Probabilistic Query Evaluation on Bounded-Treewidth Instances

SIGMOD/PODS PH.D. SYMPOSIUM

JUNE 26, 2016, SAN FRANCISCO

Mikaël Monet
Supervised by Pierre Senellart

Context

- Boolean queries (yes/no) on relational instances

Context

- Boolean queries (yes/no) on relational instances

- We want the answer to contain more information than just « yes/no »:
  - Add uncertainty
  - Obtain provenance information

Context

- Boolean queries (yes/no) on relational instances

- We want the answer to contain more information than just « yes/no »:
  - Add uncertainty
  - Obtain provenance information

- We need restrictions for all of this to be tractable

A probabilistic database

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f

A probabilistic database

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

R		
a	d	0.2
f	e	0.7
d	a	0.13
b	e	0.81

S		
d	e	0.005
f	c	0.9
a	e	0.7
c	e	0.23

Q		
c	e	f

Q			
c	e	f	0.66

Monet Mikaël

A probabilistic database

Monet Mikaël

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f

→

R		
a	d	0.2
f	e	0.7
d	a	0.13
b	e	0.81

S		
d	e	0.005
f	c	0.9
a	e	0.7
c	e	0.23

Q			
c	e	f	0.66

TID model

Probability of a possible world

R		
a	d	0.2
f	e	0.7
d	a	0.13
b	e	0.81

S		
d	e	0.005
f	c	0.9
a	e	0.7
c	e	0.23

A possible world I →

R	
f	e
d	a

S	
c	e

Q			
c	e	f	0.66

Q		
c	e	f

Monet Mikaël

Probability of a possible world

R		
a	d	0.2
f	e	0.7
d	a	0.13
b	e	0.81

S		
d	e	0.005
f	c	0.9
a	e	0.7
c	e	0.23

Q			
c	e	f	0.66

A possible world I →

R	
f	e
d	a

S	
c	e

Q		
c	e	f

Monet Mikaël

Probability Pr(I) of this possible world =
0.7*0.13*0.23*0.66

Probability of a possible world

R		
a	d	0.2
f	e	0.7
d	a	0.13
b	e	0.81

S		
d	e	0.005
f	c	0.9
a	e	0.7
c	e	0.23

A possible world I

R	
f	e
d	a

S	
c	e

Q			
c	e	f	0.66

Q		
c	e	f

Probability Pr(I) of this possible world =
0.7*0.13*0.23*0.66
(1-0.2)(1-0.81)*(1-0.005)*(1-0.9)*(1-0.7)

Monet Mikaël

Probabilistic query evaluation (PQE)

- Focus on Boolean queries (yes/no)

Probabilistic query evaluation (PQE)

- Focus on Boolean queries (yes/no)
- Probability of a query $\mathbf{Q}$ on probabilistic instance $\mathfrak{I}$:

$$\mathrm{P}(\mathbf{Q}) = \sum_{I \subseteq \mathfrak{I}, I \models \mathbf{Q}} \Pr(I)$$

Probabilistic query evaluation (PQE)

- Focus on Boolean queries (yes/no)
- Probability of a query $\mathbf{Q}$ on probabilistic instance $\mathfrak{I}$:

$$\mathrm{P}(\mathbf{Q}) = \sum_{I \subseteq \mathfrak{I}, I \vDash \mathbf{Q}} \Pr(I)$$

- Problem: in general #P-hard

3 possible directions

- Approximate
- Restrict queries
- Restrict instances

1) Approximate probability computation

1) Approximate probability computation

- Monte-Carlo sampling

1) Approximate probability computation

- Monte-Carlo sampling
- Inconvenient: running time quadratic in desired precision

1) Approximate probability computation

- Monte-Carlo sampling

- Inconvenient: running time quadratic in desired precision

  $\Rightarrow$ Not adequate for low probabilities.

2) Restricting the class of queries

- [Dalvi and Suciu 2012] shows the following dichotomy for any UCQ Q :

2) Restricting the class of queries

- [Dalvi and Suciu 2012] shows the following dichotomy for any UCQ Q :
- Either PQE is PTIME on all intances

2) Restricting the class of queries

- [Dalvi and Suciu 2012] shows the following dichotomy for any UCQ Q :
- Either PQE is PTIME on all intances
- Or PQE is #P-hard on all instances

2) Restricting the class of queries

- [Dalvi and Suciu 2012] shows the following dichotomy for any UCQ Q :
- Either PQE is PTIME on all intances
- Or PQE is #P-hard on all instances

- Simple conjunctive query ∃x,y R(x),S(x,y),T(y) is already #P-hard!

2) Restricting the class of queries

- [Dalvi and Suciu 2012] shows the following dichotomy for any UCQ Q :
- Either PQE is PTIME on all intances
- Or PQE is #P-hard on all instances
- Simple conjunctive query ∃x,y R(x),S(x,y),T(y) is already #P-hard!
- Criterion is too crisp

3) Restricting the shape of the instances

- Bound the ***treewidth*** of instances by a constant

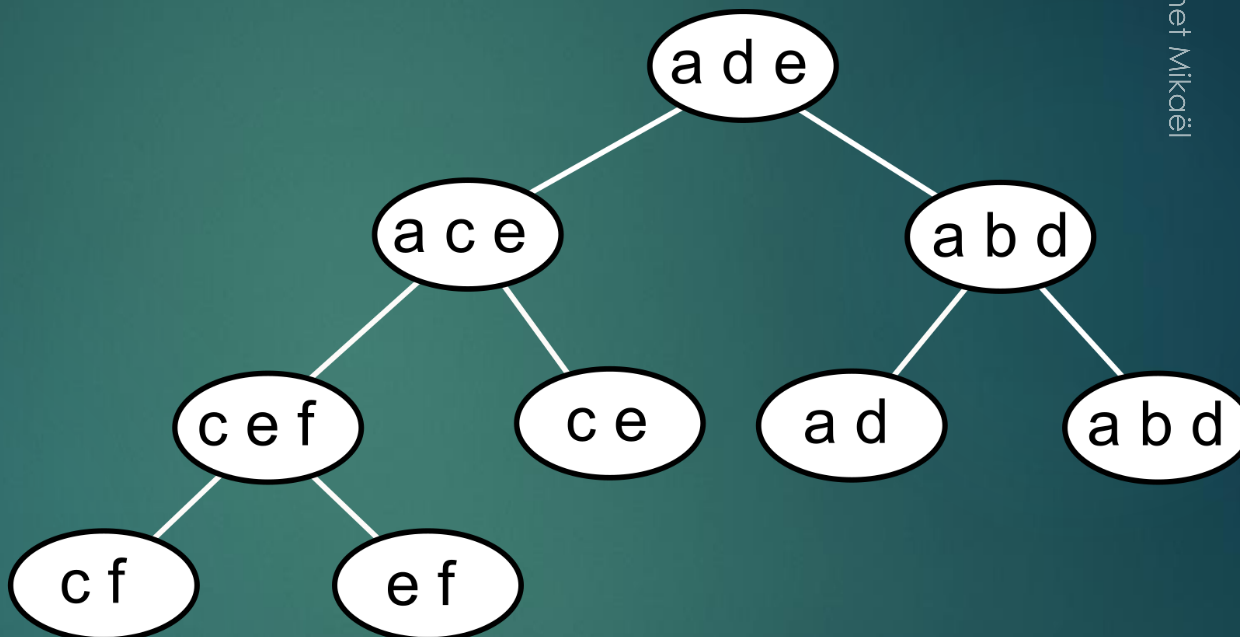
- Treewidth: measure used to tell how far a graph is from being a tree

