0	0	0

- $Q \rightsquigarrow \tilde{Q}^b$ has bn variables
- $\mathbb{D} \rightsquigarrow \tilde{\mathbb{D}}^b$ for $b = \log|\mathsf{dom}|$. Database has roughly the same bitsize but size 2 domain!

WCOJ finally

- To compute $Q(\mathbb{D})$ run simple branch and bound algorithm on $(\tilde{Q}^b, \tilde{\mathbb{D}}^b)$:
  - runs in time $\tilde{O}(m \cdot (n \log |\mathsf{dom}|) \cdot 2\mathsf{wc}(\tilde{Q}^b, N, 2))$
  - where $\mathsf{wc}(\tilde{Q}^b, N, 2)$ is the worst case for $\tilde{Q}^b$ on relations of size $\leq N$ and domain 2.
  - $\mathsf{wc}(\tilde{Q}^b, N, 2) \leq \mathsf{wc}(Q, N)$ by reconverting back to larger domain.

We hence compute $Q(\mathbb{D})$ in time $\tilde{O}(mn \cdot \mathsf{wc}(Q, N))$!

Sampling answers uniformly

Problem

Given Q and $\mathbb{D}$, sample $\tau \in Q(\mathbb{D})$ with probability $\frac{1}{|Q(\mathbb{D})|}$ or fail if $Q(\mathbb{D}) = \emptyset$.

Naive algorithm:

- materialize $Q(\mathbb{D})$ in a table
- sample $i \leq |Q(\mathbb{D})|$ uniformly
- output $Q(\mathbb{D})[i]$.

Complexity using WCOJ: $\tilde{O}(\mathsf{wc}(Q, N))$.

Problem

Given Q and $\mathbb{D}$, sample $\tau \in Q(\mathbb{D})$ with probability $\frac{1}{|Q(\mathbb{D})|}$ or fail if $Q(\mathbb{D}) = \emptyset$.

Naive algorithm:

- materialize $Q(\mathbb{D})$ in a table
- sample $i \leq |Q(\mathbb{D})|$ uniformly
- output $Q(\mathbb{D})[i]$.

Complexity using WCOJ: $\tilde{O}(\mathsf{wc}(Q, N))$.

We can do better: (expected) time $\tilde{O}(\frac{\mathsf{wc}(Q,N)}{|Q(\mathbb{D})|+1} poly(|Q|))$

PODS 23: [Deng, Lu, Tao] and [Kim, Ha, Fletcher, Han]

Problem

Given Q and $\mathbb{D}$, sample $\tau \in Q(\mathbb{D})$ with probability $\frac{1}{|Q(\mathbb{D})|}$ or fail if $Q(\mathbb{D}) = \emptyset$.

Naive algorithm:

- materialize $Q(\mathbb{D})$ in a table
- sample $i \leq |Q(\mathbb{D})|$ uniformly
- output $Q(\mathbb{D})[i]$.

