Circuits de provenance pour les arbres et les instances quasi-arborescentes

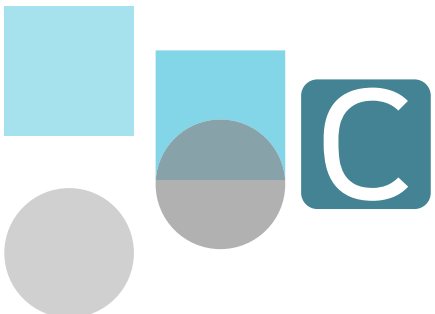
Antoine Amarilli[1], Pierre Bourhis[2], Pierre Senellart[1,3]

[1] Télécom ParisTech

[2] CNRS CRIStAL

[3] National University of Singapore

1er octobre 2015



Problème général

- Considérons une requête et une instance relationelle
- Souvent, évaluer la requête ne suffit pas :
  - → On cherche de l'information quantitative
  - → On cherche le lien entre le résultat et l'entrée

Problème général

- Considérons une requête et une instance relationelle
- Souvent, évaluer la requête ne suffit pas :
  - → On cherche de l'information quantitative
  - → On cherche le lien entre le résultat et l'entrée

→ Calculer la provenance de la requête !

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R	
a	b
b	c
d	e
e	d
f	f

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R	
a	b
b	c
d	e
e	d
f	f

- Résultat : vrai

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R		
a	b	Public
b	c	Secret
d	e	Confidentiel
e	d	Confidentiel
f	f	Secret défense

- Résultat : vrai
- Ajouter des annotations de sécurité :
  Public, Confidentiel, Secret, Secret défense, Indisponible

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R		
a	b	Public
b	c	Secret
d	e	Confidentiel
e	d	Confidentiel
f	f	Secret défense

- Résultat : vrai
- Ajouter des annotations de sécurité :
  Public, Confidentiel, Secret, Secret défense, Indisponible
- Quelle est l'accréditation nécessaire ?

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R		
a	b	Public
b	c	Secret
d	e	Confidentiel
e	d	Confidentiel
f	f	Secret défense

- Résultat : vrai
- Ajouter des annotations de sécurité :
  Public, Confidentiel, Secret, Secret défense, Indisponible
- Quelle est l'accréditation nécessaire ?

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R		
a	b	Public
b	c	Secret
d	e	Confidentiel
e	d	Confidentiel
f	f	Secret défense

- Résultat : vrai
- Ajouter des annotations de sécurité :
  Public, Confidentiel, Secret, Secret défense, Indisponible
- Quelle est l'accréditation nécessaire ?

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R		
a	b	Public
b	c	Secret
d	e	Confidentiel
e	d	Confidentiel
f	f	Secret défense

- Résultat : vrai
- Ajouter des annotations de sécurité :
  Public, Confidentiel, Secret, Secret défense, Indisponible
- Quelle est l'accréditation nécessaire ?

Exemple 1 : sécurité pour une requête conjonctive

- Considérons la requête conjonctive : $\exists xyz\ R(x, y) \wedge R(y, z)$
- Considérons l'instance relationnelle suivante :

R		
a	b	Public
b	c	Secret
d	e	Confidentiel
e	d	Confidentiel
f	f	Secret défense

- Résultat : vrai
- Ajouter des annotations de sécurité :
  Public, Confidentiel, Secret, Secret défense, Indisponible
- Quelle est l'accréditation nécessaire ?

→ Résultat : Confidentiel

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x,y) \wedge R(y,z)$.

R	
a	b
b	c
d	e
e	d
f	f

- Résultat : vrai

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité
- Combien de correspondances de la requête ?

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité
- Combien de correspondances de la requête ?

$\rightarrow$ Résultat : 1

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité
- Combien de correspondances de la requête ?

$\rightarrow$ Résultat : $1 + 1$

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité
- Combien de correspondances de la requête ?

$\rightarrow$ Résultat : $1 + 1 + 1$

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité
- Combien de correspondances de la requête ?

$\rightarrow$ Résultat : $1 + 1 + 1 + 1$

Exemple 2 : sémantique multiensembliste

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Résultat : vrai
- Ajoutons des annotations de multiplicité
- Combien de correspondances de la requête ?

$\rightarrow$ Résultat : $1 + 1 + 1 + 1 = 4$

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

- Résultat : vrai

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques
- Pour quelles sous-instances la requête est-elle vérifiée ?

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques
- Pour quelles sous-instances la requête est-elle vérifiée ?

$\rightarrow$ Résultat : $(f_1 \wedge f_2)$

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x,y) \wedge R(y,z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques
- Pour quelles sous-instances la requête est-elle vérifiée ?

$\rightarrow$ Résultat : $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x,y) \wedge R(y,z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques
- Pour quelles sous-instances la requête est-elle vérifiée ?

$\rightarrow$ Résultat : $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques
- Pour quelles sous-instances la requête est-elle vérifiée ?

$\rightarrow$ Résultat : $(f_1 \wedge f_2) \vee (f_3 \wedge f_4) \vee f_5$

Exemple 3 : faits incertains

Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Résultat : vrai
- Voyons les faits comme incertains, mettons-leur des annotations atomiques
- Pour quelles sous-instances la requête est-elle vérifiée ?

→ Résultat : $(f_1 \wedge f_2) \vee (f_3 \wedge f_4) \vee f_5$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat :

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat :

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3) \oplus (f_5 \otimes f_5)$

Exemple 4 : le semi-anneau universel $\mathbb{N}[X]$

- Considérons à nouveau : $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Mettons des annotations atomiques $X = f_1, \ldots, f_n$ aux faits
- Le semi-anneau le plus général : $\mathbb{N}[X]$, les polynômes en X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Résultat : $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3) \oplus (f_5 \otimes f_5)$

Spécialisation et homomorphismes

- Ces exemples s'expriment par des semi-anneaux commutatifs :
  - semi-anneau de sécurité $(K, \min, \max, \text{Public}, \text{Indisponible})$
  - semi-anneau des multiplicités $(\mathbb{N}, +, \times, 0, 1)$
  - semi-anneau booléen $(\mathrm{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - semi-anneau universel $(\mathbb{N}[X], +, \times, 0, 1)$

Spécialisation et homomorphismes

- Ces exemples s'expriment par des semi-anneaux commutatifs :
  - semi-anneau de sécurité $(K, \min, \max, \text{Public}, \text{Indisponible})$
  - semi-anneau des multiplicités $(\mathbb{N}, +, \times, 0, 1)$
  - semi-anneau booléen $(\mathrm{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - semi-anneau universel $(\mathbb{N}[X], +, \times, 0, 1)$
- $\mathbb{N}[X]$ est le semi-anneau universel :
  - La provenance pour $\mathbb{N}[X]$ se spécialise vers tout $K[X]$
  - Par commutation avec les homomorphismes, les annotations atomiques de X peuvent être remplacées par leur valeur dans K

Spécialisation et homomorphismes

- Ces exemples s'expriment par des semi-anneaux commutatifs :
  - semi-anneau de sécurité $(K, \min, \max, \text{Public}, \text{Indisponible})$
  - semi-anneau des multiplicités $(\mathbb{N}, +, \times, 0, 1)$
  - semi-anneau booléen $(\mathrm{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - semi-anneau universel $(\mathbb{N}[X], +, \times, 0, 1)$
- $\mathbb{N}[X]$ est le semi-anneau universel :
  - La provenance pour $\mathbb{N}[X]$ se spécialise vers tout $K[X]$
  - Par commutation avec les homomorphismes, les annotations atomiques de X peuvent être remplacées par leur valeur dans K

$\rightarrow$ Calculer la provenance $\mathbb{N}[X]$ englobe toutes les tâches

$\rightarrow$ Pour des requêtes conjonctives, c'est faisable en temps polynomial en l'instance d'entrée

Provenance et probabilités

- Évaluation probabiliste de requêtes :
  - Requête conjonctive fixée $q : \exists xyz\ R(x, y) \wedge R(y, z)$
  - Instance d'entrée TID :

R		
a	b	0,6
b	c	0,9

Provenance et probabilités

- Évaluation probabiliste de requêtes :
  - Requête conjonctive fixée $q : \exists xyz\ R(x, y) \wedge R(y, z)$
  - Instance d'entrée TID :

R		
a	b	0,6
b	c	0,9

  → La probabilité de q est $0,6 \times 0,9$

Provenance et probabilités

- Évaluation probabiliste de requêtes :
  - Requête conjonctive fixée $q : \exists xyz\ R(x, y) \wedge R(y, z)$
  - Instance d'entrée TID :

R		
a	b	0,6
b	c	0,9

  → La probabilité de q est $0,6 \times 0,9$
    → C'est la probabilité de la provenance $\mathrm{PosBool}[X]$

Provenance et probabilités

- Évaluation probabiliste de requêtes :
  - Requête conjonctive fixée $q : \exists xyz\ R(x, y) \wedge R(y, z)$
  - Instance d'entrée TID :

R		
a	b	0,6
b	c	0,9

  → La probabilité de q est $0,6 \times 0,9$

    → C'est la probabilité de la provenance $\mathrm{PosBool}[X]$

→ #P-dur en les données (≈ NP-dur pour le comptage)

Arbres et instances quasi-arborescentes

- Idées : les instances d'entrée doivent être quasi-arborescentes
  - Décomposition en arbre d'une instance : couvrir tous les faits

Arbres et instances quasi-arborescentes

- Idées : les instances d'entrée doivent être quasi-arborescentes
  - Décomposition en arbre d'une instance : couvrir tous les faits
  - Largeur d'arbre : largeur minimale d'une décomposition (nombre maximal de sommets énumérés simultanément)

Arbres et instances quasi-arborescentes

- Idées : les instances d'entrée doivent être quasi-arborescentes
  - Décomposition en arbre d'une instance : couvrir tous les faits
  - Largeur d'arbre : largeur minimale d'une décomposition (nombre maximal de sommets énumérés simultanément)
    - Les arbres sont de largeur 1
    - Les cycles sont de largeur 2
    - Les k-cliques et les k-grilles sont de largeur $k-1$

Arbres et instances quasi-arborescentes

- Idées : les instances d'entrée doivent être quasi-arborescentes
  - Décomposition en arbre d'une instance : couvrir tous les faits
  - Largeur d'arbre : largeur minimale d'une décomposition (nombre maximal de sommets énumérés simultanément)
    - Les arbres sont de largeur 1
    - Les cycles sont de largeur 2
    - Les k-cliques et les k-grilles sont de largeur $k-1$
  - Quasi-arborescent : largeur d'arbre bornée par une constante

Énoncé du problème

- De nombreuses tâches sont tractables en les données sur les instances quasi-arborescentes :
  - L'évaluation de requêtes MSO est linéaire [Courcelle, 1990]
  - Le comptage de résultats aussi [Arnborg et al., 1991]
  - L'évaluation probabiliste est linéaire pour les arbres [Cohen et al., 2009]
  - (MSO = premier ordre + quantification sur les ensembles ; couvre l'algèbre relationelle, Datalog monadique, etc.)

