

Direct Access for Conjunctive Queries with Negations

Florent Capelli, Oliver Irwin

CRIL, Université d'Artois

June 13, 2024

Direct Access on Join Queries

Join Queries

Join Query : $Q (x_1, \dots, x_n) = \bigwedge_{i=1}^k R_i (\mathbf{x}_i)$
where \mathbf{x}_i is a tuple over $X = \{x_1, \dots, x_n\}$

Join Queries

$$\text{Join Query : } Q (x_1, \dots, x_n) = \bigwedge_{i=1}^k R_i (\mathbf{x}_i)$$

where \mathbf{x}_i is a tuple over $X = \{x_1, \dots, x_n\}$

Example:

$$Q (\text{city, country, name, id}) = \text{People} (\text{id, name, city}) \wedge \text{Capitals} (\text{city, country})$$

id	name	city
1	Alice	Paris
2	Bob	Lens
3	Chiara	Rome
4	Djibril	Berlin
5	Émile	Dortmund
6	Francesca	Rome

Capitals	
city	country
Berlin	Germany
Paris	France
Rome	Italy

Join Queries

$$\text{Join Query : } Q (x_1, \dots, x_n) = \bigwedge_{i=1}^k R_i (\mathbf{x}_i)$$

where \mathbf{x}_i is a tuple over $X = \{x_1, \dots, x_n\}$

Example:

$$Q (\text{city, country, name, id}) = \text{People} (\text{id, name, city}) \wedge \text{Capitals} (\text{city, country})$$

id	name	city
1	Alice	Paris
2	Bob	Lens
3	Chiara	Rome
4	Djibril	Berlin
5	Émile	Dortmund
6	Francesca	Rome

Capitals	
city	country
Berlin	Germany
Paris	France
Rome	Italy

Join Queries

$$\text{Join Query : } Q (x_1, \dots, x_n) = \bigwedge_{i=1}^k R_i (\mathbf{x}_i)$$

where \mathbf{x}_i is a tuple over $X = \{x_1, \dots, x_n\}$

Example:

$$Q (\text{city, country, name, id}) = \text{People} (\text{id, name, city}) \wedge \text{Capitals} (\text{city, country})$$

id	name	city
1	Alice	Paris
2	Bob	Lens
3	Chiara	Rome
4	Djibril	Berlin
5	Émile	Dortmund
6	Francesca	Rome

Capitals	
city	country
Berlin	Germany
Paris	France
Rome	Italy

Q (D)			
city	country	name	id
Paris	France	Alice	1
Rome	Italy	Chiara	3
Berlin	Germany	Djibril	4
Rome	Italy	Francesca	6

Direct Access

Quickly access $Q(\mathbb{D})[k]$, **the k^{th} element of $Q(\mathbb{D})$.**

city	country	name	id
Paris	France	Alice	1
Rome	Italy	Chiara	3
Berlin	Germany	Djibril	4
Rome	Italy	Francesca	6

Direct Access

Quickly access $Q(\mathbb{D})[k]$, **the k^{th} element of $Q(\mathbb{D})$.**

city	country	name	id
Paris	France	Alice	1
Rome	Italy	Chiara	3
Berlin	Germany	Djibril	4
Rome	Italy	Francesca	6

$Q(\mathbb{D})[2]?$
 $(\textit{Rome}, \textit{Italy}, \textit{Chiara}, 3)$.

Naive Direct Access

Naive algorithm: materialize $Q(\mathbb{D})$ in an array, sort it. Access.

city	country	name	id
...
Berlin	Germany	Djibril	4
...
Paris	France	Alice	1
...
Rome	Italy	Chiara	3
Rome	Italy	Francesca	6
...

$Q(\mathbb{D})[1432] = ??$

Naive Direct Access

Naive algorithm: materialize $Q(\mathbb{D})$ in an array, sort it. Access.

city	country	name	id
...
Berlin	Germany	Djibril	4
...
Paris	France	Alice	1
...
Rome	Italy	Chiara	3
Rome	Italy	Francesca	6
...

$Q(\mathbb{D})[1432] = ??$

Precomputation : $O(\#Q(\mathbb{D}))$ at least (may be worse), **very costly**

Access : $O(1)$, **nearly free**

Naive Direct Access

Naive algorithm: materialize $Q(\mathbb{D})$ in an array, **sort it**. Access.

city	country	name	id
...
Berlin	Germany	Djibril	4
...
Paris	France	Alice	1
...
Rome	Italy	Chiara	3
Rome	Italy	Francesca	6
...

$Q(\mathbb{D})[1432] = ??$

Precomputation : $O(\#Q(\mathbb{D}))$ at least (may be worse), **very costly**

Access : $O(1)$, **nearly free**

Orders ?

1. Order by **weights**
2. **Lexicographical orders**
 - order on the vars of Q
 - order on domain D of \mathbb{D}

Orders ?

1. Order by **weights**

2. **Lexicographical orders**

- order on the vars of Q
- order on domain D of \mathbb{D}

Variable order ($city, country, name, id$) :

city	country	name	id
Berlin	Germany	Djibril	4
Paris	France	Alice	1
Rome	Italy	Chiara	3
Rome	Italy	Francesca	6

Orders ?

1. Order by **weights**

2. **Lexicographical orders**

- order on the vars of Q
- order on domain D of \mathbb{D}

Variable order ($city, country, name, id$) :

city	country	name	id
Berlin	Germany	Djibril	4
Paris	France	Alice	1
Rome	Italy	Chiara	3
Rome	Italy	Francesca	6

In this talk: only lexicographical orders.

Applications

Direct Access generalizes many tasks that have been previously studied:

- **Uniform sampling** without repetitions
- **Ranked enumeration**
- **Counting queries:**
 - how many answers between τ_1 and τ_2 ?
 - how many answers extend a *partial answer* etc.

Beating the Naive Approach

Beating Naive Direct Access

Naive Direct Access:

- Preprocessing at least $O(\#Q(\mathbb{D}))$.
- Access time $O(1)$.

Can we have better preprocessing and reasonable access time?

Beating Naive Direct Access

Naive Direct Access:

- Preprocessing at least $O(\#Q(\mathbb{D}))$.
- Access time $O(1)$.

Can we have better preprocessing and reasonable access time?

“Ideal” complexity:

- $O(|\mathbb{D}|)$ preprocessing
- $O(\log |\mathbb{D}|)$ access time

Complexity of Direct Access

Computing $\#Q(\mathbb{D})$ given Q and \mathbb{D} is $\#P$ -hard.

No Direct Access algorithm with good guarantees for every Q and \mathbb{D} .

Complexity of Direct Access

Computing $\#Q(\mathbb{D})$ given Q and \mathbb{D} is $\#P$ -hard.

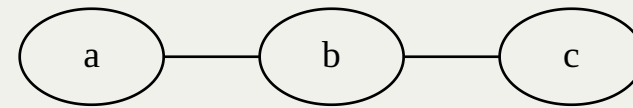
No Direct Access algorithm with good guarantees for every Q and \mathbb{D} .

Data complexity assumption: for a fixed Q , what is the best preprocessing $f(|\mathbb{D}|)$ for an access time $O(\text{polylog}|\mathbb{D}|)$?

In this work, all presented complexity in data complexity will also be polynomial for combined complexity.

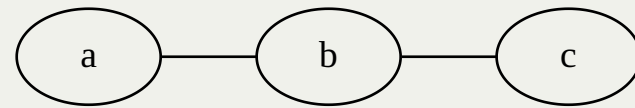
An easy query?

$$Q(a, b, c) = A(a, b) \wedge B(b, c).$$



An easy query?

$$Q(a, b, c) = A(a, b) \wedge B(b, c).$$

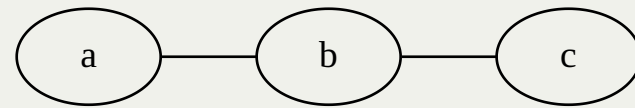


Direct Access for lexicographical order induced by (a, b, c) ?

- Precomputation $O(|\mathbb{D}|)$
- Access time $O(\log |\mathbb{D}|)$

An easy query?

$$Q(a, b, c) = A(a, b) \wedge B(b, c).$$



Direct Access for lexicographical order induced by (a, b, c) ?

- Precomputation $O(|\mathbb{D}|)$
- Access time $O(\log |\mathbb{D}|)$

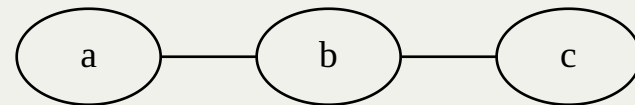
a	b	b	c
0	0	0	0
0	0	0	1
1	1	0	2
1	1	1	1
2	1	1	2

Precomputation :

- $\#Q(0, 0, _) = 3$
- $\#Q(1, 1, _) = 2$
- $\#Q(2, 1, _) = 2$

An easy query?

$$Q(a, b, c) = A(a, b) \wedge B(b, c).$$



Direct Access for lexicographical order induced by (a, b, c) ?

- Precomputation $O(|\mathbb{D}|)$
- Access time $O(\log |\mathbb{D}|)$

a	b
0	0
1	1
2	1

b	c
0	0
0	1
0	2
1	1
1	2

Precomputation :

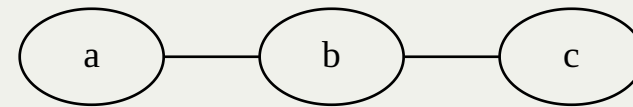
- $\#Q(0, 0, _) = 3$
- $\#Q(1, 1, _) = 2$
- $\#Q(2, 1, _) = 2$

Access $Q[5]$:

- $a = 0, b = 0$: not enough solutions
- $a = 1, b = 1$: enough! 3 solutions smaller than $(1, 1, _)$
- Look for the **second** solution of $B(1, _) : a = 1, b = 1, c = 2$

A not so easy query

$$Q(a, c, b) = A(a, b) \wedge B(b, c).$$

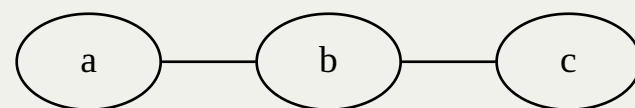


Direct Access for lexicographical order induced by (a, c, b) ?

- **Precomputation** $O(|\mathbb{D}|^2)$
- **Access time** $O(\log |\mathbb{D}|)$

A not so easy query

$$Q(a, c, b) = A(a, b) \wedge B(b, c).$$



Direct Access for lexicographical order induced by (a, c, b) ?

- **Precomputation** $O(|\mathbb{D}|^2)$
- **Access time** $O(\log |\mathbb{D}|)$

Reduces to multiplying two $\{0, 1\}$ -matrices M, N over \mathbb{N} :

- $(i, j) \in A$ iff $M[i, j] = 1$, $(j, k) \in N$ iff $N[j, k] = 1$
- $\#Q(i, j, -) = (MN)[i, j]$
- Direct Access can be used to find $\#Q(i, j, -)$ with $O(\log |\mathbb{D}|)$ queries.

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :
 - preprocessing $\tilde{O}\left(|\mathbb{D}|^{\iota(Q, \pi)}\right)$

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :
 - preprocessing $\tilde{O}\left(|\mathbb{D}|^{\iota(Q, \pi)}\right)$
 - access $O(\log |\mathbb{D}|)$

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :
 - preprocessing $\tilde{O}\left(|\mathbb{D}|^{\iota(Q, \pi)}\right)$
 - access $O(\log |\mathbb{D}|)$
- **Tight fine-grained lower bounds:**

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :
 - preprocessing $\tilde{O}\left(|\mathbb{D}|^{\iota(Q, \pi)}\right)$
 - access $O(\log |\mathbb{D}|)$
- **Tight fine-grained lower bounds**:
 - if possible with $\tilde{O}\left(|\mathbb{D}|^k\right)$ preprocessing with $k < \iota(Q, \pi)$

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :
 - preprocessing $\tilde{O}\left(|\mathbb{D}|^{\iota(Q, \pi)}\right)$
 - access $O(\log |\mathbb{D}|)$
- **Tight fine-grained lower bounds**:
 - if possible with $\tilde{O}\left(|\mathbb{D}|^k\right)$ preprocessing with $k < \iota(Q, \pi)$
 - then *Zero-Clique Conjecture* is false

(we can find 0-weighted k -cliques in graphs in time $< |G|^{k-\varepsilon}$)

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

Characterizing preprocessing time

Given a query Q and order π on its variables, we can compute $\iota(Q, \pi)$ such that:

- **Tractable Direct access** for Q on \mathbb{D} :
 - preprocessing $\tilde{O}\left(|\mathbb{D}|^{\iota(Q, \pi)}\right)$
 - access $O(\log |\mathbb{D}|)$
- **Tight fine-grained lower bounds:**
 - if possible with $\tilde{O}\left(|\mathbb{D}|^k\right)$ preprocessing with $k < \iota(Q, \pi)$
 - then *Zero-Clique Conjecture* is false

(we can find 0-weighted k -cliques in graphs in time $< |G|^{k-\varepsilon}$)

- Function ι closely related to fractional hypertree width.

1. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries*, N. Carmeli, N. Tziavelis, W. Gatterbauer, B. Kimelfeld, M. Riedewald
2. *Tight Fine-Grained Bounds for Direct Access on Join Queries*, K. Bringmann, N. Carmeli, S. Mengel

End of the story?

So, if we understand everything for Direct Access and lexicographical orders, what is **our contribution**?