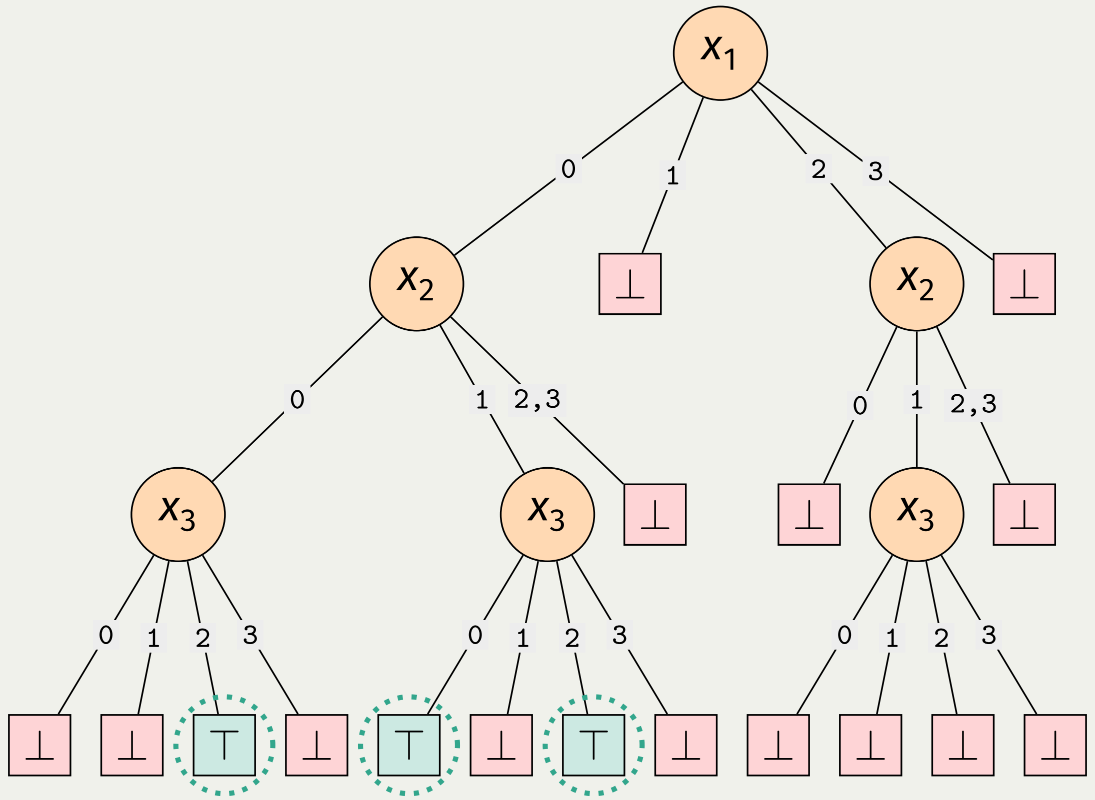
Complexity using WCOJ: $\tilde{O}(\mathsf{wc}(Q, N))$.

We can do better: (expected) time $\tilde{O}(\frac{\mathsf{wc}(Q,N)}{|Q(\mathbb{D})|+1} poly(|Q|))$

PODS 23: [Deng, Lu, Tao] and [Kim, Ha, Fletcher, Han]

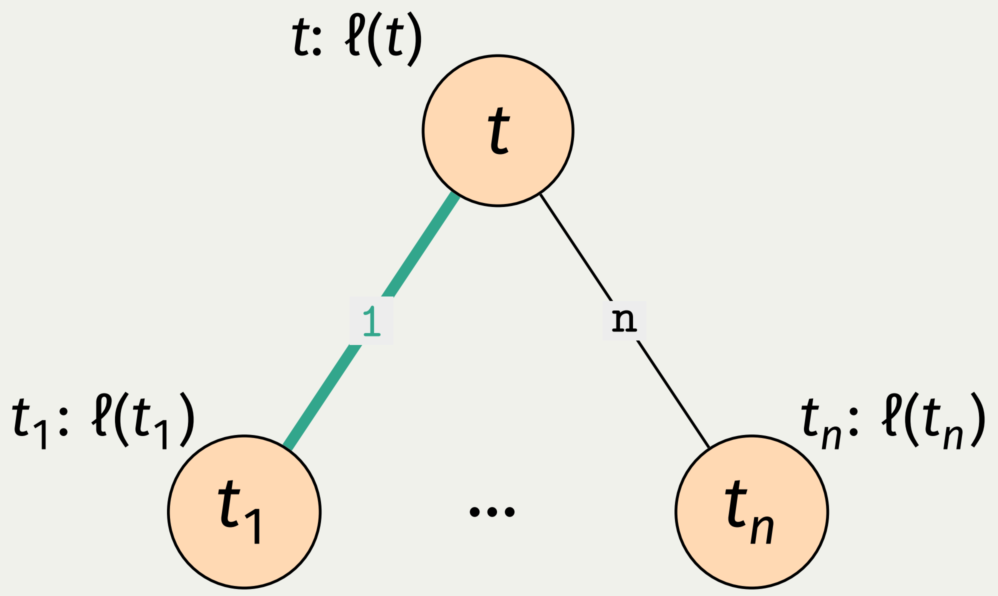
Let's do a modular proof of this fact!

