Université Paris Cité

FACULTÉ DES SCIENCES
Directrice : Carole DELPORTE

Rapport de stage de Master 2

COMPATIBILITÉ DE LA SÉMANTIQUE BINDING TRAIL AVEC GQL

Tuteur pédagogique : Cristina SIRANGELO
Tuteurs de recherche : Nadime FRANCIS & Victor MARSAULT

Étudiant : Steven SAILLY

ANNÉE ACADÉMIQUE 2023/2024

Table des matières

1 Introduction **3**

2 Préliminaires **4**
2.1 Langages, expressions régulières 4
2.2 Graphes 4
2.3 Bases de données 5
2.4 GQL 6
2.5 Complexité 11

3 Problèmes computationnels **12**
3.1 Sémantiques 12
3.2 Problèmes de décision 13

4 Résultats négatifs **14**
4.1 Répétition fixe 14
4.2 Variable avec répétitions imbriquées 16
4.3 Variable sans répétitions imbriquées 17
4.4 Variable hors des répétitions 21

5 Résultats positifs **23**

6 Conclusion **23**

A Vue d'ensemble des gadgets du Théorème 37 **25**

B Variantes du Théorème 37 **26**
B.1 Avec unions imbriquées 26
B.2 Avec unions non imbriquées 27

Table des figures

1 Un exemple introductif 3
2 Base de données de l'Exemple 20 9
3 Base de données de l'Exemple 25 11
4 Base de données utilisée comme intuition du Théorème 34 15
5 Gadgets du Théorème 37 18
6 Base de données utilisée dans la preuve du Théorème 39 21
7 Vue d'ensemble des gadgets du Théorème 37 25
8 Gadgets de la réduction avec une requête contenant des disjonctions imbriquées 27
9 Gadgets de la réduction avec une requête contenant un niveau de disjonction 28

1 Introduction

Ce rapport de stage de recherche de Master 2 Informatique fondamentale et appliquée de l'université Paris Cité traite de la compatibilité de la sémantique Binding Trail avec GQL. Le stage s'est déroulé à l'université Gustave Eiffel, au sein du Laboratoire d'Informatique Gaspard Monge, encadré par Nadime Francis et Victor Marsault.

Nos objets d'étude sont les graphes à propriétés et le langage GQL. Un graphe à propriétés est un multigraphe étiqueté où les nœuds et les arêtes sont associés à des couples clé-valeur, appelés *propriétés*. Le langage de requête GQL est un nouveau standard publié en avril 2024 par l'ISO, développé pour interroger des graphes à propriétés [1]. Les réponses aux requêtes de GQL sont des chemins dans des graphes. On s'intéressera plus particulièrement à des *fragments* de GQL. Supposons qu'on souhaite interroger le graphe à propriétés G décrit par la Figure 1.

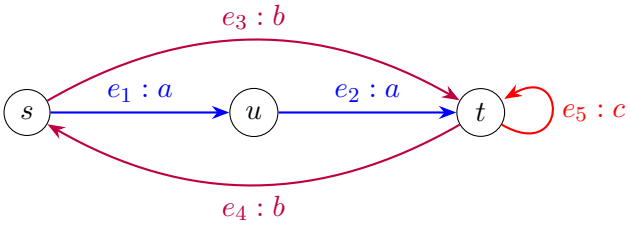


FIGURE 1 – Un exemple introductif

On écrit $i : l$ pour indiquer que l'étiquette de l'arête i est l. Par exemple, dans G, l'arête e_4 qui va de t à s est étiquetée par b. On veut connaître les chemins qui vont de s à t parmi les résultats de la requête $Q = [\xrightarrow{:a}|\xrightarrow{:b}]^* \xrightarrow{:c} [\xrightarrow{:a}|\xrightarrow{:b}]^*$. Intuitivement, Q renvoie les chemins qui commencent avec un nombre quelconque, éventuellement nul, d'arêtes étiquetées par a ou par b, avec une arête étiquetée par c, et qui terminent avec un nombre quelconque, éventuellement nul, d'arêtes étiquetées par a ou par b.

Il y a un nombre infini de chemins de G qui satisfont Q. En effet, on peut prendre la boucle $s \xrightarrow{e_3} t \xrightarrow{e_4} s$ autant de fois qu'on veut. Pour éviter d'avoir une réponse infinie à une requête, on restreint l'ensemble des réponses avec des *sémantiques*. Parmi les plus classiques, on peut citer Shortest, qui ne renvoie que les plus courts chemins, et Trail, qui renvoie les chemins sans répétition d'arête. Si on interroge G avec la requête Q pour trouver les chemins de s à t en utilisant ces sémantiques, on obtient les chemins :

- pour Shortest : $\pi_1 = s \xrightarrow{e_3} t \xrightarrow{e_5} t$, l'unique résultat
- pour Trail :
  - π_1,
  - $\pi_2 = s \xrightarrow{e_1} u \xrightarrow{e_2} t \xrightarrow{e_5} t$,
  - $\pi_3 = s \xrightarrow{e_1} u \xrightarrow{e_2} t \xrightarrow{e_5} t \xrightarrow{e_4} s \xrightarrow{e_3} t$,
  - $\pi_4 = s \xrightarrow{e_1} u \xrightarrow{e_2} t \xrightarrow{e_4} s \xrightarrow{e_3} t \xrightarrow{e_5} t$,
  - $\pi_5 = s \xrightarrow{e_3} t \xrightarrow{e_4} s \xrightarrow{e_1} u \xrightarrow{e_2} t \xrightarrow{e_5} t$,
  - $\pi_6 = s \xrightarrow{e_3} t \xrightarrow{e_5} t \xrightarrow{e_4} s \xrightarrow{e_1} u \xrightarrow{e_2} t$.

On remarque que la sortie de Q sous la sémantique Shortest est courte, et son calcul est facile puisque cela revient à appliquer un algorithme de plus court chemin. À l'opposé, la sémantique Trail permet d'avoir plus de résultats, mais calculer un résultat sous cette sémantique est **NP**-complet [2]. Ainsi, choisir une sémantique trouver un compromis entre *expressivité* et *complexité* d'évaluation.

Une nouvelle sémantique, Binding Trail, a été décrite dans [3]. Intuitivement, les chemins renvoyés par une requête en utilisant cette sémantique peuvent utiliser deux fois la même arête uniquement si ce n'est pas le même atome de la requête qui a permis de l'utiliser. Si on interroge G avec la requête Q pour trouver les chemins de s à t en utilisant la sémantique Binding Trail, on obtient par exemple le chemin : $\pi_7 = s \xrightarrow{e_3} t \xrightarrow{e_5} t \xrightarrow{e_4} s \xrightarrow{e_3} t$. En effet, la première occurrence de e_3 dans π_7 correspond à la première occurrence de $\xrightarrow{:b}$ dans Q, et la seconde occurrence de e_3 dans π_7 correspond à la seconde occurrence de $\xrightarrow{:b}$ dans Q.

Dans [3], la sémantique est étudiée pour les requêtes qui sont des *regular path queries* (RPQs) [4], une RPQ étant exprimée sous la forme d'une expression régulière. Dans ce cadre théorique, Binding Trail a de bonnes propriétés. En effet, calculer un résultat à une requête est un problème **NL**-complet, et énumérer tous les résultats est possible avec un délai polynomial. On veut savoir si Binding Trail conserve ses bonnes propriétés dans GQL.

Dans la Section 2 on donne les préliminaires nécessaires et on définit GPC, qui est l'abstraction de GQL qu'on considère, et ses fragments considérés. Dans la Section 3 on définit les problèmes de décision qui nous intéressent ainsi que la sémantique Binding Trail étendue à GQL. On montre dans la Section 4 que le problème est difficile même dans des fragments restreints de GQL, et dans la Section 5 on identifie un fragment pour lequel le problème est tractable.

2 Préliminaires

2.1 Langages, expressions régulières

Une expression régulière R sur un alphabet Σ s'obtient à partir de la grammaire suivante :

$$R ::= \varepsilon \mid a \mid R^* \mid R \cdot R \mid R|R \text{ (avec } a \in \Sigma)$$

On note $L(R) \subseteq \Sigma^*$ l'ensemble des mots décrits par R, défini de façon usuelle.

Une *regular path query* (RPQ) est exprimée sous la forme d'une expression régulière [4].

2.2 Graphes

Un multigraphe mixte, ou *graphe* dans la suite, est un triplet $G = (V, E, \phi)$ avec :

— V un ensemble de nœuds,

— E un ensemble d'arêtes,

— ϕ une fonction qui, à toute arête $e \in E$, associe soit un couple (u, v), soit un ensemble $\{u, v\}$ avec $u, v \in V$; si $\phi(e) = (u, v)$, alors e est une arête orientée de u à v, sinon c'est une arête non orientée.

Une *walk* w dans un graphe est une suite finie non vide de nœuds et d'arêtes alternés, commençant et finissant par un nœud, de la forme $w = (n_0, e_1, n_2, \ldots, e_{k-1}, n_k)$ avec $n_0, \ldots, n_k \in V$, $e_1, \ldots, e_{k-1} \in E$, telle que pour tout $1 \leq i < k$ impair, $\phi(e_i) = (n_i, n_{i+1})$, $\phi(e_i) = (n_{i+1}, n_i)$, ou $\phi(e_i) = \{n_i, n_{i+1}\}$.
On note $\text{len}(w) = k + 1$ la *longueur* de w, $w[i]$ le $i^{\text{ème}}$ élément de w, et $\text{endpoints}(w) = (w[0], w[k])$.