Signed Join Queries

Definition

$$Q = \bigwedge_{i=1}^k P_i(\mathbf{z}_i) \bigwedge_{i=1}^l \neg N_i(\mathbf{z}_i)$$

Negation interpreted **over a given domain D** :

Definition

$$Q = \bigwedge_{i=1}^k P_i(\mathbf{z}_i) \bigwedge_{i=1}^l \neg N_i(\mathbf{z}_i)$$

Negation interpreted **over a given domain D** :

N		
x_1	x_2	x_3
0	1	0

Definition

$$Q = \bigwedge_{i=1}^k P_i(\mathbf{z}_i) \bigwedge_{i=1}^l \neg N_i(\mathbf{z}_i)$$

Negation interpreted **over a given domain D** :

N		
x_1	x_2	x_3
0	1	0

$\neg N$ on $\{0, 1\}$		
x_1	x_2	x_3
0	0	0
0	0	1
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Definition

$$Q = \bigwedge_{i=1}^k P_i(\mathbf{z}_i) \bigwedge_{i=1}^l \neg N_i(\mathbf{z}_i)$$

Negation interpreted **over a given domain D** :

N		
x_1	x_2	x_3
0	1	0

$\neg N$ on $\{0, 1\}$		
x_1	x_2	x_3
0	0	0
0	0	1
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

- $\neg N(x_1, \dots, x_k)$ encoded with $|D|^k - \#N$ tuples.
- Relation with SAT: $\neg N$ is $x_1 \vee \neg x_2 \vee x_3$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ?$
- $Q(\mathbb{D}) [2] ?$
- $Q(\mathbb{D}) [3] ?$
- $Q(\mathbb{D}) [k] ?$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ? x_1 = 0, x_2 = 0, x_3 = 0$ ie $[0]_2!$
- $Q(\mathbb{D}) [2] ?$
- $Q(\mathbb{D}) [3] ?$
- $Q(\mathbb{D}) [k] ?$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ? x_1 = 0, x_2 = 0, x_3 = 0$ ie $[0]_2!$
- $Q(\mathbb{D}) [2] ? x_1 = 0, x_2 = 0, x_3 = 1$ ie $[1]_2!$
- $Q(\mathbb{D}) [3] ?$
- $Q(\mathbb{D}) [k] ?$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ? x_1 = 0, x_2 = 0, x_3 = 0$ ie $[0]_2!$
- $Q(\mathbb{D}) [2] ? x_1 = 0, x_2 = 0, x_3 = 1$ ie $[1]_2!$
- $Q(\mathbb{D}) [3] ?$
 $x_1 = 0, x_2 = 1, x_3 = 0$ ie $[2]_2?$
- $Q(\mathbb{D}) [k] ?$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ? x_1 = 0, x_2 = 0, x_3 = 0$ ie $[0]_2!$
- $Q(\mathbb{D}) [2] ? x_1 = 0, x_2 = 0, x_3 = 1$ ie $[1]_2!$
- $Q(\mathbb{D}) [3] ?$
 $x_1 = 0, x_2 = 1, x_3 = 1$ ie $[3]_2!$
- $Q(\mathbb{D}) [k] ?$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ? x_1 = 0, x_2 = 0, x_3 = 0$ ie $[0]_2!$
- $Q(\mathbb{D}) [2] ? x_1 = 0, x_2 = 0, x_3 = 1$ ie $[1]_2!$
- $Q(\mathbb{D}) [3] ?$
 $x_1 = 0, x_2 = 1, x_3 = 1$ ie $[3]_2!$
- $Q(\mathbb{D}) [k] ? [k - 1 + p_k]_2$
 where $p_k = \{t \in N \mid t \leq k\}$

Positive Encoding not Optimal

$$Q(x_1, \dots, x_n) = \neg N(x_1, \dots, x_n), \text{ domain } \{0, 1\}.$$

Positive encoding: preprocessing $O(2^n)$

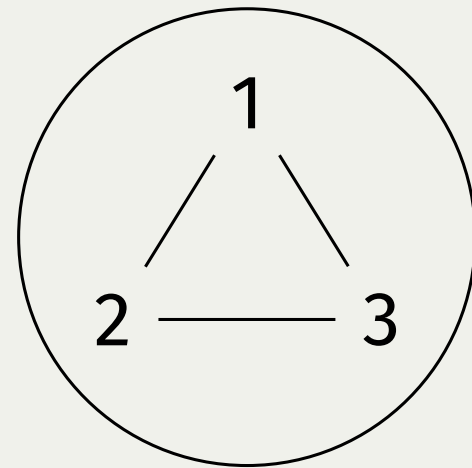
N		
x_1	x_2	x_3
0	1	0
1	0	1

- $Q(\mathbb{D}) [1] ? x_1 = 0, x_2 = 0, x_3 = 0$ ie $[0]_2!$
- $Q(\mathbb{D}) [2] ? x_1 = 0, x_2 = 0, x_3 = 1$ ie $[1]_2!$
- $Q(\mathbb{D}) [3] ?$
 $x_1 = 0, x_2 = 1, x_3 = 1$ ie $[3]_2!$
- $Q(\mathbb{D}) [k] ? [k - 1 + p_k]_2$
 where $p_k = \{t \in N \mid t \leq k\}$

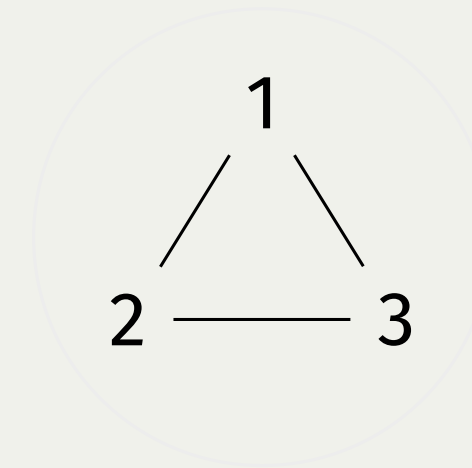
Linear preprocessing!

Hardness of subqueries

$$Q_1 = R(1, 2, 3) \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



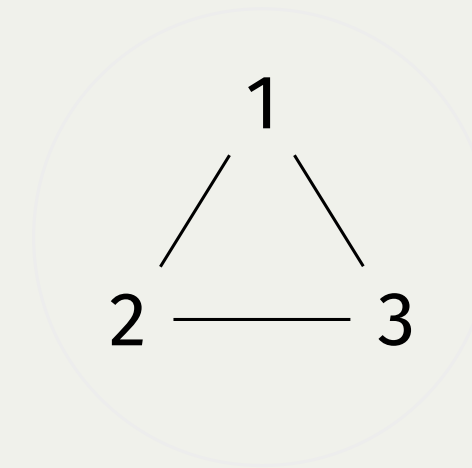
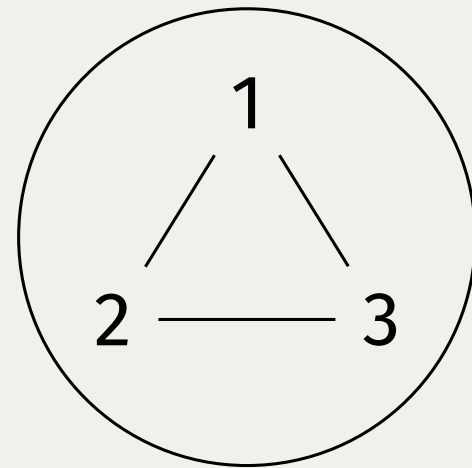
$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



Hardness of subqueries

$$Q_1 = R(1, 2, 3) \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$

$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$

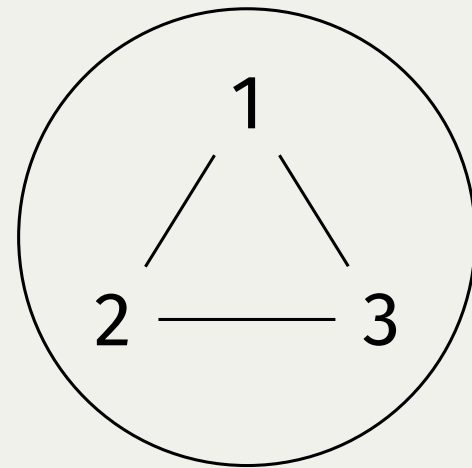


linear preprocessing

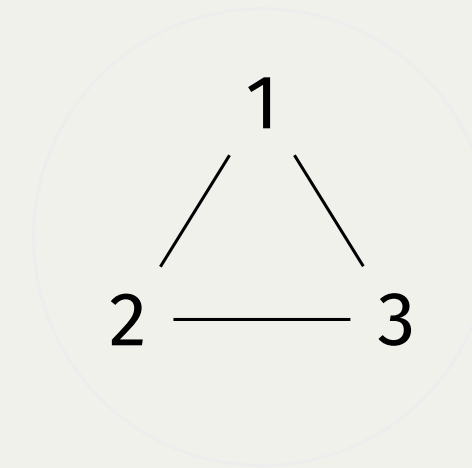
Hardness of subqueries

$$Q_1 = R(1, 2, 3) \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$

$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



linear preprocessing

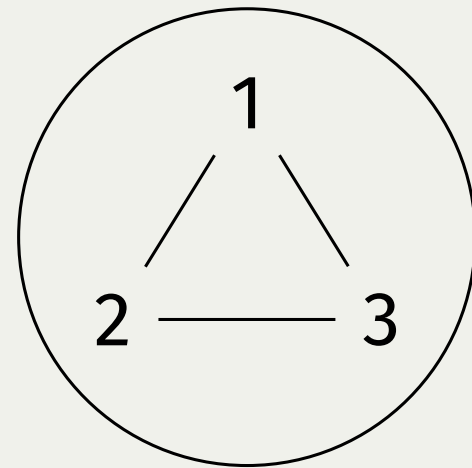


non-linear preprocessing

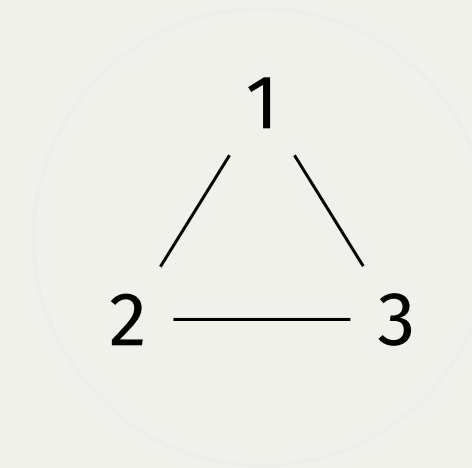
Hardness of subqueries

$$Q_1 = R(1, 2, 3) \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$

$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



linear preprocessing

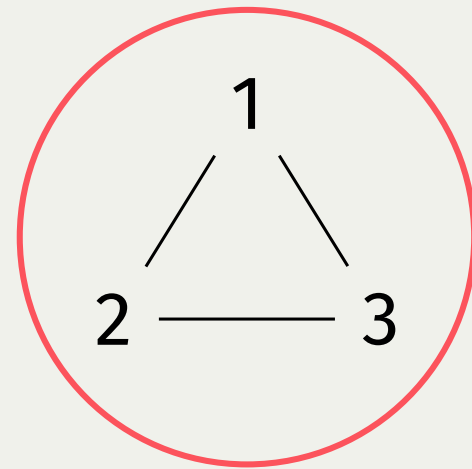


non-linear preprocessing

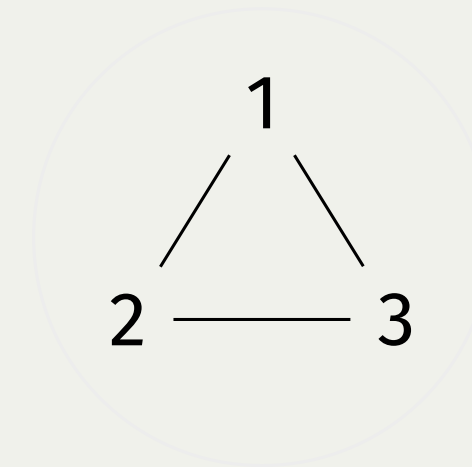
Subqueries may be harder to solve than the query itself!

Subqueries and negative atoms

$$Q_1' = \neg R(1, 2, 3) \\ \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



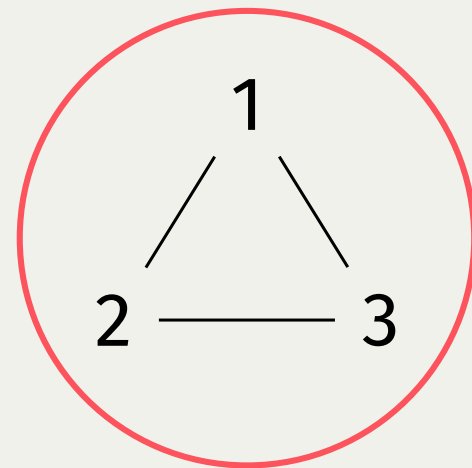
$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



non-linear preprocessing (triangle)

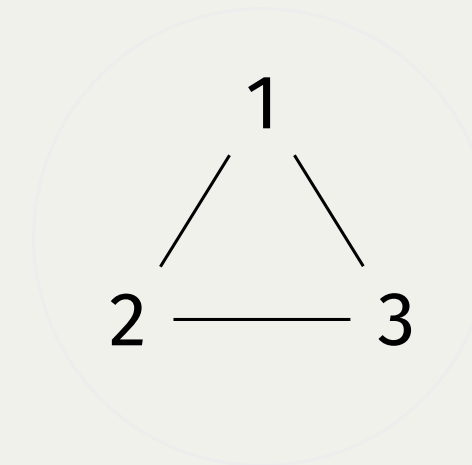
Subqueries and negative atoms

$$Q_1' = \neg R(1, 2, 3) \\ \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



Equivalent to Q_2 if $R = \emptyset$

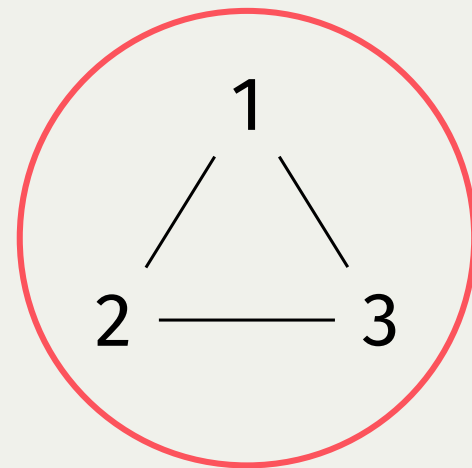
$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



non-linear preprocessing (triangle)

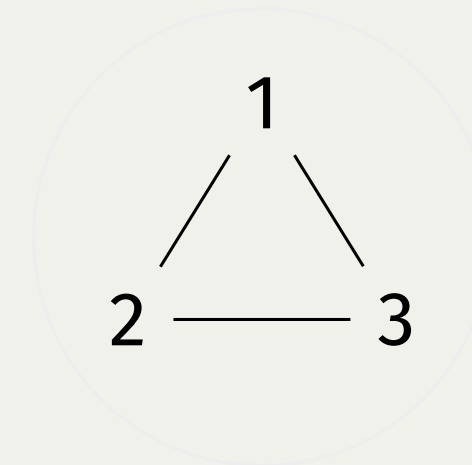
Subqueries and negative atoms

$$Q_1' = \neg R(1, 2, 3) \\ \wedge S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



Equivalent to Q_2 if $R = \emptyset$

$$Q_2 = S(1, 2) \wedge T(2, 3) \wedge U(3, 1)$$



non-linear preprocessing (triangle)

DA for $Q = P \wedge N$ implies DA for $Q = P \wedge N'$ for every $N' \subseteq N$!