Revisiting the problem



Sampling answers reduces to sampling $\top$-leaves in a tree with ($\top$,$\bot$)-labeled leaves.

Sampling leaves, the easy way

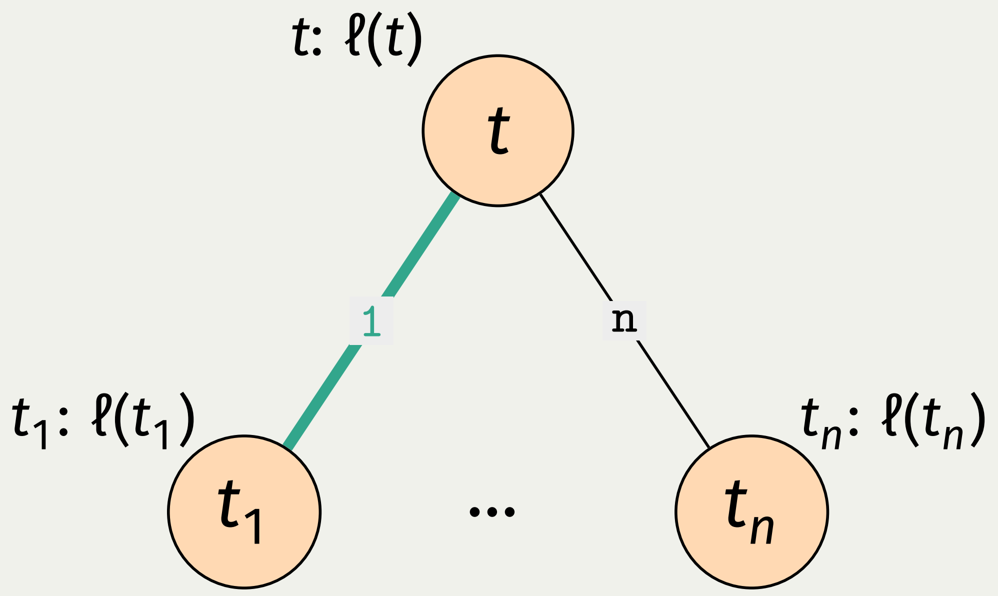


- $\ell(t)$: number of ⊤-leaves below t is known
- Recursively sample uniformly a ⊤-leaf in t_i with probability $\frac{\ell(t_i)}{\ell(t)}$.
- A leaf in $\ell(t_i)$ will hence be sampled with probability

$$\frac{1}{\ell(t_i)} \times \frac{\ell(t_i)}{\ell(t)} = \frac{1}{\ell(t)}$$

Uniform!