Énoncé du problème

- De nombreuses tâches sont tractables en les données sur les instances quasi-arborescentes :
  - L'évaluation de requêtes MSO est linéaire [Courcelle, 1990]
  - Le comptage de résultats aussi [Arnborg et al., 1991]
  - L'évaluation probabiliste est linéaire pour les arbres [Cohen et al., 2009]
  - (MSO = premier ordre + quantification sur les ensembles ; couvre l'algèbre relationelle, Datalog monadique, etc.)

→ Peut-on définir la provenance dans ce contexte ?

Énoncé du problème

- De nombreuses tâches sont tractables en les données sur les instances quasi-arborescentes :
  - L'évaluation de requêtes MSO est linéaire [Courcelle, 1990]
  - Le comptage de résultats aussi [Arnborg et al., 1991]
  - L'évaluation probabiliste est linéaire pour les arbres [Cohen et al., 2009]
  - (MSO = premier ordre + quantification sur les ensembles ; couvre l'algèbre relationelle, Datalog monadique, etc.)

→ Peut-on définir la provenance dans ce contexte ?

→ Peut-on la calculer efficacement ?

Énoncé du problème

- De nombreuses tâches sont tractables en les données sur les instances quasi-arborescentes :
  - L'évaluation de requêtes MSO est linéaire [Courcelle, 1990]
  - Le comptage de résultats aussi [Arnborg et al., 1991]
  - L'évaluation probabiliste est linéaire pour les arbres [Cohen et al., 2009]
  - (MSO = premier ordre + quantification sur les ensembles ; couvre l'algèbre relationelle, Datalog monadique, etc.)

→ Peut-on définir la provenance dans ce contexte ?

→ Peut-on la calculer efficacement ?

→ Peut-on généraliser ces résultats ?

Table des matières

Idée générale

- Provenance $\mathrm{Bool}[X]$ sur arbres / instances quasi-arborescentes
- Le monde des arbres :
  - Requête : MSO sur les arbres
- Le monde des instances quasi-arborescentes :
  - Requête : MSO sur l'instance

  $\rightarrow$ Réductible aux arbres [Courcelle, 1990]

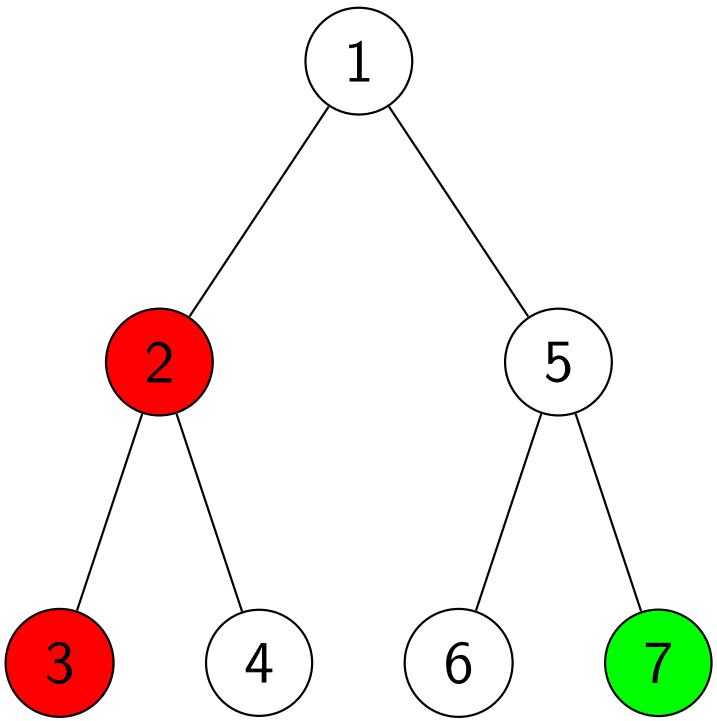
Idée générale

- Provenance $\mathrm{Bool}[X]$ sur arbres / instances quasi-arborescentes
- Le monde des arbres :
  - Requête : MSO sur les arbres
- Le monde des instances quasi-arborescentes :
  - Requête : MSO sur l'instance

  $\rightarrow$ Réductible aux arbres [Courcelle, 1990]

$\rightarrow$ Pour commencer : provenance $\mathrm{Bool}[X]$, requêtes sur les arbres

Arbres incertains



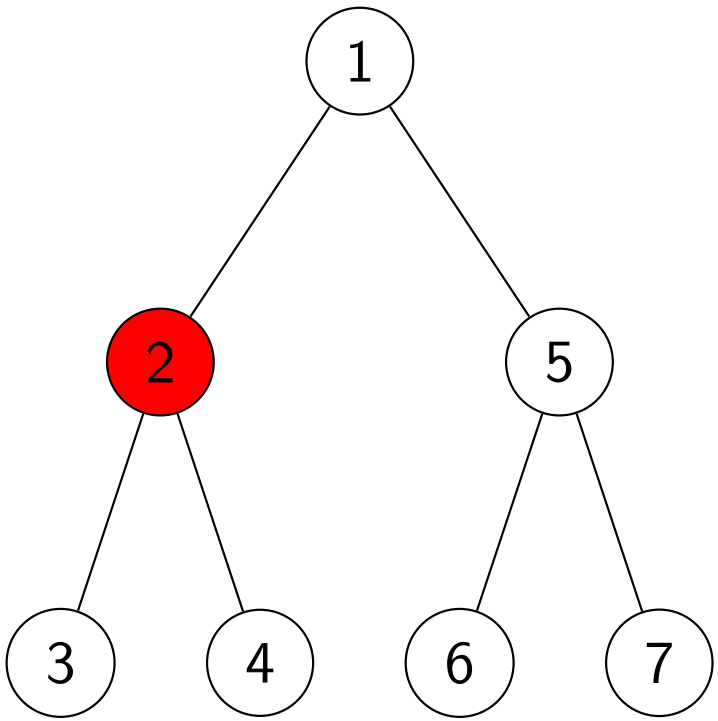
Une valuation d'un arbre dit s'il faut garder ou effacer la couleur des nœuds

Requête d'exemple :
« Y a-t-il un nœud vert et un rouge ? »

Valuation : $\{2, 3, 7\}$

La requête est vraie

Arbres incertains



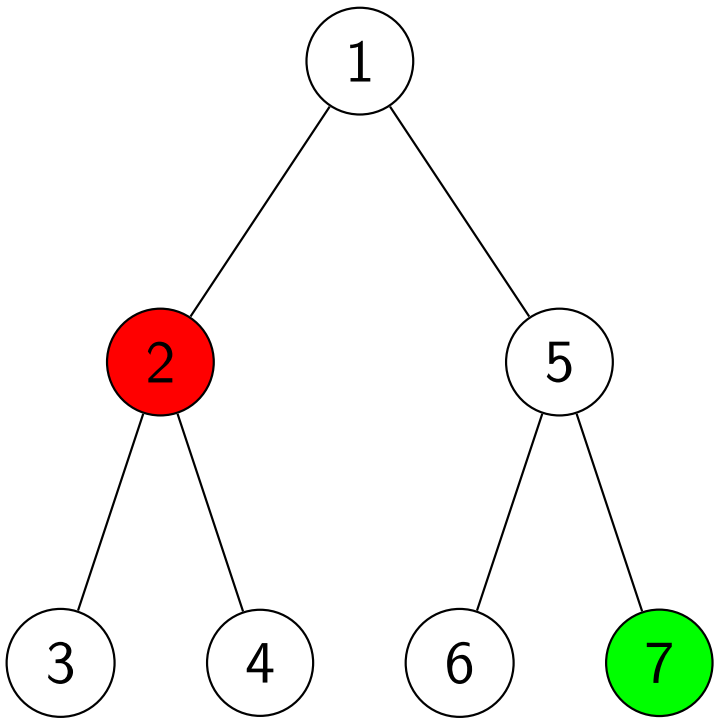
Une valuation d'un arbre dit s'il faut garder ou effacer la couleur des nœuds

Requête d'exemple :
« Y a-t-il un nœud vert et un rouge ? »

Valuation : $\{2\}$

La requête est fausse

Arbres incertains



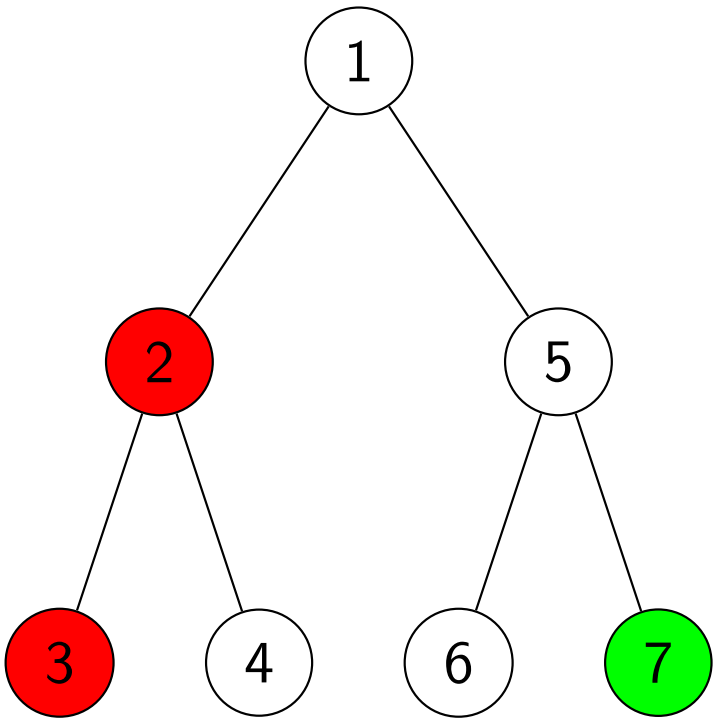
Une valuation d'un arbre dit s'il faut garder ou effacer la couleur des nœuds