Treewidth

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f



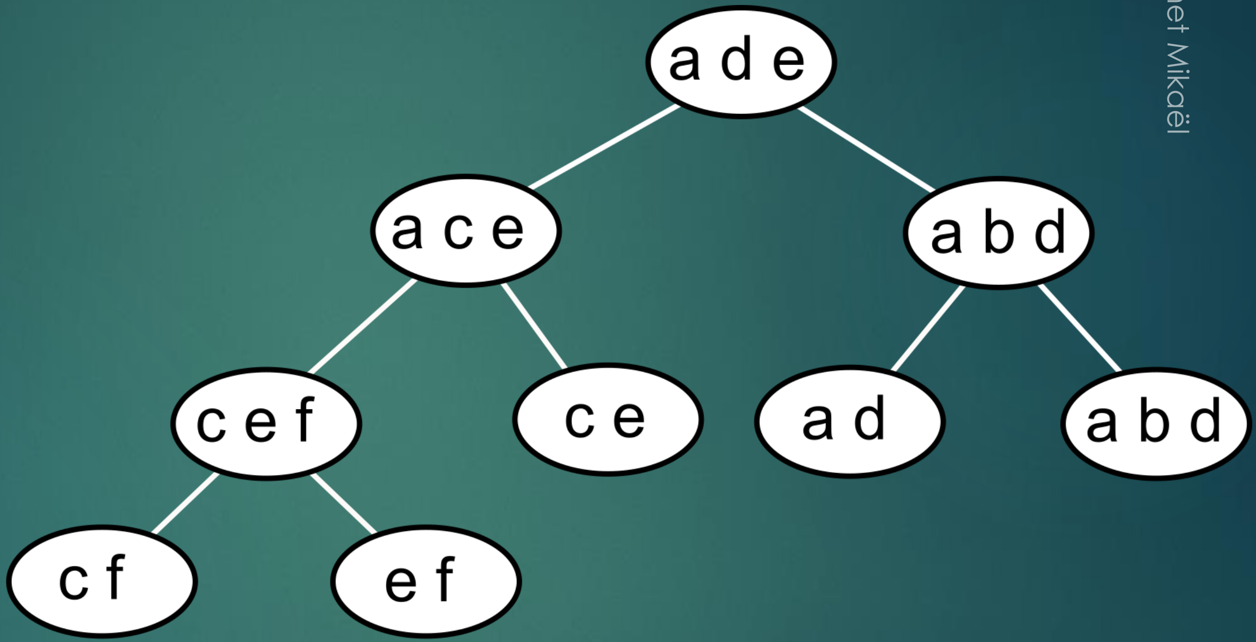
Monet Mikaël

Treewidth

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f



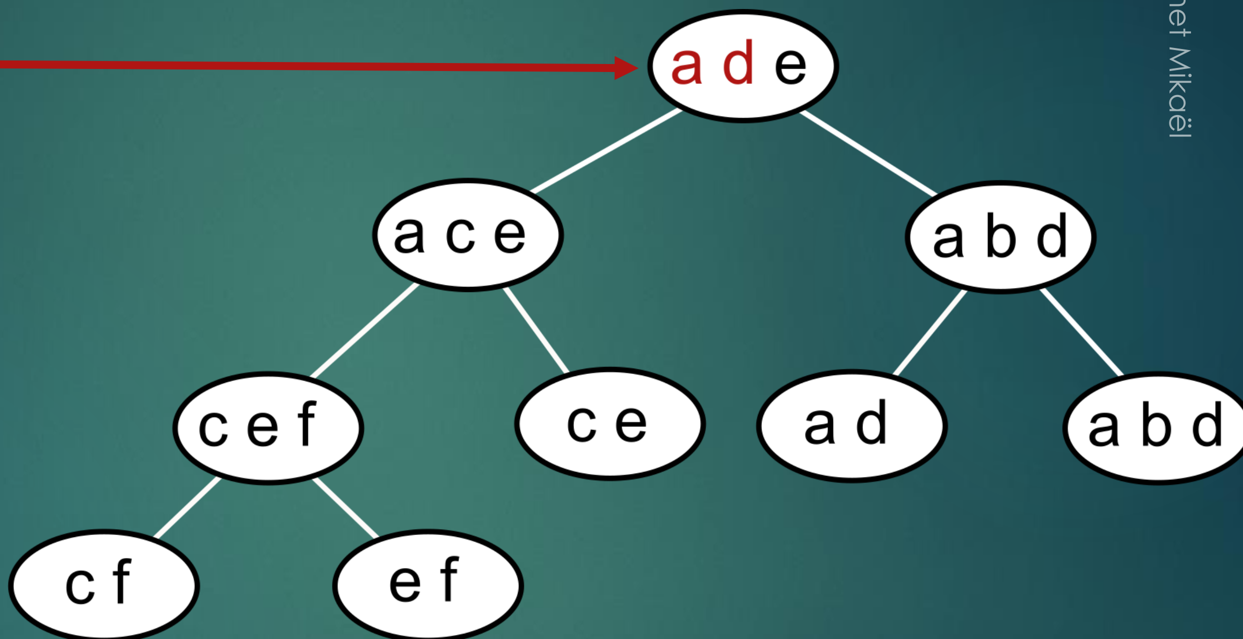
Treewidth

Monet Mikaël

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f

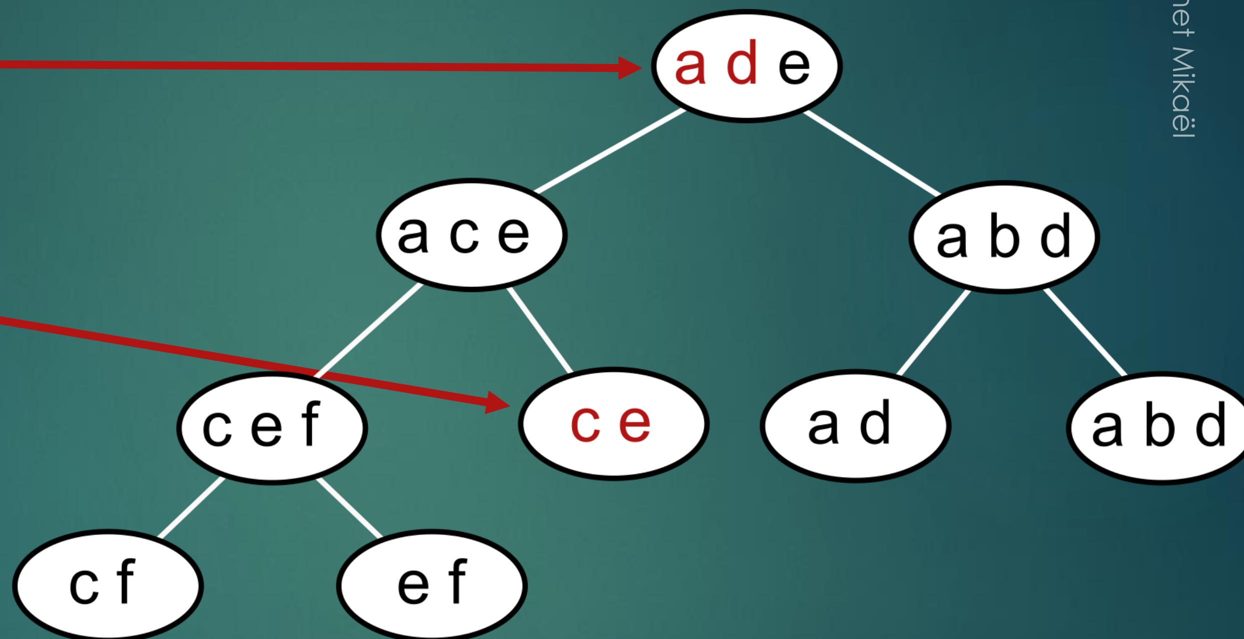


Treewidth

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f



Monet Mikaël