Sampling leaves, the easy way



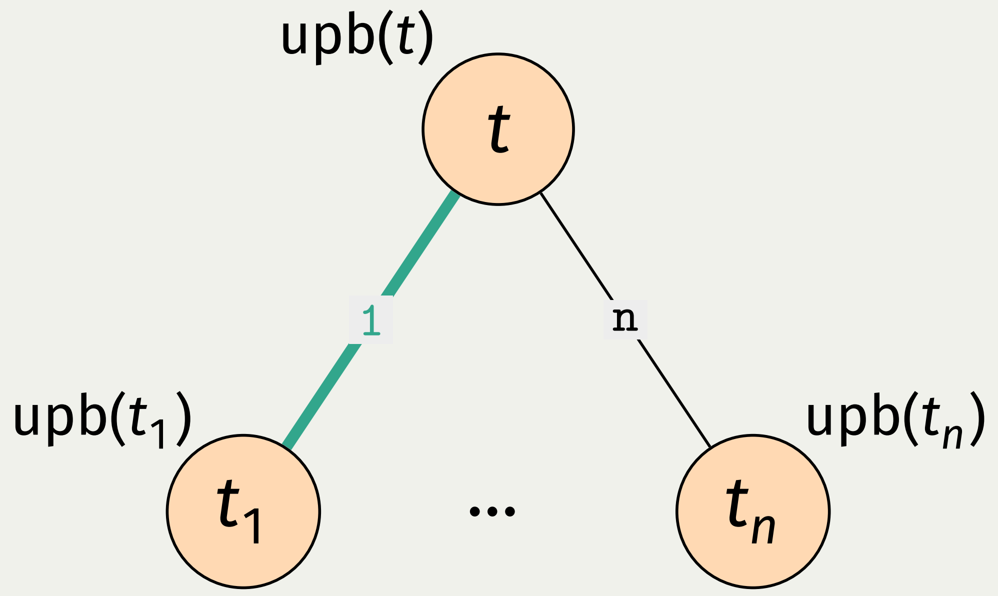
- $\ell(t)$: number of $\top$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{\ell(t_i)}{\ell(t)}$.
- A leaf in $\ell(t_i)$ will hence be sampled with probability

$$\frac{1}{\ell(t_i)} \times \frac{\ell(t_i)}{\ell(t)} = \frac{1}{\ell(t)}$$

Uniform!

In our case, we do not know $\ell(t)\ldots$

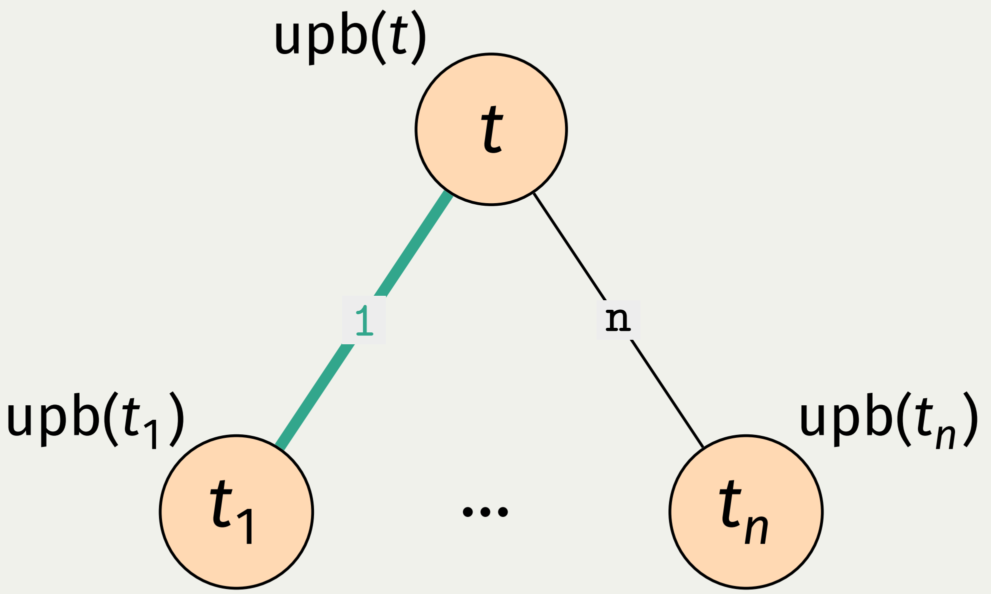
Sampling leaves with a nice oracle



- $upb(t)$: **upperbound** on the number of $\top$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{upb(t_i)}{upb(t)}$.
- **Fail** with probability $1 - \sum_i \frac{upb(t_i)}{upb(t)}$ or upon encountering $\bot$.

Only makes sense if $\sum_i upb(t_i) \leq upb(t)$.