Remarque 1 (Longueur)**.** On n'utilise pas la définition usuelle de la longueur : la longueur d'une walk n'est pas son nombre d'arêtes, mais la somme de son nombre d'arêtes et de son nombre de sommets.

2.3 Bases de données

Dans la suite, on suppose l'existence d'un ensemble infini dénombrable $\mathcal{K}$ de clés, et d'un ensemble infini dénombrable Const de constantes.

Définition 2. Un graphe à propriétés, ou *base de données* dans la suite, est un tuple $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ avec :

- Σ un ensemble fini de symboles, ou *étiquettes*,
- V un ensemble fini d'identifiants de nœuds (plus simplement *nœuds*),
- E_d un ensemble fini d'identifiants d'arêtes orientées (plus simplement *arêtes orientées*),
- E_u un ensemble fini d'identifiants d'arêtes non orientées (plus simplement *arêtes non orientées*),
- $\text{src} : E_d \longrightarrow V$ qui associe à chaque arête orientée son nœud source,
- $\text{tgt} : E_d \longrightarrow V$ qui associe à chaque arête orientée son nœud cible,
- $\text{endpoints} : E_u \longrightarrow 2^V$, avec pour tout $e \in E_u$ $|endpoints(e)| \in \{1, 2\}$, qui associe à chaque arête non orientée ses extrémités,
- $\lambda : V \cup E_d \cup E_u \longrightarrow 2^\Sigma$ la fonction qui associe à chaque nœud et à chaque arête un ensemble (éventuellement vide) d'étiquettes,
- $\delta : (V \cup E_d \cup E_u) \times \mathcal{K} \longrightarrow$ Const la fonction partielle qui associe à chaque couple (nœud ou arête, clé) une constante. Si $\delta(c, k)$ n'est pas défini, on dira que c n'a pas la propriété k. Pour tout c ayant au moins une propriété, l'ensemble de ses propriétés est fini.

Définition 3 (Concaténation d'étiquettes)**.** Soit e, e' deux arêtes. $\lambda(e)$ et $\lambda(e')$ sont des langages, et leur concaténation est définie de façon usuelle.

Définition 4 (Walk)**.** Une *walk*[1] w dans une base de données est une suite finie non vide de nœuds et d'arêtes alternés, commençant et finissant par un nœud, de la forme $w = (n_0, e_1, n_2, \ldots, e_{k-1}, n_k)$ avec $n_0, \ldots, n_k \in V$, $e_1, \ldots, e_{k-1} \in E_d \times E_u$, telle que pour tout $1 \leq i \leq k$ impair, soit $\{\text{src}(e_i), \text{tgt}(e_i)\} = \{n_{i-1}, n_{i+1}\}$, soit $\text{endpoints}(e_i) = \{n_{i-1}, n_{i+1}\}$. On note $\text{len}(w) = k + 1$ la *longueur* de w, $w[i]$ le $i^{\text{ème}}$ élément de w, et $\text{endpoints}(w) = (w[0], w[k])$.
Pour à la fois simplifier et désambiguïser les notations des walks, au lieu d'écrire une walk sous la forme (n_{i-1}, e_i, n_{i+1}), on pourra l'écrire $n_{i-1} \xrightarrow{e_i:\lambda(e_i)} n_{i+1}$, $n_{i-1} \xrightarrow{e_i} n_{i+1}$, $n_{i-1} \xrightarrow{:\lambda(\text{e}_\text{i})} n_{i+1}$ ou $n_{i-1} \to n_{i+1}$ si $\text{src}(e_i) = n_{i-1}$ et $\text{tgt}(e_i) = n_{i+1}$; on changera la direction de la flèche comme suit : $n_{i-1} \leftarrow n_{i+1}$ si $\text{src}(e_i) = n_{i+1}$ et $\text{tgt}(e_i) = n_{i-1}$, et n_{i-1} — n_{i+1} si $\text{endpoints}(e_i) = \{n_{i-1}, n_{i+1}\}$. Pour désambiguïser la notation d'un identifiant de nœud x, on pourra noter x ou $\langle x \rangle$.

Soit deux walks $w = n_0 \xrightarrow{e_1} \ldots \xrightarrow{e_{k-1}} n_k$ et $w' = n'_0 \xrightarrow{e'_1} \ldots \xrightarrow{e'_{l-1}} n'_l$. Si $n_k = n'_0$, alors on peut définir la concaténation $w \cdot w' = n_0 \xrightarrow{e_1} \ldots \xrightarrow{e_{k-1}} n_k \xrightarrow{e'_1} \ldots \xrightarrow{e'_{l-1}} n'_l$.
S'il existe une walk $w \in \mathcal{D}$ telle que $\text{endpoints}(w) = \{s, t\}$, et un mot $u \in \Sigma^*$ tel que $u \in \lambda(w)$, alors on écrit $s \xrightarrow{u} t$.

1. Le terme français *marche* serait correct, mais puisqu'il est peu utilisé par la communauté des bases de données, on conserve le terme anglais.

2.4 GQL

Graph Query Language (GQL) est un langage standardisé de requête pour les graphes à propriétés développé par l'ISO et publié en avril 2024 [1]. Par sa standardisation, on peut comparer son rôle à celui de SQL pour les bases de données relationnelles. On étudiera une simplification de GQL, *Graph Pattern Calculus* (GPC), décrite par [5]. Pour une description de GQL au-delà de la simplification qu'on étudie, on peut se référer à [6].

Dans cette section, on considère $\mathcal{X}$ un ensemble de variables, $\mathcal{L}$ un ensemble d'étiquettes, $\mathcal{K}$ un ensemble de clés, et Const un ensemble de constantes.

Définition 5 (Grammaire d'une expression GPC [5])**.**
Pour $x, y \in \mathcal{X}$, $l \in \mathcal{L}$, $a, b \in \mathcal{K}$, $c \in$ Const :

- Descripteur : $d ::= x \mid :l \mid x:l$
- Direction : $\Longleftrightarrow ::= \rightarrow \mid \leftarrow \mid —$
- Condition : $\theta ::= x.a = c \mid x.a = y.b \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta$
- Motif atomique de nœud : $u ::= () \mid (d)$
- Motif atomique d'arête : $e ::= \Longleftrightarrow \mid \overset{d}{\Longleftrightarrow}$
- Motif de walk : $\pi ::= u \mid e \mid \pi_{<\theta>} \mid \pi^{n..m} \mid \pi \cdot \pi \mid \pi|\pi \quad$ avec $0 \leq n \leq m \leq \infty$

Un motif atomique de nœud est de la forme $(x : l)$, où x est une variable et l est une étiquette, qui sont tous les deux optionnels. De la même façon, un motif atomique d'arête est de la forme $\overset{x:l}{\Longleftrightarrow}$ avec x et $: l$ optionnels, et $\Longleftrightarrow$ représente la direction de l'arête : directe ($\rightarrow$), indirecte ($\leftarrow$), ou non dirigée ($—$). Si une variable x apparaît, alors elle est *liée* au nœud ou à l'arête correspondant dans la base de données. Si une étiquette l apparaît, alors les nœuds ou arêtes pouvant être matchés par le motif atomique doivent être étiquetés par l.
Une condition θ permet de vérifier l'égalité d'une propriété avec une constante, ou l'égalité entre deux propriétés. On peut appliquer à une condition les opérateurs booléens habituels ET $\wedge$, OU $\vee$, et NON $\neg$.
Un motif est construit par l'union $|$, la concaténation $\cdot$, le conditionnement $_{<\theta>}$, et la répétition entre n et m fois $^{n..m}$ de motifs.

Remarque 6 (Simplification d'écriture)**.** Généralement, on se permettra d'omettre l'opérateur de concaténation, et on écrira $\pi_1\pi_2$ au lieu de $\pi_1 \cdot \pi_2$. De la même façon, on écrira π^* et π^+ au lieu de $\pi^{0..\infty}$ et $\pi^{1..\infty}$. On pourra désigner par « motif répété » tout motif de la forme $\pi^{n..m}$.

Remarque 7 (Motifs de nœud implicites)**.** Un motif atomique d'arête représente implicitement des motifs de nœud : tout motif atomique d'arête $\overset{d}{\Longleftrightarrow}$ est équivalent au motif $() \overset{d}{\Longleftrightarrow} ()$.

Remarque 8 (Motif bien formé)**.** On ne considérera que des motifs *bien formés* au sens de GPC. En particulier, les motifs seront bien typés et la portée des variables sera respectée. On pourra se référer aux sections 3, 4, et 5 de [5].

Exemple 9. Le motif suivant est conforme à GPC :

$$\pi = \left[(\text{var}_1 : \text{label}_1) \xrightarrow{:\text{label}_2} \xleftarrow{:\text{label}_2} (\text{var}_2)_{<\text{var}_2.x=\text{var}_1.x>}\right] \mid [\xleftarrow{:\text{label}_3}{}^{2..1024}]$$

En ajoutant les motifs de nœud implicites, on obtient :

$$\pi = \left[(\text{var}_1 : \text{label}_1) \xrightarrow{:\text{label}_2} () \xleftarrow{:\text{label}_2} (\text{var}_2)_{<\text{var}_2.x=\text{var}_1.x>}\right] \mid \left[[() \xleftarrow{:\text{label}_3} ()]^{2..1024}\right]$$

On remarque que les deux motifs atomiques d'arête étiquetés par label_2 partagent un même motif atomique de nœud implicite : si deux arêtes sont matchées par les motifs d'arête étiquetés par label_2, alors celles-ci auront le même nœud pour cible.