Measuring hardness of SJQ

Good candidate for $Q = Q^+ \wedge Q^-$:

Signed-HyperOrder Width

$$\text{show} (Q, \pi) = \max_{Q' \subseteq Q^-} \iota (Q^+ \wedge Q', \pi)$$

For Q a (positive) JQ, and π a variable ordering, we can solve DA with

- Preprocessing $\tilde{O} \left(|\mathbb{D}|^{\iota(Q, \pi)} \right)$
- Access time $O \left(\log |\mathbb{D}| \right)$

Measuring hardness of SJQ

Good candidate for $Q = Q^+ \wedge Q^-$:

Signed-HyperOrder Width

$$show(Q, \pi) = \max_{Q' \subseteq Q^-} \iota(Q^+ \wedge Q', \pi)$$

For Q a **signed JQ**, and π a variable ordering, we can solve DA with

- Preprocessing $\tilde{O}\left(|\mathbb{D}|^{1 + show(Q, \pi)}\right)$
- Access time $O(\log |\mathbb{D}|)$

Our contribution : new island of tractability for Signed JQ!

A word on show

Signed HyperOrder Width (and incidentally, our result) generalizes:

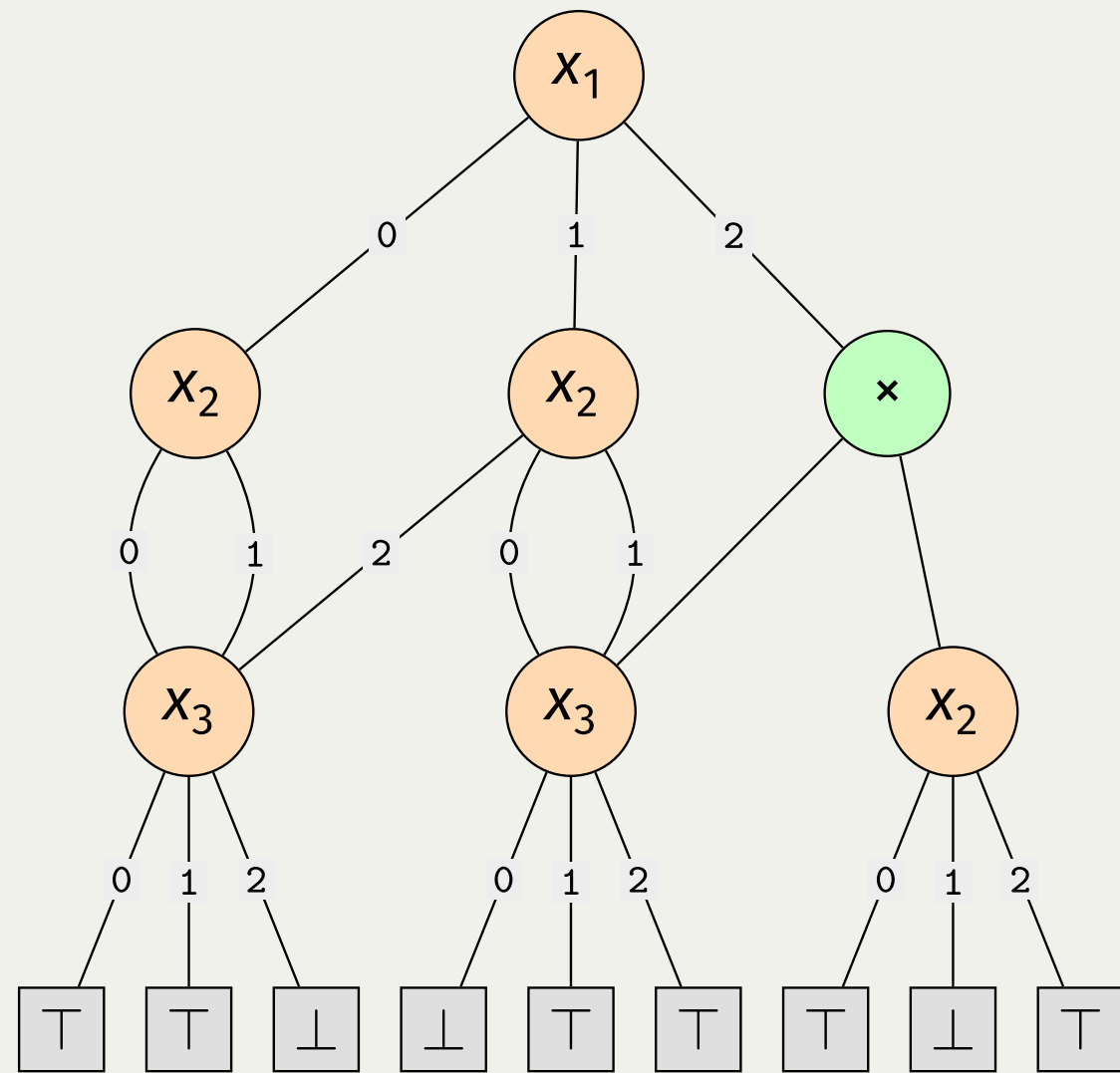
- β -acyclicity (#SAT and #NCQ are already known tractable)
- *signed*-acyclicity (Model Checking for SCQ known to be tractable)
- Nest set width (SAT / Model Checking for NCQ known to be tractable)

Basically, everything that is known to be tractable on SCQ/NCQ.

1. *Understanding model counting for β -acyclic CNF-formulas*, J. Brault-Baron, F. C., S. Mengel
2. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*, J. Brault-Baron
3. *Tractability Beyond β -Acyclicity for Conjunctive Queries with Negation*, M. Lanzinger

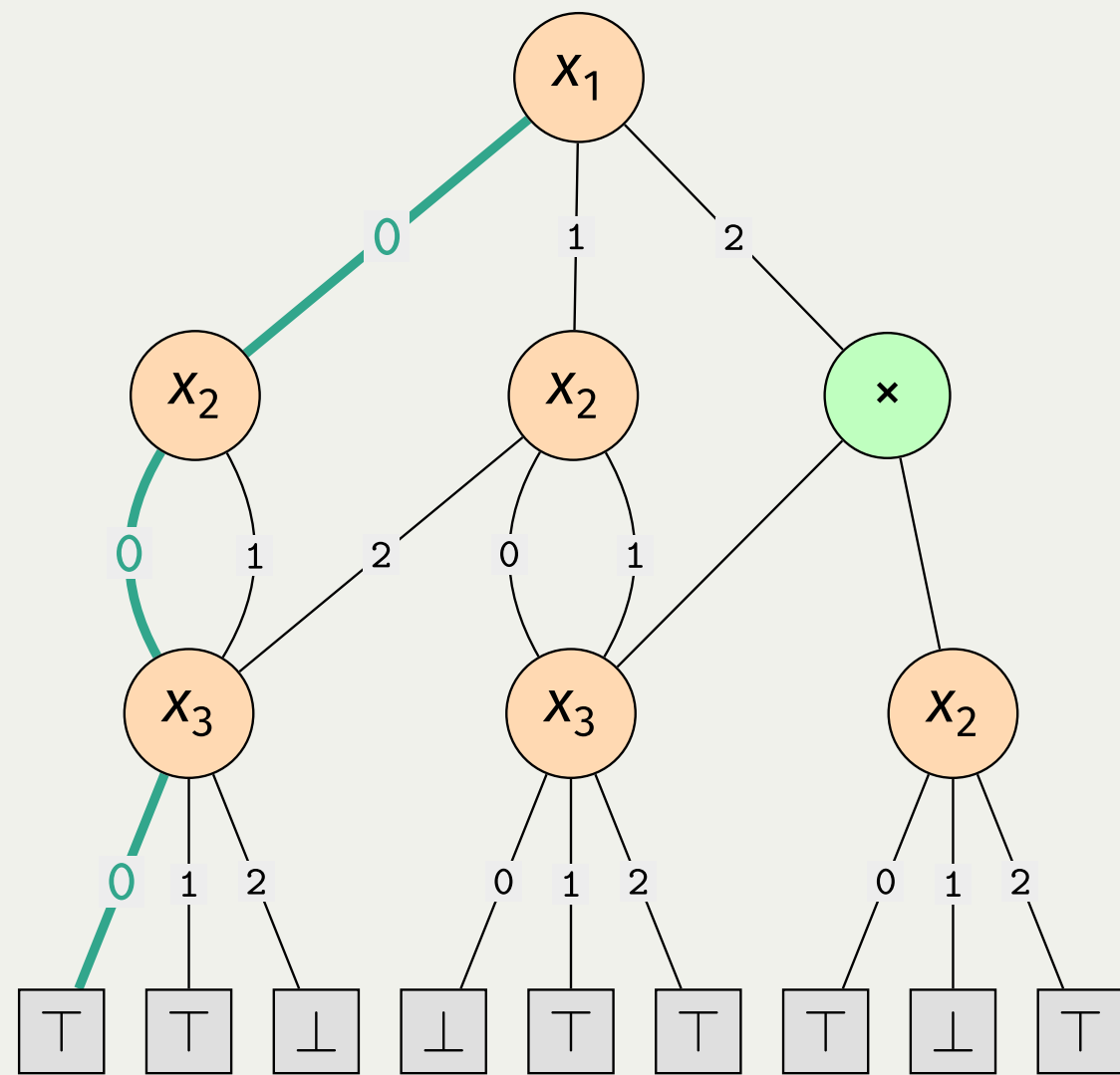
Our algorithm: a circuit approach

Relational Circuits



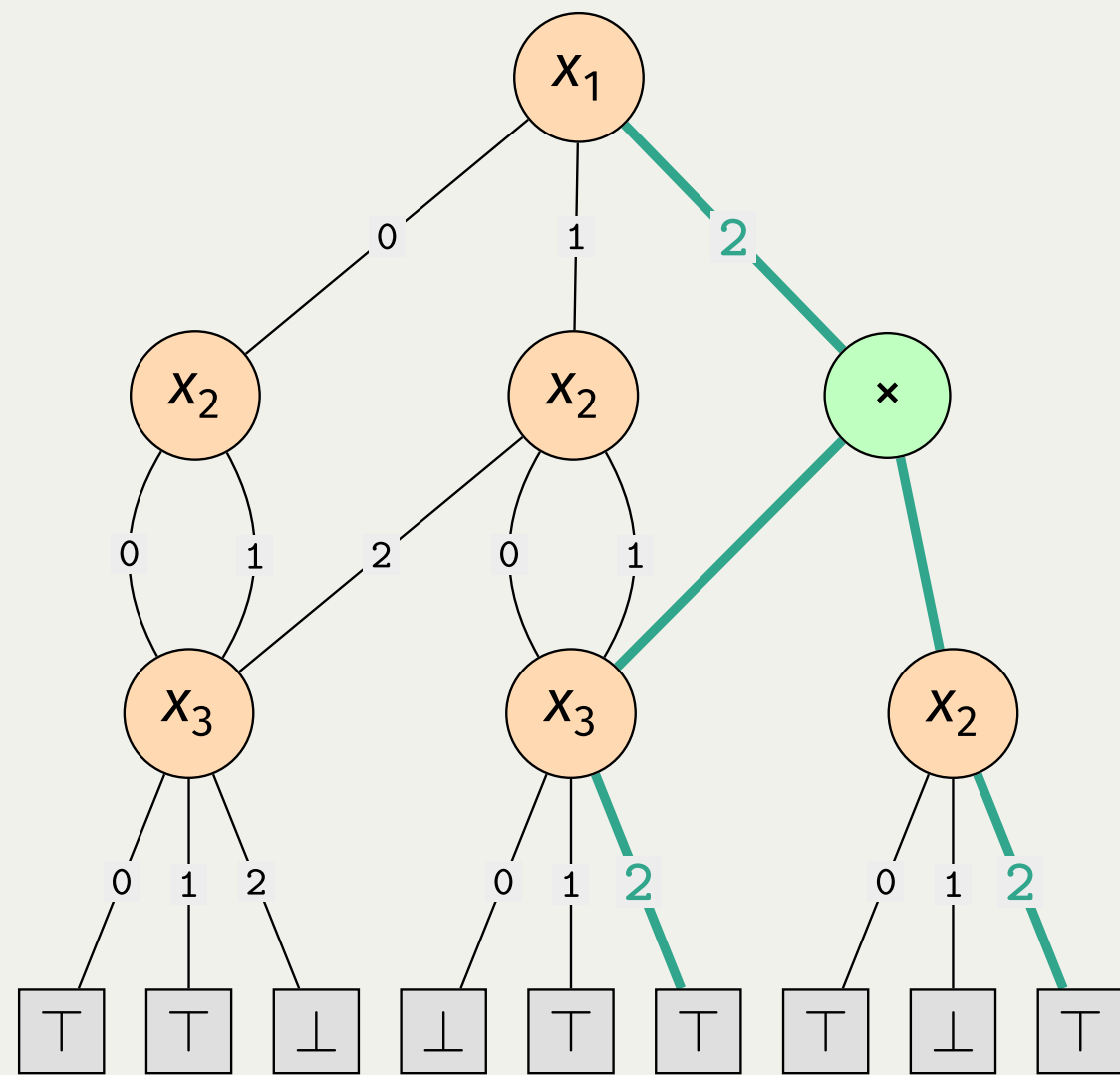
x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	1
1	0	2
1	1	1
1	1	2
1	2	0
1	2	1
2	0	1
2	0	2
2	2	1
2	2	2