Sampling leaves with a nice oracle



- $upb(t)$: **upperbound** on the number of $\top$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{upb(t_i)}{upb(t)}$.
- **Fail** with probability $1 - \sum_i \frac{upb(t_i)}{upb(t)}$ or upon encountering $\bot$.

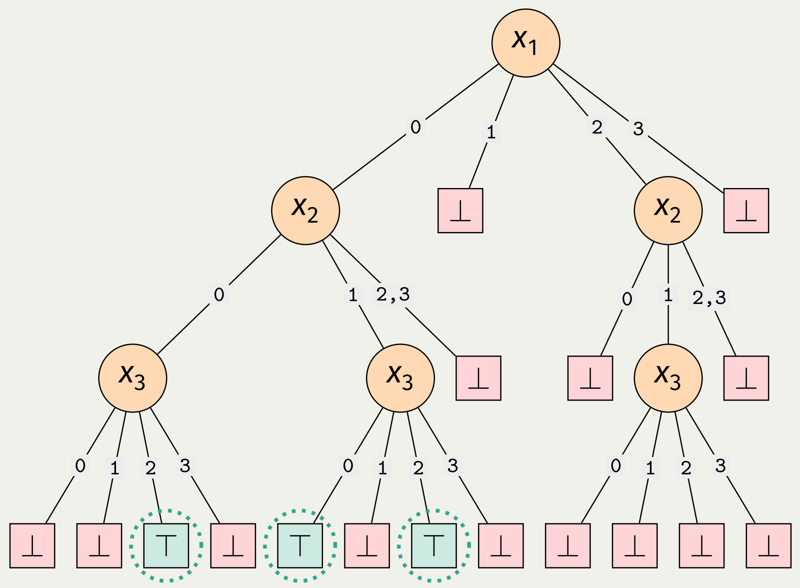
Only makes sense if $\sum_i upb(t_i) \leq upb(t)$.

Las Vegas uniform sampling algorithm:

- **each leaf is output with probability $\frac{1}{ubp(t)}$,**
- **fails with proba $1 - \frac{\ell(t)}{upb(t)}$ where $\ell(t)$ is the number of $\top$-leaves under t.**

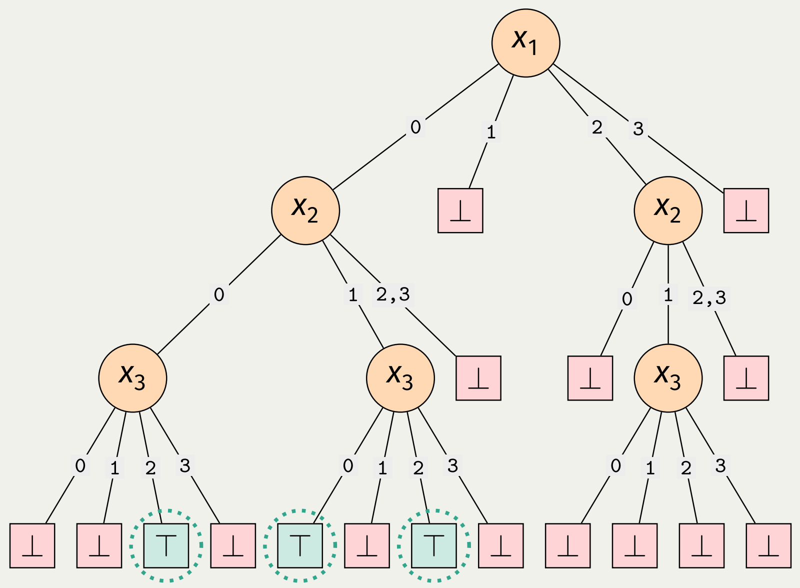
Repeat until output: $O(\frac{upb(r)}{\ell(r)})$ expected calls, where r is the root.