Remarque 10 (Désambiguïsation)**.** On utilise les crochets pour désambiguïser et non les parenthèses car celles-ci sont utilisées pour représenter les motifs atomiques de nœud.

Remarque 11 (Restricteurs)**.** Les restricteurs des expressions GPC telles que définies par [5] correspondent à des *sémantiques* telles que celles définies à la Section 3.1.

Dans la suite, on utilise les motifs comme requêtes, aussi pourra-t-on confondre les deux termes.

On définit plusieurs restrictions successives de GPC. Le fragment GPC$^-$ interdit les comparaisons entre propriétés en conservant la comparaison entre une propriété et une constante uniquement dans un motif atomique, et interdit les répétitions $^{n..m}$ où $n \notin \{0, 1\}$ ou $m \neq \infty$.

Définition 12 (Grammaire d'une expression GPC$^-$)**.**
Pour $x \in \mathcal{X}$, $l \in \mathcal{L}$, $a \in \mathcal{K}$, $c \in \text{Const}$:

— Descripteur : $d ::= x \mid :l \mid x : l$

— Direction : $\iff ::= \rightarrow \mid \leftarrow \mid -$

— Condition : $\theta ::= x.a = c \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta \mid \top$

— Motif atomique de nœud : $u ::= ()_{<\theta>} \mid (d)_{<\theta>}$

— Motif atomique d'arête : $e ::= \iff_{<\theta>} \mid \overset{d}{\iff}_{<\theta>}$

— Motif de walk : $\pi ::= u \mid e \mid \pi^{0..\infty} \mid \pi^{1..\infty} \mid \pi \cdot \pi \mid \pi|\pi$

Remarque 13 (Simplification d'écriture)**.** On simplifie la notation des motifs atomiques lorsque $\theta = \top$ en omettant la condition. Par exemple, $(d)_{<\top>}$ est noté (d).

Le fragment GPC^0 interdit en plus la présence de variables dans les motifs répétés. Notons que les étiquettes dans les motifs répétés ne sont pas interdites.

Définition 14 (Grammaire d'une expression GPC^0)**.**
Pour $x \in \mathcal{X}$, $l \in \mathcal{L}$, $a \in \mathcal{K}$, $c \in \text{Const}$:

— Descripteur : $d ::= x \mid :l \mid x : l$

— Direction : $\iff ::= \rightarrow \mid \leftarrow \mid -$

— Condition : $\theta ::= x.a = c \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta \mid \top$

— Motif atomique de nœud sans variable : $u ::= ()_{<\theta>} \mid (:l)_{<\theta>}$

— Motif atomique d'arête sans variable : $e ::= \iff_{<\theta>} \mid \overset{:l}{\iff}_{<\theta>}$

— Motif répétable : $\pi_{rep} ::= u \mid e \mid \pi_{rep}^{0..\infty} \mid \pi_{rep}^{1..\infty}$

— Motif de walk : $\pi ::= (d)_{<\theta>} \mid \overset{d}{\iff}_{<\theta>} \mid \pi_{rep} \mid \pi \cdot \pi \mid \pi|\pi$

Le fragment GPC^{--} interdit les motifs répétés.

Définition 15 (Grammaire d'une expression GPC^{--})**.**
Pour $x \in \mathcal{X}$, $l \in \mathcal{L}$, $a \in \mathcal{K}$, $c \in \text{Const}$:

— Descripteur : $d ::= x \mid :l \mid x:l$

— Direction : $\Longleftrightarrow ::= \rightarrow \mid \leftarrow \mid —$

— Condition : $\theta ::= x.a = c \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta \mid \top$

— Motif atomique de nœud : $u ::= ()_{<\theta>} \mid (d)_{<\theta>}$

— Motif atomique d'arête : $e ::= \Longleftrightarrow_{<\theta>} \mid \overset{d}{\Longleftrightarrow}_{<\theta>}$

— Motif de walk : $\pi ::= u \mid e \mid \pi \cdot \pi \mid \pi|\pi$

Remarque 16. Un motif conforme à GPC^{--} est aussi conforme à GPC^0, et un motif conforme à GPC^0 est aussi conforme à GPC$^-$.

Remarque 17. Dans un motif conforme à GPC$^-$, les conditions n'apparaissent que dans les motifs atomiques. Pour rester *bien typé* au sens de la Remarque 8, une condition ne peut contenir que la variable qui apparaît dans son motif atomique.

On veut utiliser les motifs pour trouver des walks dans une base de données. On va intuiter la notion de walk satisfaisant un motif, qu'on définira formellement dans la Définition 24.

Intuition 18 (Évaluation d'une condition)**.** Soit $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ une base de données, u un nœud de $\mathcal{D}$, et $(x:l)_{<\theta>}$ un motif atomique de nœud. On *évalue* θ de façon usuelle, avec la condition $x.a = c$ évaluée à vrai ssi $\delta(u, a) = c$, et on note $\theta(u)$ l'évaluation de θ sur u. De la même façon, avec une arête e et un motif atomique d'arête $\overset{x:l}{\Longleftrightarrow}_{<\theta>}$, la condition $x.a = c$ est évaluée à vrai ssi $\delta(u, a) = c$, et on note $\theta(e)$ l'évaluation de θ sur e. Ces cas englobent les cas des motifs atomiques de la forme $(x)_{<\theta>}$ et $\overset{x}{\Longleftrightarrow}_{<\theta>}$.

Intuition 19 (Satisfaction)**.** Soit $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ une base de données et Q un motif conforme à GPC$^-$. On note $\text{MATCH}_Q(\mathcal{D})$ l'ensemble des walks dans $\mathcal{D}$ telles qu'elles satisfont le motif.

— Un nœud u de $\mathcal{D}$ satisfait un motif atomique de nœud $(x_1 : l_1)_{<\theta_1>}$ ssi $\theta_1(u) = \top$, et $l_1 \in \lambda(u)$. Ce cas englobe les cas des motifs atomiques de la forme $()_{<\theta_1>}$, $(x_1)_{<\theta_1>}$, ou $(:l_1)_{<\theta_1>}$.

— Une walk $w = n_{i-1} \xrightarrow{e_i} n_{i+1}$ (respectivement $w' = n_{i-1} \xleftarrow{e_i} n_{i+1}$ et $w'' = n_{i-1} \xrightarrow{e_i} n_{i+1}$) de $\mathcal{D}$ satisfait un motif atomique d'arête $\overset{x_2:l_2}{\Longleftrightarrow}_{<\theta_2>}$ ssi $\theta_2(e) = \top$, $l_2 \in \lambda(e)$, et $\Longleftrightarrow = \rightarrow$ (respectivement $\Longleftrightarrow = \leftarrow$ et $\Longleftrightarrow = —$). Ces cas englobent les cas des motifs atomiques de la forme $\Longleftrightarrow_{<\theta_2>}$, $\overset{x_2}{\Longleftrightarrow}_{<\theta_2>}$, ou $\overset{:l_2}{\Longleftrightarrow}_{<\theta_2>}$.

— Une walk w satisfait un motif de la forme $\pi_1|\pi_2$ ssi w satisfait π_1 ou w satisfait π_2.

— Une walk w satisfait un motif de la forme $\pi_1\pi_2$ ssi il existe w_1 et w_2 tels que $w = w_1w_2$, w_1 satisfait π_1 et w_2 satisfait π_2, et les variables de π_1 et celles de π_2 sont correctement unifiées : pour toute variable x qui apparaît à la fois dans un motif atomique de π_1 et dans un motif atomique de π_2, les deux éléments de w qui satisfont exactement ces deux motifs atomiques sont égaux.

— Une walk w satisfait un motif de la forme π^+ ssi il existe $w_1, \ldots, w_k$ qui satisfont π tels que $w = w_1 \cdot \ldots \cdot w_k$.

— Une walk w satisfait un motif de la forme π^* ssi $\text{len}(w) = 1$, ou si elle satisfait π^+.

Exemple 20. Avec la base de données $\mathcal{D}$ telle que représentée par la Figure 2 [2].

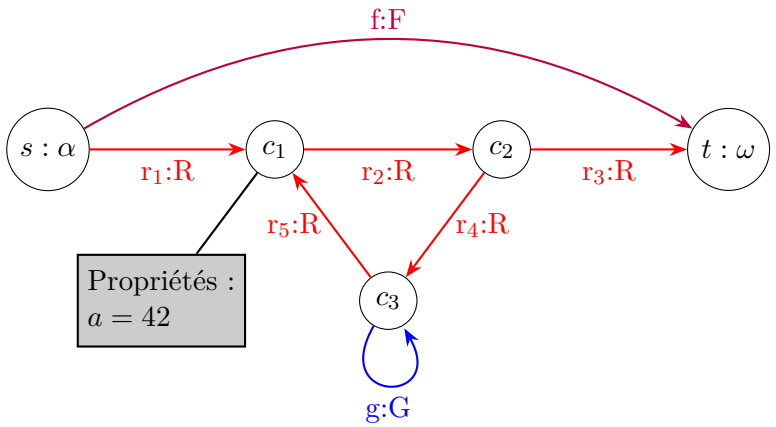


FIGURE 2 – Base de données de l'Exemple 20

Voici des listes non exhaustives de walks satisfaisant ou non des motifs donnés.