Relational Circuits



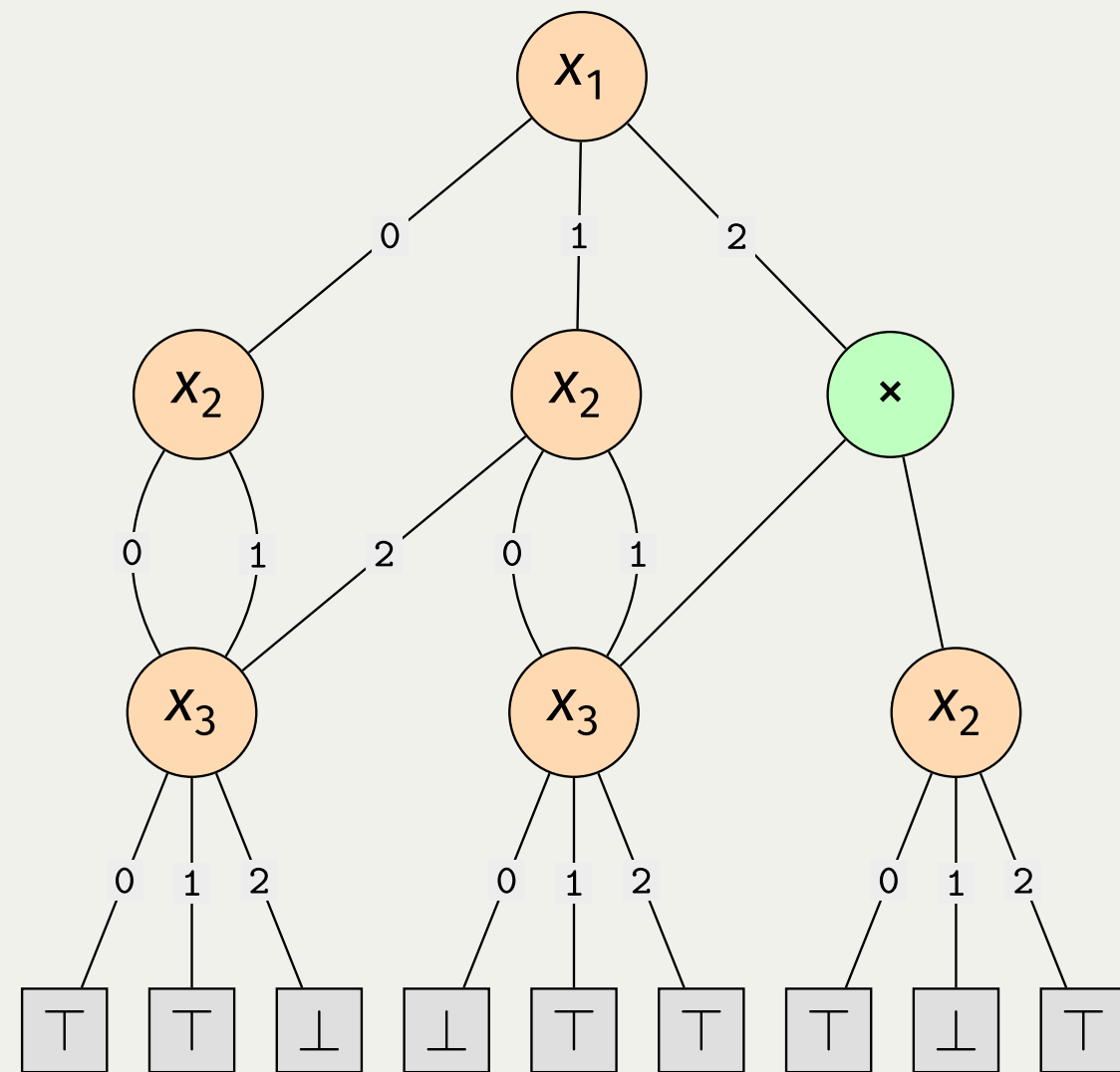
x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	1
1	0	2
1	1	1
1	1	2
1	2	0
1	2	1
2	0	1
2	0	2
2	2	1
2	2	2

Relational Circuits



x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	1
1	0	2
1	1	1
1	1	2
1	2	0
1	2	1
2	0	1
2	0	2
2	2	1
2	2	2

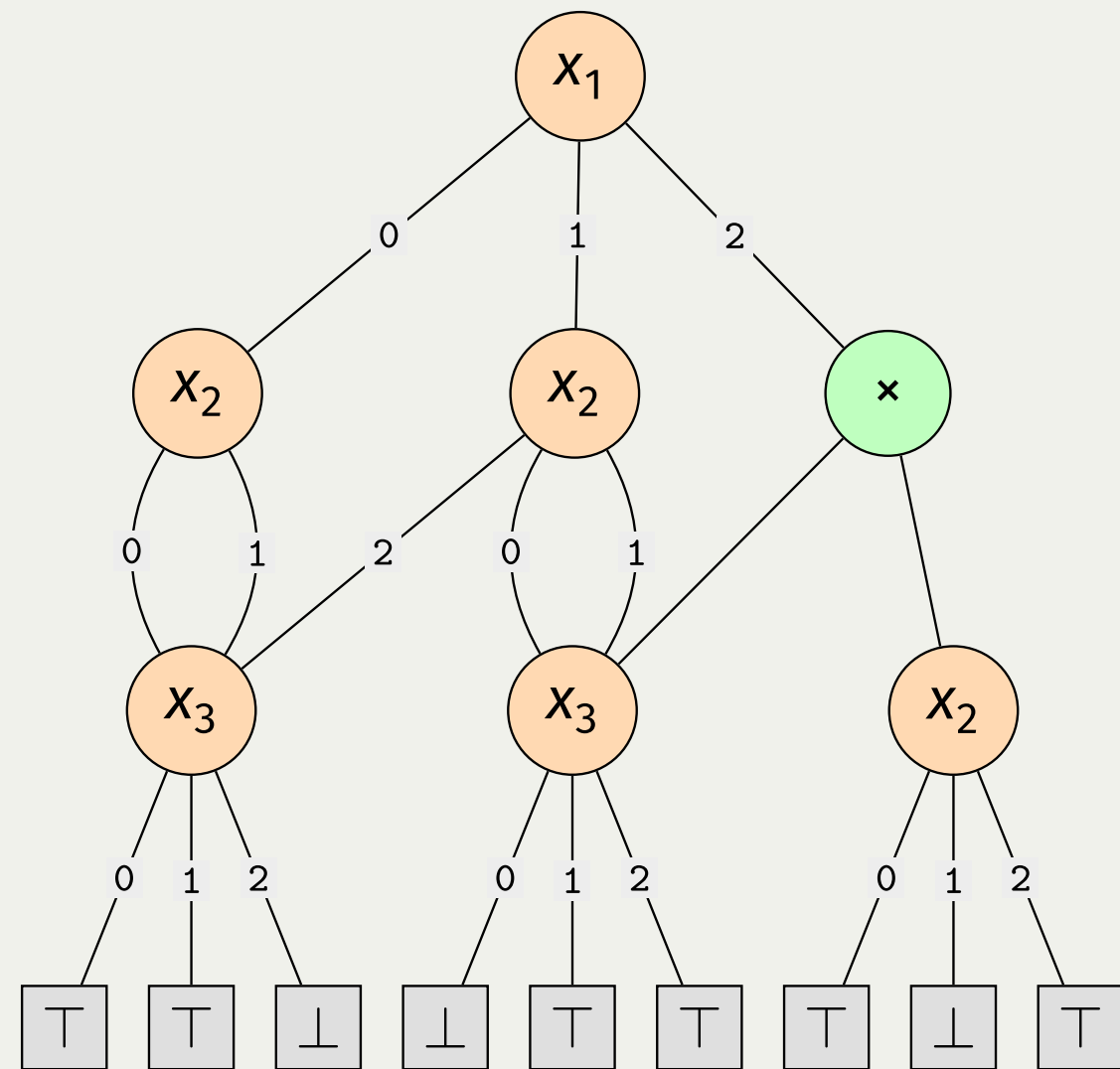
Ordered Relational Circuits



Factorized representation of relation $R \subseteq D^X$:

- **Inputs** gates : \top & \perp
- **Decision** gates
- **Cartesian products**: \times -gates

Ordered Relational Circuits

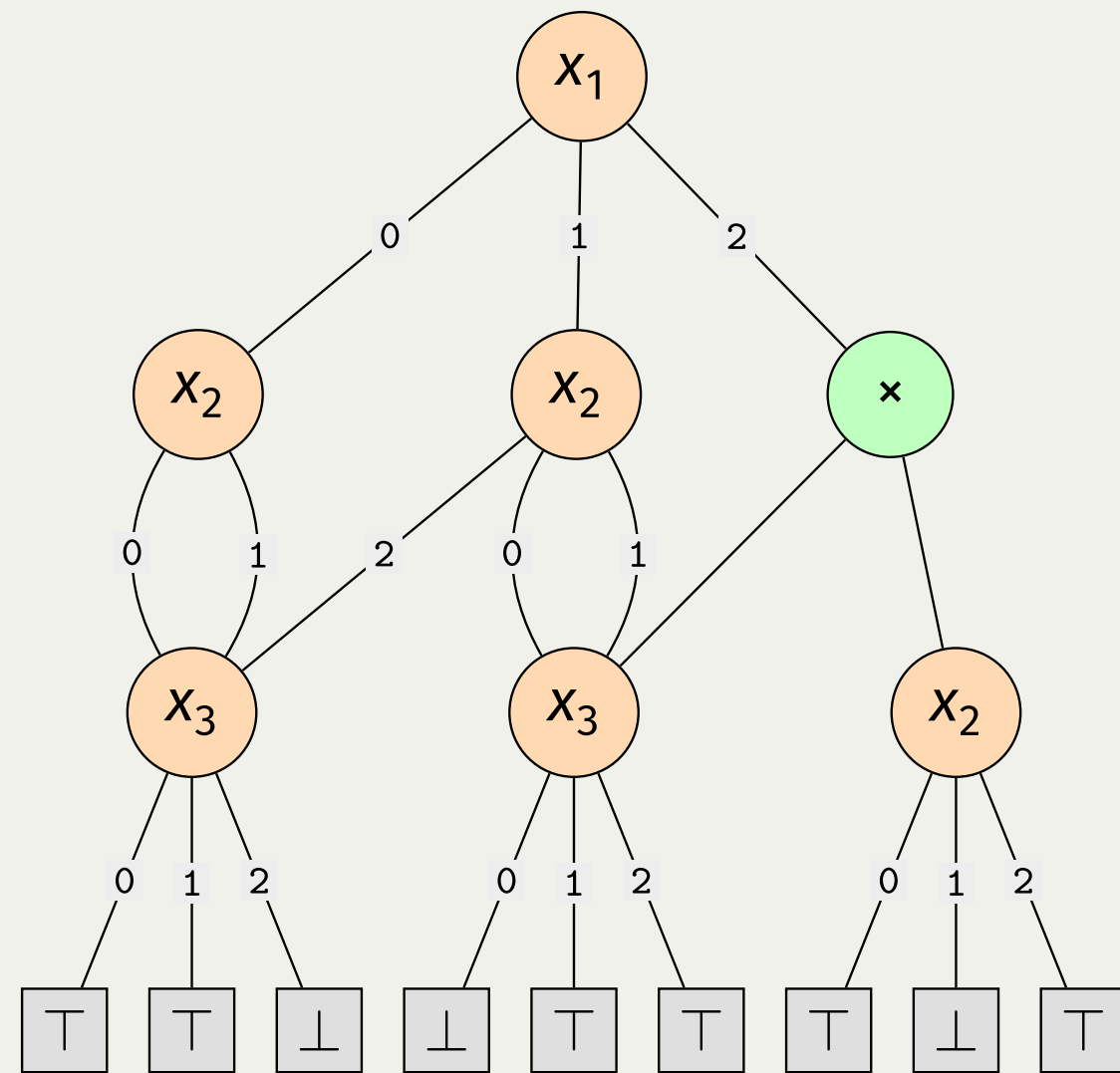


Factorized representation of relation $R \subseteq D^X$:

- **Inputs** gates : \top & \perp
- **Decision** gates
- **Cartesian products**: \times -gates

Ordered: decision gates below x_i only mention x_j with $j > i$.

Direct Access on Relational Circuits



For C on domain D , variables x_1, \dots, x_n , DA possible :