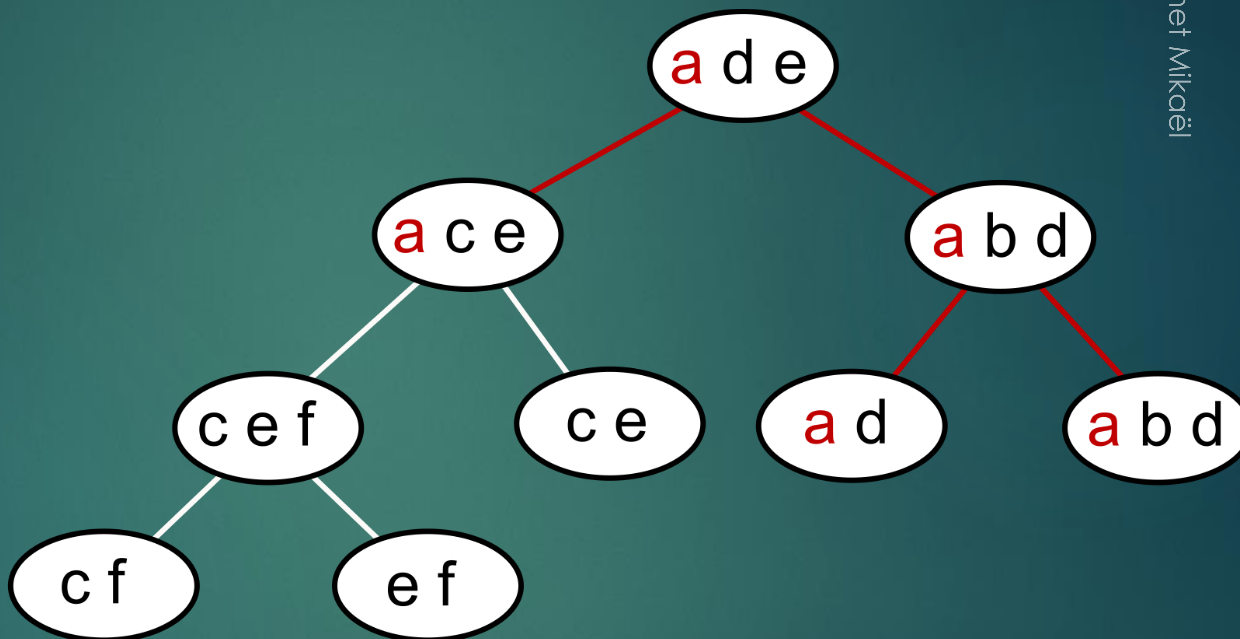
Treewidth

Monet Mikaël

R	
a	d
f	e
d	a
b	e

S	
d	e
f	c
a	e
c	e

Q		
c	e	f

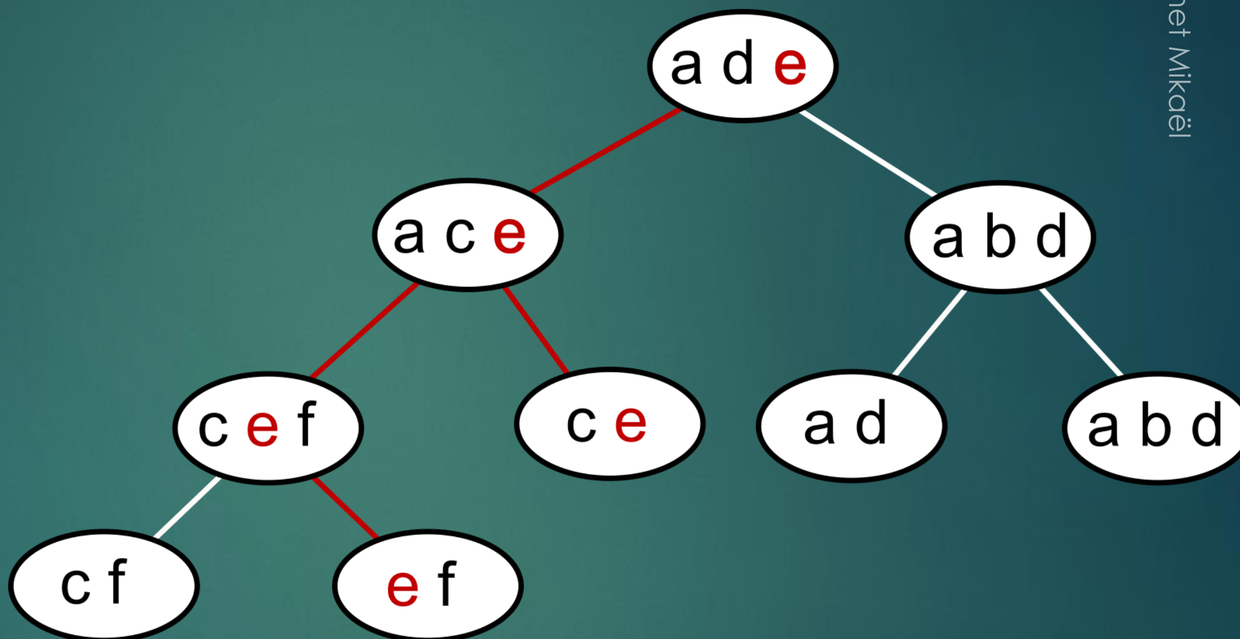


Treewidth

R	
a	d
f	e
d	a
b	e

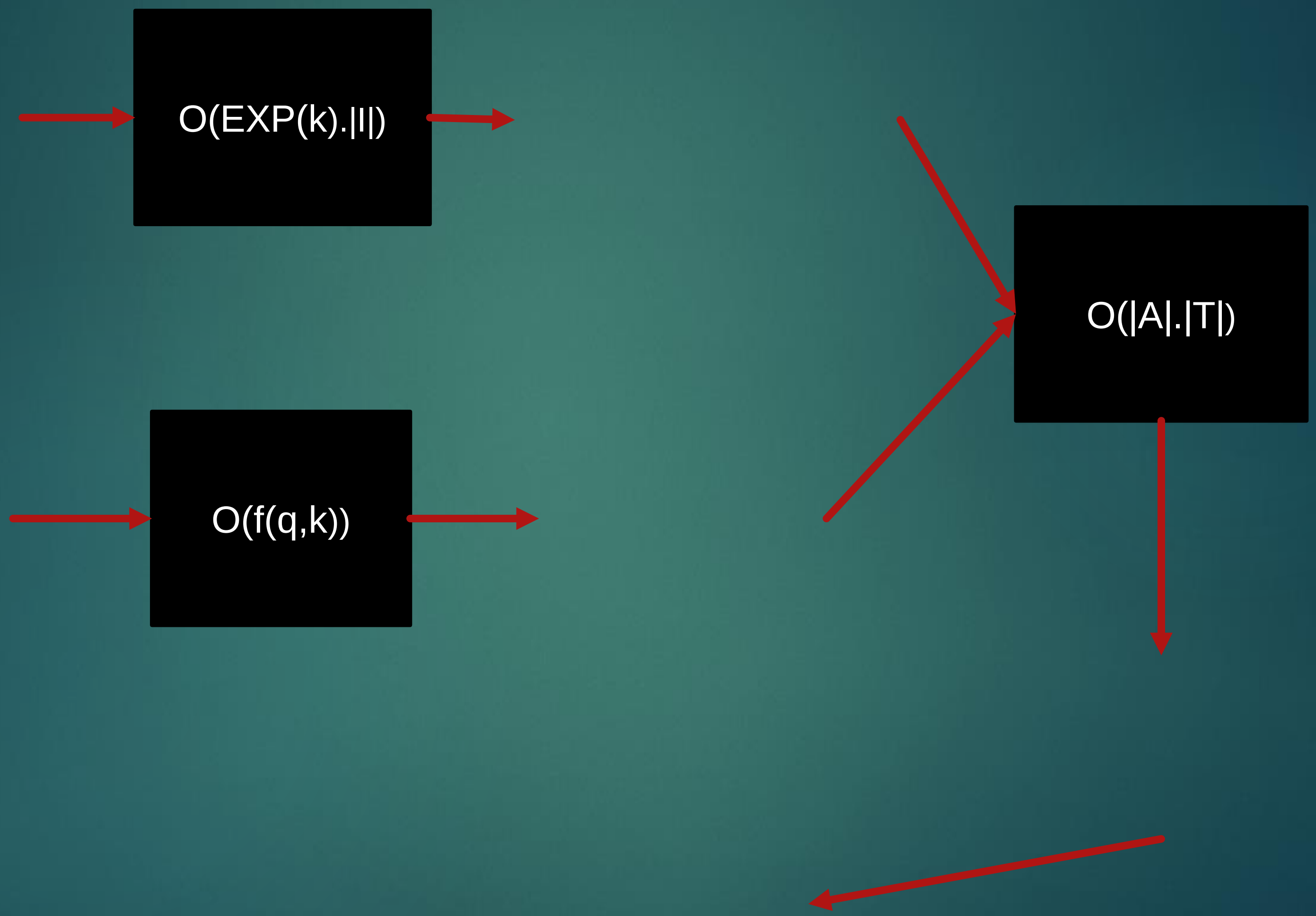
S	
d	e
f	c
a	e
c	e

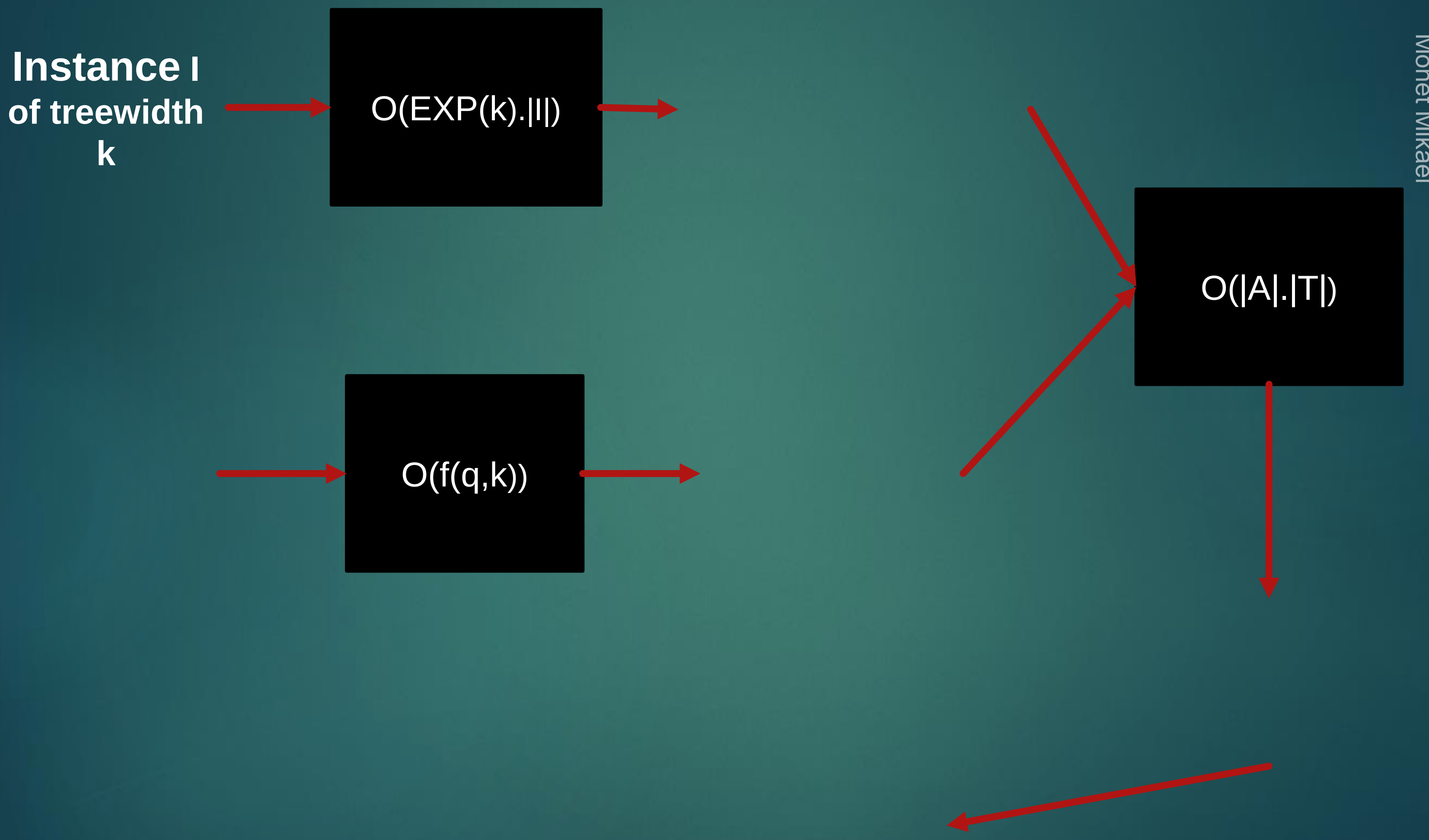
Q		
c	e	f

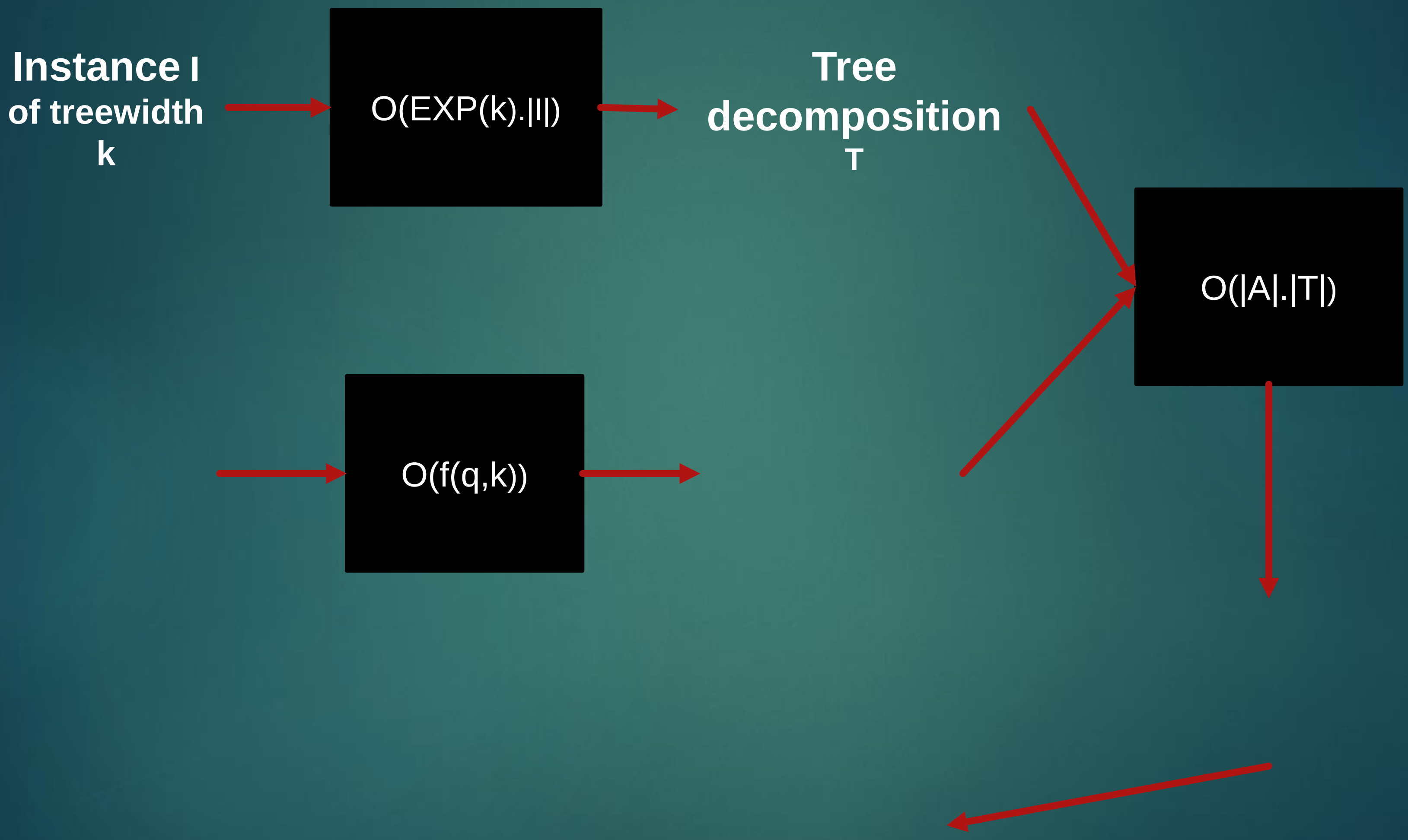


Divide and conquer !

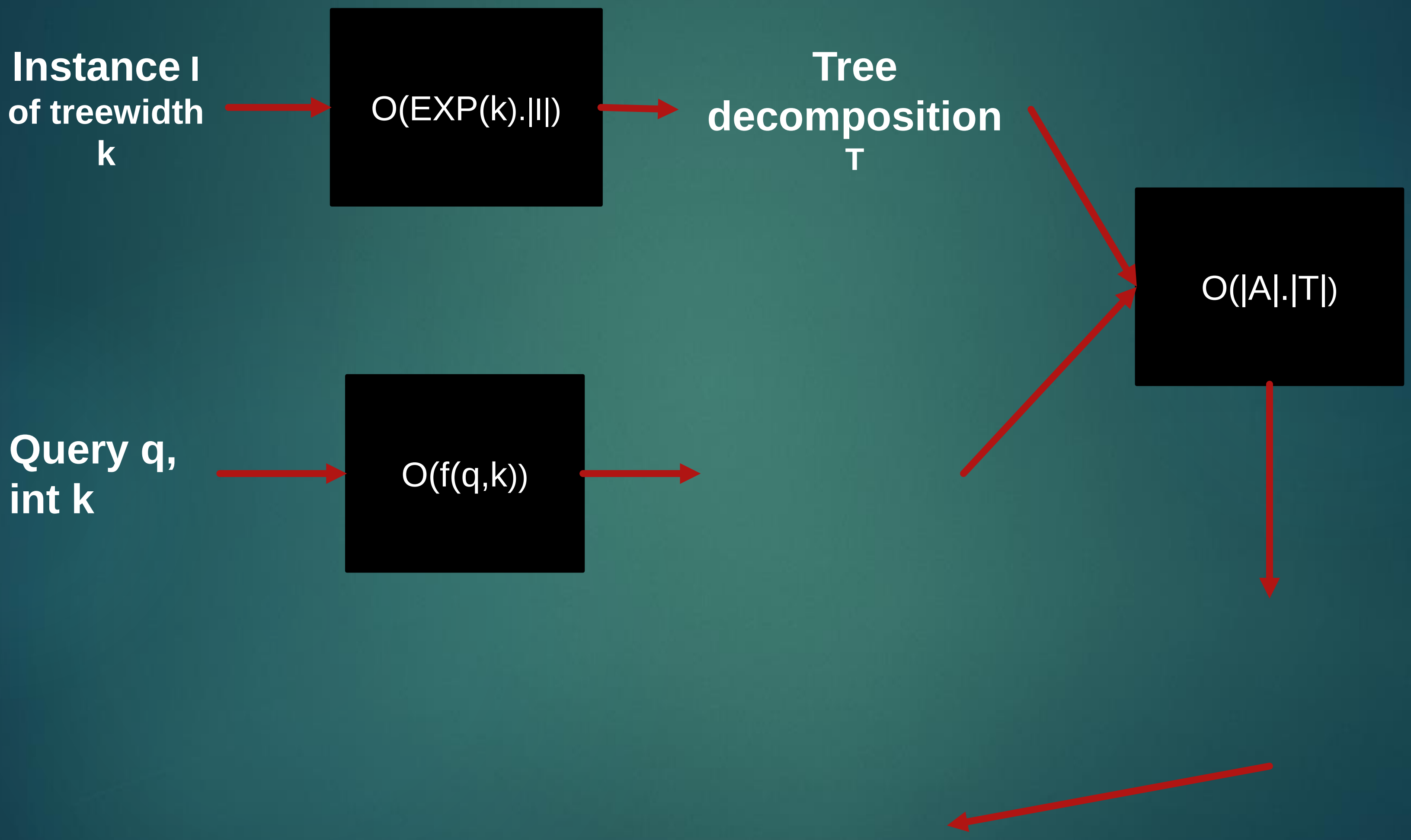
Monet Mikaël


O(EXP(k).|I|)
O(f(q,k))
O(|A|.|T|)