— $w_1 = s$, $w_2 = c_1$, $w_3 = c_2$ satisfont $\pi_1 = ()$; $\bar{w}_1 = c_1 \xrightarrow{r_2} c_2$ ne satisfait pas π_1.

— $w_4 = s \xrightarrow{r_1} c_1$, $w_5 = s \xrightarrow{f} t$ satisfont $\pi_2 = \rightarrow$; $\bar{w}_2 = t \xleftarrow{f} s$ ne satisfait pas π_2.

— $w'_4 = c_1 \xleftarrow{r_1} s$ satisfait $\pi_3 = (x)_{<x.a=42>} \leftarrow ()$.

— w_5 satisfait $\pi_4 = (:\alpha) \rightarrow ()$; w'_4 ne satisfait pas π_4.

— $w_6 = c_3$, $w_7 = c_3 \xrightarrow{g} c_3 \xrightarrow{g} c_3$, $w_8 = c_3 \xrightarrow{r_5} c_1 \xrightarrow{r_2} c_2 \xrightarrow{r_4} c_3$ satisfont $\pi_5 = (x)[\xrightarrow{:G}|\xrightarrow{:R}]^*(x)$; $\bar{w}_5 = c_3 \xrightarrow{r_5} c_1 \xrightarrow{r_2} c_2 \xrightarrow{r_4} c_3 \xrightarrow{r_5} c_1$ ne satisfait pas π_5.

— w_7, w_8 satisfont $\pi_6 = (x)[\xrightarrow{:G}|\xrightarrow{:R}]^+(x)$; w_6 ne satisfait pas π_6.

— w_5, $w_9 = s \xrightarrow{r_1} c_1 \xleftarrow{r_1} s \xrightarrow{f} t$ satisfont $\pi_7 = (:\alpha)[\rightarrow | \leftarrow]^*(:\omega)$.

Pour une requête et une walk données, on va construire un tuple appelé *binding walk* permettant d'associer un élément de la walk au motif atomique de la requête qui a permis que cet élément soit dans la walk.

Définition 21 (Position d'un motif atomique)**.** Soit Q un motif conforme à GPC$^-$ auquel on a ajouté les motifs de nœud implicites. La *position* d'un motif atomique dans Q est le nombre de motifs atomiques à sa gauche dans Q.

Exemple 22. Avec $Q = (:l_1)[\rightarrow | \leftarrow]^*(:l_2)$. On ajoute les motifs implicites pour obtenir $Q' = (:l_1)[() \rightarrow ()|() \leftarrow ()]^*(:l_2)$ et on a :

Q' :	$(:l_1)$	[	()	$\rightarrow$	()	|	()	$\leftarrow$	()	]*	$(:l_2)$
Position :	0		1	2	3		4	5	6		7

Définition 23 (Binding walk)**.** Soit $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ une base de données, et Q un motif conforme à GPC$^-$ auquel on a ajouté les motifs de nœud implicites. On note $\text{Pos} = \{0, \ldots, k\}$ l'ensemble des positions des motifs atomiques de Q. On pose γ_n (respectivement γ_e) la fonction partielle qui associe aux positions le motif atomique de nœud (respectivement d'arête) correspondant.
Une *binding walk* de Q dans $\mathcal{D}$ est un tuple $b = (w, g_e, g_n)$ où w est une walk dans $\mathcal{D}$, $g_e : \{i \mid i \text{ impair}, i < \text{len}(w)\} \longrightarrow \text{Pos}$, et $g_n : \{i \mid i \text{ pair}, i < \text{len}(w)\} \longrightarrow 2^{\text{Pos}}$. Un tel tuple

2. Cette base de données est largement utilisée dans [3]

est un *candidat* de réponse pour Q.
On dit que le nœud n est *associé* dans w à une variable x par g_n s'il existe $i < \text{len}(w)$ tel que $w[i] = n$ et il existe une position $\text{pos} \in g_n(i)$ telle que le motif $\gamma_n(\text{pos})$ contient la variable x. On le note $n \diamond_{g_n} x$. De façon similaire, on dit que l'arête e est associée dans w à une variable x par g_e, et on note $e \diamond_{g_e} x$.
On note $(x : l)^k$ pour affirmer que le motif atomique à la position k dans Q est $(x : l)$ et, de la même façon, on note $\overset{x:l}{\Longleftrightarrow}^{k}$ pour affirmer que le motif atomique à la position k dans Q est $\overset{x:l}{\Longleftrightarrow}$.

On veut maintenant connaître l'ensemble des binding walks d'une base de données qui satisfont une requête donnée.

Définition 24 (Satisfaction)**.** Soit $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ une base de données et Q un motif conforme à GPC$^-$ auquel on a ajouté les motifs de nœud implicites. On note $\text{BW}_{Q,\mathcal{D}}$ l'ensemble des binding walks de Q dans $\mathcal{D}$, et on définit la fonction φ qui associe un motif et sa position à un ensemble de binding walks de Q dans $\mathcal{D}$ telle que :

— $\varphi((x : l)^k_{<\theta>}, \mathcal{D}) = \left\{ ((n_0), \emptyset, \{0 \mapsto \{k\}\}) \;\middle|\; \begin{matrix} l \in \lambda(n_0) \\ \theta(x) = \top \end{matrix} \right\}$

— $\varphi(\overset{x:l}{\longrightarrow}^{k}_{<\theta>}, \mathcal{D}) = \left\{ ((n_0 \xrightarrow{e_1} n_2), \{1 \mapsto k\}, \begin{Bmatrix} 0 \mapsto \emptyset \\ 2 \mapsto \emptyset \end{Bmatrix}) \;\middle|\; \begin{matrix} l \in \lambda(e_1) \\ \theta(x) = \top \end{matrix} \right\}$

— $\varphi(\overset{x:l}{\longleftarrow}^{k}_{<\theta>}, \mathcal{D}) = \left\{ ((n_0 \xleftarrow{e_1} n_2), \{1 \mapsto k\}, \begin{Bmatrix} 0 \mapsto \emptyset \\ 2 \mapsto \emptyset \end{Bmatrix}) \;\middle|\; \begin{matrix} l \in \lambda(e_1) \\ \theta(x) = \top \end{matrix} \right\}$

— $\varphi(\overset{x:l}{-}^{k}_{<\theta>}, \mathcal{D}) = \left\{ ((n_0 \xrightarrow{e_1} n_2), \{1 \mapsto k\}, \begin{Bmatrix} 0 \mapsto \emptyset \\ 2 \mapsto \emptyset \end{Bmatrix}) \;\middle|\; \begin{matrix} l \in \lambda(e_1) \\ \theta(x) = \top \end{matrix} \right\}$

— $\varphi(Q_1 \cdot Q_2, \mathcal{D}) = \left\{ (w, h_e, h_n) \;\middle|\; \begin{matrix} (w^1, g_e^1, g_n^1) \in \varphi(Q_1, \mathcal{D}), (w^2, g_e^2, g_n^2) \in \varphi(Q_2, \mathcal{D}) \\ w = w^1 \cdot w^2 \\ \forall i, j,\ \forall x \in \mathcal{X},\ (w^1[i] \diamond_{g_e^1} x \wedge w^2[j] \diamond_{g_e^2} x) \\ \qquad \Longrightarrow\ w^1[i] = w^2[j] \\ \forall i, j,\ \forall x \in \mathcal{X},\ (w^1[i] \diamond_{g_n^1} x \wedge w^2[j] \diamond_{g_n^2} x) \\ \qquad \Longrightarrow\ w^1[i] = w^2[j] \end{matrix} \right\}$

avec $h_e = i \mapsto \begin{cases} g_e^1(i) & \text{si } i < \text{len}(w^1) \\ g_e^2(i - \text{len}(w^1) + 1) & \text{sinon} \end{cases}$

$h_n = i \mapsto \begin{cases} g_n^1(i) & \text{si } i < \text{len}(w^1) - 1 \\ g_n^1(i) \cup g_n^2(0) & \text{si } i = \text{len}(w^1) - 1 \\ g_n^2(i - \text{len}(w^1) + 1) & \text{sinon} \end{cases}$

— $\varphi(Q_1 | Q_2, \mathcal{D}) = \varphi(Q_1, \mathcal{D}) \cup \varphi(Q_2, \mathcal{D})$

— $\varphi(Q'^+, \mathcal{D}) = \left\{ \left(w, h_e, i \mapsto \bigcup_{m=1}^{k} h_n^m(i)\right) \;\middle|\; \begin{matrix} k \in \mathbb{N}^* \\ w = w^1 \cdot \ldots \cdot w^k \\ (w^1, g_e^1, g_n^1), \ldots, (w^k, g_e^k, g_n^k) \in \varphi(Q', \mathcal{D}) \end{matrix} \right\}$

avec $\text{len}^0 = 0,\ \text{len}^m = \sum_{j=1}^{m} (\text{len}(w^j) - 1)$

$h_e = i \mapsto g_e^m(i - \text{len}^{m-1}) \quad (m \in \{1, \ldots, k\} \text{ tel que } \text{len}^{m-1} \leq i < \text{len}^m)$

$h_n^m : i \mapsto \begin{cases} g_n^m(i - \text{len}^{m-1}) & \text{si } i \in \left\{\text{len}^{m-1}, \ldots, \text{len}^m\right\} \\ \emptyset & \text{sinon} \end{cases}$

— $\varphi(Q'^*, \mathcal{D}) = \varphi(Q'^+, \mathcal{D}) \cup \{((n_0), \emptyset, \{0 \mapsto \emptyset\}) \mid n_0 \in V\}$

Une walk w *satisfait* une requête Q dans une base de données $\mathcal{D}$ ssi il existe une binding walk $b = (w, g_e, g_n)$ telle que $b \in \varphi(Q, \mathcal{D})$.