- **Preprocessing:** $O(|C| \log |D|)$
- **Access time:** $O(n \log |D|)$

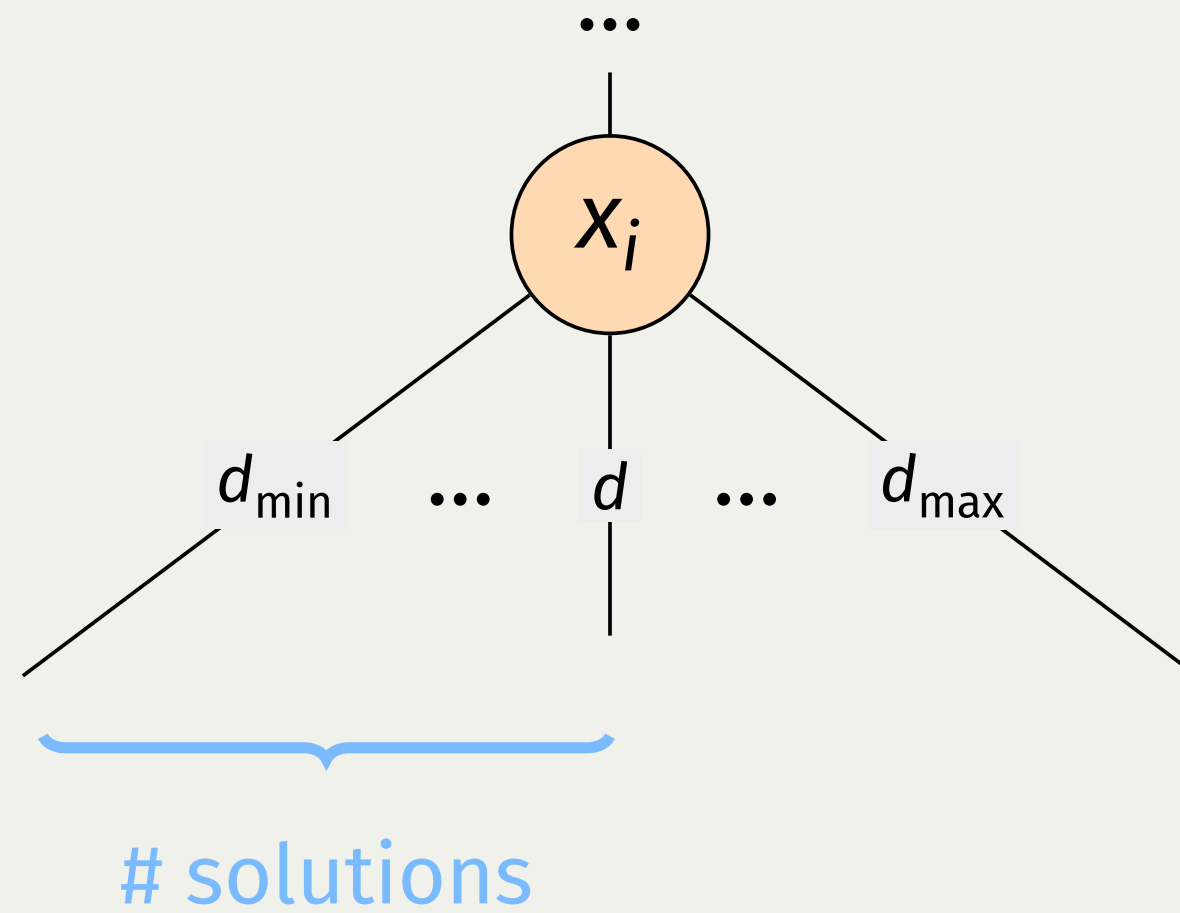
Preprocessing

Preprocessing

Idea : for each gate v over x_i and for each domain value d

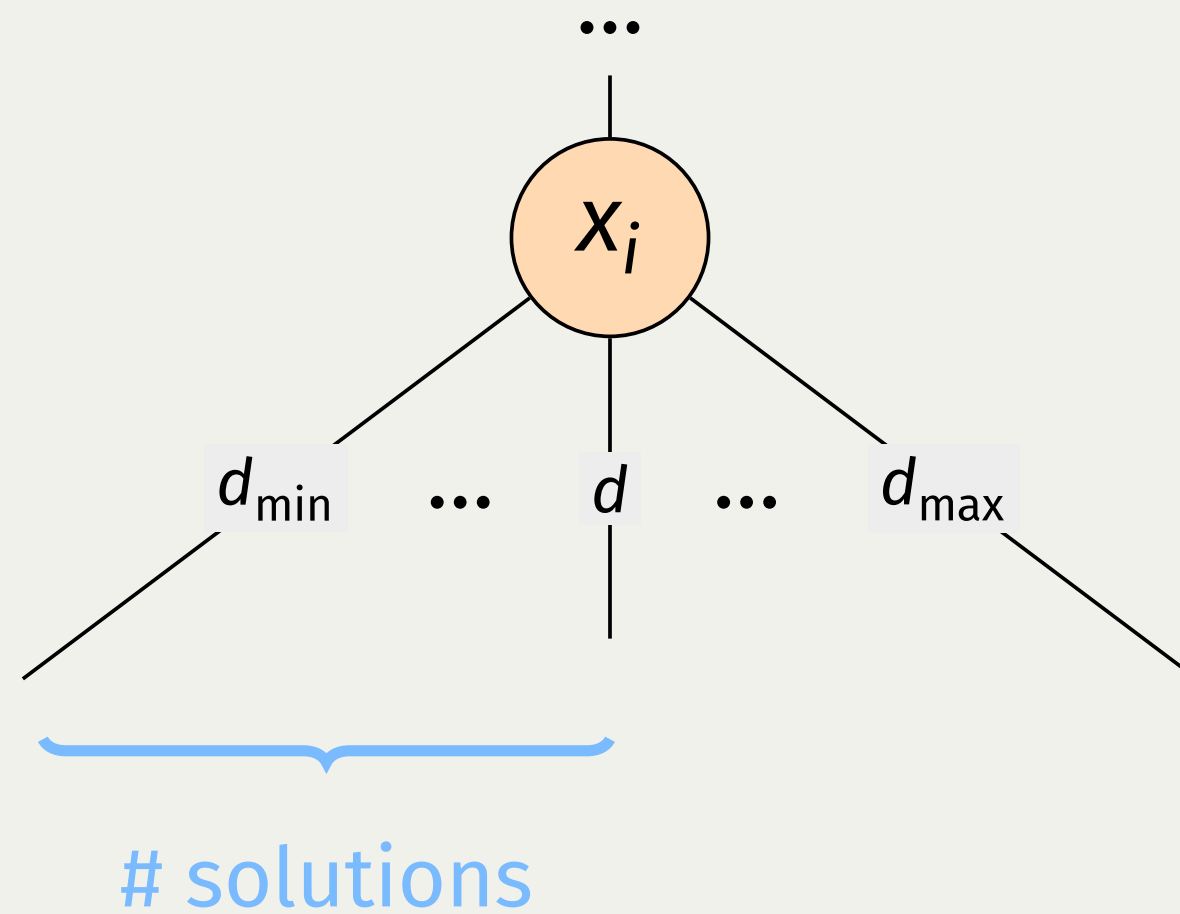
Preprocessing

Idea : for each gate v over x_i and for each domain value d



Preprocessing

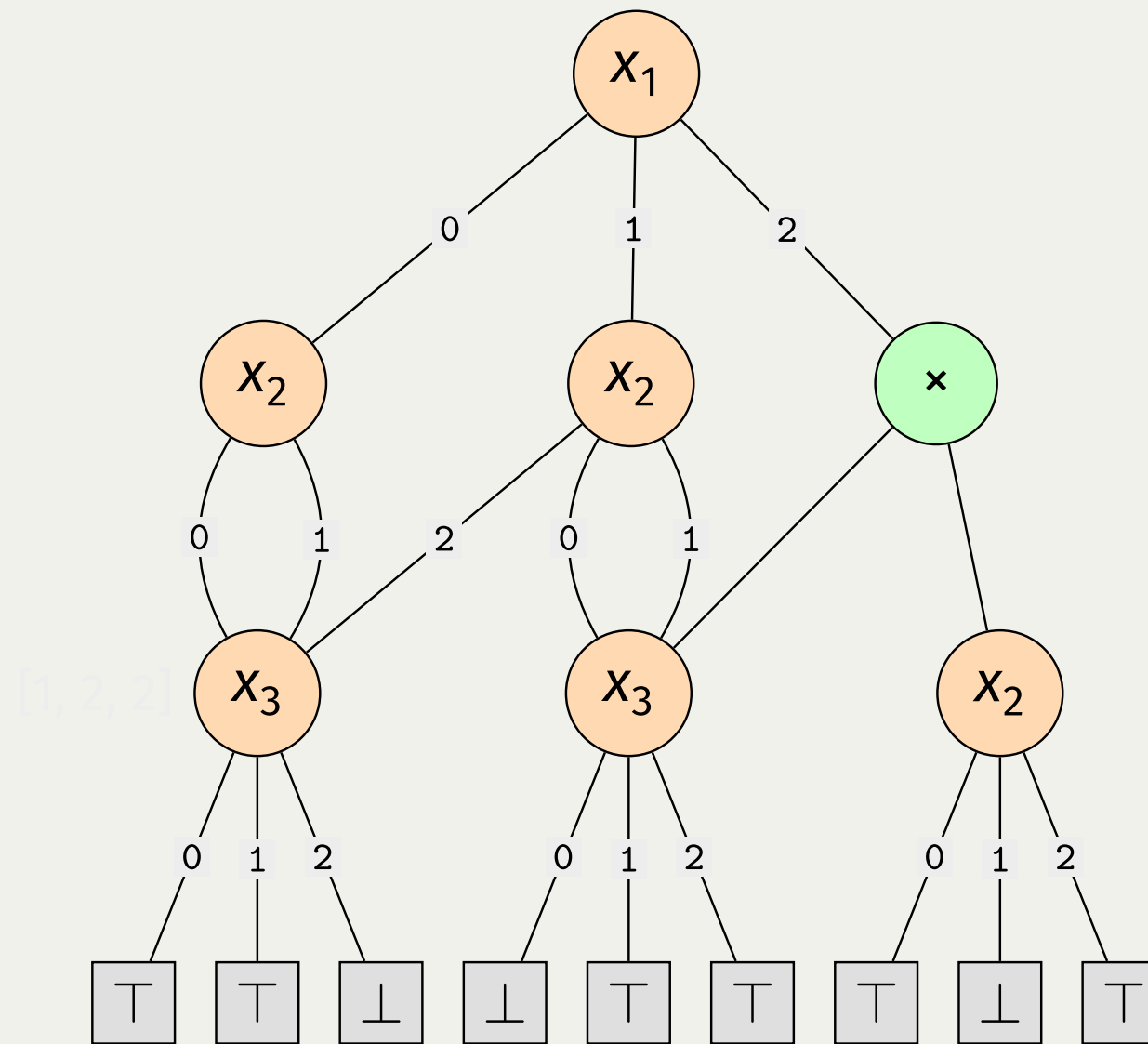
Idea : for each gate v over x_i and for each domain value d



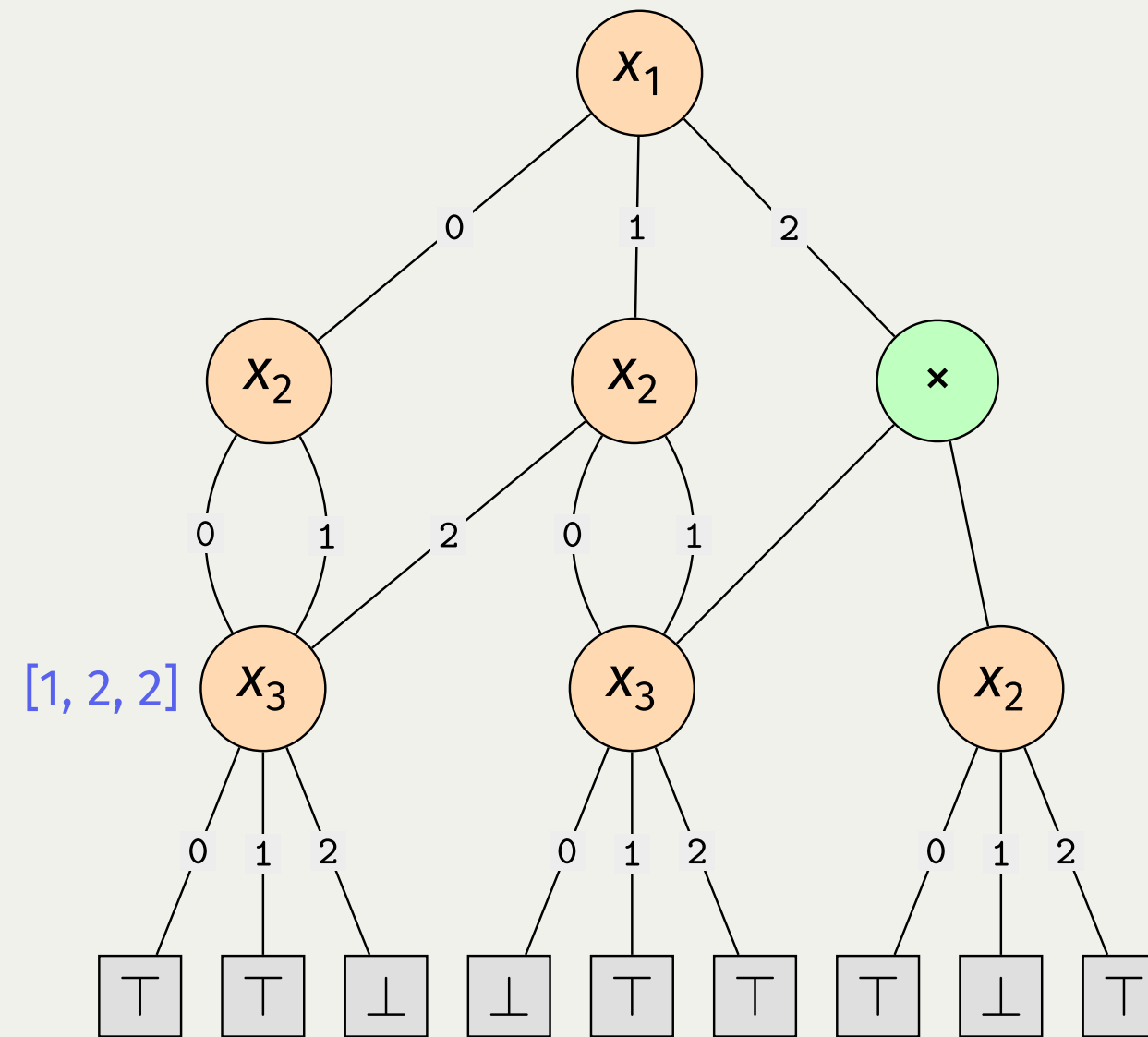
compute the size of the relation where x_i is set to a value $d' \leq d$