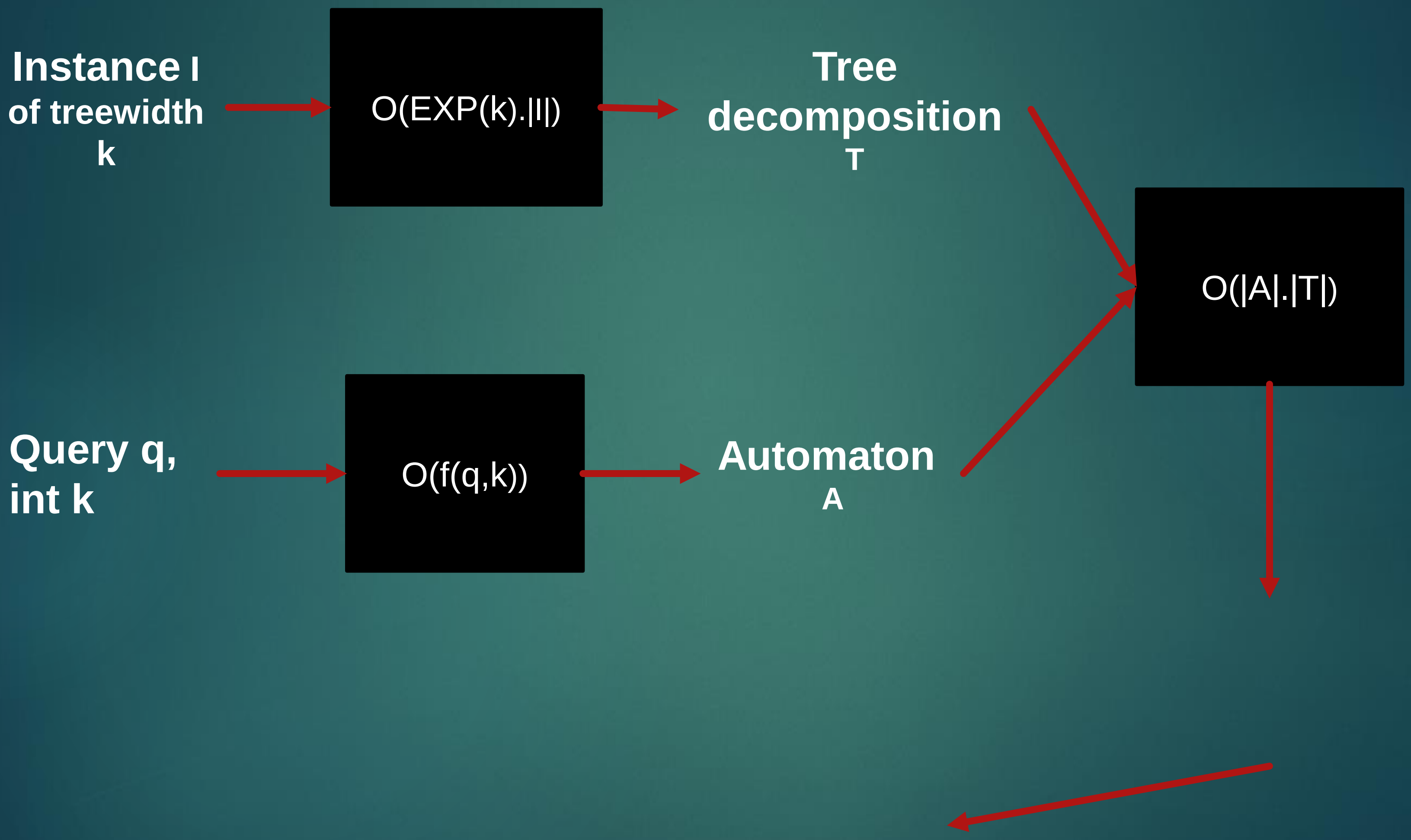

Instance I
of treewidth
k
O(EXP(k).|I|)
O(f(q,k))
O(|A|.|T|)