Exemple 25. Avec la base de données $\mathcal{D}$ décrite par la Figure 3, et les requêtes suivantes :

— $Q_1 = (x)$
Positions : 0

— $Q_2 = () \to ()$
Positions : 0 1 2

— $Q_3 = () [() \to ()]^+ ()$
Positions : 0 1 2 3 4

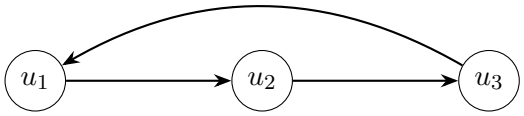


FIGURE 3 – Base de données de l'Exemple 25

$$\varphi(Q_1, \mathcal{D}) = \left\{ \begin{array}{l} \big((u_1), \emptyset, \{0 \mapsto \{0\}\}\big), \\ \big((u_2), \emptyset, \{0 \mapsto \{0\}\}\big), \\ \big((u_3), \emptyset, \{0 \mapsto \{0\}\}\big) \end{array} \right\}$$

$$\varphi(Q_2, \mathcal{D}) = \left\{ \begin{array}{l} \big((u_1 \to u_2), \{1 \mapsto 1\}, \left\{ \begin{array}{l} 0 \mapsto \{0\} \\ 2 \mapsto \{2\} \end{array} \right\}\big), \\ \big((u_2 \to u_3), \{1 \mapsto 1\}, \left\{ \begin{array}{l} 0 \mapsto \{0\} \\ 2 \mapsto \{2\} \end{array} \right\}\big), \\ \big((u_3 \to u_1), \{1 \mapsto 1\}, \left\{ \begin{array}{l} 0 \mapsto \{0\} \\ 2 \mapsto \{2\} \end{array} \right\}\big) \end{array} \right\}$$

$$\big((u_1 \to u_2 \to u_3 \to u_1), \left\{ \begin{array}{l} 1 \mapsto 1 \\ 3 \mapsto 1 \\ 5 \mapsto 1 \end{array} \right\}, \left\{ \begin{array}{l} 0 \mapsto \{0\} \\ 2 \mapsto \{0, 2\} \\ 4 \mapsto \{0, 2\} \\ 6 \mapsto \{2\} \end{array} \right\}\big) \in \varphi(Q_2^+, \mathcal{D})$$

$$\big((u_1 \to u_2 \to u_3 \to u_1), \left\{ \begin{array}{l} 1 \mapsto 2 \\ 3 \mapsto 2 \\ 5 \mapsto 2 \end{array} \right\}, \left\{ \begin{array}{l} 0 \mapsto \{0, 1\} \\ 2 \mapsto \{1, 3\} \\ 4 \mapsto \{1, 3\} \\ 6 \mapsto \{3, 4\} \end{array} \right\}\big) \in \varphi(Q_3, \mathcal{D})$$

2.5 Complexité

On utilise les définitions usuelles des problèmes de décision, et des classes de complexité **P** et **NP**. On peut se référer aux chapitres 1 et 2 de [7].

Jusqu'à la fin de cette section, on suppose donnés une requête Q, une base de données $\mathcal{D}$, et un problème de décision P prenant en entrée une requête et une base de données.

On peut fixer une des deux entrées de P pour définir d'une part la *complexité en données* et d'autre part la *complexité en requête* de P. Intuitivement, si on fixe la requête, la complexité de P ne dépend que de la base de données, et on calcule ainsi sa complexité en données ; de façon similaire, si on fixe la base de données, la complexité de P ne dépend que de la requête, et on calcule ainsi sa complexité en requête.

Définition 26 (Complexité en données et en requête [8])**.** Soit PROB un problème de décision prenant en entrée une requête Q et une base de données $\mathcal{D}$. On a PROB :

— Entrée : une base de données $\mathcal{D}$, une requête Q

— Question : ...

Considérer la complexité en données de PROB revient à considérer le problème PROB(Q) :

— Entrée : une base de données $\mathcal{D}$

— Question : [la même que celle de PROB]

De la même façon, considérer la complexité en requête de PROB revient à considérer le problème PROB($\mathcal{D}$) :

— Entrée : une requête Q

— Question : [la même que celle de PROB]

Un problème d'énumération consiste en l'énumération de tous les éléments d'un ensemble et, intuitivement, plus cet ensemble est grand, plus le temps de calcul sera élevé. Comme on veut une mesure indépendante de la taille de l'ensemble, on ne s'intéresse plus au temps pour obtenir tous ses éléments, mais à l'écart entre l'obtention de deux éléments.

Définition 27 (Complexité énumérative [9])**.** Soit Σ un alphabet, et $A \subseteq \Sigma^* \times \Sigma^*$ une relation. On note $A(x)$ l'ensemble des y tels que $(x, y) \in A$. Le problème d'énumération ENUM$_A$ est la fonction qui associe $A(x)$ à x. Une *solution* du problème est un élément de ENUM$_A$. On considère la *complexité énumérative* de ENUM$_A$ comme le temps de précalcul, entre le début du calcul et la production de la première solution, et le délai maximal entre la production de deux solutions consécutives.

On signale l'existence de la classe de complexité énumérative **DelayP** qui contient les problèmes énumérables avec un précalcul et un délai polynomiaux en la taille de l'entrée.

3 Problèmes computationnels

3.1 Sémantiques

Idéalement, lors de l'évaluation d'une requête Q sur une base de données $\mathcal{D}$, on voudrait calculer MATCH$_Q(\mathcal{D})$, mais c'est un ensemble qui peut être infini, comme on peut le remarquer dans l'Exemple 20. Ainsi, on restreint cet ensemble selon des *sémantiques*. On donne la définition de deux sémantiques communes avant de donner celle de Binding Trail, la sémantique étudiée.

Définition 28 (Trail)**.** Une *trail* est une walk sans répétition d'arête. Une walk w satisfait un motif Q dans une base de données $\mathcal{D}$ sous la sémantique Trail ssi il existe une binding walk $b = (w, g_e, g_n)$ de Q dans $\mathcal{D}$ avec $w[i] \neq w[j]$ pour tout $i \neq j$ impairs.

Définition 29 (Shortest)**.** Soit $b = (w, g_e, g_n)$ une binding trail de Q dans $\mathcal{D}$. w satisfait Q dans $\mathcal{D}$ sous la sémantique Shortest ssi il n'existe pas de binding walk $b' = (w', g'_e, g'_n)$ de Q dans $\mathcal{D}$ telle que endpoints(w) = endpoints(w') et len(w') < len(w).

Ces deux sémantiques ont l'inconvénient, pour la première, de rendre l'évaluation de la requête non tractable [2] et, pour la seconde, de tant restreindre l'ensemble qu'un grand nombre d'information est perdu. Intuitivement, Binding Trail permet à une walk d'utiliser plusieurs fois une même arête uniquement si celle-ci n'est pas matchée deux fois par le même atome. Avant de pouvoir définir Binding Trail pour les RPQs, il faut d'abord définir la linéarisation d'une expression régulière.

Définition 30 (Linéarisation, positions)**.** Soit R une expression régulière sur un alphabet Σ. Une *linéarisation* de R est une copie R' de R où chaque symbole $\sigma_i \in \Sigma$ à la position i dans R est remplacé par un symbole σ'_i d'un nouvel alphabet Σ', appelé *positions* dans R, avec $\sigma_i \neq \sigma'_i$ et $\sigma'_i \neq \sigma'_j$ pour tout $i \neq j$.

On peut maintenant définir Binding Trail pour les RPQs, puis adapter cette définition aux motifs.

Définition 31 (Binding Trail pour les RPQs [3])**.** Soit $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ une base de données, et R une expression régulière. Soit R' une linéarisation de R et Σ' les positions correspondantes dans R. Une binding trail dans $\mathcal{D}$ satisfaisant R est une suite de couples $\big((e_0, \alpha_0), \ldots, (e_{k-1}, \alpha_{k-1})\big) \in (E_d \cup E_u) \times \Sigma'$ telle que :

— $n_0 \xrightarrow{e_0} \ldots \xrightarrow{e_{k-1}} n_k$ est une walk dans $\mathcal{D}$,

— $\alpha_0 \cdot \ldots \cdot \alpha_{k-1} \in L(R')$ et $(\alpha_0 \cdot \ldots \cdot \alpha_{k-1})^{-1} \in \lambda(e_1 \cdot \ldots \cdot e_{k-1})$,

— les (e_i, α_i) sont deux à deux distincts.

On note Proj_1 la projection sur la première composante. On définit la sémantique Binding Trail pour les RPQs comme :

$$[\![R]\!]_{\text{BT,RPQ}}(\mathcal{D}) = \text{Proj}_1(\{w \mid w \text{ est une binding trail dans } \mathcal{D} \text{ satisfaisant } R\})$$

Remarque 32 (Répétition de taille arbitraire sous Binding Trail)**.** Généralement, on considère que les expressions $R = a^2$ et $R' = aa$ sont équivalentes. Ce n'est pas le cas sous Binding Trail car la position de l'atome dans la requête est importante. Si w et w' sont deux binding trails conformes respectivement à R et à R', alors les deux contiendront deux arêtes étiquetées par a, mais celles de w seront forcément différentes, alors que celles de w' pourront être les mêmes. L'effet de l'opérateur de répétition arbitraire sur les problèmes de complexité n'avait pas été étudié dans [3], mais on le traitera dans le Théorème 34.