Upper bound oracles for conjunctive queries



- Node t: partial assignment $\tau_t := (x_1 = d_1, \ldots, x_i = d_i)$
- Number of $\boxed{\top}$ leaves below t: $|Q(\mathbb{D})[\tau_t]|$.
- $upb(t)$???: **look for worst case bounds**!

Upper bound oracles for conjunctive queries

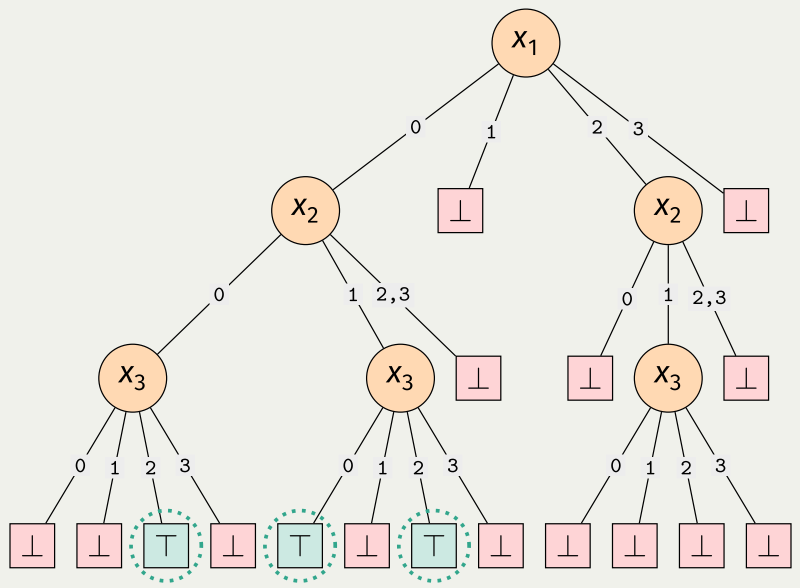


- Node t: partial assignment $\tau_t := (x_1 = d_1, \ldots, x_i = d_i)$
- Number of $\top$ leaves below t: $|Q(\mathbb{D})[\tau_t]|$.
- $upb(t)$???: **look for worst case bounds**!

AGM bound: there exists positive rational numbers $(\lambda_R)_{R \in Q}$ such that

$$|Q(\mathbb{D})| \leq \prod_{R \in Q} |R^{\mathbb{D}}|^{\lambda_R} \leq \mathsf{wc}(Q, N)$$

Upper bound oracles for conjunctive queries



- Node t: partial assignment $\tau_t := (x_1 = d_1, \dots, x_i = d_i)$
- Number of $\top$ leaves below t: $|Q(\mathbb{D})[\tau_t]|$.
- $upb(t)$???: **look for worst case bounds**!

AGM bound: there exists positive rational numbers $(\lambda_R)_{R \in Q}$ such that

$$|Q(\mathbb{D})| \leq \prod_{R \in Q} |R^{\mathbb{D}}|^{\lambda_R} \leq \mathsf{wc}(Q, N)$$

Define $upb(t) = \prod_{R \in Q} |R^{\mathbb{D}}[\tau_t]|^{\lambda_R} \leq \mathsf{wc}(Q, N)$:

- it is an upper bound on $|Q(\mathbb{D})[\tau_t]|$,
- it is supperadditive: $upb(t) \geq \sum_{d \in \mathsf{dom}} upb(t_d)$
- value of upb at the root of the tree: $\mathsf{wc}(Q, N)$!