Requête d'exemple :
« Y a-t-il un nœud vert et un rouge ? »

Valuation : $\{2, 7\}$

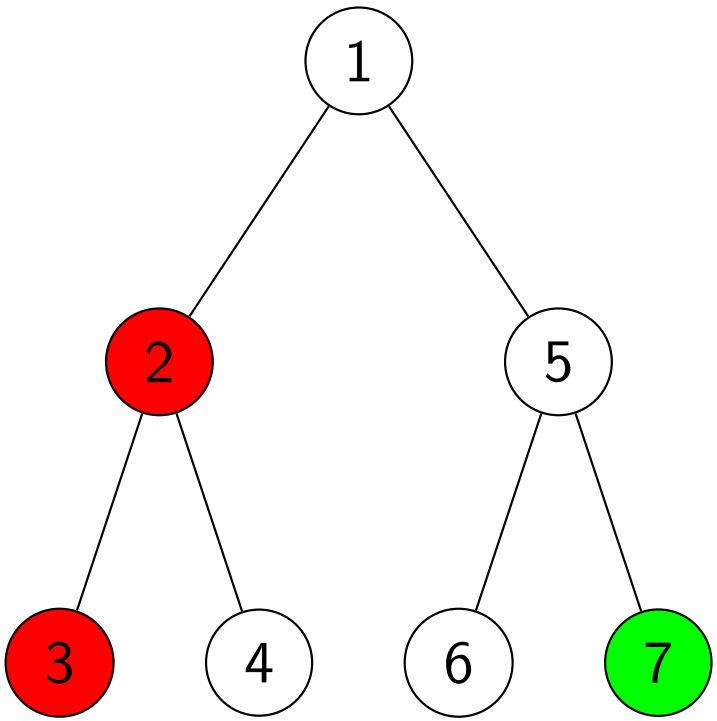
La requête est vraie

Formules et circuits de provenance



- Quelles valuations satisfont la requête q ?

Formules et circuits de provenance



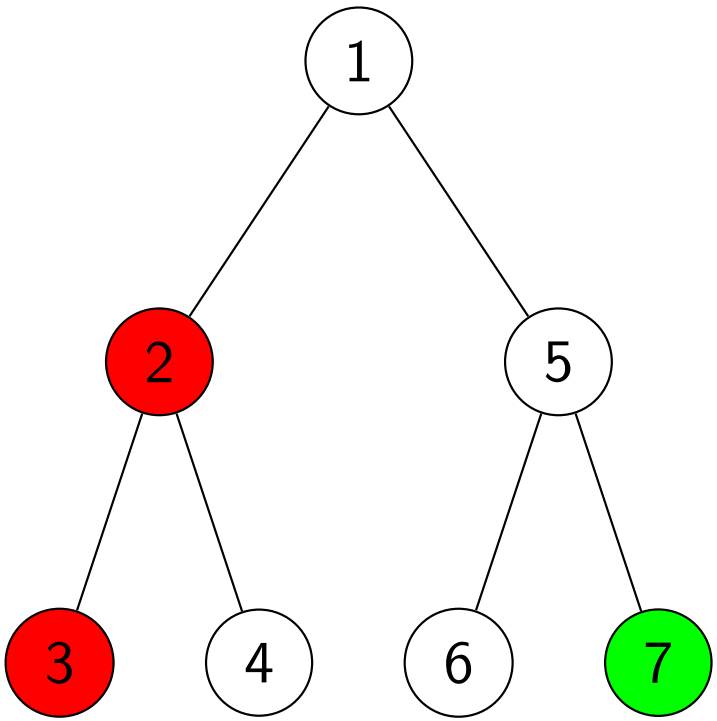
- Quelles valuations satisfont la requête q ?

$\rightarrow$ Formule de provenance de q sur un arbre incertain T :

- Formule booléenne ϕ
- en les variables x_2, x_3, x_7

$\rightarrow$ $\nu(T)$ satisfait q ssi $\nu(\phi)$ est vraie

Formules et circuits de provenance



- Quelles valuations satisfont la requête q ?

$\rightarrow$ Formule de provenance de q sur un arbre incertain T :

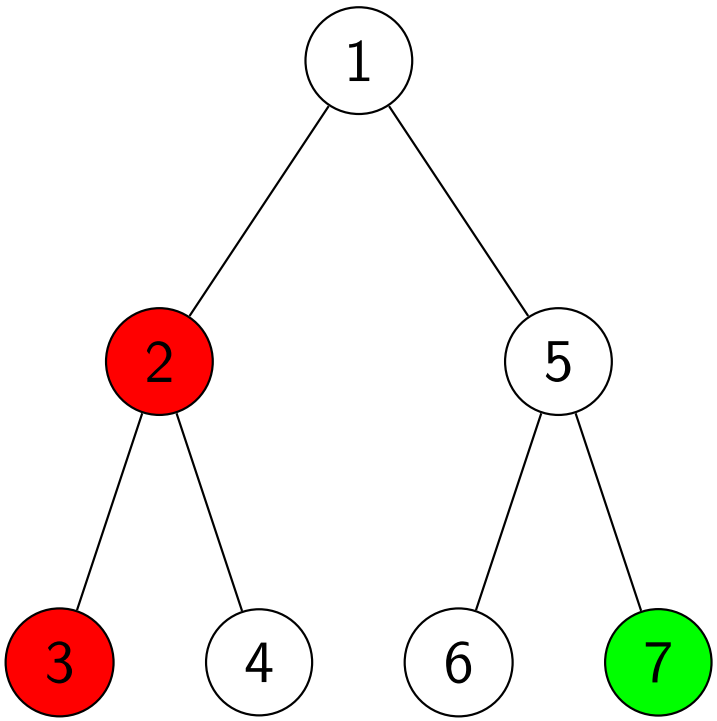
  - Formule booléenne ϕ
  - en les variables x_2, x_3, x_7

  $\rightarrow$ $\nu(T)$ satisfait q ssi $\nu(\phi)$ est vraie

- Circuit de provenance de q sur T [Deutch et al., 2014]
  - Circuit booléen C
  - avec des portes d'entrée g_2, g_3, g_7

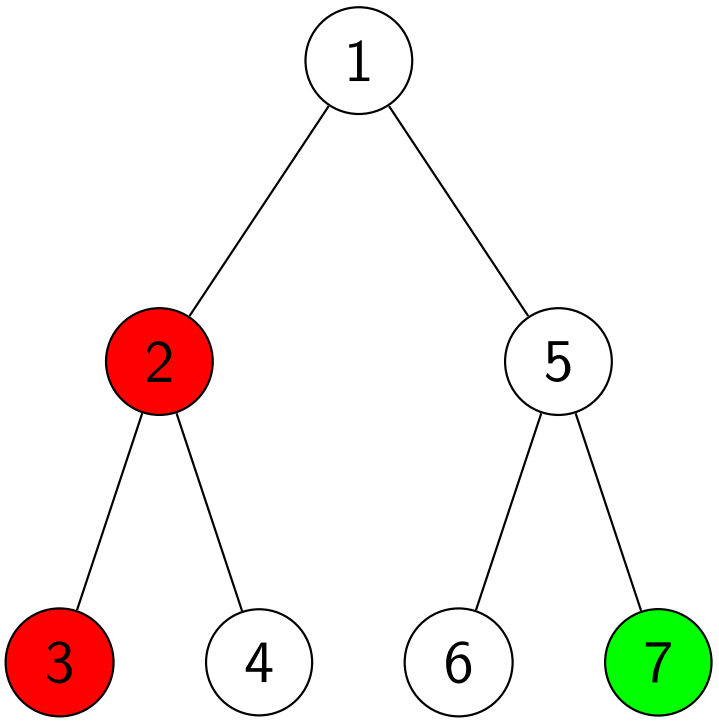
  $\rightarrow$ $\nu(T)$ satisfait q ssi $\nu(C)$ est vraie

Exemple



Y a-t-il un nœud vert et un rouge ?

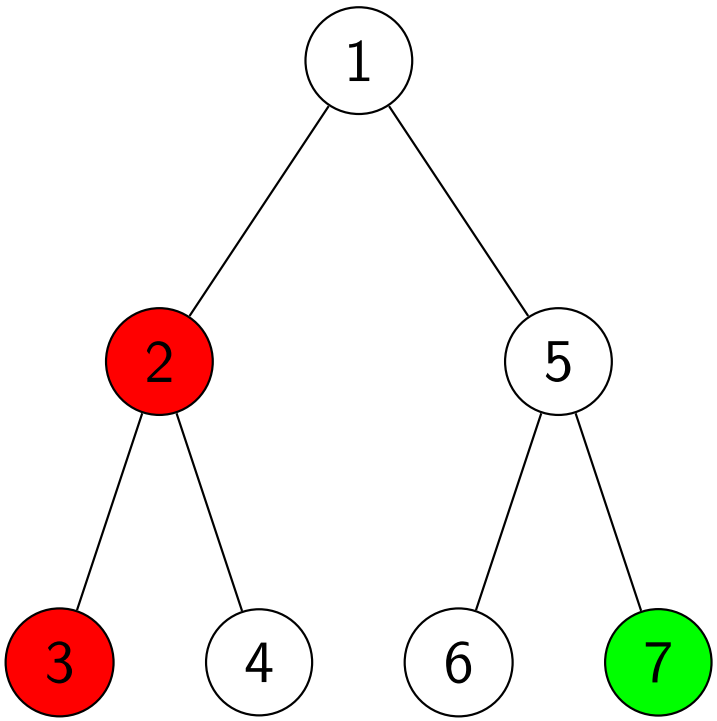
Exemple



Y a-t-il un nœud vert et un rouge ?