Pour traiter le cas des motifs, on s'appuie sur la fonction φ définie à la Définition 24 car elle permet de savoir quels motifs atomiques ont permis de construire une walk donnée. En particulier, elle permet de retrouver si une même arête a été matchée deux fois à la même position d'un motif atomique.

Définition 33 (Binding Trail pour les motifs)**.** Soit $\mathcal{D} = (\Sigma, V, E_d, E_u, \text{src}, \text{tgt}, \text{endpoints}, \lambda, \delta)$ une base de données, et Q un motif conforme à GPC$^-$. Une walk w satisfait un motif Q dans une base de données $\mathcal{D}$ sous la sémantique Binding Trail ssi il existe une binding walk $b = (w, g_e, g_n)$ de Q dans $\mathcal{D}$ telle que $b \in \varphi(Q, \mathcal{D})$ et pour toute arête e de w, si e apparaît à deux positions distinctes i et j de w, alors $g_e(i) \neq g_e(j)$.
On note $[\![Q]\!]_{\text{BT}}(\mathcal{D})$ l'ensemble des walks qui satisfont Q dans $\mathcal{D}$ sous la sémantique Binding Trail.

3.2 Problèmes de décision

Le problème Query Evaluation consiste en l'énumération des walks entre deux nœuds satisfaisant une requête. C'est un problème proche de ce qu'on souhaiterait faire dans un système réel.

Problème 1. Query Evaluation sous Binding Trail (Query Evaluation$_{\text{BT}}$)

— Entrée : une base de données $\mathcal{D}$, une requête Q, deux nœuds s, t de $\mathcal{D}$

— Sortie : $\{w \mid w \in [\![Q]\!]_{\text{BT}}(\mathcal{D}) \wedge \text{endpoints}(w) = \{s, t\}\}$

On s'intéressera principalement au problème de décision Tuple Membership qui consiste à savoir s'il existe une walk entre deux nœuds donnés satisfaisant une requête. En effet, pour deux nœuds donnés, si Tuple Membership est vrai alors Query Evaluation a au moins une solution et si Tuple Membership est faux alors Query Evaluation n'a aucune solution. Ainsi, si Tuple Membership est un problème *difficile*, alors le précalcul ou le calcul de la première solution pour Query Evaluation sera lui aussi difficile. Or, on montrera que Tuple Membership est déjà difficile pour la plupart de nos restrictions.

Problème 2. Tuple Membership sous Binding Trail (Tuple Membership$_{\mathrm{BT}}$) :

— Entrée : une base de données $\mathcal{D}$, une requête Q, deux nœuds s, t de $\mathcal{D}$

— Question : Existe-t-il une walk $w \in [\![Q]\!]_{\mathrm{BT}}(\mathcal{D})$ telle que endpoints$(w) = \{s, t\}$?

On rappelle que [3] a prouvé que, si Q est une RPQ, alors Tuple Membership$_{\mathrm{BT}}$ est **NL**-complet et Query Evaluation$_{\mathrm{BT}}$ est dans **DelayP**.

4 Résultats négatifs

Dans la suite, on restreint notre modèle de base de données aux multigraphes multiétiquetés sans arêtes non orientées ni propriétés. Ainsi, on enlève E_u, endpoints, et δ de notre modèle, et une base de données devient un tuple $\mathcal{D} = (\Sigma, V, E_d, \mathrm{src}, \mathrm{tgt}, \lambda)$. On verra que Tuple Membership$_{\mathrm{BT}}$ est déjà **NP**-difficile dans ce cadre restreint.

On procèdera par réduction soit depuis 3-SAT, soit depuis 2-Disjoint Paths. On rappelle 2-Disjoint Paths, dont on trouvera la preuve de **NP**-complétude à la section 10.2 de [10].

Problème 3. 2-Disjoint Paths

— Entrée : un graphe $G = (V, E)$, quatre nœuds s_1, t_1, s_2, t_2 de G deux à deux distincts

— Question : Existe-t-il deux walks π_1 de s_1 à t_1, et π_2 de s_2 à t_2 telles que π_1 et π_2 n'ont pas d'arêtes en commun ?

4.1 Répétition fixe

On va montrer que, même pour certaines RPQs, le problème est difficile.

Théorème 34. Tuple Membership$_{\mathrm{BT}}$ *est* ***NP****-difficile pour les RPQs avec répétition arbitraire. C'est déjà vrai en complexité en données : il existe une requête Q pour laquelle* Tuple Membership$_{\mathrm{BT}}(Q)$ *est* ***NP****-difficile.*

Intuition de la preuve. Avec $Q = (ba^*r)^2$, G le graphe et G' la base de données construite à partir de G décrits par la Figure 4.

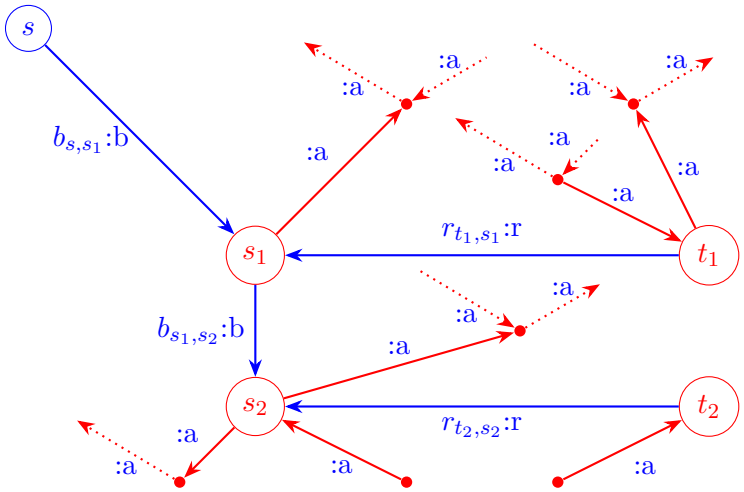


FIGURE 4 – Base de données utilisée comme intuition du Théorème 34

Soit G le graphe qui contient tous les nœuds et arêtes en rouge. On pose G' la base de données qui est une copie de G', en ajoutant le sommet s et les arêtes et étiquettes en bleu. On cherche à réduire depuis 2-DISJOINT PATHS.
En partant de s, on ne peut aller qu'en s_1 par b_{s,s_1} puis, pour pouvoir terminer avec une arête étiquetée par r, on doit aller soit en t_1, soit en t_2, puis on ira respectivement soit en s_1, soit en s_2. Aller en s_2 empêcherait de faire le second b, donc on ne doit pas aller en t_2. Enfin, en arrivant en s_2, la sémantique empêche de reprendre les mêmes arêtes que celles utilisées pour aller de s_1 à t_1.

Preuve. Par réduction depuis 2-DISJOINT PATHS.
On pose $\Sigma = \{a, b, r\}$, et $Q = (ba^*r)^2$ une expression sur Σ. On veut montrer que 2-DISJOINT PATHS se réduit à TUPLE MEMBERSHIP$_{\text{BT}}(Q)$.
Soit (G, s_1, t_1, s_2, t_2) une instance de 2-DISJOINT PATHS. Soit V, E tels que $G = (V, E)$. On va créer G' une copie de G dont les arêtes seront étiquetées par a, avec un nouveau nœud s qui sera la source des walks matchées par Q, deux nouveaux nœuds s_1 et s_2 qui seront chacun la cible d'une arête étiquetée par b, et deux nouveaux nœuds t_1 et t_2 qui seront chacun la source d'une arête étiquetée par r. Plus formellement, on pose $G' = (\Sigma, V', E_d, \text{src}, \text{tgt}, \lambda)$ avec :

- $V' = V \uplus \{s\}$
- $E_d = E \uplus \{b_{s,s_1}, b_{s_1,s_2}, r_{t_1,s_1}, r_{t_2,s_2}\}$
- src: $E_d \longrightarrow V'$ telle que pour $(u, v) \in E$, $\text{src}(u, v) = u$, et $\text{src}(b_{s,s_1}) = s$, $\text{src}(b_{s_1,s_2}) = s_1$, $\text{src}(r_{t_1,s_1}) = t_1$, $\text{src}(r_{t_2,s_2}) = t_2$
- tgt: $E_d \longrightarrow V'$ telle que pour $(u, v) \in E$, $\text{tgt}(u, v) = v$, et $\text{tgt}(b_{s,s_1}) = s_1$, $\text{tgt}(b_{s_1,s_2}) = s_2$, $\text{tgt}(r_{t_1,s_1}) = s_1$, $\text{tgt}(r_{t_2,s_2}) = s_2$
- λ: $E_d \longrightarrow \Sigma$ telle que pour $e \in E$, $\lambda(e) = a$, et $\lambda(b_{s,s_1}) = b$, $\lambda(b_{s_1,s_2}) = b$, $\lambda(r_{t_1,s_1}) = r$, $\lambda(r_{t_2,s_2}) = r$.