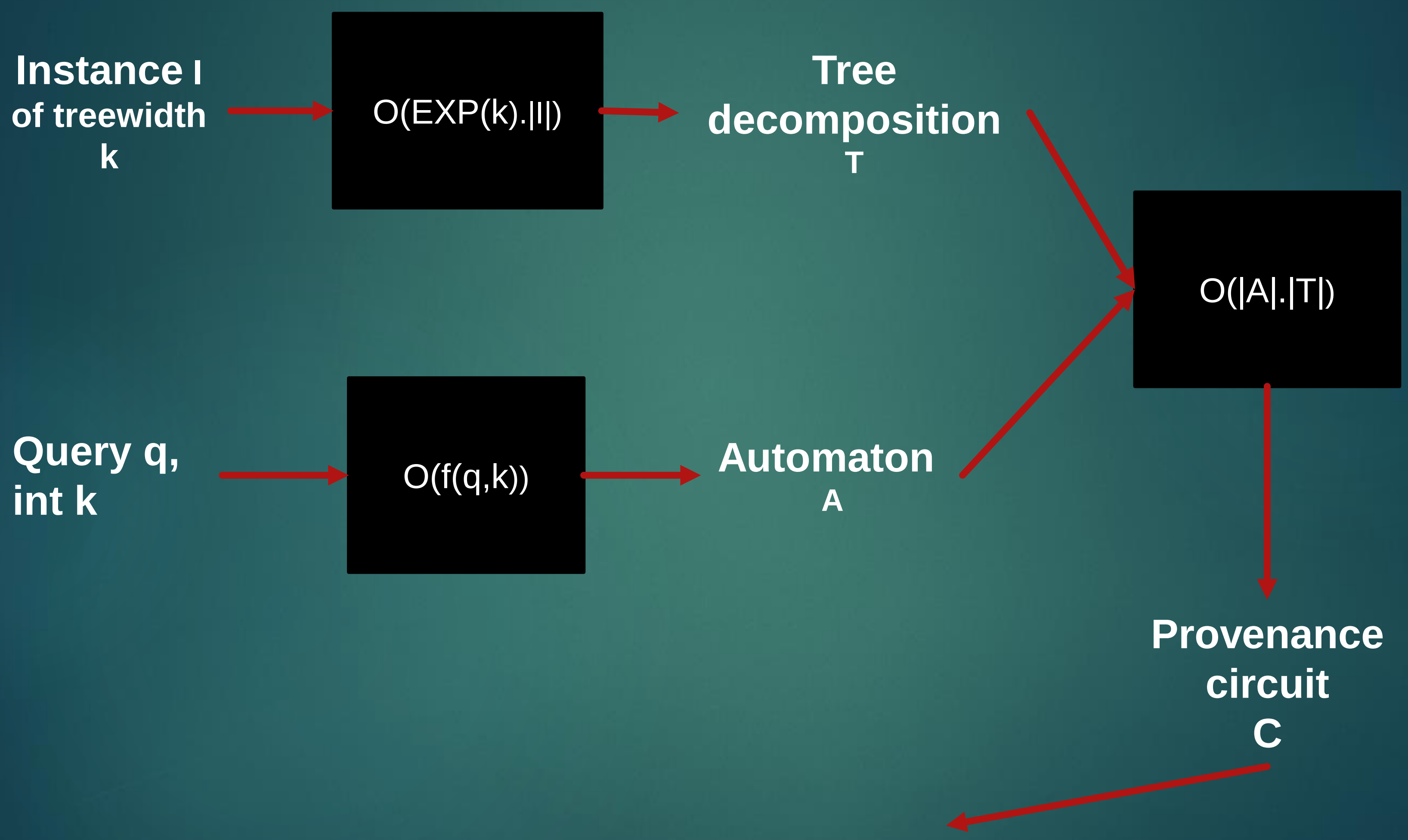
Monet Mikaël

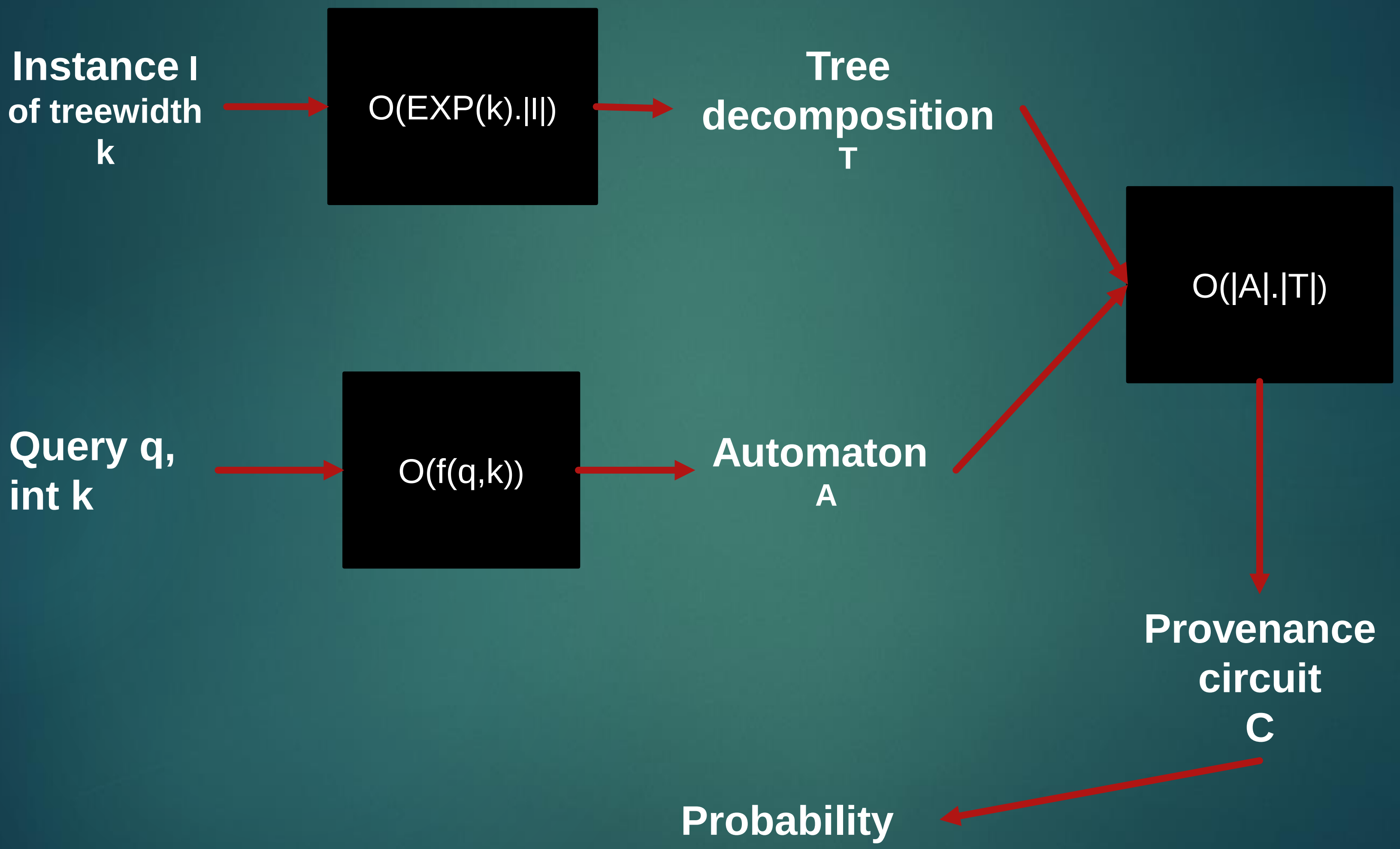


Monet Mikaël

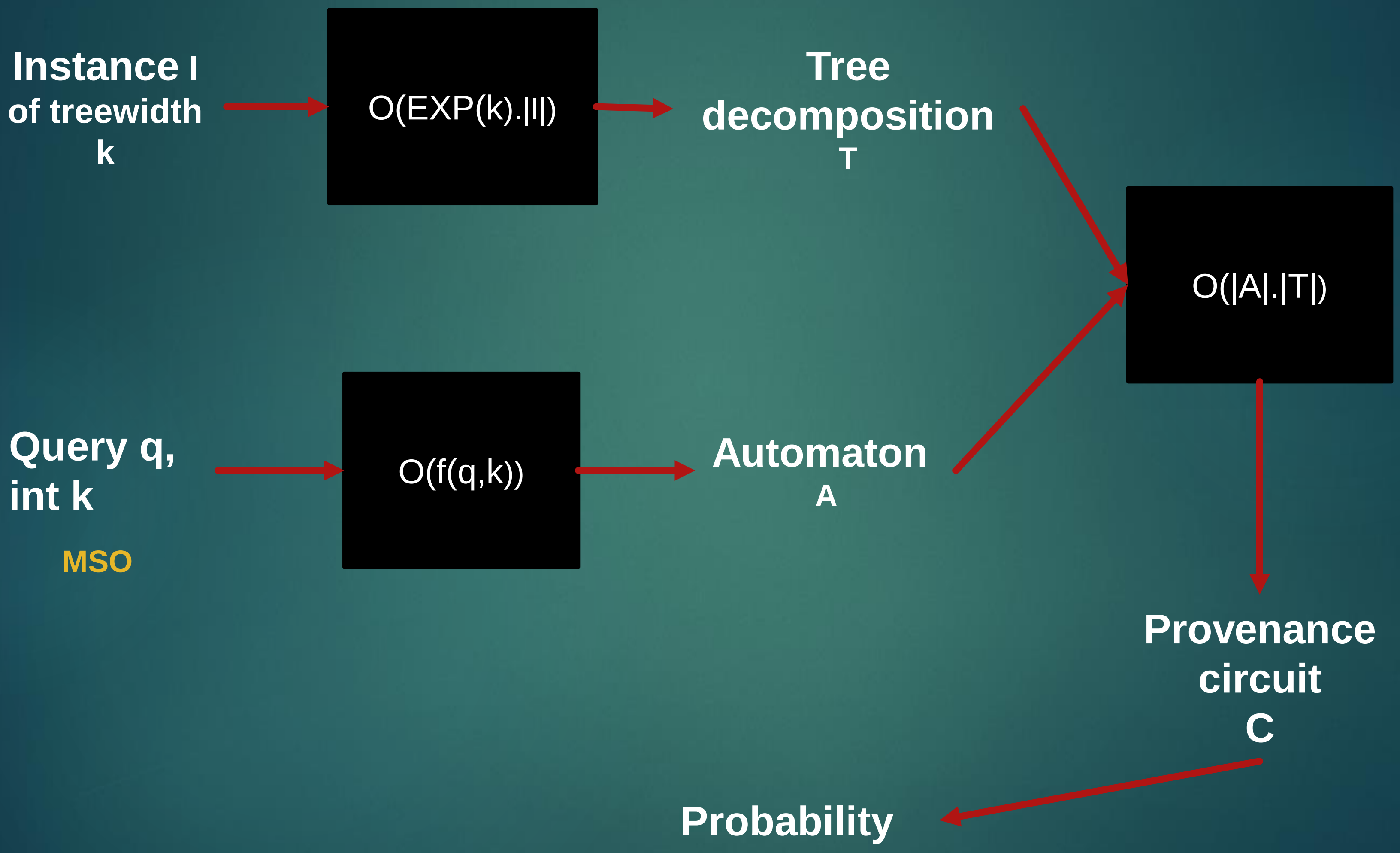


Monet Mikaël


Instance I
of treewidth
k
O(EXP(k).|I|)
Tree
decomposition
T
Query q,
int k
O(f(q,k))
Automaton
A
O(|A|.|T|)
Provenance
circuit
C

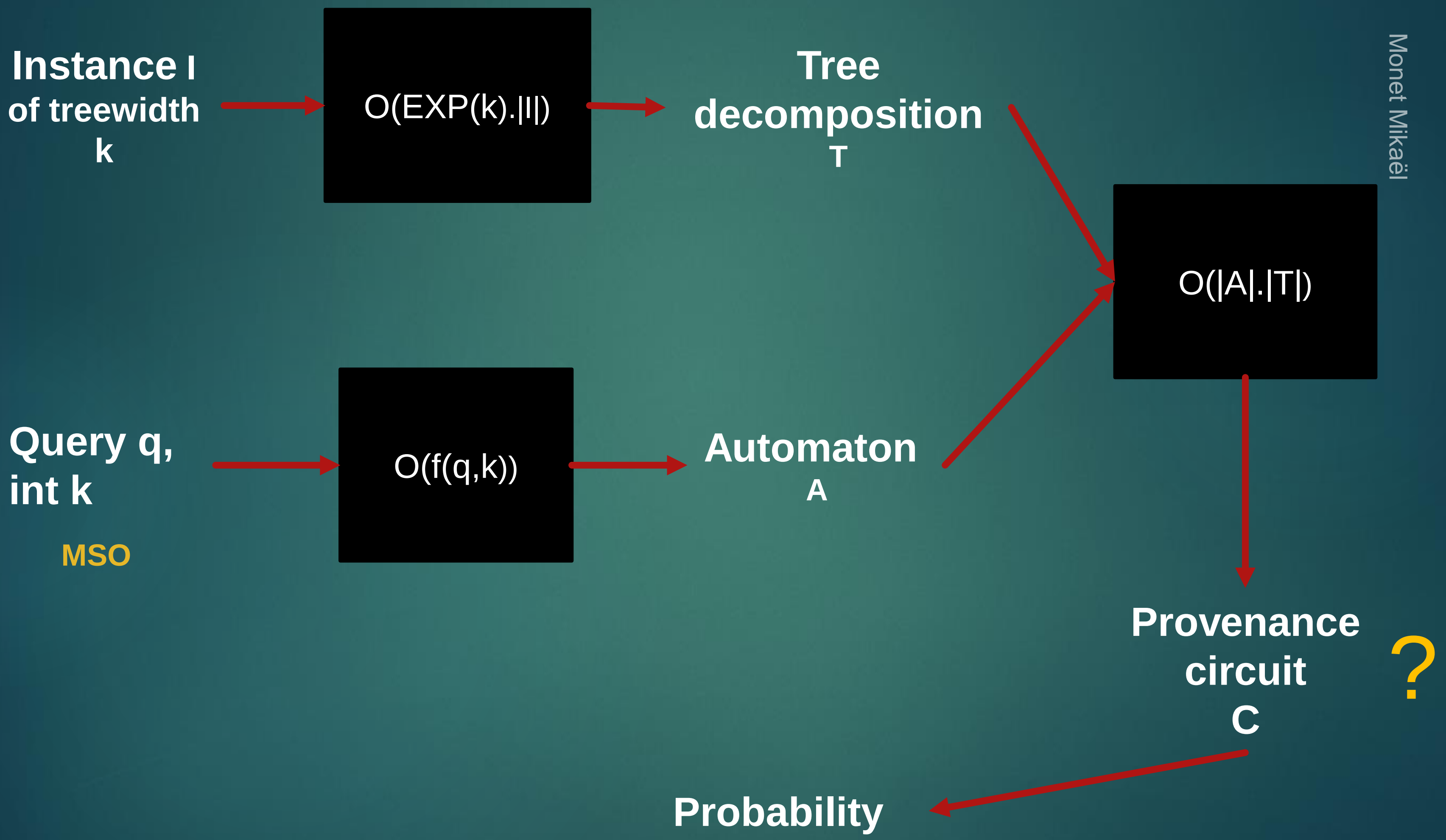


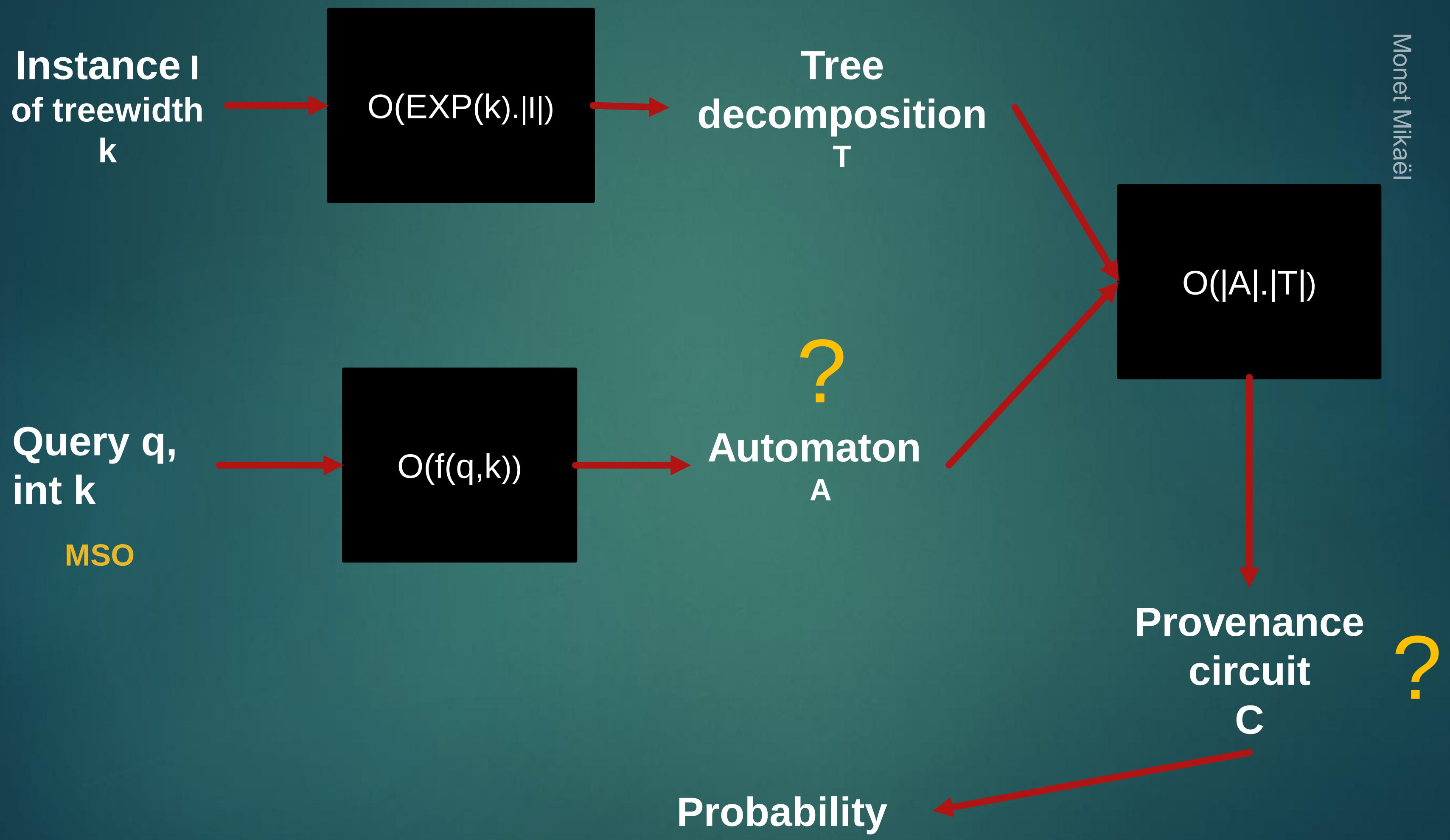
Monet Mikaël



Monet Mikaël

Monet Mikaël





Monet Mikaël

Provenance circuit C of query Q on instance I

Provenance circuit C of query Q on instance I

- Boolean circuit (AND, OR, NOT gates)

Provenance circuit C of query Q on instance I

- Boolean circuit (AND, OR, NOT gates)
- Inputs = the facts of I

Provenance circuit C of query Q on instance I

- Boolean circuit (AND, OR, NOT gates)
- Inputs = the facts of I

- For every $\nu : I \rightarrow \{\text{true}, \text{false}\}$

  $\nu(I) \vDash Q$ iff $\nu(C) = 1$

Tree automata

- A bottom-up deterministic tree automaton on {a, b}-trees is a tuple A = (Q, F, **ι**, **δ**) where :
- Q : finite set of states
- F ⊆ Q : accepting states
- **ι** : {a, b} → Q , determining state for the leaves
- **δ** : {a, b} X Q^2 → Q , determining the state for internal nodes