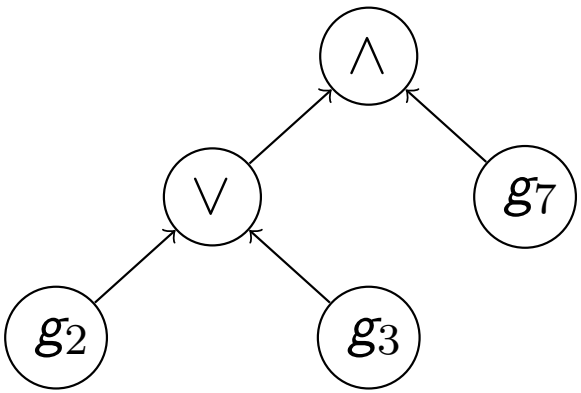
- Formule de provenance :
  $(x_2 \vee x_3) \wedge x_7$

Exemple



Y a-t-il un nœud vert et un rouge ?

- Formule de provenance :
  $(x_2 \vee x_3) \wedge x_7$
- Circuit de provenance :



Notre résultat principal sur les arbres

Théorème

Pour toute requête MSO q fixée,
pour tout arbre T donné en entrée,
on peut construire un circuit de provenance $\mathrm{Bool}[X]$ de q sur T
en temps linéaire en T.

Notre résultat principal sur les arbres

Théorème

Pour toute requête MSO q fixée,
pour tout arbre T donné en entrée,
on peut construire un circuit de provenance $\mathrm{Bool}[X]$ de q sur T
en temps linéaire en T.

$\rightarrow$ Étapes clés :

- Faire de q un automate d'arbres [Thatcher and Wright, 1968]
- Matérialiser ses transitions possibles sur T

Notre résultat principal sur les arbres

Théorème

Pour toute requête MSO q fixée,
pour tout arbre T donné en entrée,
on peut construire un circuit de provenance $\mathrm{Bool}[X]$ de q sur T
en temps linéaire en T.

→ Étapes clés :

- Faire de q un automate d'arbres [Thatcher and Wright, 1968]
- Matérialiser ses transitions possibles sur T

Corollaire

Si les nœuds ont des probabilités d'être gardés indépendamment,
on peut calculer la probabilité de la requête en temps linéaire.

Instances quasi-arborescentes

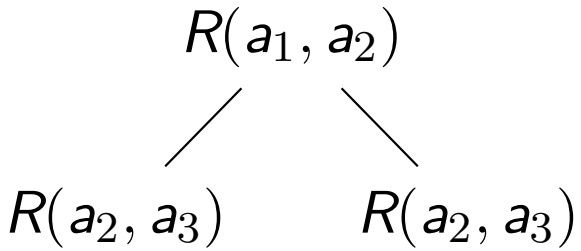
- Encodage en arbre : un arbre E sur un alphabet fini qui représente une instance quasi-arborescente I
- Requête MSO sur $I \rightarrow$ requête MSO sur E

Instances quasi-arborescentes

- Encodage en arbre : un arbre E sur un alphabet fini qui représente une instance quasi-arborescente I
- Requête MSO sur $I \rightarrow$ requête MSO sur E
- Instance incertaine : chaque fait peut être présent ou absent

$\rightarrow$ Chaque $I' \subseteq I$ est une valuation de E

R	
a	b
b	c
b	d

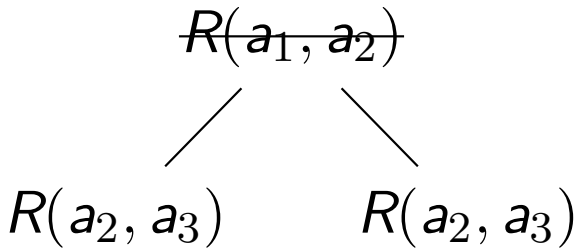


Instances quasi-arborescentes

- Encodage en arbre : un arbre E sur un alphabet fini qui représente une instance quasi-arborescente I
- Requête MSO sur $I \rightarrow$ requête MSO sur E
- Instance incertaine : chaque fait peut être présent ou absent

$\rightarrow$ Chaque $I' \subseteq I$ est une valuation de E

R	
~~a~~	~~b~~
b	c
b	d

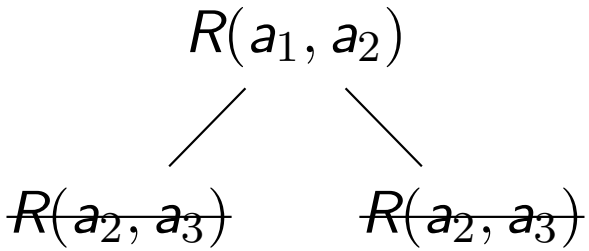


Instances quasi-arborescentes

- Encodage en arbre : un arbre E sur un alphabet fini qui représente une instance quasi-arborescente I
- Requête MSO sur $I \rightarrow$ requête MSO sur E
- Instance incertaine : chaque fait peut être présent ou absent

$\rightarrow$ Chaque $I' \subseteq I$ est une valuation de E

R	
a	b
~~b~~	~~c~~
~~b~~	~~d~~

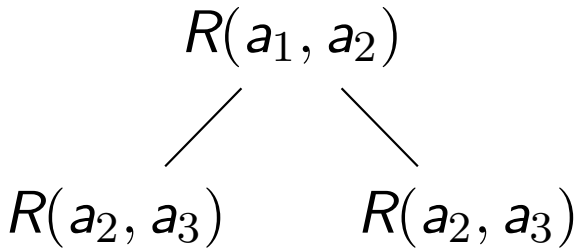


Instances quasi-arborescentes

- Encodage en arbre : un arbre E sur un alphabet fini qui représente une instance quasi-arborescente I
- Requête MSO sur $I \rightarrow$ requête MSO sur E
- Instance incertaine : chaque fait peut être présent ou absent

$\rightarrow$ Chaque $I' \subseteq I$ est une valuation de E

R	
a	b
b	c
b	d



Résultat et conséquences

Théorème

Pour toute requête MSO q fixée et $k \in \mathbb{N}$,
pour toute instance d'entrée I de largeur d'arbre $\leq k$,
on peut construire en temps linéaire
un circuit de provenance $\mathrm{Bool}[X]$ de q sur I.

Résultat et conséquences

Théorème

Pour toute requête MSO q fixée et $k \in \mathbb{N}$,
pour toute instance d'entrée I de largeur d'arbre $\leq k$,
on peut construire en temps linéaire
un circuit de provenance $\mathrm{Bool}[X]$ de q sur I.

Corollaire

Les requêtes MSO ont une complexité linéaire en données sur les instances TID quasi-arborescentes.

Résultat et conséquences

Théorème

Pour toute requête MSO q fixée et $k \in \mathbb{N}$,
pour toute instance d'entrée I de largeur d'arbre $\leq k$,
on peut construire en temps linéaire
un circuit de provenance $\mathrm{Bool}[X]$ de q sur I.

Corollaire

Les requêtes MSO ont une complexité linéaire en données sur les instances TID quasi-arborescentes.

Corollaire

Le comptage pour MSO a une complexité linéaire (déjà connu).

Table des matières

Premier problème : requêtes non-monotones

- On veut généraliser de $\mathrm{Bool}[X]$ à $\mathbb{N}[X]$
- Les semi-anneaux gèrent mal la négation [Amsterdamer et al., 2011]
- Notre construction utilise des portes NOT

Premier problème : requêtes non-monotones

- On veut généraliser de $\mathrm{Bool}[X]$ à $\mathbb{N}[X]$
- Les semi-anneaux gèrent mal la négation [Amsterdamer et al., 2011]
- Notre construction utilise des portes NOT

$\rightarrow$ q est monotone si $I \models q$ implique $I' \models q$ pour tout $I' \supseteq I$

Premier problème : requêtes non-monotones

- On veut généraliser de $\mathrm{Bool}[X]$ à $\mathbb{N}[X]$
- Les semi-anneaux gèrent mal la négation [Amsterdamer et al., 2011]
- Notre construction utilise des portes NOT

→ q est monotone si $I \models q$ implique $I' \models q$ pour tout $I' \supseteq I$

→ Les circuits de provenance pour les requêtes monotones peuvent être monotones

Deuxième problème : définition intrinsèque

- La provenance booléenne a une définition intrinsèque :
  « Caractériser quelles sous-instances satisfont la requête »
  - $\rightarrow$ Indépendant de l'écriture de la requête
  - $\rightarrow$ Indépendant de l'encodage en arbre
- La provenance $\mathbb{N}[X]$ est souvent définie opérationnellement :
  - $\rightarrow$ Dépend de l'écriture de la requête

Deuxième problème : définition intrinsèque

- La provenance booléenne a une définition intrinsèque :
  « Caractériser quelles sous-instances satisfont la requête »
  - $\rightarrow$ Indépendant de l'écriture de la requête
  - $\rightarrow$ Indépendant de l'encodage en arbre
- La provenance $\mathbb{N}[X]$ est souvent définie opérationnellement :
  - $\rightarrow$ Dépend de l'écriture de la requête

$\rightarrow$ On se limite désormais aux unions de requêtes conjonctives

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1)$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1)$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3)$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3)$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$
  c'est-à-dire $x_1^2 + 2x_2x_3$

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$
  c'est-à-dire $x_1^2 + 2x_2x_3$
- Définition :
  - Sommer sur les correspondances
  - Multiplier les faits utilisés

Provenance d'une requête conjonctive sans variables libres

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Requête : $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance :
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$
  c'est-à-dire $x_1^2 + 2x_2x_3$
- Définition :
  - Sommer sur les correspondances
  - Multiplier les faits utilisés

En quoi $\mathbb{N}[X]$ est-il plus expressif que $\mathrm{PosBool}[X]$?

$\rightarrow$ Coefficients : compter les dérivations multiples

$\rightarrow$ Exposants : utiliser les faits plusieurs fois

Notre résultat pour les circuits de provenance $\mathbb{N}[X]$

Théorème

Pour toute union de requêtes conjonctives q fixée et $k \in \mathbb{N}$,
pour toute instance d'entrée I de largeur d'arbre $\leq k$,
on peut construire en temps linéaire
un circuit de provenance $\mathbb{N}[X]$ de q sur I.

Notre résultat pour les circuits de provenance $\mathbb{N}[X]$

Théorème

Pour toute union de requêtes conjonctives q fixée et $k \in \mathbb{N}$,
pour toute instance d'entrée I de largeur d'arbre $\leq k$,
on peut construire en temps linéaire
un circuit de provenance $\mathbb{N}[X]$ de q sur I.