Preprocessing

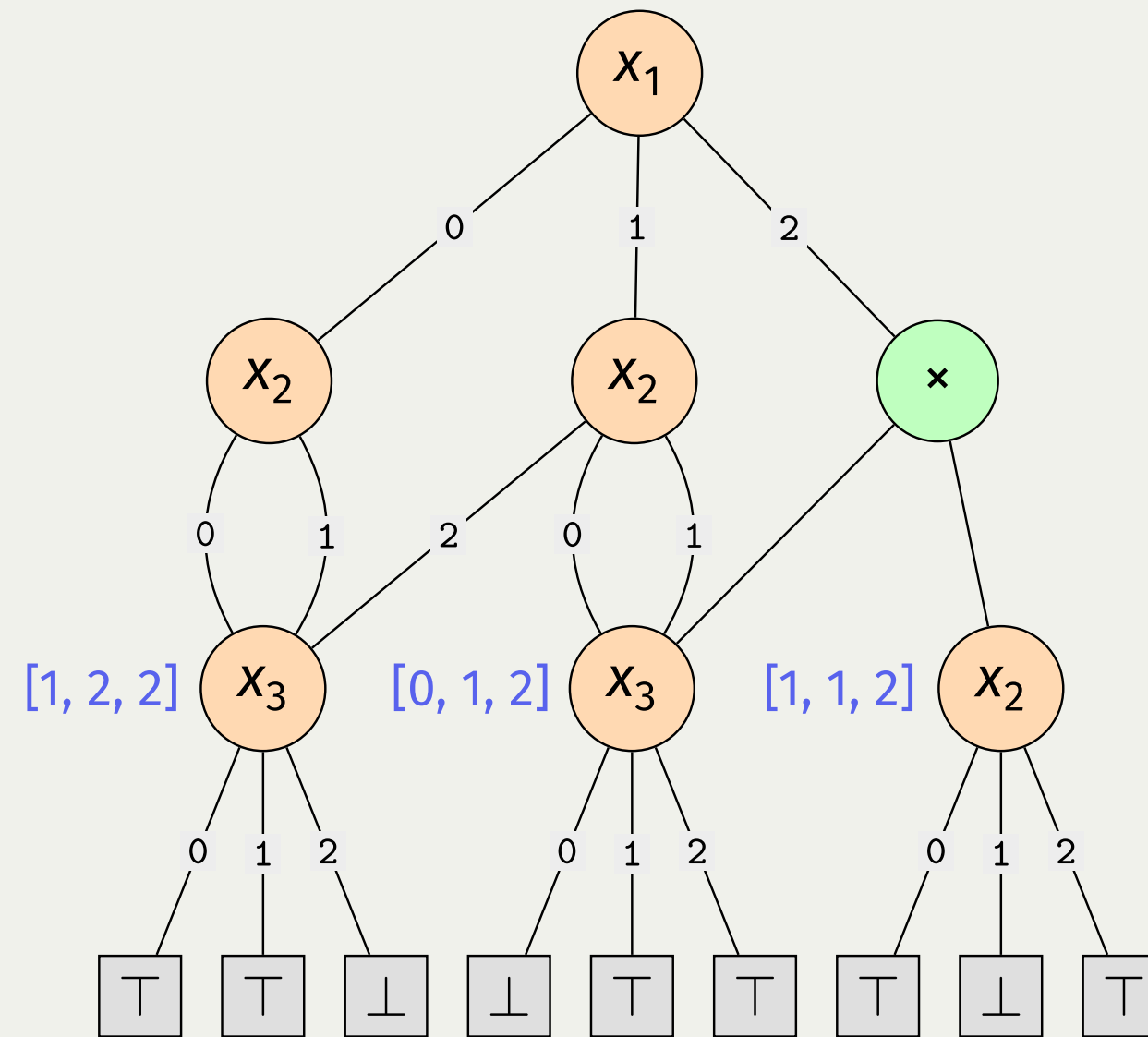
Preprocessing



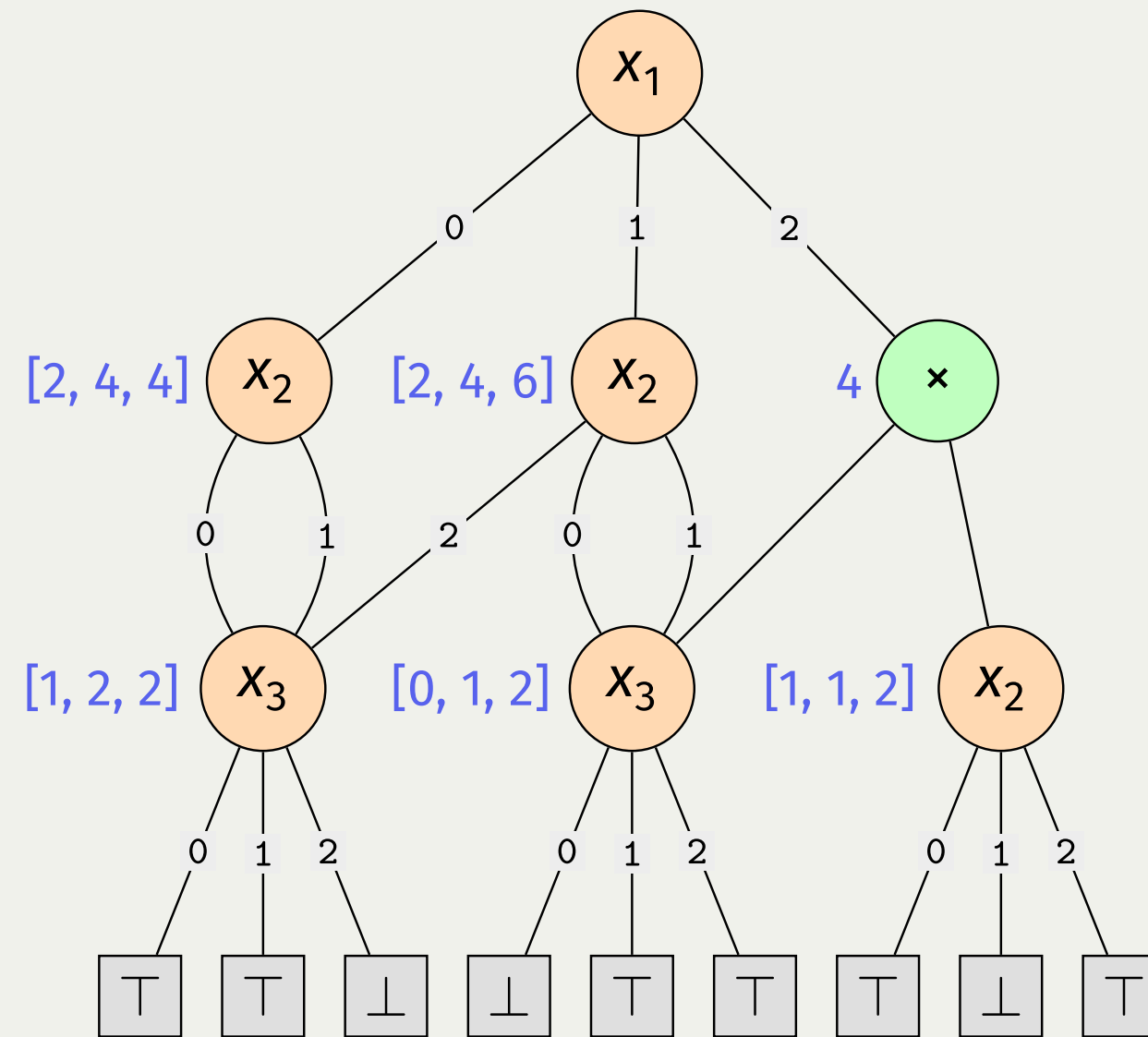
Preprocessing



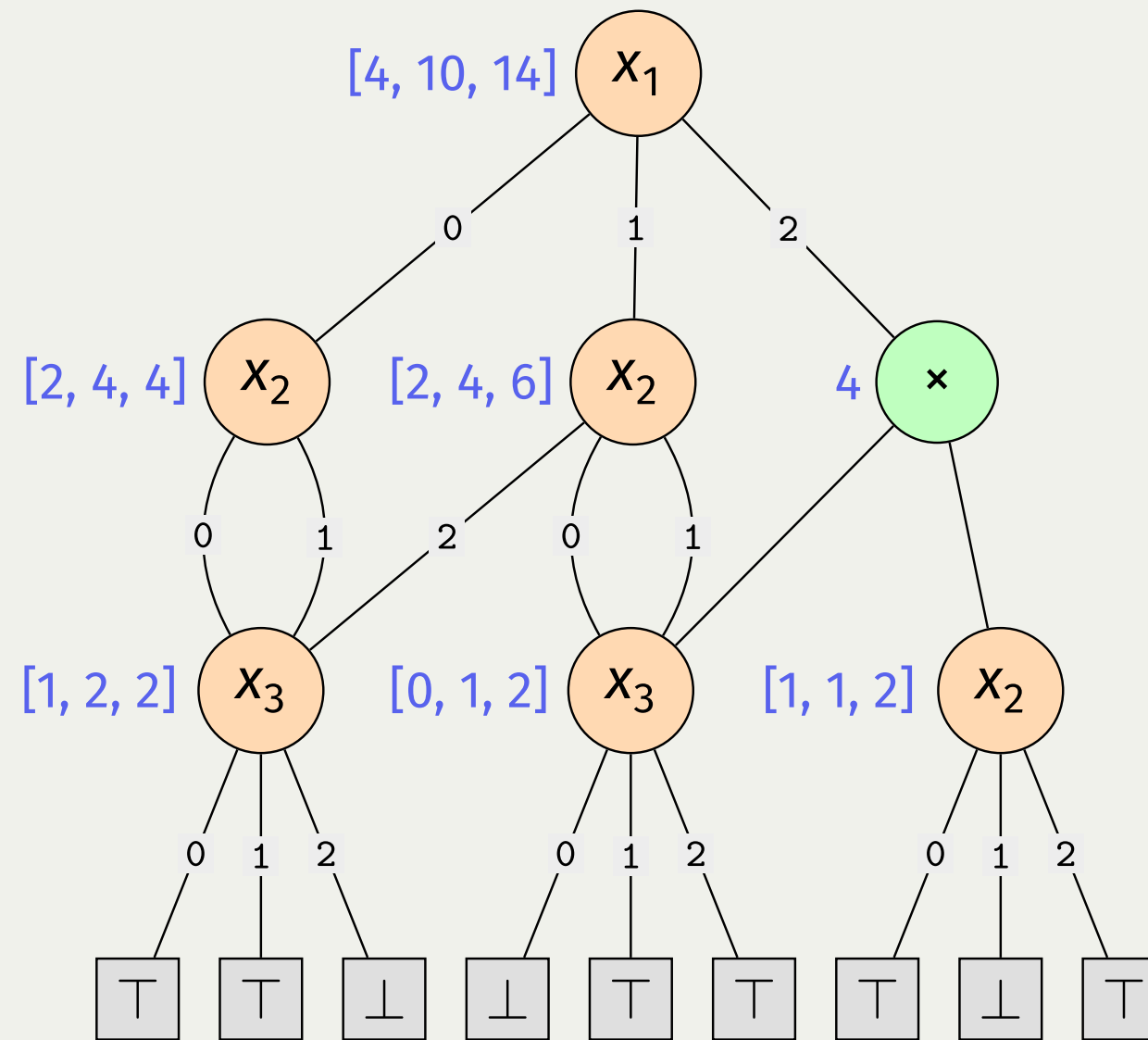
Preprocessing



Preprocessing



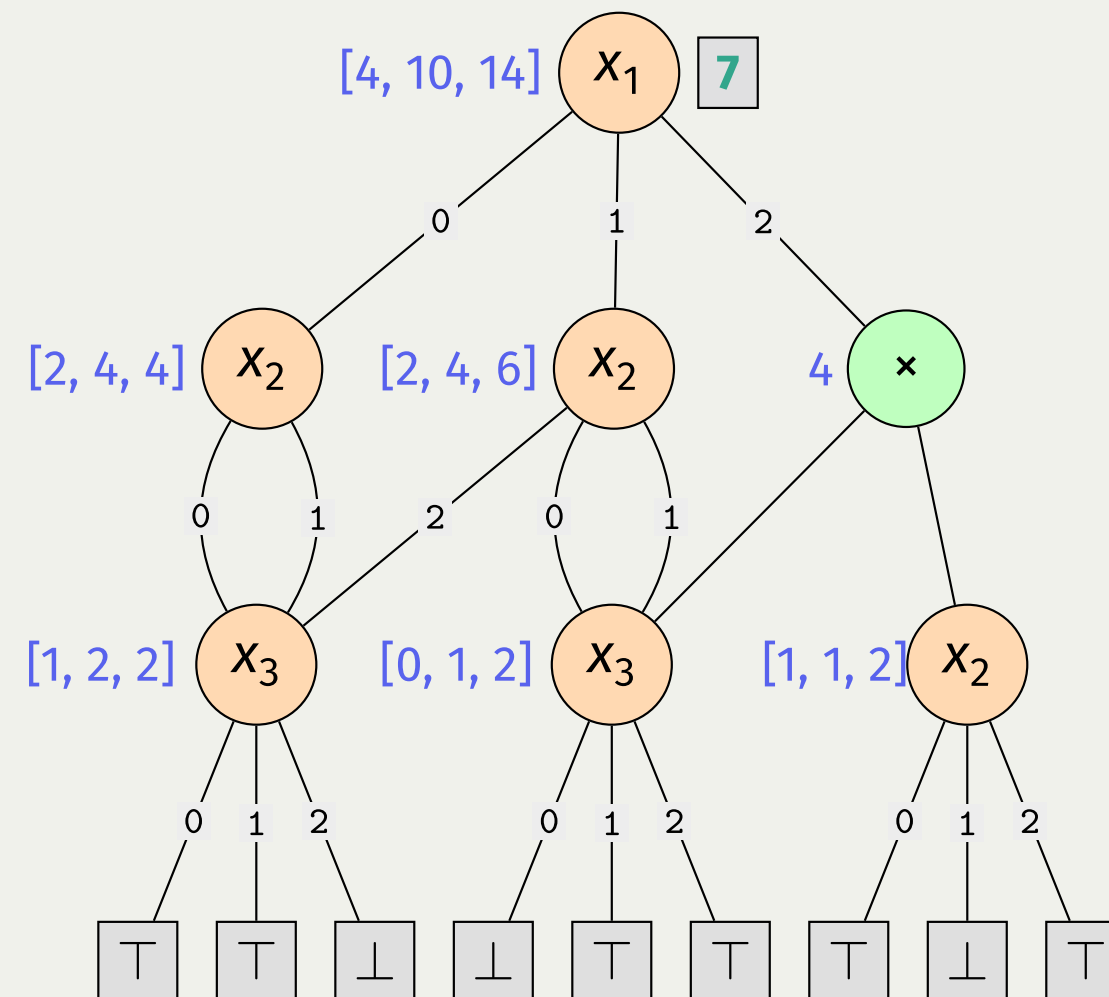
Preprocessing



Direct Access 7th solution

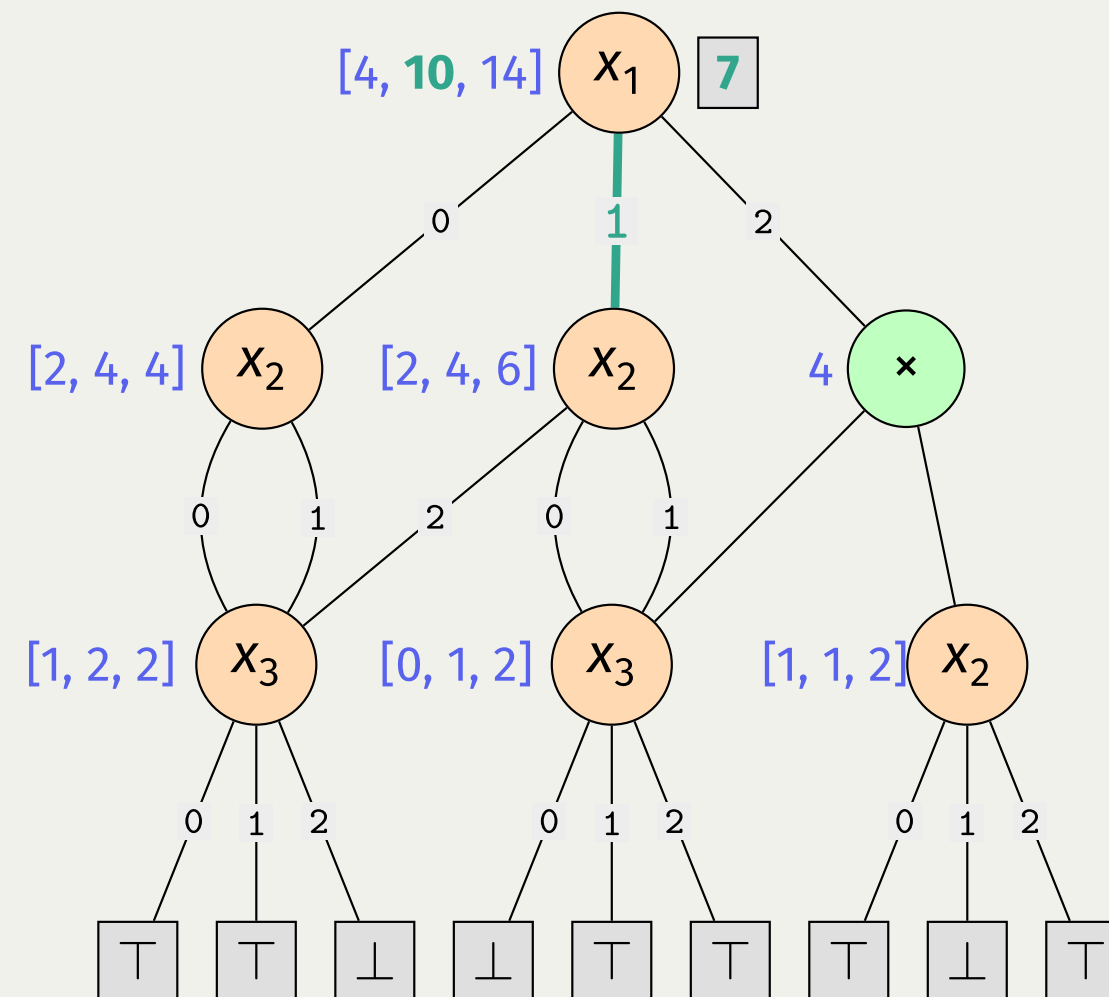
Compute the 7th solution

Direct Access 7th solution



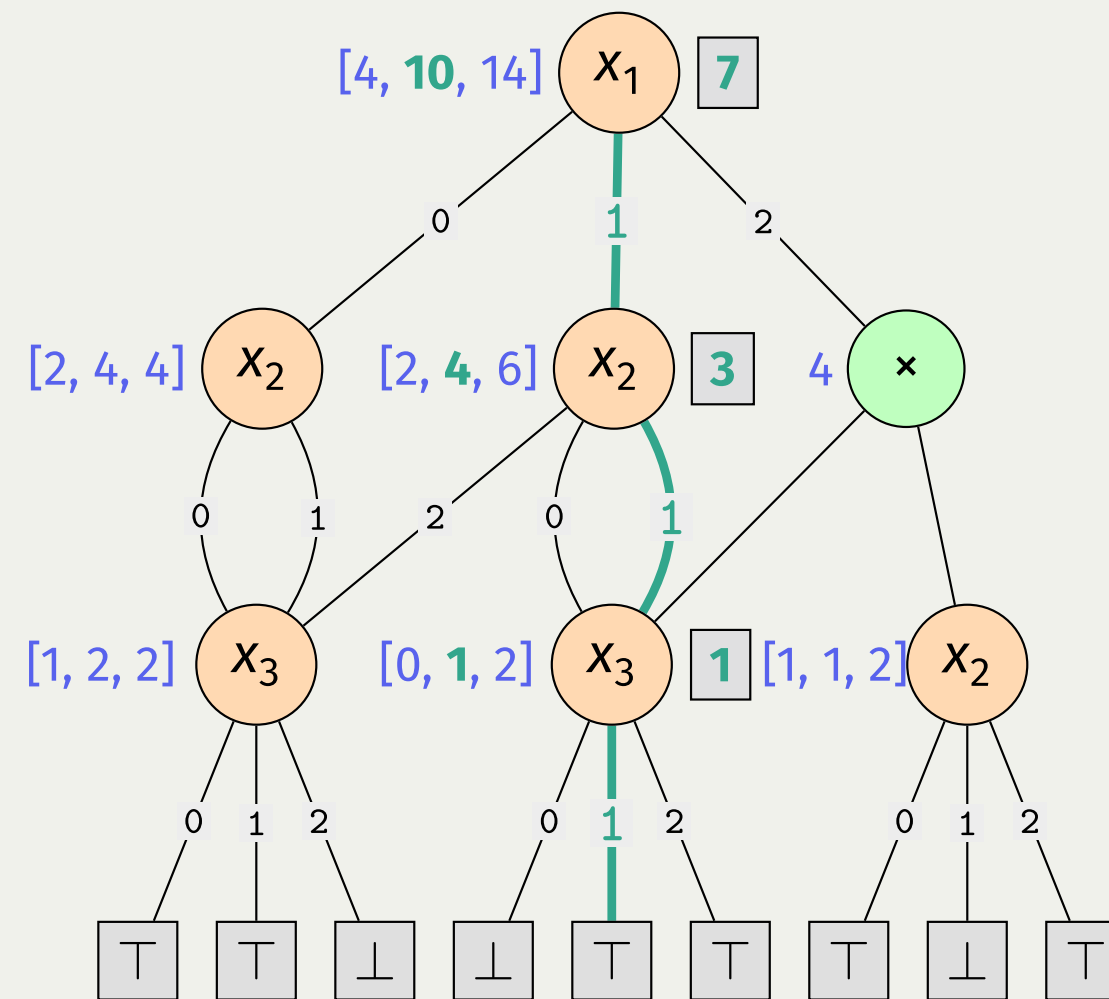
Compute the 7th solution

Direct Access 7th solution



Compute the 7th solution

Direct Access 7th solution

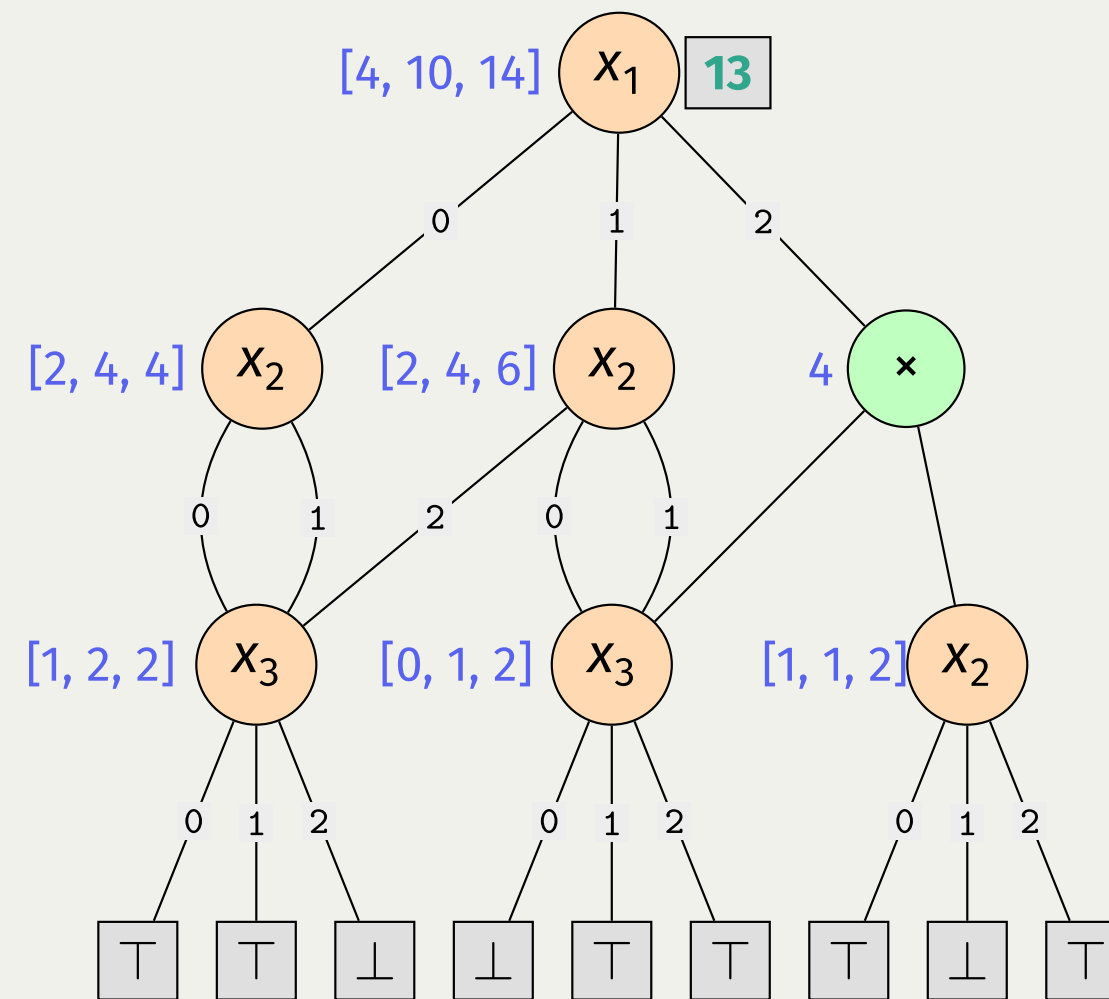


Compute the 7th solution \rightarrow 111

Direct Access the 13th solution

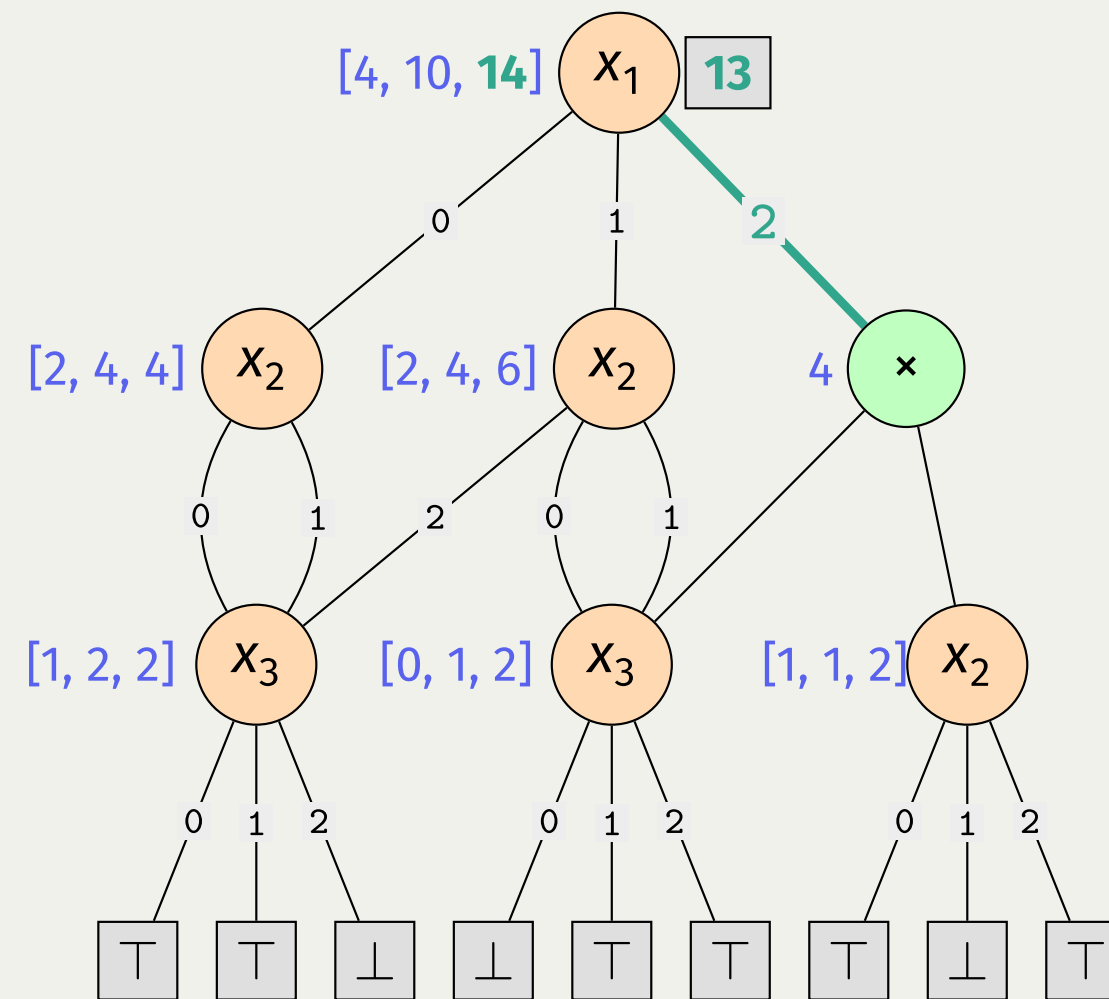
Compute the 13th solution

Direct Access the 13th solution



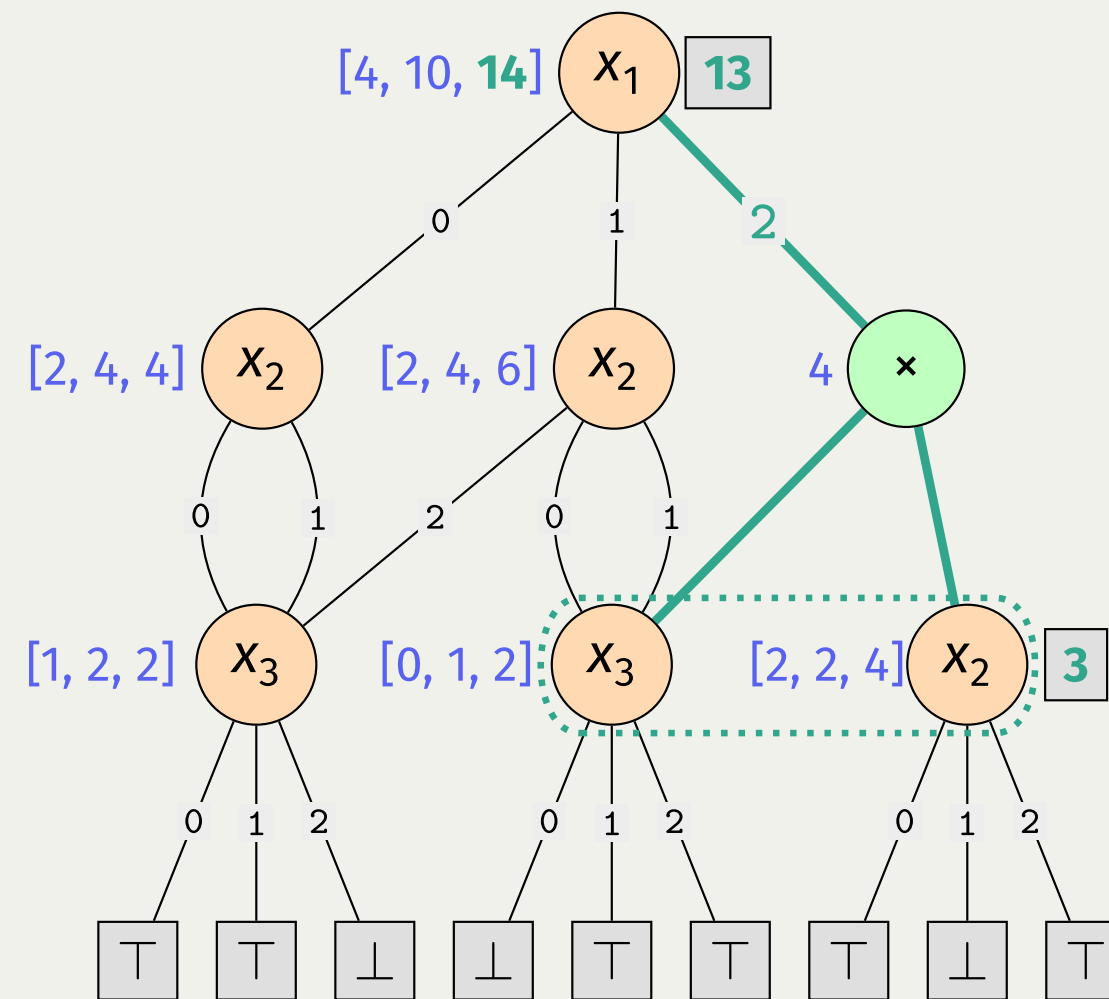
Compute the 13th solution

Direct Access the 13th solution



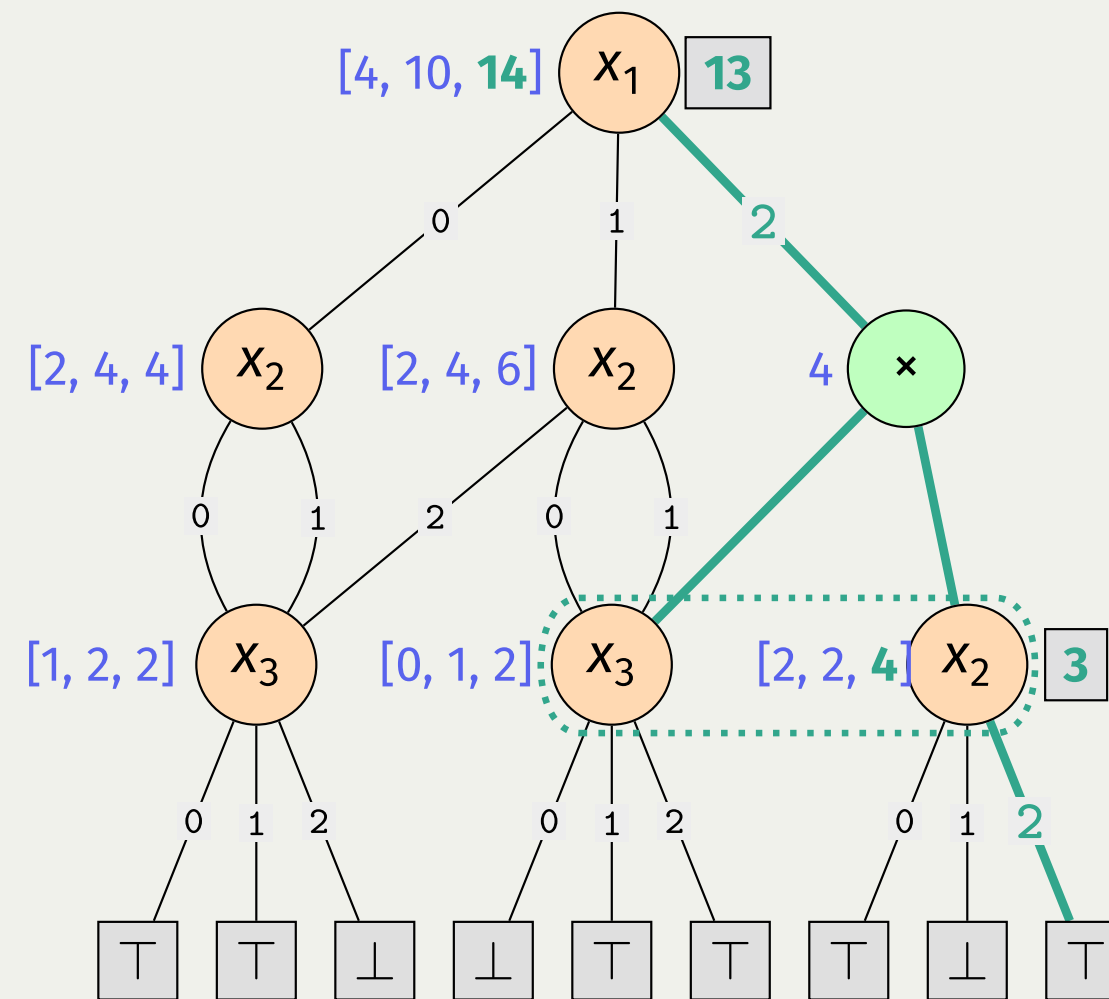
Compute the 13th solution

Direct Access the 13th solution



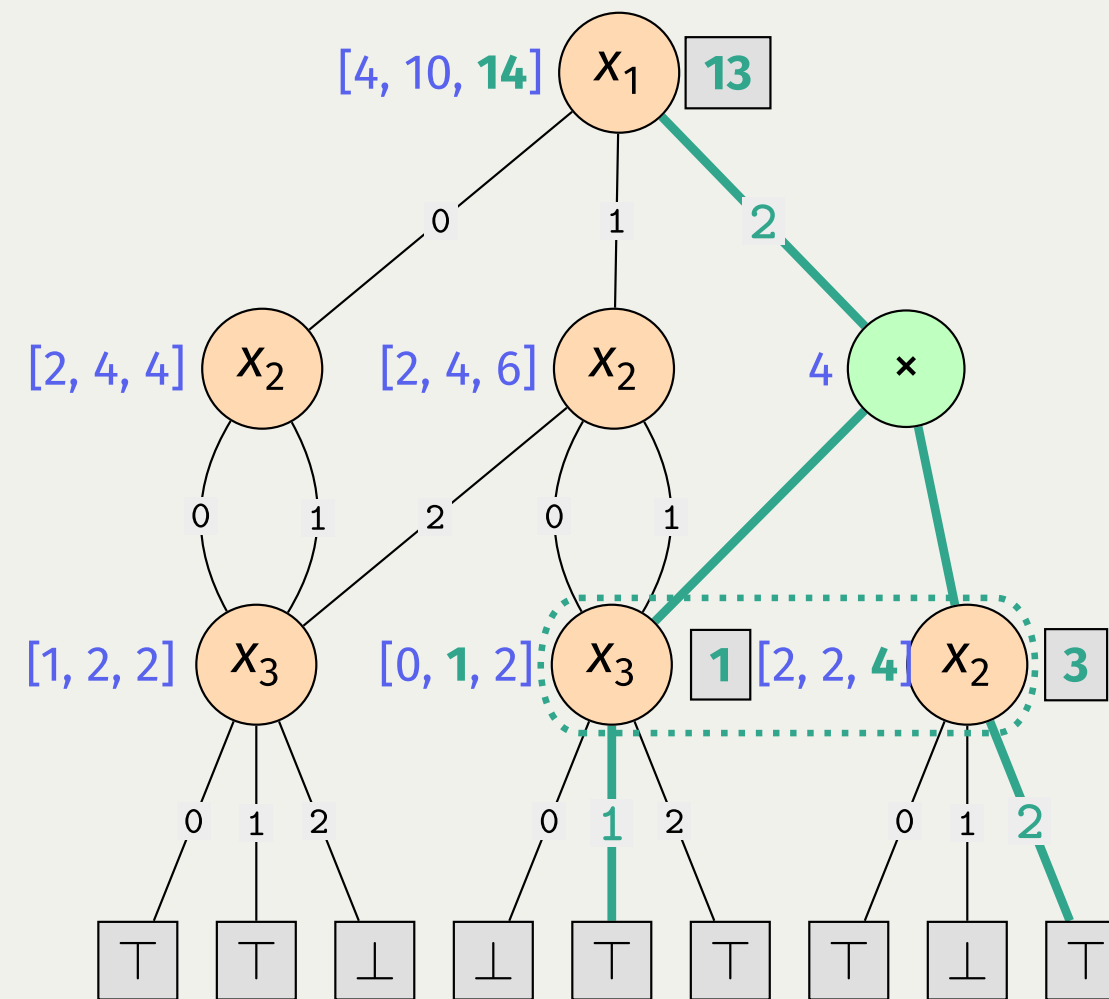
Compute the 13th solution

Direct Access the 13th solution



Compute the 13th solution

Direct Access the 13th solution



Compute the 13th solution \rightarrow 221

Solving DA for SCQ

SCQ $Q (x_1, \dots, x_n)$, $\pi = (x_1, \dots, x_n)$.

Preprocessing:

1. Construct π -ordered circuit C of size $\tilde{O} \left(|\mathbb{D}|^{1 + \text{show}(Q, \pi)} \text{poly}(Q) \right)$
2. Preprocess C in time $O (|C| \log |\mathbb{D}|)$.

Direct Access :

1. Directly on C
2. in time $O (n \log |D|)$!