Run of an automaton on a tree

- Q = {O, O, O}

Run of an automaton on a tree

- Q = {O, O, O}
- F = {O}

Run of an automaton on a tree

- Q = {O, O, O}
- F = {O}
- ι = { (a, O), (b, O)}

Run of an automaton on a tree

- Q = {O, O, O}
- F = {O}
- ι = { (a, O), (b, O)}

δ=

lab	q1	q2	out
a	O	O	O
a	O	?	O
a	?	O	O
a	O	?	O
a	?	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	?	O
b	?	O	O

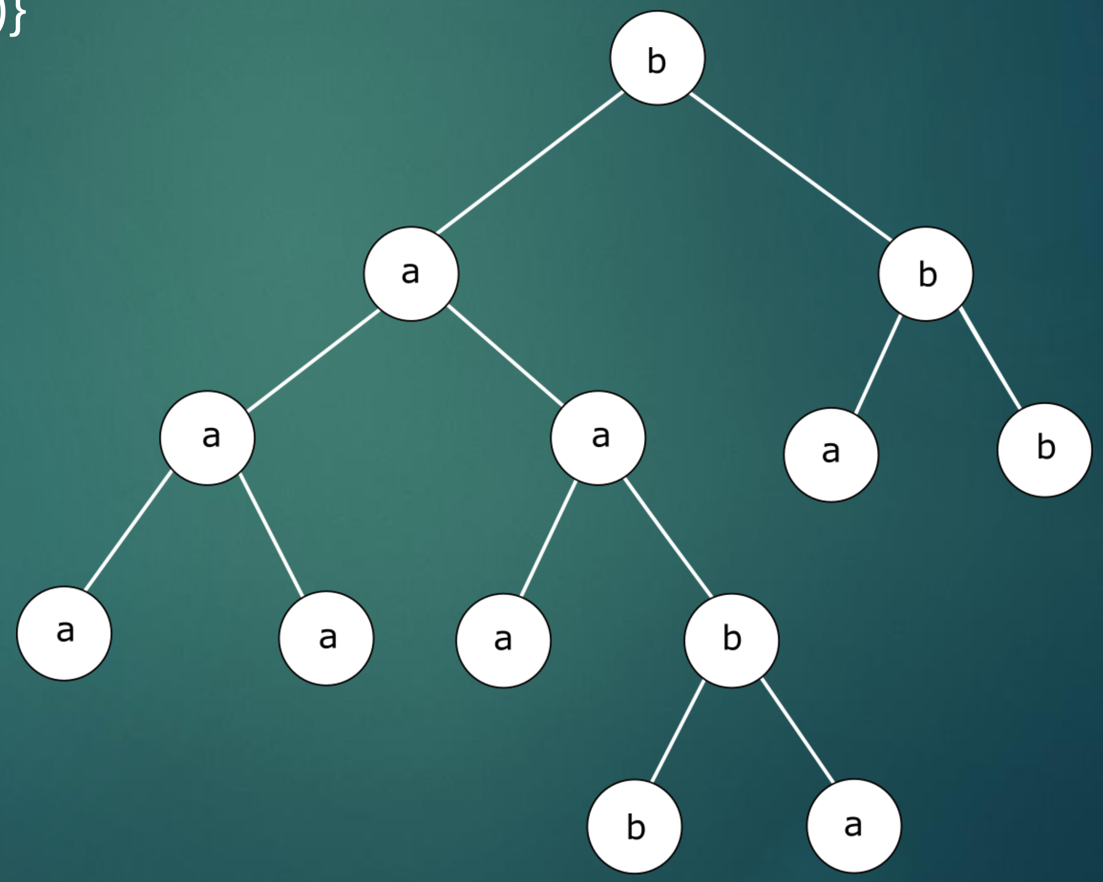
Monet Mikaël

Run of an automaton on a tree

- Q = {O (red), O (yellow), O (green)}
- F = {O (green)}
- ι = { (a, O (red)), (b, O (yellow))}

δ=

lab	q1	q2	out
a	O (red)	O (red)	O (red)
a	O (yellow)	?	O (green)
a	?	O (yellow)	O (green)
a	O (green)	?	O (green)
a	?	O (green)	O (green)
b	O (red)	O (red)	O (yellow)
b	O (red)	O (yellow)	O (yellow)
b	O (yellow)	O (red)	O (yellow)
b	O (yellow)	O (yellow)	O (yellow)
b	O (green)	?	O (green)
b	?	O (green)	O (green)



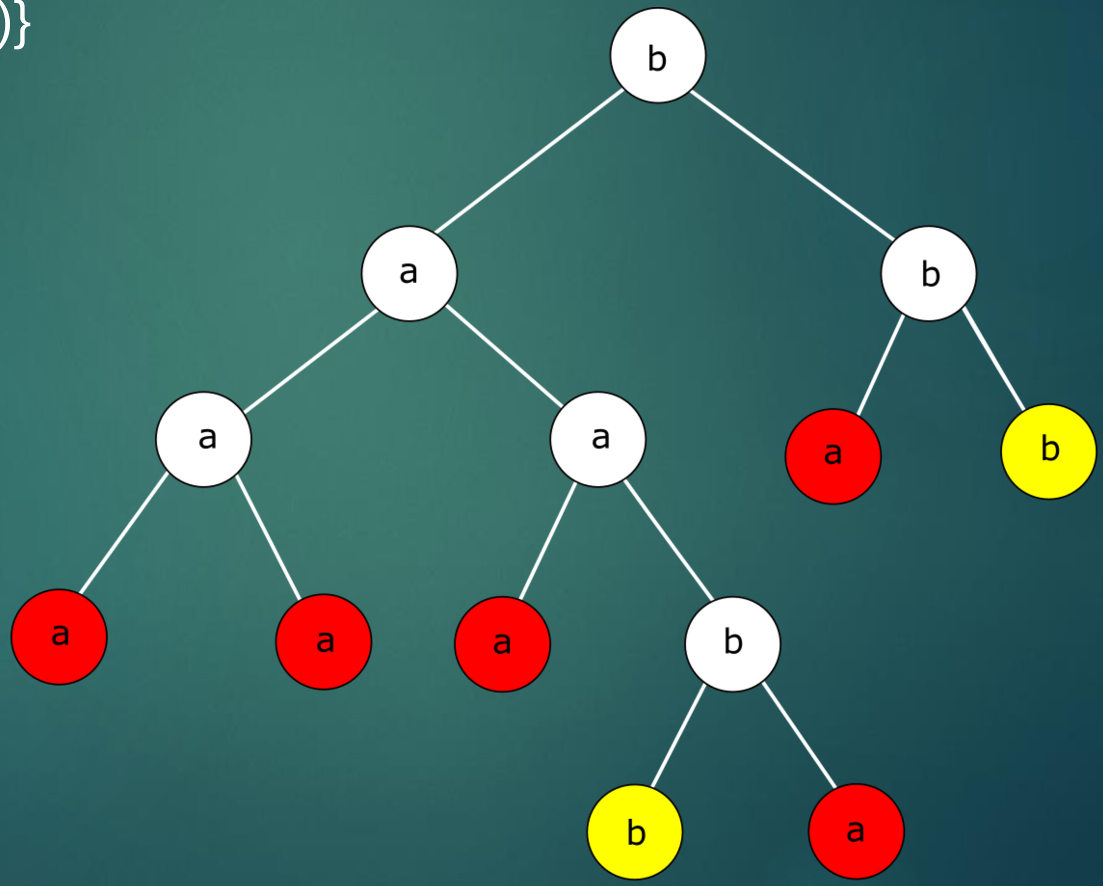
Monet Mikaël

Initialization of the leaves

- Q = {O (red), O (yellow), O (green)}
- F = {O (green)}
- ι = { (a, O (red)), (b, O (yellow))}

δ=

lab	q1	q2	out
a	O (red)	O (red)	O (red)
a	O (yellow)	?	O (green)
a	?	O (yellow)	O (green)
a	O (green)	?	O (green)
a	?	O (green)	O (green)
b	O (red)	O (red)	O (yellow)
b	O (red)	O (yellow)	O (yellow)
b	O (yellow)	O (red)	O (yellow)
b	O (yellow)	O (yellow)	O (yellow)
b	O (green)	?	O (green)
b	?	O (green)	O (green)



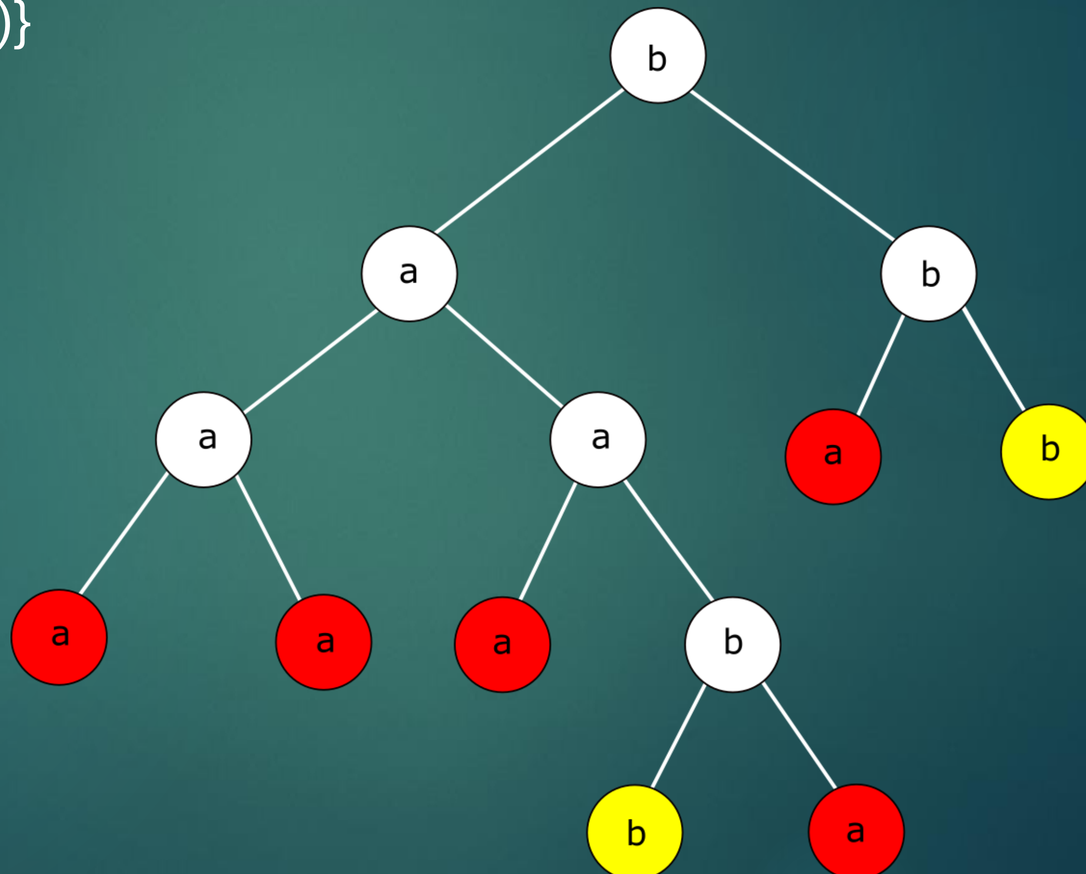
Monet Mikaël

Initialization of the leaves

- Q = {O (red), O (yellow), O (green)}
- F = {O (green)}
- ι = { (a, O (red)), (b, O (yellow))}

δ=

lab	q1	q2	out
a	O (red)	O (red)	O (red)
a	O (yellow)	?	O (green)
a	?	O (yellow)	O (green)
a	O (green)	?	O (green)
a	?	O (green)	O (green)
b	O (red)	O (red)	O (yellow)
b	O (red)	O (yellow)	O (yellow)
b	O (yellow)	O (red)	O (yellow)
b	O (yellow)	O (yellow)	O (yellow)
b	O (green)	?	O (green)
b	?	O (green)	O (green)