$\rightarrow$ Quels problèmes empêchent de généraliser à MSO/Datalog ?

- Multiplicité maximale non-bornée
- Pas de définition logique de la multiplicité

Table des matières

Résumé

- Résultat :
  - → Calcul linéaire de circuits de provenance sur les arbres et les instances quasi-arborescentes :
    - pour MSO, $\mathrm{Bool}[X]$
    - pour MSO monotone, $\mathrm{PosBool}[X]$
    - pour les unions de requêtes conjonctives, $\mathbb{N}[X]$
  - → plus rapide que le cas général (linéaire et non polynomial)
  - → pas plus coûteux que l'évaluation de requête

Résumé

- Résultat :
  - → Calcul linéaire de circuits de provenance sur les arbres et les instances quasi-arborescentes :
    - pour MSO, $\mathrm{Bool}[X]$
    - pour MSO monotone, $\mathrm{PosBool}[X]$
    - pour les unions de requêtes conjonctives, $\mathbb{N}[X]$
  - → plus rapide que le cas général (linéaire et non polynomial)
  - → pas plus coûteux que l'évaluation de requête
- Techniques :
  - Représentations inhabituelles de la provenance (circuits)
  - Définitions intrinsèques de la provenance
  - Extension de la provenance à MSO (seulement $\mathrm{PosBool}[X]$)

Résumé

- Résultat :
  - → Calcul linéaire de circuits de provenance sur les arbres et les instances quasi-arborescentes :
    - pour MSO, $\mathrm{Bool}[X]$
    - pour MSO monotone, $\mathrm{PosBool}[X]$
    - pour les unions de requêtes conjonctives, $\mathbb{N}[X]$
  - → plus rapide que le cas général (linéaire et non polynomial)
  - → pas plus coûteux que l'évaluation de requête
- Techniques :
  - Représentations inhabituelles de la provenance (circuits)
  - Définitions intrinsèques de la provenance
  - Extension de la provenance à MSO (seulement $\mathrm{PosBool}[X]$)
- Applications :
  - → Capturer des résultats existants (découpler le calcul symbolique et le calcul numérique)
  - → Étendre à de nouvelles applications (probabilités)

Directions futures

- Étendre $\mathbb{N}[X]$ à MSO ? (idées : séries formelles)
- Datalog monadique ? [Gottlob et al., 2010]
- Autres applications ? agrégation, énumération ?
- Faire des expériences sur ces techniques
- Décompositions en arbre spécifiques à une requête ?
- Est-il nécessaire de borner la largeur d'arbre ?
  - → Résultats d'intractabilité sur toute famille constructible si la largeur d'arbre n'est pas bornée, via le théorème du « mineur grille » exclu [Robertson and Seymour, 1986]

Directions futures

- Étendre $\mathbb{N}[X]$ à MSO ? (idées : séries formelles)
- Datalog monadique ? [Gottlob et al., 2010]
- Autres applications ? agrégation, énumération ?
- Faire des expériences sur ces techniques
- Décompositions en arbre spécifiques à une requête ?
- Est-il nécessaire de borner la largeur d'arbre ?
  - → Résultats d'intractabilité sur toute famille constructible si la largeur d'arbre n'est pas bornée, via le théorème du « mineur grille » exclu [Robertson and Seymour, 1986]

Merci pour votre attention !

Bibliographie I


Amsterdamer, Y., Deutch, D., and Tannen, V. (2011).
On the limitations of provenance for queries with difference.
In *TaPP*.


Arnborg, S., Lagergren, J., and Seese, D. (1991).
Easy problems for tree-decomposable graphs.
J. Algorithms, 12(2) :308–340.


Chaudhuri, S. and Vardi, M. Y. (1992).
On the equivalence of recursive and nonrecursive Datalog programs.
In *PODS*.


Cohen, S., Kimelfeld, B., and Sagiv, Y. (2009).
Running tree automata on probabilistic XML.
In *PODS*.

Bibliographie II


Courcelle, B. (1990).
The monadic second-order logic of graphs. I. Recognizable sets of finite graphs.
Inf. Comput., 85(1).


Deutch, D., Milo, T., Roy, S., and Tannen, V. (2014).
Circuits for datalog provenance.
In *ICDT*.


Gottlob, G., Pichler, R., and Wei, F. (2010).
Monadic datalog over finite structures of bounded treewidth.
TOCL, 12(1) :3.


Green, T. J., Karvounarakis, G., and Tannen, V. (2007).
Provenance semirings.
In *PODS*.

Bibliographie III


Robertson, N. and Seymour, P. D. (1986).
Graph minors. v. excluding a planar graph.
Journal of Combinatorial Theory, Series B, 41(1) :92–114.


Thatcher, J. W. and Wright, J. B. (1968).
Generalized finite automata theory with an application to a decision problem of second-order logic.
Mathematical systems theory, 2(1) :57–81.

Semiring provenance [Green et al., 2007]

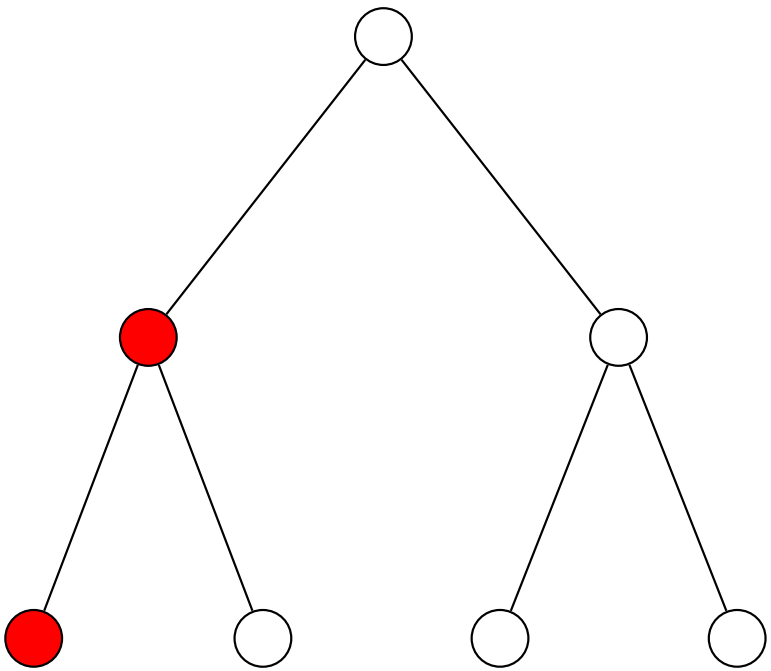
- Semiring $(K, \oplus, \otimes, 0, 1)$
  - $(K, \oplus)$ commutative monoid with identity 0
  - $(K, \otimes)$ commutative monoid with identity 1
  - $\otimes$ distributes over $\oplus$
  - 0 absorptive for $\otimes$

Semiring provenance [Green et al., 2007]

- Semiring $(K, \oplus, \otimes, 0, 1)$
  - $(K, \oplus)$ commutative monoid with identity 0
  - $(K, \otimes)$ commutative monoid with identity 1
  - $\otimes$ distributes over $\oplus$
  - 0 absorptive for $\otimes$
- Idea : Maintain annotations on tuples while evaluating :
  - Union : annotation is the sum of union tuples
  - Select : select as usual
  - Project : annotation is the sum of projected tuples
  - Product : annotation is the product

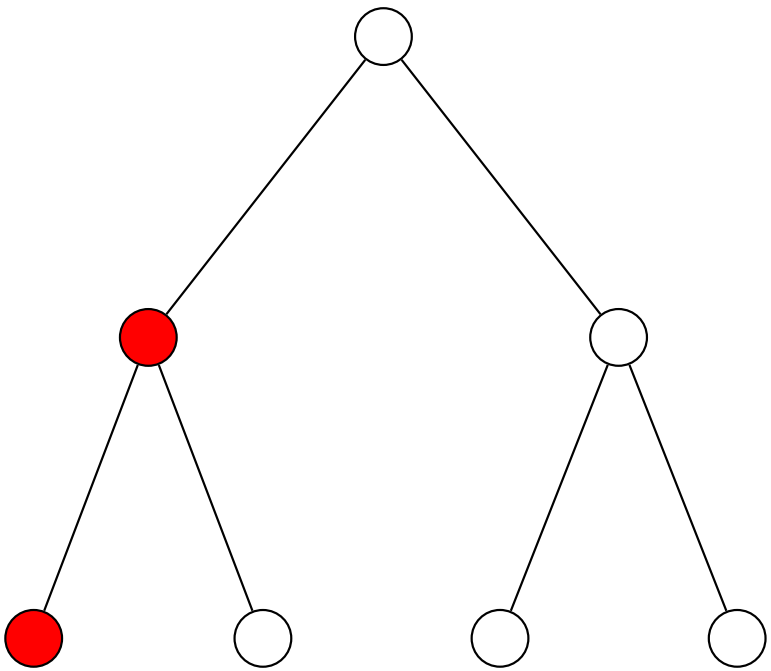
Tree automata

Tree alphabet : 



Tree automata

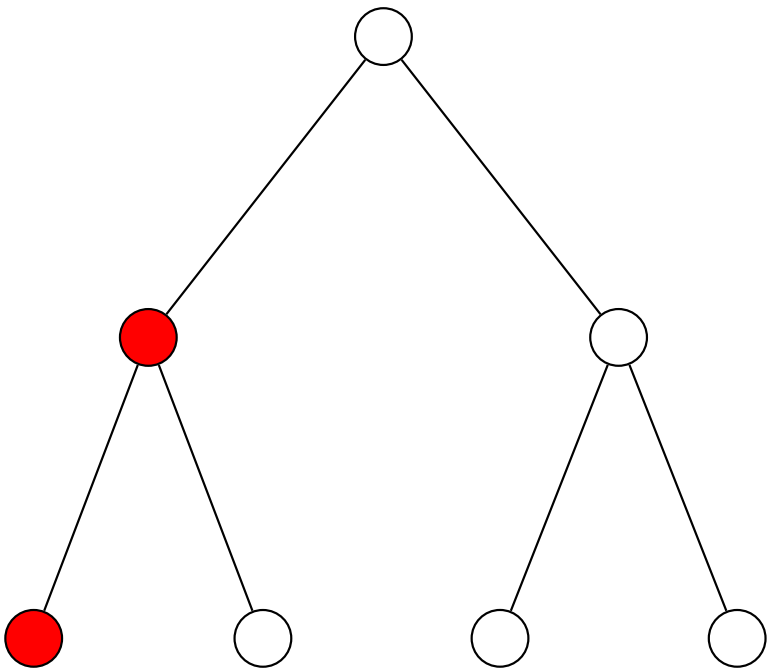
Tree alphabet : 



- bNTA : bottom-up nondeterministic tree automaton
- “Is there both a red and green node ?”