Wrapping up sampling

Given a super-additive function upperbounding the number of T-leaves in a tree at each node, we have:

Las Vegas uniform sampling algorithm:

- **each leaf is output with probability** $\frac{1}{ubp(t)}$
- **fails with proba** $1 - \frac{\ell(t)}{upb(t)}$

Repeat until output: $O(\frac{upb(r)}{\ell(r)})$ **expected calls.**

Wrapping up sampling

Given a super-additive function upperbounding the number of $\top$-leaves in a tree at each node, we have:

Las Vegas uniform sampling algorithm:

- **each leaf /answer is output with probability** $\frac{1}{ubp(t)} = \frac{1}{wc(Q,N)}$
- **fails with proba** $1 - \frac{\ell(t)}{upb(t)} = 1 - \frac{|Q(\mathbb{D})|}{wc(Q,N)}$

Repeat until output: $O(\frac{upb(r)}{\ell(r)}) = \frac{\mathsf{wc}(Q,N)}{1+|Q(\mathbb{D})|}$ **expected calls.**

Wrapping up sampling

Given a super-additive function upperbounding the number of $\top$-leaves in a tree at each node, we have:

Las Vegas uniform sampling algorithm:

- **each leaf /answer is output with probability** $\frac{1}{ubp(t)} = \frac{1}{wc(Q,N)}$
- **fails with proba** $1 - \frac{\ell(t)}{upb(t)} = 1 - \frac{|Q(\mathbb{D})|}{wc(Q,N)}$

Repeat until output: $O(\frac{upb(r)}{\ell(r)}) = \frac{\mathsf{wc}(Q,N)}{1+|Q(\mathbb{D})|}$ **expected calls.**

Final complexity: binarize to navigate the tree in $\tilde{O}(nm)$: $\tilde{O}(nm \cdot \frac{\mathsf{wc}(Q,N)}{1+|Q(\mathbb{D})|})$

Matches existing results, proof more modular.

Beyond Cardinality Constraints

Worst case and constraints

So far we have considered worst case wrt this class:

- $\mathcal{D}_Q^{\leqslant N} = \{\mathbb{D} \mid \forall R \in Q, |R^{\mathbb{D}}| \leqslant N\}$
- $\mathsf{wc}(Q, N) = \sup_{\mathbb{D} \in \mathcal{D}_Q^{\leqslant N}} |Q(\mathbb{D})|$

Each relation is subject to a cardinality constraint of size N.

What if we know that our instance has some extra properties (e.g., a *functional dependency*)

- We know $\mathbb{D} \in \mathcal{C} \subseteq \mathcal{D}_Q^{\leqslant N}$
- We want the join to run in $\tilde{O}(f(|Q|) \cdot \mathsf{wc}(Q, \mathcal{C}))$ where $\mathsf{wc}(Q, \mathcal{C}) := \sup_{\mathbb{D} \in \mathcal{C}} |Q(\mathbb{D})|$.

In this case, we say that our algorithm is worst case optimal wrt $\mathcal{C}$.

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

- Let $\mathcal{C}$ be the class of databases where $|R| \leq N, |S| \leq N$ and R respect functional dependency $x_2 \to x_1$.
- $\mathsf{wc}(Q, \mathcal{C}) \leq N$ because each tuple of $S^{\mathbb{D}}$ can be extended to *at most one solution*.

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

- Let $\mathcal{C}$ be the class of databases where $|R| \leq N, |S| \leq N$ and R respect functional dependency $x_2 \to x_1$.
- $\mathsf{wc}(Q, \mathcal{C}) \leq N$ because each tuple of $S^{\mathbb{D}}$ can be extended to *at most one solution*.

Is our simple join worst case optimal for this class?

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

- Let $\mathcal{C}$ be the class of databases where $|R| \leq N, |S| \leq N$ and R respect functional dependency $x_2 \to x_1$.
- $\mathsf{wc}(Q, \mathcal{C}) \leq N$ because each tuple of $S^{\mathbb{D}}$ can be extended to *at most one solution*.

Is our simple join worst case optimal for this class?

Short answer: yes if x_2 is set before x_1.

Prefix closed classes

Recall the complexity of our algorithm: $\tilde{O}(m|\mathsf{dom}|\sum_{i=1}^{n}|Q_i(\mathbb{D})|))$ where $Q_i = \bigwedge_{R \in Q} \prod_{x_1,\dots,x_i} R$

A class of database $\mathcal{C}$ for Q is **prefix closed for order** $\pi = (x_1, \dots, x_n)$ if for each i and $\mathbb{D} \in \mathcal{C}$:

$$|Q_i(\mathbb{D})| \leq \mathsf{wc}(\mathcal{C})$$

$\mathcal{D}_Q^{\leq N}$ is prefix closed (for any order)!