On veut montrer que (G, s_1, t_1, s_2, t_2) est une instance acceptante de 2-DISJOINT PATHS ssi (G', s, s_2) est une instance acceptante de TUPLE MEMBERSHIP$_{\text{BT}}(Q)$, c'est-à-dire ssi il existe une walk $\pi : s \xrightarrow{\pi} s_2$ telle que $\pi \in [\![Q]\!]_{\text{BT}}(G')$.

($\Rightarrow$) On suppose que (G, s_1, t_1, s_2, t_2) est acceptant pour 2-DISJOINT PATHS.
Soit π_1 une walk de s_1 à t_1 et π_2 une walk de s_2 à t_2 telles que π_1 et π_2 sont disjoints en arêtes dans G.

On pose $\pi = (s, b_{s,s_1}, s_1) \cdot \pi_1 \cdot (t_1, r_{t_1,s_1}, s_1, b_{s_1,s_2}, s_2) \cdot \pi_2 \cdot (t_2, r_{t_2,s_2}, s_2)$.
On remarque que $\mathrm{src}(\pi) = s$ et $\mathrm{tgt}(\pi) = s_2$. Il reste à montrer que π est acceptée par Q sous Binding Trail.

1. $\lambda(\pi) = \lambda(s, b_{s,s_1}, s_1) \cdot \lambda(\pi_1) \cdot \lambda(t_1, r_{t_1,s_1}, s_1, b_{s_1,s_2}, s_2) \cdot \lambda(\pi_2) \cdot \lambda(t_2, r_{t_2,s_2}, s_2)$
   $\lambda(\pi) = b \cdot \lambda(\pi_1) \cdot rb \cdot \lambda(\pi_2) \cdot r$
   Par définition de G', $\lambda(\pi_1) \in a^*$ et $\lambda(\pi_2) \in a^*$.
   On en déduit $\lambda(\pi) \in (ba^*r)^2$.
2. On remarque que π est une trail. En effet, π_1 et π_2 sont disjointes par hypothèse, et ne contiennent pas $b_{s,s_1}, b_{s_1,s_2}, r_{t_1,s_1}, r_{t_2,s_2}$ par construction de G'.

Finalement : $\pi \in [\![Q]\!]_{\mathrm{BT}}(G')$.

($\Leftarrow$) On suppose que (G', s, s_2) est acceptant pour Tuple Membership$_{\mathrm{BT}}(Q)$.
Soit π une walk de s à s_2 telle que $\pi \in [\![Q]\!]_{\mathrm{BT}}(G')$.
On sait que $\lambda(\pi) \in (ba^*r)^2$, donc il existe $\pi_b^{(1)}$, π_1, $\pi_r^{(1)}$, $\pi_b^{(2)}$, π_2, $\pi_r^{(2)}$ telles que :

— $\pi = \pi_b^{(1)} \cdot \pi_1 \cdot \pi_r^{(1)} \cdot \pi_b^{(2)} \cdot \pi_2 \cdot \pi_r^{(2)}$
— $\lambda(\pi_b^{(1)}) = b = \lambda(\pi_b^{(2)})$
— $\lambda(\pi_1) \in a^*$ et $\lambda(\pi_2) \in a^*$
— $\lambda(\pi_r^{(1)}) = r = \lambda(\pi_r^{(2)})$.

Comme $\pi \in [\![Q]\!]_{\mathrm{BT}}(G')$, π_1 et π_2 sont disjointes en arêtes car elles sont toutes associées à l'atome a. Par définition de G', π_1 et π_2 sont aussi des walks de G.
Il reste à montrer que $\mathrm{src}(\pi_1) = s_1, \mathrm{tgt}(\pi_1) = t_1$,

$$\mathrm{src}(\pi_2) = s_2, \mathrm{tgt}(\pi_2) = t_2.$$

On remarque que $\mathrm{src}(\pi_b^{(1)}) = s$, donc $\mathrm{tgt}(\pi_b^{(1)}) = s_1$ par définition de G'. On en déduit $\mathrm{src}(\pi_a^{(1)}) = s_1$ car ce sont les deux seules arêtes étiquetées par r dans G'. On remarque que soit $\pi_r^{(1)} = (t_1, r_{t_1,s_1}, s_1)$ ($\bigstar_1$), soit $\pi_r^{(1)} = (t_2, r_{t_2,s_2}, s_2)$ ($\bigstar_2$).
Si ($\bigstar_1$), alors $\mathrm{tgt}(\pi_1) = t_1$, et il n'y a rien de plus à montrer pour π_1.
Si ($\bigstar_2$), alors $\mathrm{src}(\pi_b^{(2)}) = s_2$, ce qui est impossible car s_2 n'a pas d'arête sortante étiquetée par b.
On en déduit que π_1 est bien une walk de s_1 à t_1. De la même façon, on trouve que π_2 est une walk de s_2 à t_2.
Finalement, π_1 et π_2 sont deux walks disjointes de G allant respectivement de s_1 à t_1 et de s_2 à t_2, donc (G, s_1, t_1, s_2, t_2) est acceptant pour 2-Disjoint Paths. □

4.2 Variable avec répétitions imbriquées

Le Théorème 35 est similaire au Théorème 34, mais on s'interdit la répétition $^{n..m}$ où $n = m$. On peut quand même montrer la borne inférieure grâce aux variables.

Théorème 35. Tuple Membership$_{\mathrm{BT}}$ *est* ***NP****-difficile pour les expressions de GPC. C'est déjà vrai en complexité en données : il existe une requête Q pour laquelle* Tuple Membership$_{\mathrm{BT}}(Q)$ *est* ***NP****-difficile.*

Intuition de la preuve. Avec $Q = \left[() \xrightarrow{:\mathrm{b}} (x)[() \xrightarrow{:\mathrm{a}} ()]^* \xrightarrow{:\mathrm{r}} (x)\right]^{0..2}$, G le graphe et G' la base de données construite à partir de G décrits par la Figure 4, on cherche à réduire depuis 2-Disjoint Paths. On veut aller de s à s_2 dans G'. La variable dans la requête impose de terminer la première itération sur s_1 puis de terminer la seconde itération sur s_2, et la sémantique empêche de prendre plusieurs fois la même arête étiquetée par a. Comme ce sont exactement les arêtes étiquetées par a qui sont présentes dans G, on a des walks

disjointes dans G entre s_1 et t_1, et entre s_2 et t_2.

Preuve. Par réduction depuis 2-Disjoint Paths.
On pose $\Sigma = \{b, a, r\}$, et $Q = (() \xrightarrow{:b} (x)(() \xrightarrow{:a} ())^* \xrightarrow{:r} (x))^{0..2}$ une expression sur Σ. On veut montrer que 2-Disjoint Paths se réduit à Tuple Membership$_{\text{BT}}(Q)$.
Soit $(G, s_1, t_1, s_2, t, 2)$ une instance de 2-Disjoint Paths, et V, E tels que $G = (V, E)$.
On pose $G' = (\Sigma, V', E_d, \text{src}, \text{tgt}, \lambda)$ avec :

- $V' = V \uplus \{s\}$
- $E_d = E \uplus \{b_{s,s_1}, b_{s_1,s_2}, r_{t_1,s_1}, r_{t_2,s_2}\}$
- src: $E_d \longrightarrow V'$ telle que pour $(u, v) \in E$, $\text{src}(u, v) = u$, et $\text{src}(b_{s,s_1}) = s$, $\text{src}(b_{s_1,s_2}) = s_1$, $\text{src}(r_{t_1,s_1}) = t_1$, $\text{src}(r_{t_2,s_2}) = t_2$
- tgt: $E_d \longrightarrow V'$ telle que pour $(u, v) \in E$, $\text{tgt}(u, v) = v$, et $\text{tgt}(b_{s,s_1}) = s_1$, $\text{tgt}(b_{s_1,s_2}) = s_2$, $\text{tgt}(r_{t_1,s_1}) = s_1$, $\text{tgt}(r_{t_2,s_2}) = s_2$
- λ: $E_d \longrightarrow \Sigma$ telle que pour $e \in E$, $\lambda(e) = a$, et $\lambda(b_{s,s_1}) = b$, $\lambda(b_{s_1,s_2}) = b$, $\lambda(r_{t_1,s_1}) = r$, $\lambda(r_{t_2,s_2}) = r$.

On veut montrer que (G, s_1, t_1, s_2, t_2) est une instance acceptante de 2-Disjoint Paths ssi (G', s, s_2) est une instance acceptante de Tuple Membership$_{\text{BT}}(Q)$, c'est-à-dire ssi il existe une walk π: $s \xrightarrow{\pi} s_2$ telle que $\pi \in [\![Q]\!]_{\text{BT}}(G')$.

($\Rightarrow$) On suppose que (G, s_1, t_1, s_2, t_2) est acceptant pour 2-Disjoint Paths. On procède de la même façon que dans la preuve du Théorème 34, et on remarque en plus que la variable x dans Q est correctement unifiée. En effet, les nœuds cibles de chaque arête étiquetée par b sont aussi les nœuds cibles de la prochaine arête étiquetée par r.
Finalement : $\pi \in [\![Q]\!]_{\text{BT}}(G')$.