Tree automata

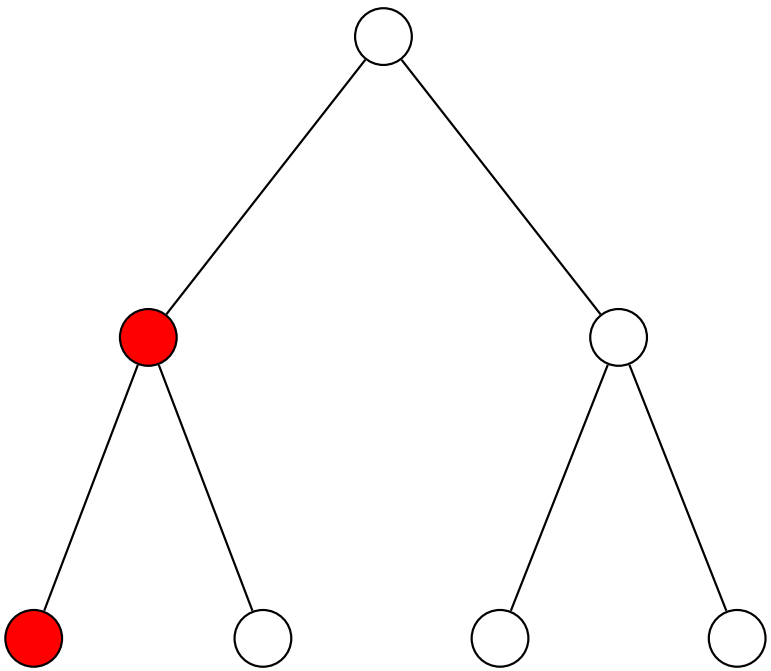
Tree alphabet : 



- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$

Tree automata

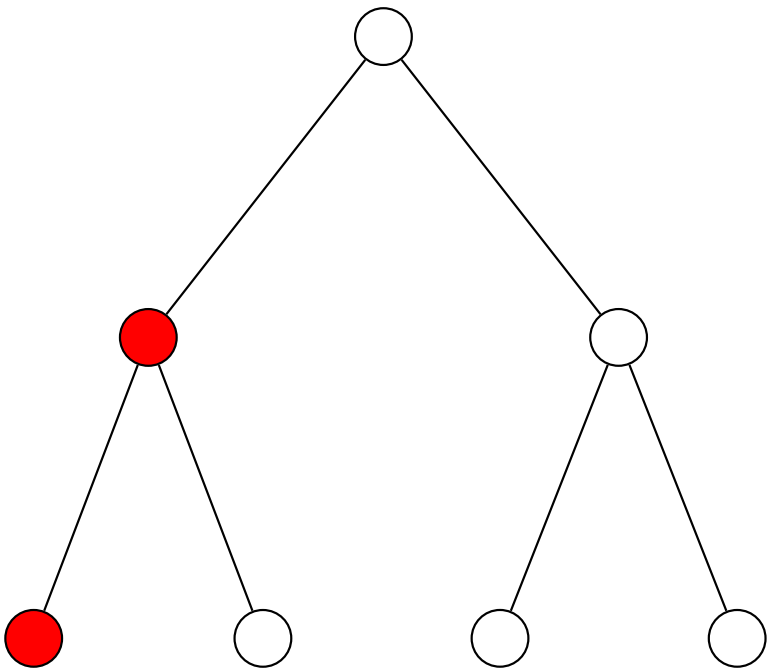
Tree alphabet : 



- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$
- Final states : $\{\top\}$

Tree automata

Tree alphabet : ○ ● ●

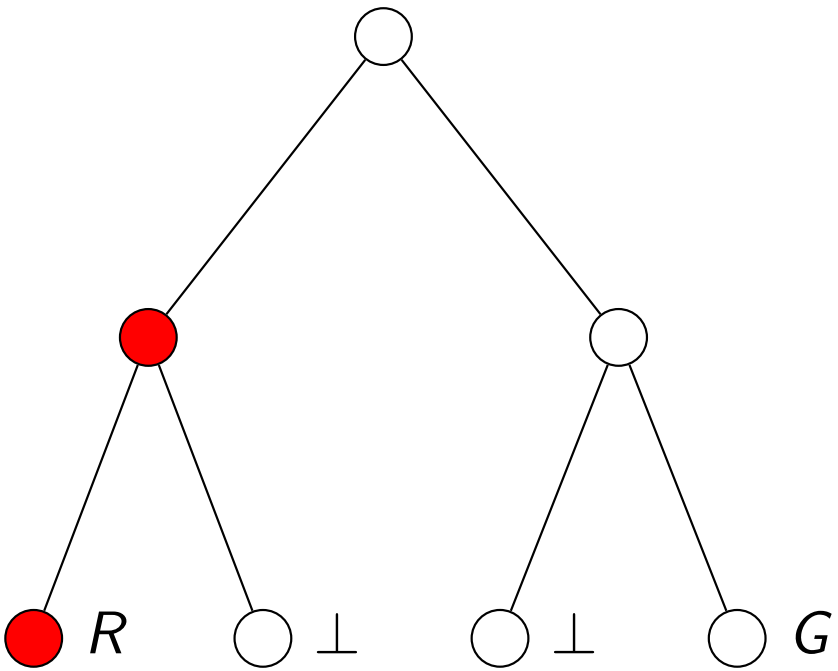


- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$
- Final states : $\{\top\}$
- Initial function :

○ $\bot$ ● R ● G

Tree automata

Tree alphabet : ○ ● ●

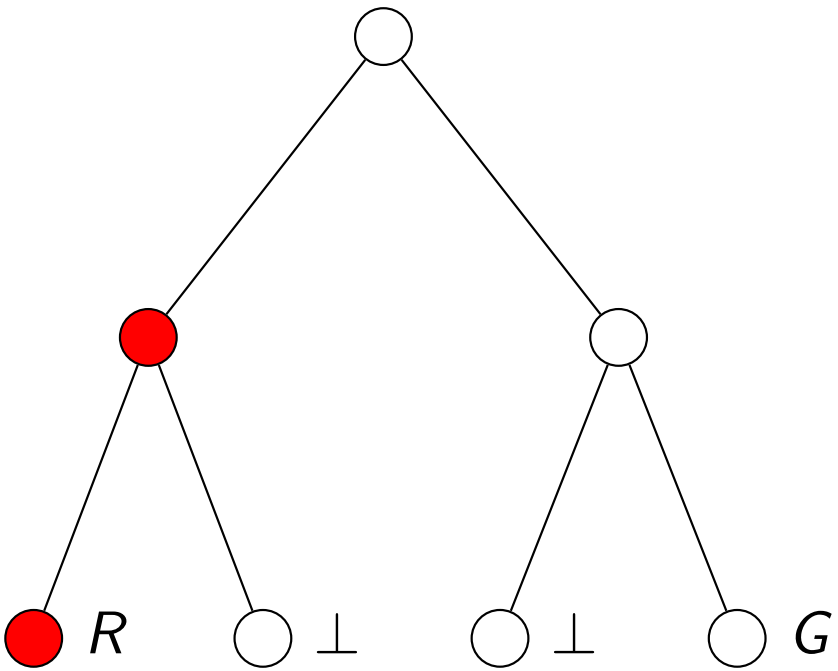


- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$
- Final states : $\{\top\}$
- Initial function :

○ $\bot$ ● R ● G

Tree automata

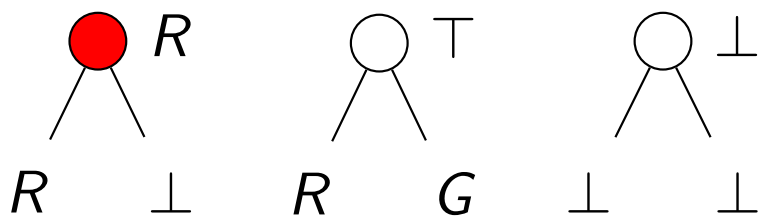
Tree alphabet : ○ ● ●



- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$
- Final states : $\{\top\}$
- Initial function :

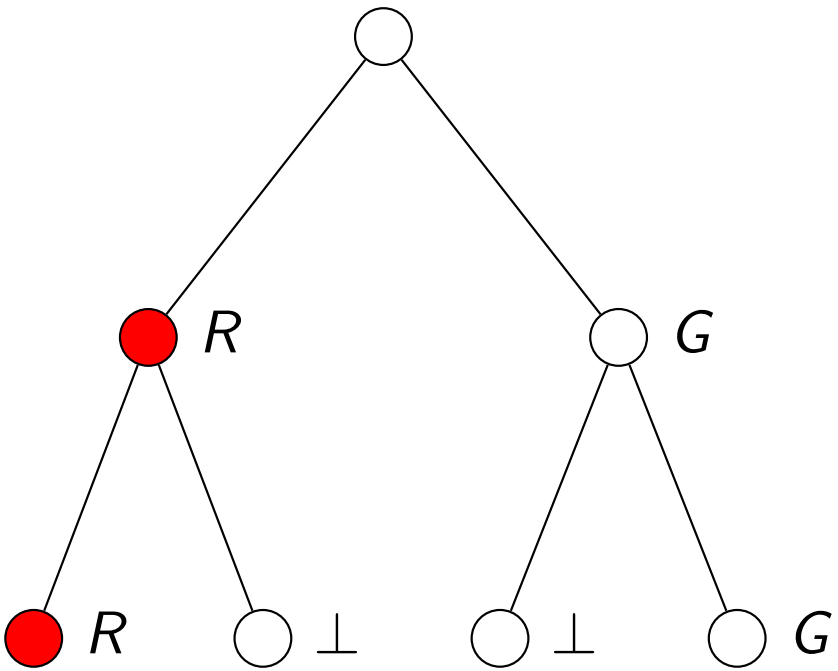
○ $\bot$ ● R ● G

- Transitions (examples) :