Solving DA for SCQ

SCQ $Q (x_1, \dots, x_n)$, $\pi = (x_1, \dots, x_n)$.

Preprocessing:

1. Construct π -ordered circuit C of size $\tilde{O} \left(|\mathbb{D}|^{1 + \text{show}(Q, \pi)} \text{poly}(Q) \right)$
2. Preprocess C in time $O (|C| \log |\mathbb{D}|)$.

Direct Access :

1. Directly on C
2. in time $O (n \log |D|)$!

Q, n considered constant here!

Solving DA for SCQ

SCQ $Q (x_1, \dots, x_n)$, $\pi = (x_1, \dots, x_n)$.

Preprocessing: $\tilde{O} \left(|\mathbb{D}|^{1 + show(Q, \pi)} \right)$

1. Construct π -ordered circuit C of size $\tilde{O} \left(|\mathbb{D}|^{1 + show(Q, \pi)} poly(Q) \right)$
2. Preprocess C in time $O (|C| \log |\mathbb{D}|)$.

Direct Access :

1. Directly on C
2. in time $O (n \log |D|)$!

Q, n considered constant here!

Solving DA for SCQ

SCQ $Q (x_1, \dots, x_n)$, $\pi = (x_1, \dots, x_n)$.

Preprocessing: $\tilde{O} \left(|\mathbb{D}|^{1 + show(Q, \pi)} \right)$

1. Construct π -ordered circuit C of size $\tilde{O} \left(|\mathbb{D}|^{1 + show(Q, \pi)} poly(Q) \right)$
2. Preprocess C in time $O (|C| \log |\mathbb{D}|)$.

Direct Access : $O (\log |\mathbb{D}|)$

1. Directly on C
2. in time $O (n \log |D|)$!

Q, n considered constant here!

DPLL: building circuits

Compilation based on a variation of DPLL :

1. $Q(\mathbb{D}) = \bigsqcup_{d \in D} [x_1 = d] \times Q[x_1 = d](\mathbb{D})$
2. $Q(\mathbb{D}) = Q_1(\mathbb{D}) \times Q_2(\mathbb{D})$ if $Q = Q_1 \wedge Q_2$ with $\text{var}(Q_1) \cap \text{var}(Q_2) = \emptyset$
3. Top down induction + caching