Our algorithm is (almost) worst case optimal as long as we use an order for which $\mathcal{C}$ is prefix closed!

Acyclic functional dependencies

$F = (X_1 \to Y_1, \ldots, X_k \to Y_k)$ is a set of functional dependencies:

- $G(F)$: vertices are the variables and $x \to y$ if $x \in X_i$ and $y \in Y_i$ for some i.
- If $G(F)$ is acyclic, then let $\pi = x_1, \ldots, x_n$ be a topological sort of $G(F)$. Then

$$\mathcal{C}_F^N = \{\mathbb{D} \mid \mathbb{D} \text{ respects } F\} \cap \mathcal{D}_Q^{\leqslant N}$$

is **prefix closed for order π** (exactly the same proof as for cardinality constraints).

Hence our algorithm is worst case optimal wrt $\mathcal{C}_F^N$ (as long as we follow π).

We need to show that this functional dependencies transfer in the binarised setting but it is almost immediate.

Degree constraints

A **degree constraint** is a constraint $(X, Y, N_{Y|X})$ where $X \subseteq Y$. A relation R verifies the constraint if

$$|\max_{\tau \in \text{dom}^X} \prod_Y R[\tau]| \leq N_{Y|X}$$

- Cardinality constraint = degree constraint with $X = \emptyset$.
- Functional dependency = degree constraint with $N_{Y|X} = 1$.

Acyclic degree constraints

$\Delta = \{(X_1, Y_1, N_1) \dots, (X_k, Y_k, N_k)\}$ set of degree constraints.

- $G(\Delta)$: vertices are the variables and $x \to y$ if $x \in X_i$ and $y \in Y_i$ for some i.
- If $G(\Delta)$ is acyclic, then let $\pi = x_1, \dots, x_n$ be a topological sort of $G(\Delta)$. Then

$$\mathcal{C}_\Delta^N = \{\mathbb{D} \mid \mathbb{D} \text{ respects } \Delta\} \cap \mathcal{D}_Q^{\leqslant N}$$

is **prefix closed for order π** (exactly the same proof as for cardinality constraints).

Hence our algorithm is worst case optimal wrt $\mathcal{C}_\Delta^N$ (as long as we follow π).

We need to show that this functional dependencies transfer in the binarised setting but it is almost immediate.

Bonus: sampling acyclic degree constraints

We can find (λ_R) such that $\prod_{R \in Q} |R^{\mathbb{D}}|^{\lambda_R} \leq \tilde{O}(\mathsf{wc}(Q, \mathcal{C}_\Delta^N))$ for any $\mathbb{D} \in \mathcal{C}_\Delta^N$ (**polymatroid bound**).

Define $upb(t) := \prod_{R \in Q} |R^{\mathbb{D}}[\tau_t]|^{\lambda_R}$:

- upperbound of $Q(\mathbb{D})[\tau_t]$ for any $\mathbb{D} \in \mathcal{C}_\Delta^N$,
- superadditive.

We have sampling with complexity $\tilde{O}(nm \cdot \frac{\mathsf{wc}(Q, \mathcal{C}_\Delta^N)}{1+|Q(\mathbb{D})|})$

Conclusion

- Simple algorithms and analysis
- Modular:
    - join is worst-case optimal as soon as the class is prefix closed
    - sampling is in $\frac{\mathsf{wc}(Q,\mathcal{C})}{|Q(\mathbb{D})|}$ as long as one can provide a super additive upper bound

Future work:

- Other classes such as:
    - cyclic FD,
    - general system of degree constraints (as PANDA)
- Explore dynamic ordering: can we capture more classes?