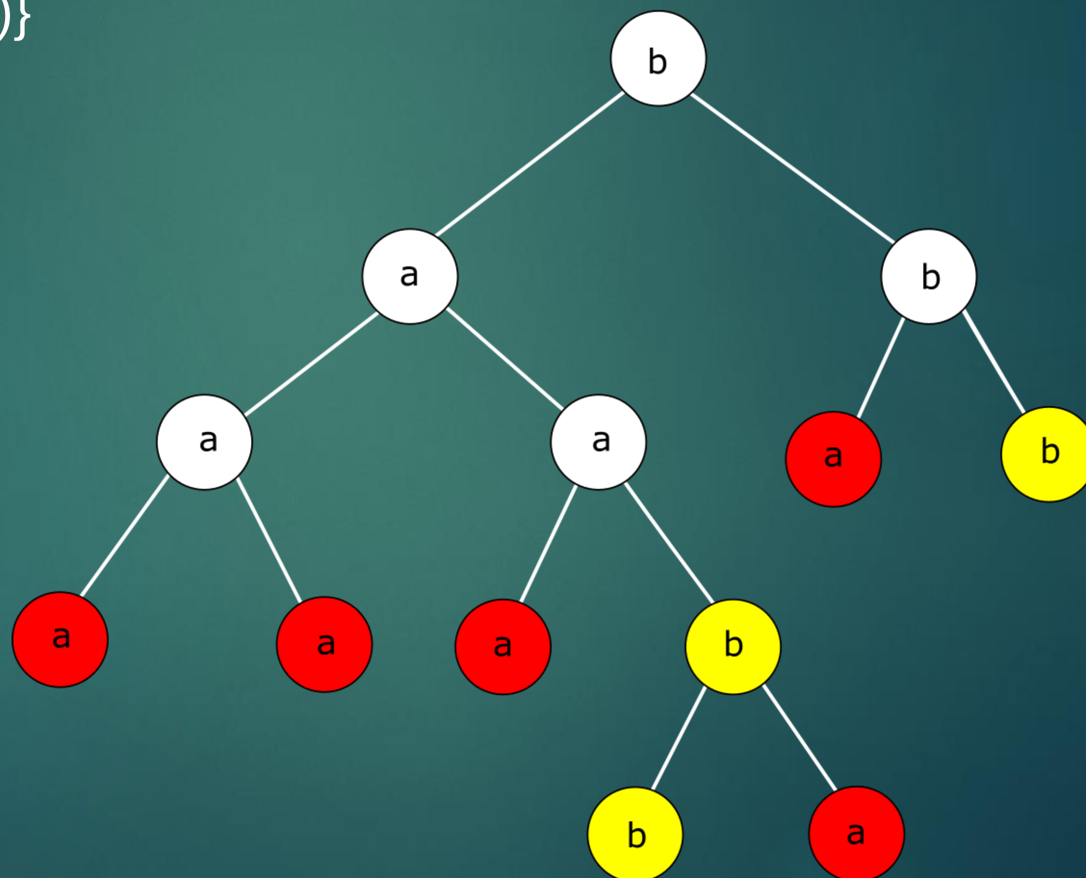
Monet Mikaël

Internal nodes

- Q = {O, O, O}
- F = {O}
- ι = { (a, O), (b, O)}

δ=

lab	q1	q2	out
a	O	O	O
a	O	?	O
a	?	O	O
a	O	?	O
a	?	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	?	O
b	?	O	O

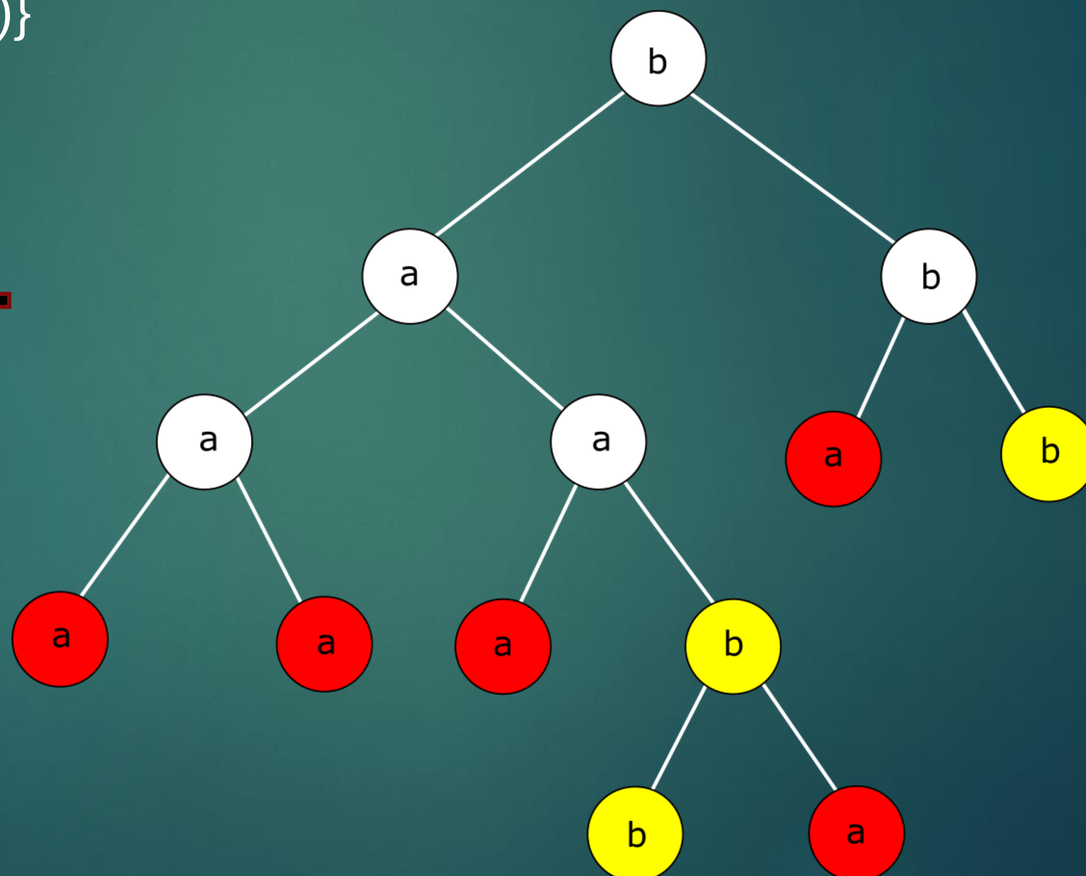


Internal nodes

- Q = {O, O, O}
- F = {O}
- ι = { (a, O), (b, O)}

δ=

lab	q1	q2	out
a	O	O	O
a	O	?	O
a	?	O	O
a	O	?	O
a	?	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	?	O
b	?	O	O

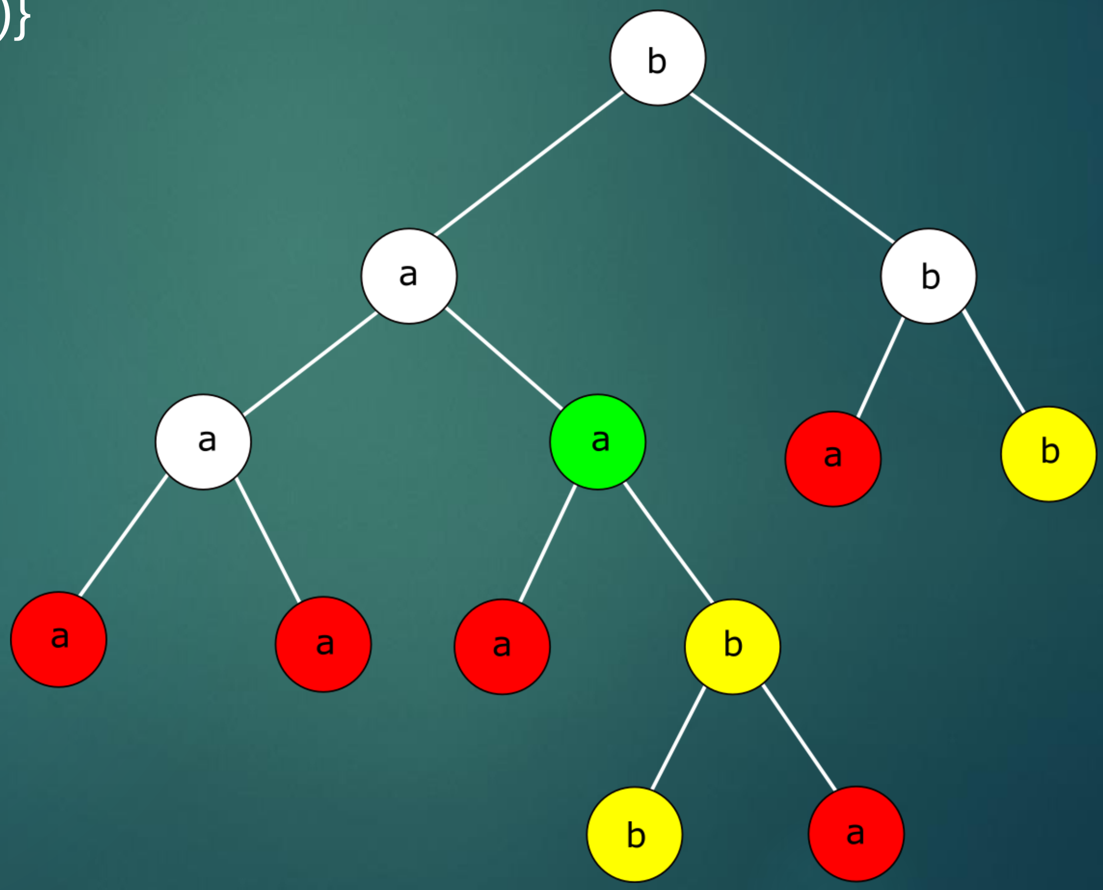


And so on…

- Q = {O, O, O}
- F = {O}
- ι = { (a, O), (b, O)}

δ=

lab	q1	q2	out
a	O	O	O
a	O	?	O
a	?	O	O
a	O	?	O
a	?	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	O	O
b	O	?	O
b	?	O	O

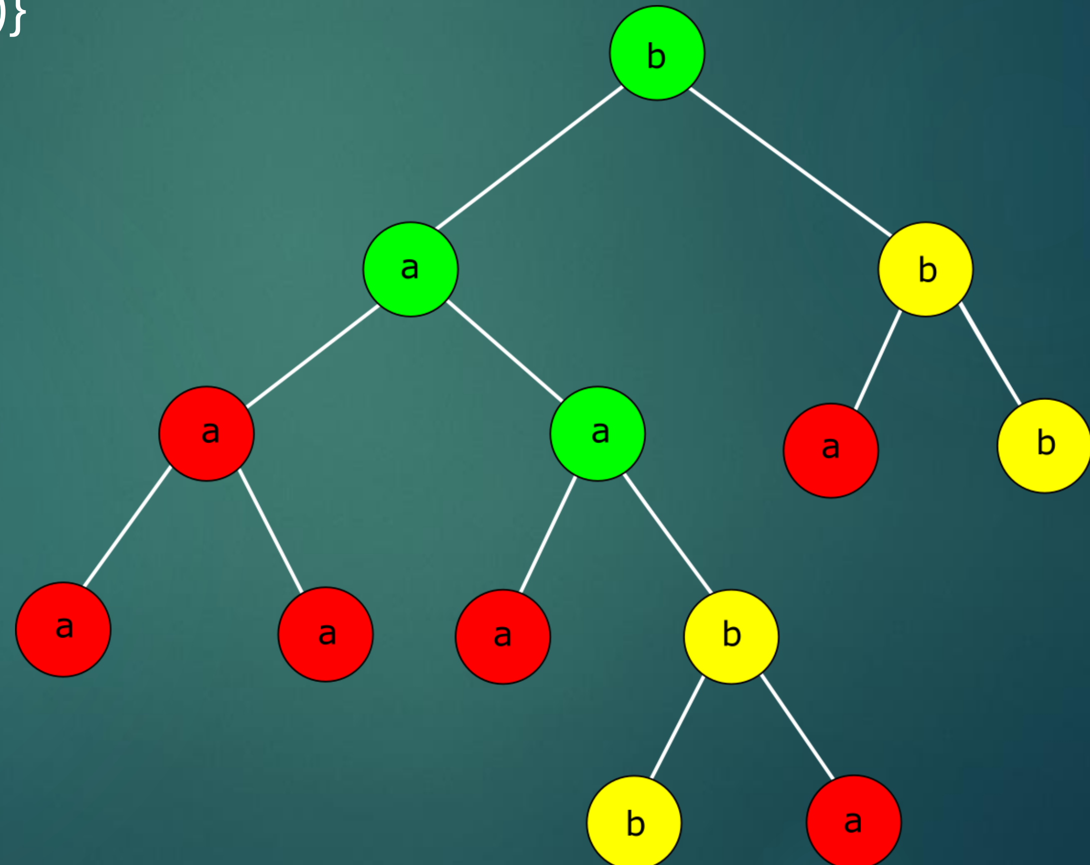


This tree is in the language of A

- Q = {O (red), O (yellow), O (green)}
- F = {O (green)}
- ι = { (a, O (red)), (b, O (yellow))}