<https://florent.capelli.me/cytoscape/dpll.html>

Going further

Other usage of circuits

1. Extension to \exists SJQ:

- Last variable in C can be **existentially projected** without increase in circuit size
- Give DA for $\exists x_k, \dots, x_n Q (x_1, \dots, x_n)$.

2. Semi-ring Aggregation

- $w: X \times D \rightarrow (\mathbb{K}, \oplus, \otimes)$
- Compute $\bigoplus_{\tau \in Q(D)} \bigotimes_{x \in X} w (x, \tau (x))$

Work in progress

1. Improve preprocessing
 $\tilde{O}\left(|\mathbb{D}|^{show(Q, \pi) + 1}\right)$

Work in progress

1. Improve preprocessing
 $\tilde{O}\left(|\mathbb{D}|^{show(Q, \pi)}\right)$

Work in progress

1. Improve preprocessing

$$\tilde{O}\left(|\mathbb{D}|^{show(Q, \pi)}\right)$$

doable with a few tweaks in DPLL, joint work with S. Salvati.

Work in progress

1. Improve preprocessing

$$\tilde{O}\left(|\mathbb{D}|^{show(Q, \pi)}\right)$$

doable with a few tweaks in DPLL, joint work with S. Salvati.

2. Lower bounds: preprocessing in $|\mathbb{D}|^{show(Q, \pi)}$

unavoidable under Zero-clique conjecture (joint work with N. Carmeli).

Work in progress

1. Improve preprocessing

$$\tilde{O}\left(|\mathbb{D}|^{show(Q, \pi)}\right)$$

doable with a few tweaks in DPLL, joint work with S. Salvati.

2. Lower bounds: preprocessing in $|\mathbb{D}|^{show(Q, \pi)}$

unavoidable under Zero-clique conjecture (joint work with N. Carmeli).

3. Aggregation

$Q(x_1, \dots, x_k, F(x_{k+1}, \dots, x_n))$, generalizing work by I. Eldar, N. Carmeli, B. Kimelfeld.