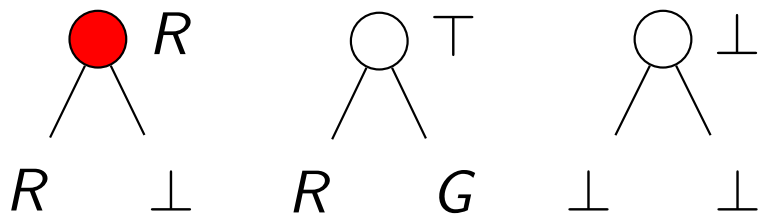
Tree automata

Tree alphabet : 



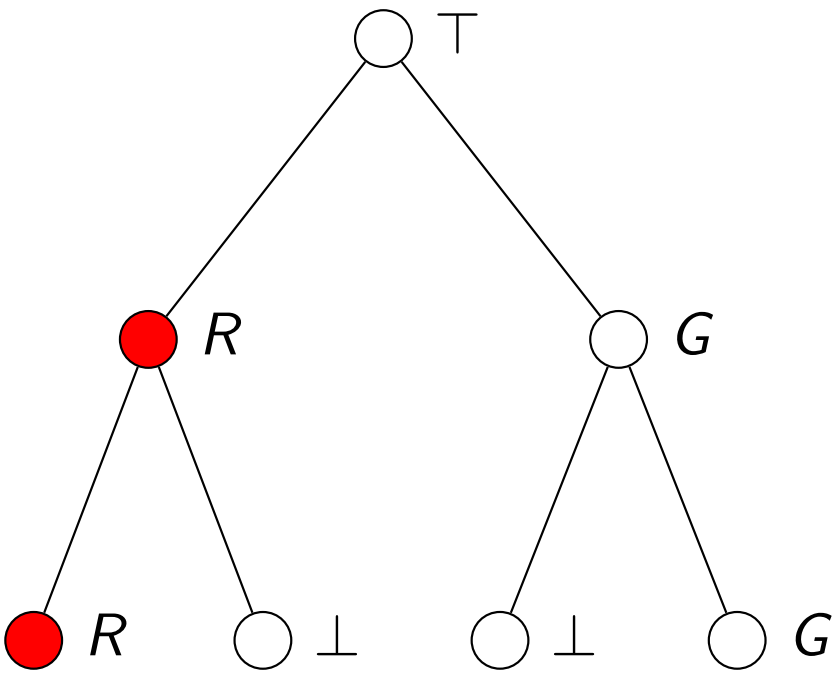
- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$
- Final states : $\{\top\}$
- Initial function :

- Transitions (examples) :



Tree automata

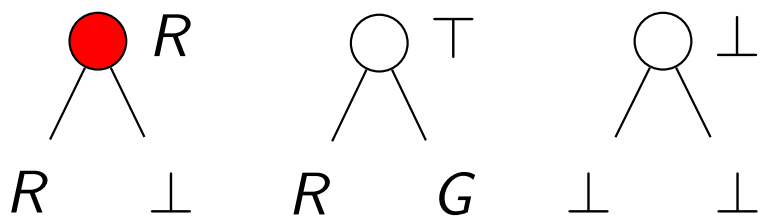
Tree alphabet : ○ ● ●



- bNTA : bottom-up nondeterministic tree automaton
- "Is there both a red and green node ?"
- States : $\{\bot, G, R, \top\}$
- Final states : $\{\top\}$
- Initial function :

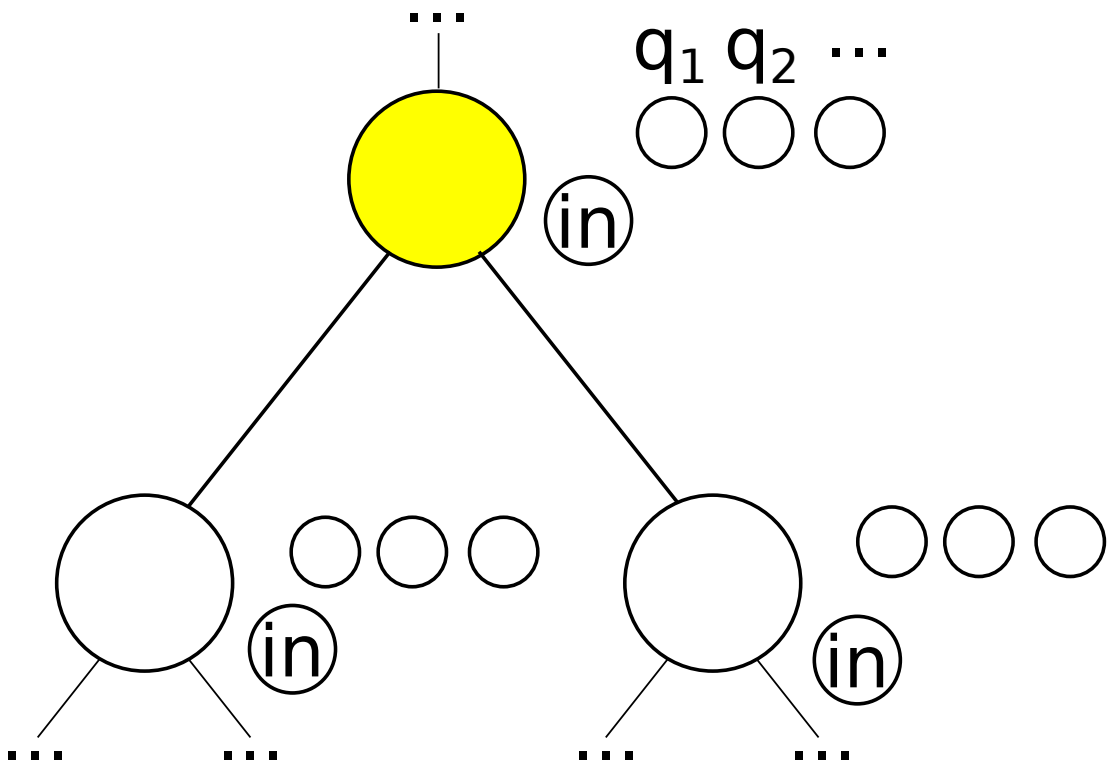
○ $\bot$ ● R ● G

- Transitions (examples) :



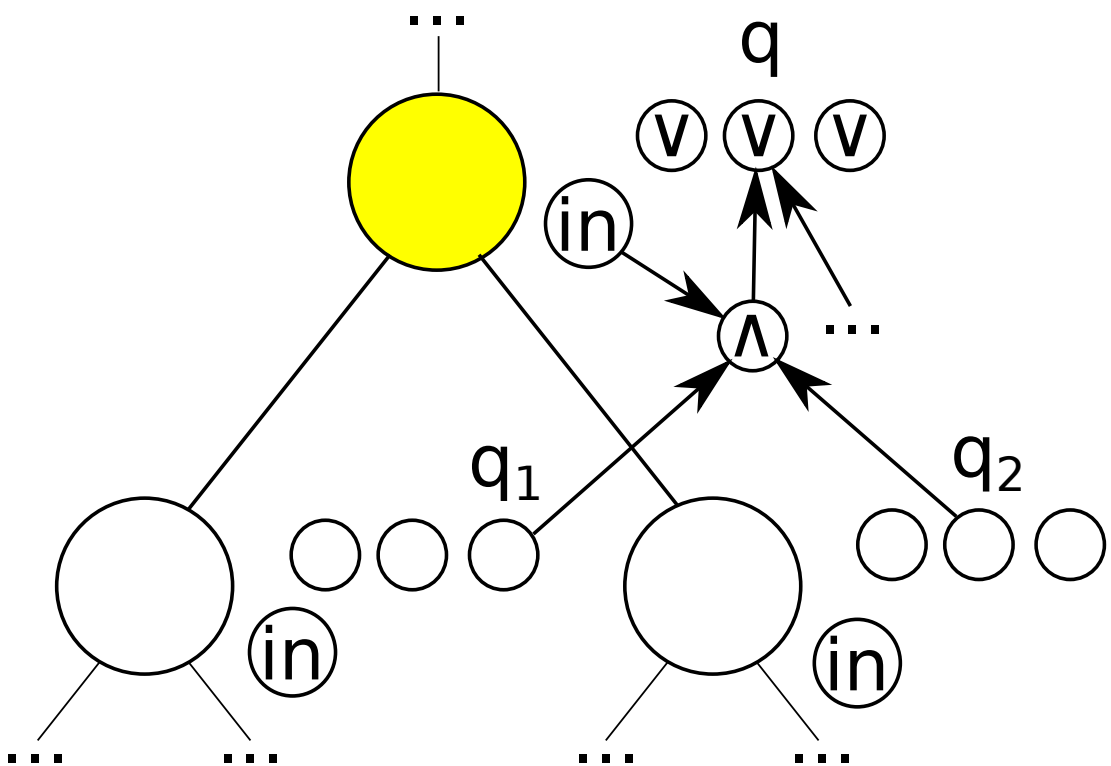
Constructing the provenance circuit

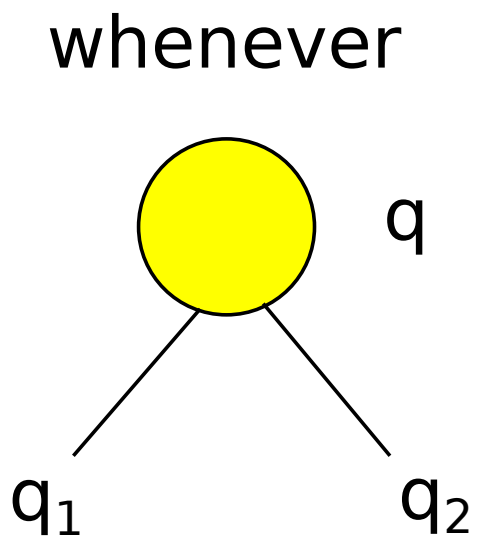
→ Construct a Boolean provenance circuit bottom-up