δ=

lab	q1	q2	out
a	O (red)	O (red)	O (red)
a	O (yellow)	?	O (green)
a	?	O (yellow)	O (green)
a	O (green)	?	O (green)
a	?	O (green)	O (green)
b	O (red)	O (red)	O (yellow)
b	O (red)	O (yellow)	O (yellow)
b	O (yellow)	O (red)	O (yellow)
b	O (yellow)	O (yellow)	O (yellow)
b	O (green)	?	O (green)
b	?	O (green)	O (green)



Monet Mikaël

Major drawbacks

- In general, computing the automaton has non-elementary complexity in the query

Major drawbacks

- In general, computing the automaton has non-elementary complexity in the query
- Exponential dependence in the instance treewidth

Major drawbacks

- In general, computing the automaton has non-elementary complexity in the query
- Exponential dependence in the instance treewidth

- Natural question: restrict queries to obtain tractable combined complexity of PQE on bounded treewidth instances?

Bad news…

Monet Mikaël

- We proved that:

Path queries on tree instances (treewidth = 1) is already #P-hard. (reduction from #MONOTONE-2-SAT)

Bad news…

- We proved that:

Path queries on tree instances (treewidth = 1) is already #P-hard. (reduction from #MONOTONE-2-SAT)

- What to do now?

Lower ambitions

- Restrict queries to obtain tractable combined complexity of ~~probabilistic~~ query evaluation on bounded treewidth instances?

Lower ambitions

- Restrict queries to obtain tractable combined complexity of ~~probabilistic~~ query evaluation on bounded treewidth instances?

- We now aim at a tractable combined complexity for deterministic query evaluation: which queries, which automata, which provenance representation?

Two-way Alternating Tree Automata

- Can navigate the tree in every direction, can launch simultaneous runs

Two-way Alternating Tree Automata

- Can navigate the tree in every direction, can launch simultaneous runs

$$\boldsymbol{\delta}(a,q) = (q,l) \vee [\quad \wedge \quad]$$

Two-way Alternating Tree Automata

- Can navigate the tree in every direction, can launch simultaneous runs

$$\boldsymbol{\delta}(a,q) = (q,l) \vee [\ (q,p) \wedge \qquad]$$

Two-way Alternating Tree Automata

- Can navigate the tree in every direction, can launch simultaneous runs

$$\boldsymbol{\delta}(a,q) = (q,l) \vee [\ (q,p) \wedge (q',r)\ ]$$

Two-way Alternating Tree Automata

- Can navigate the tree in every direction, can launch simultaneous runs

$$\boldsymbol{\delta}(a,q) = (q,l) \vee [\ (q,p) \wedge (q',r)\ ]$$

- Intuition: less things to remember, more parallelizable

Cycluits

- Boolean circuits with cycles

Cycluits

- Boolean circuits with cycles
- Least fixed-point semantics

Cycluits

- Boolean circuits with cycles
- Least fixed-point semantics
- Beware of negations!

Cycluits

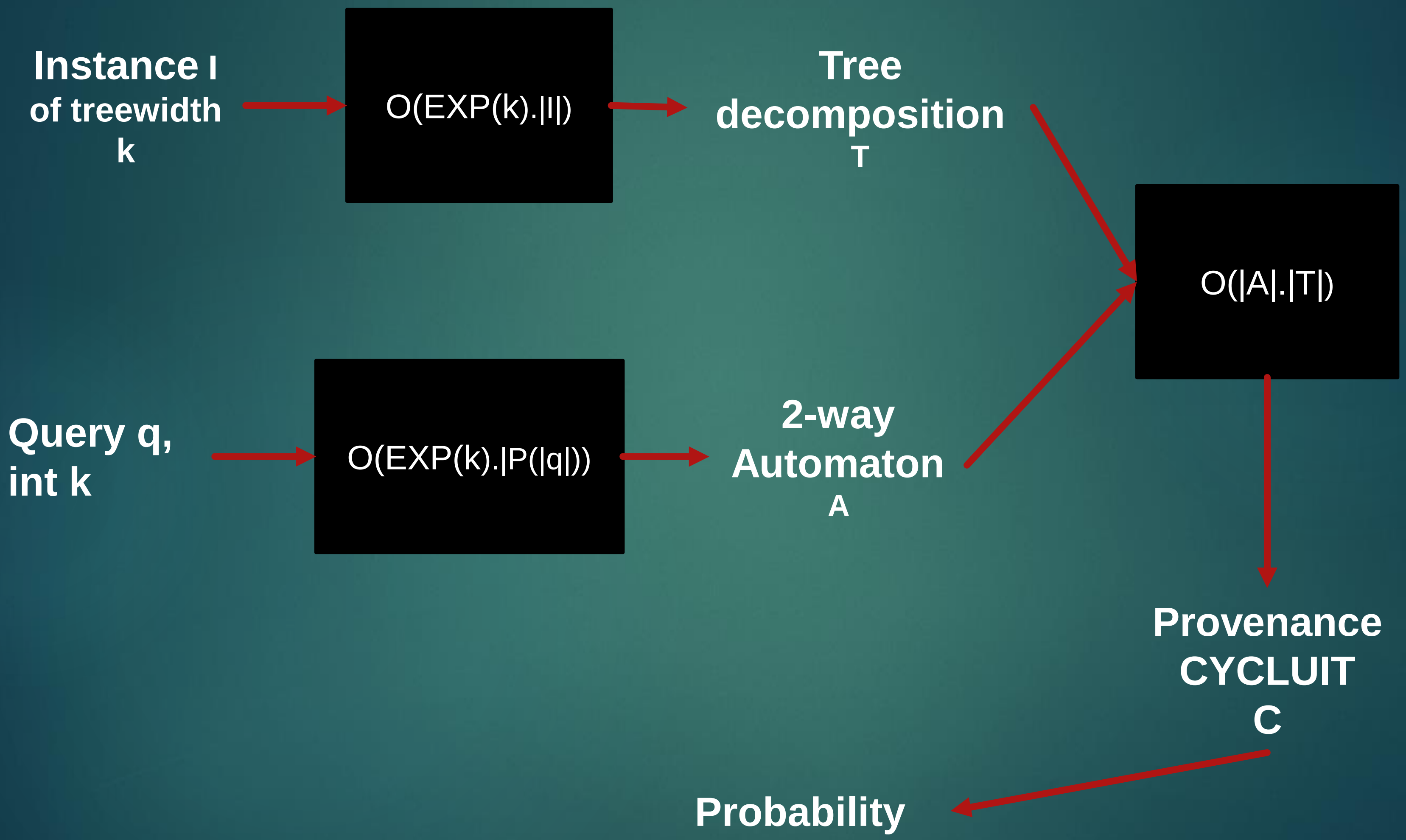
- Boolean circuits with cycles
- Least fixed-point semantics
- Beware of negations!
- Linear time evaluation

Cycluits

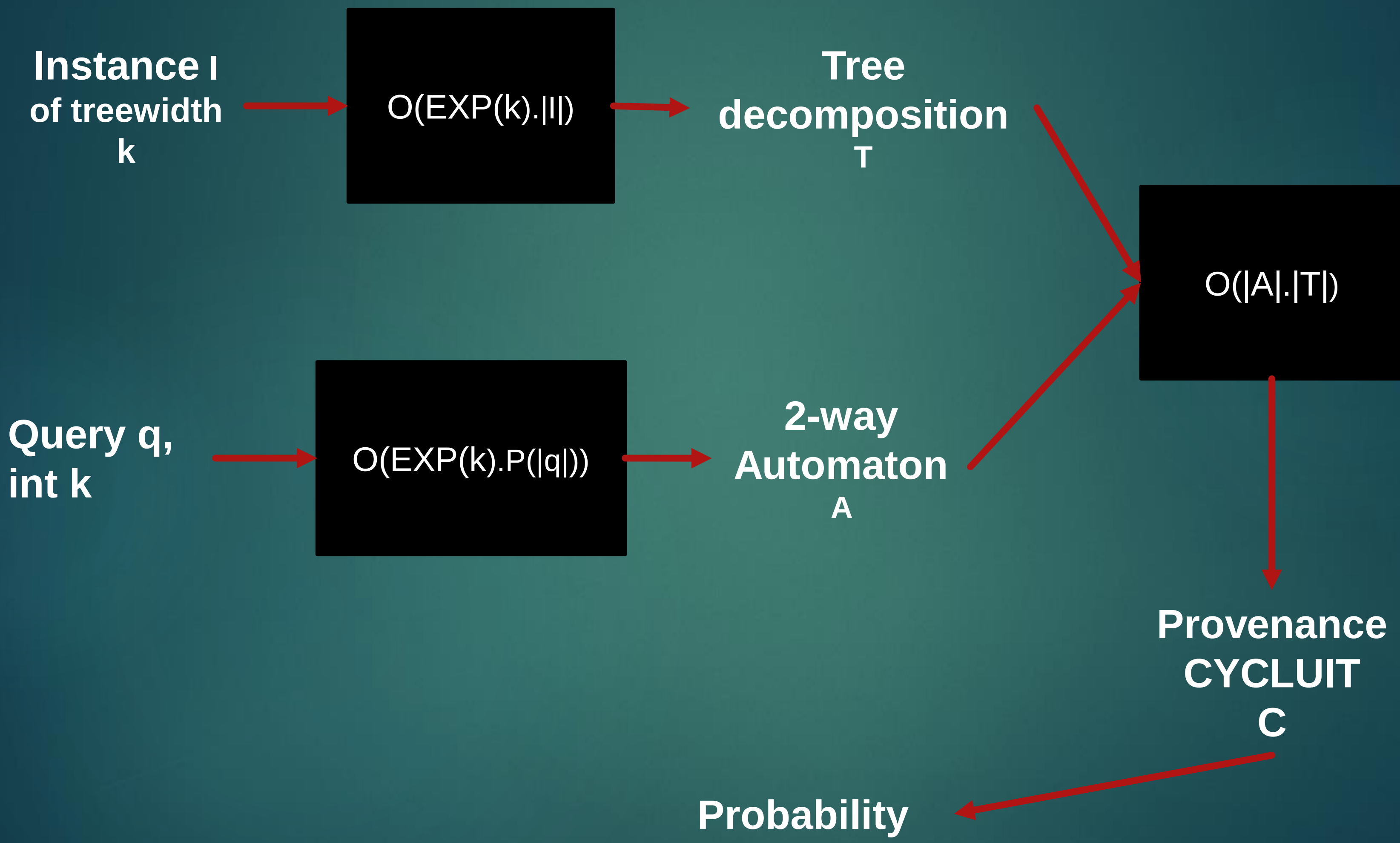
- Boolean circuits with cycles
- Least fixed-point semantics
- Beware of negations!
- Linear time evaluation
- Can be acyclified in quadratic time

Cycluits

- Boolean circuits with cycles
- Least fixed-point semantics
- Beware of negations!
- Linear time evaluation
- Can be acyclified in quadratic time
- Are they more concise?



Monet Mikaël



Thanks for your attention!