Constructing the provenance circuit

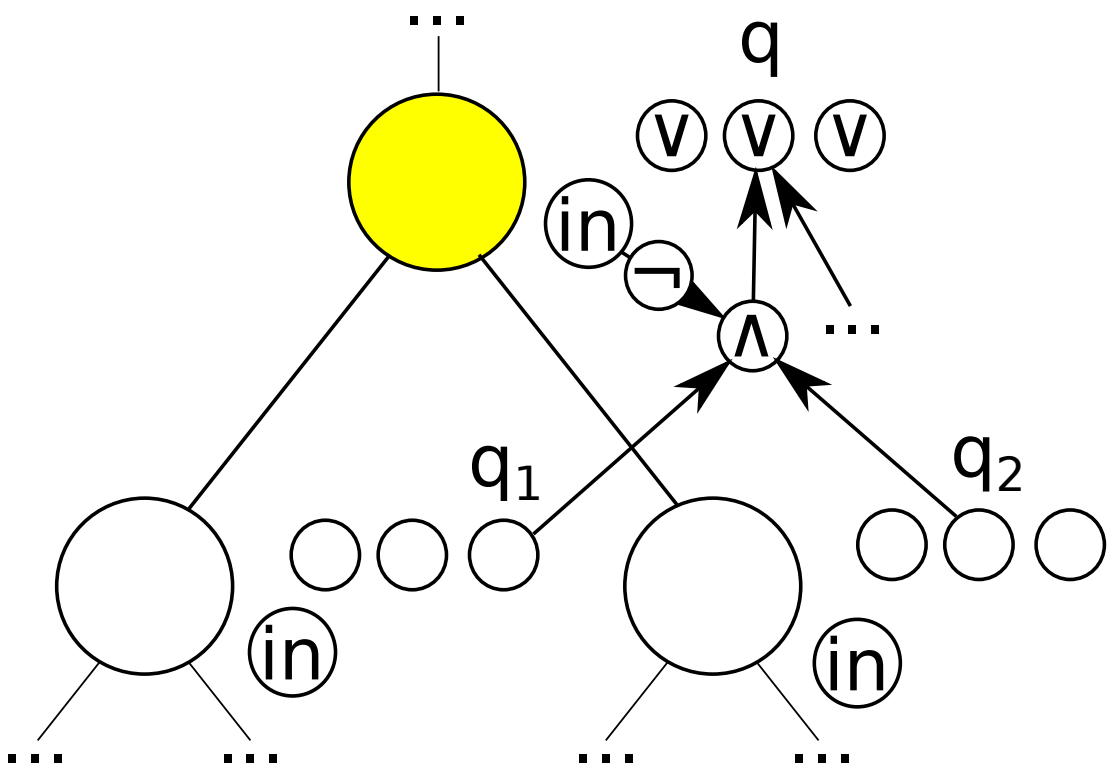
→ Construct a Boolean provenance circuit bottom-up

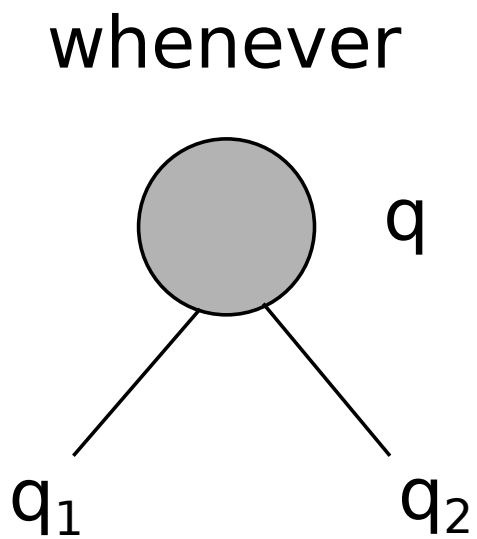




Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up





Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance :

N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

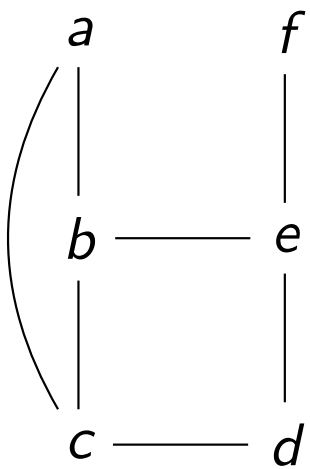
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance :

Gaifman graph :

N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e



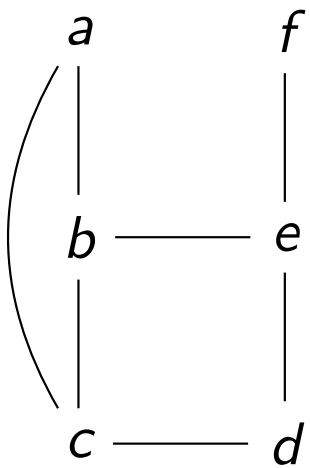
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance :

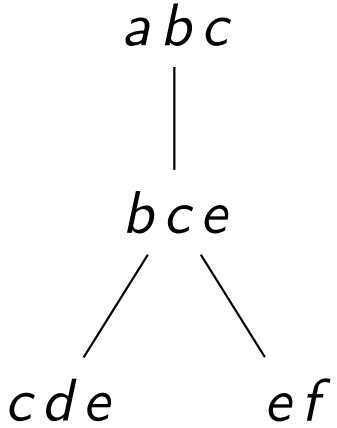
N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

Gaifman graph :



Tree decomp. :



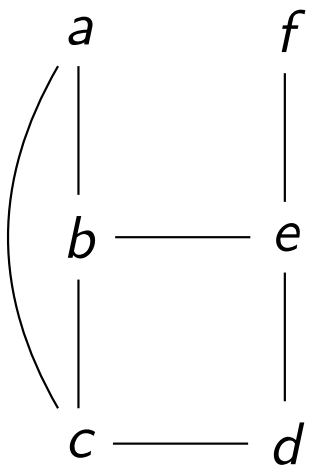
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance :

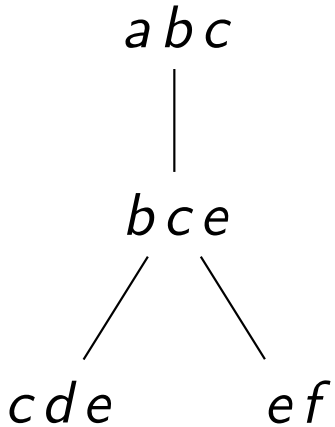
N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

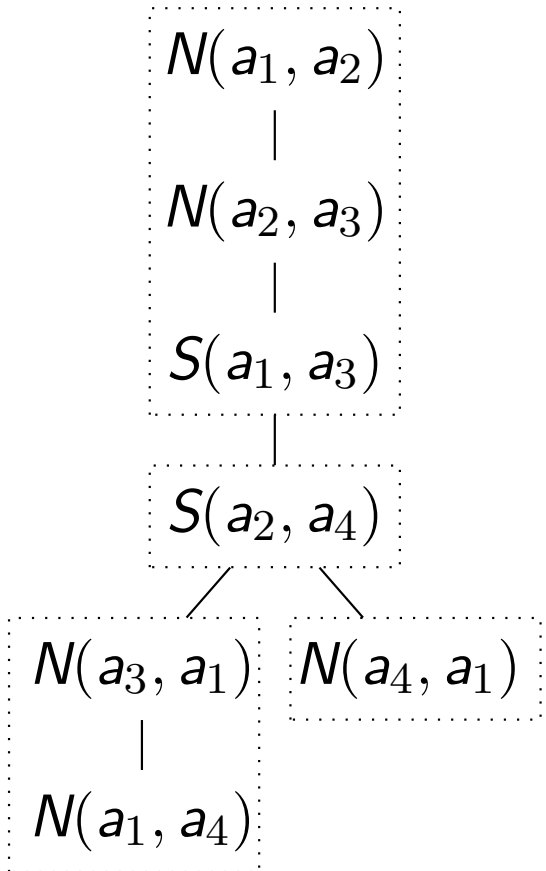
Gaifman graph :



Tree decomp. :



Tree encoding :



Example : block-independent disjoint (BID) instances

name	**city**	**iso**	p
pods	melbourne	au	0.8
pods	sydney	au	0.2
icalp	tokyo	jp	0.1
icalp	kyoto	jp	0.9

Example : block-independent disjoint (BID) instances

name	**city**	**iso**	p
pods	melbourne	au	0.8
pods	sydney	au	0.2
icalp	tokyo	jp	0.1
icalp	kyoto	jp	0.9

- Evaluating a fixed CQ is #P-hard in general

Example : block-independent disjoint (BID) instances

name	**city**	**iso**	p
pods	melbourne	au	0.8
pods	sydney	au	0.2
icalp	tokyo	jp	0.1
icalp	kyoto	jp	0.9

- Evaluating a fixed CQ is #P-hard in general

→ For a treelike instance, linear time !

Supporting coefficients

- In the world of trees
  - The same valuation can be accepted multiple times

  $\rightarrow$ Number of accepting runs of the bNTA
- In the world of treelike instances
  - The same match can be the image of multiple homomorphisms

Supporting coefficients

- In the world of trees
  - The same valuation can be accepted multiple times

  → Number of accepting runs of the bNTA
- In the world of treelike instances
  - The same match can be the image of multiple homomorphisms

→ Add assignment facts to represent possible assignments

→ Encode to a bNTA that guesses them

Supporting exponents

- In the world of trees
  - The same fact can be used multiple times
  - Annotate nodes with a multiplicity
  - The bNTA is monotone for that multiplicity
  - Use each input gate as many times as we read its fact
- In the world of treelike instances
  - The same fact can be the image of multiple atoms
  - Maximal multiplicity is query-dependent but instance-independent

Supporting exponents

- In the world of trees
  - The same fact can be used multiple times
  - Annotate nodes with a multiplicity
  - The bNTA is monotone for that multiplicity
  - Use each input gate as many times as we read its fact
- In the world of treelike instances
  - The same fact can be the image of multiple atoms
  - Maximal multiplicity is query-dependent but instance-independent

→ Encodes CQs to bNTAs that read multiplicities

- Consider all possible CQ self-homomorphisms
- Count the multiplicities of identical atoms
- Rewrite relations to add multiplicities
- Usual compilation on the modified signature