($\Leftarrow$) On suppose que (G', s, s_2) est acceptant pour Tuple Membership$_{\text{BT}}(Q)$. Soit π une walk de s à s_2 telle que $\pi \in [\![Q]\!]_{\text{BT}}(G')$.
On sait que $\lambda(\pi) \in \lambda(Q)$, donc le motif $p = () \xrightarrow{:b} (x)(() \xrightarrow{:a} ())^* \xrightarrow{:r} (x)$ est répété exactement zéro, une, ou deux fois dans π.
Si p est répété exactement zéro fois dans π, alors $s = s_2$, ce qui est impossible par construction de G'. Si p est répété exactement une fois dans π, comme $\text{src}(\pi) = s$, alors $x = s_1$, donc $\text{tgt}(\pi) = s_1$, ce qui est impossible par hypothèse. On en déduit que p est répété exactement deux fois dans π.
En procédant de la même façon que dans la preuve du Théorème 34, on obtient que π_1 et π_2 sont deux walks disjointes de G allant respectivement de s_1 à t_1 et de s_2 à t_2, donc (G, s_1, t_1, s_2, t_2) est acceptant pour 2-Disjoint Paths. □

Remarque 36. Les bornes utilisées pour la répétition dans la requête pourraient aussi être $^{0..\infty}$ sans modifier significativement la preuve.

4.3 Variable sans répétitions imbriquées

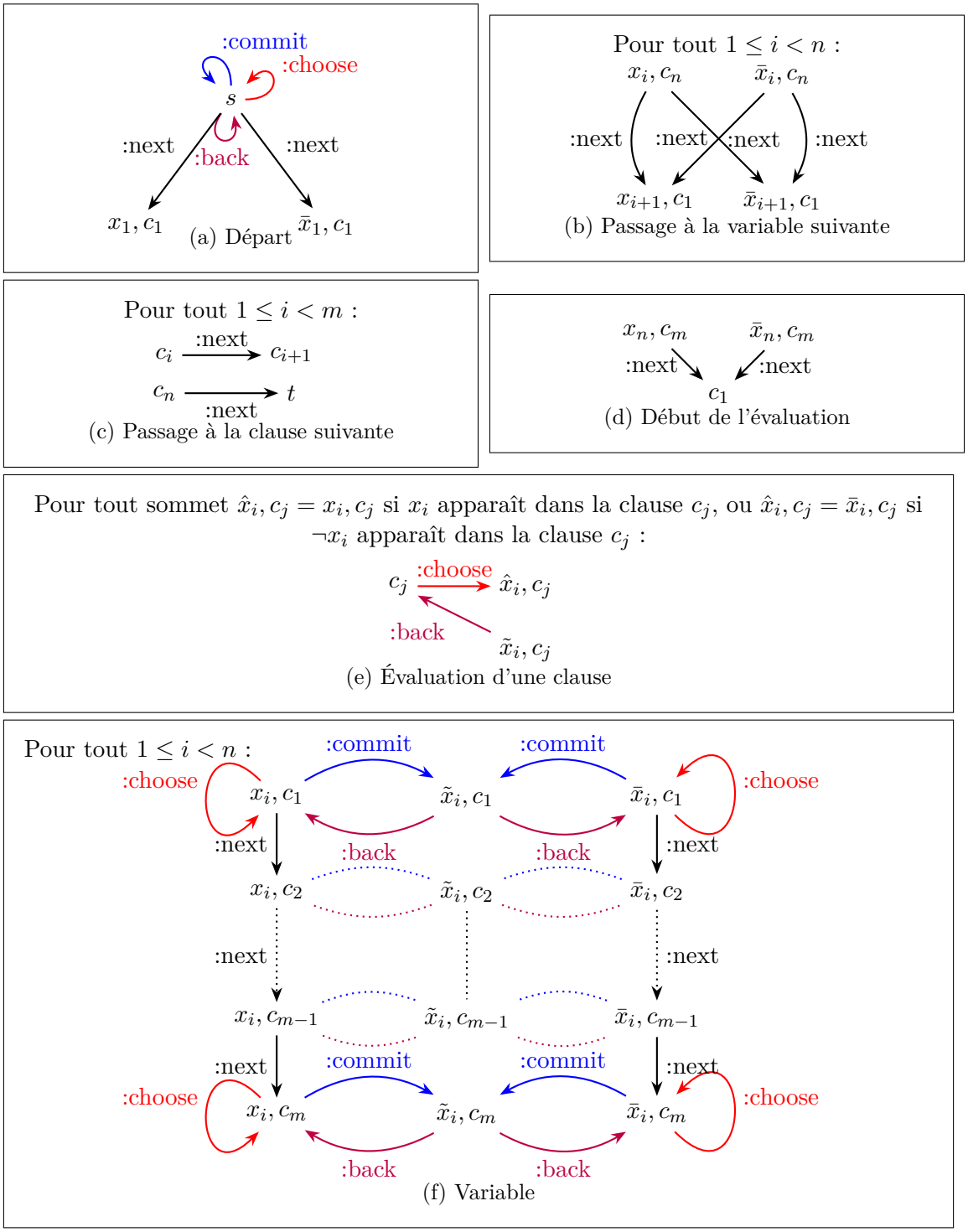
Les systèmes réels imposent des restrictions aux requêtes. Par exemple, GQL interdit les requêtes avec une hauteur d'étoile supérieure à 1 : la requête du Théorème 37 ne serait donc pas autorisée. On prouve que, même avec cette restriction, le problème est **NP**-difficile.

Théorème 37. Tuple Membership$_{\text{BT}}$ *est* ***NP****-difficile pour les expressions de GPC*$^-$*. C'est déjà vrai en complexité en données : il existe une requête Q pour laquelle* Tuple Membership$_{\text{BT}}(Q)$ *est* ***NP****-difficile.*

Preuve. Par réduction depuis 3-SAT.

On pose $\Sigma = \{\text{choose}, \text{commit}, \text{back}, \text{next}\}$, et $Q = ((u) \xrightarrow{:\text{choose}} () \xrightarrow{:\text{commit}} () \xrightarrow{:\text{back}} (u) \xrightarrow{:\text{next}} ())^*$ une expression sur Σ. On veut montrer que 3-SAT se réduit à $\textsc{Tuple Membership}_{\text{BT}}(Q)$.

Soit $I = c_1 \wedge \ldots \wedge c_m$ une instance de 3-SAT avec m clauses et n variables $x_1, \ldots, x_n$. On pose $G = (\Sigma, V, E_d, \text{src}, \text{tgt}, \lambda)$ la base de données composée des gadgets de la Figure 5. On trouvera une vue d'ensemble de la base de données en Annexe A.



(a) Départ (b) Passage à la variable suivante (c) Passage à la clause suivante (d) Début de l'évaluation (e) Évaluation d'une clause (f) Variable

FIGURE 5 – Gadgets du Théorème 37

On veut montrer que I est une instance acceptante de 3-SAT ssi (G, s, t) est une instance acceptante de $\textsc{Tuple Membership}_{\mathrm{BT}}(Q)$.

($\Rightarrow$) On suppose que I est acceptant pour 3-SAT.

On note Vars l'ensemble des variables $\{x_1, ..., x_n\}$. Soit $v : \text{Vars} \longrightarrow \{\top, \bot\}$ une valuation qui satisfait I. On note $v(x_k)$ la valuation affectée à la variable x_k telle que I est acceptant.

On note $\hat{x}_i = \begin{cases} x_i & \text{si } v(x_i) = \top \\ \bar{x}_i & \text{sinon} \end{cases}$ et $\bar{\hat{x}}_i = \begin{cases} x_i & \text{si } v(x_i) = \bot \\ \bar{x}_i & \text{sinon.} \end{cases}$

Comme I est acceptant, on peut fixer pour chaque clause une variable qui la rend vraie. On pose t la suite des témoins : pour toute clause c_j, t_j est égal à la variable qui rend la clause c_j vraie.

On pose les suites de chemins suivantes :

- pour tout $1 \leq i \leq n$, $1 \leq j \leq m$: $\pi_{x_i,c_j} = \langle \bar{\hat{x}}_i, c_j \rangle \xrightarrow{:\text{choose}} \langle \bar{\hat{x}}_i, c_j \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_i, c_j \rangle \xrightarrow{:\text{back}} \langle \bar{\hat{x}}_i, c_j \rangle (\bigstar_1)$
- pour tout $1 \leq i \leq n$, $1 \leq j < m$: $\pi_{\text{nextClauseInVar}_{i,j}} = \langle \bar{\hat{x}}_i, c_j \rangle \xrightarrow{:\text{next}} \langle \bar{\hat{x}}_i, c_{j+1} \rangle$
- pour tout $1 \leq i < n$: $\pi_{\text{nextVar}_i} = \langle \bar{\hat{x}}_i, c_m \rangle \xrightarrow{:\text{next}} \langle \bar{\hat{x}}_{i+1}, c_1 \rangle$
- pour tout $1 \leq i \leq n$: $\pi_{x_i} = \pi_{x_i,c_1} \cdot \pi_{\text{nextClauseInVar}_{i,1}} \cdot \pi_{x_i,c_2} \cdot \ldots \cdot \pi_{\text{nextClauseInVar}_{i,m-1}} \cdot \pi_{x_i,c_m}$
- pour tout $1 \leq j \leq m$: $\pi_{\text{checkClause}_j} = \langle c_j \rangle \xrightarrow{:\text{choose}} \langle \hat{x}_k, c_j \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_k, c_j \rangle \xrightarrow{:\text{back}} \langle c_j \rangle$, avec $x_k = t_j$ $(\bigstar_2)$.

On pose :

- $\pi_{\text{start}} = \langle s \rangle \xrightarrow{:\text{choose}} \langle s \rangle \xrightarrow{:\text{commit}} \langle s \rangle \xrightarrow{:\text{back}} \langle s \rangle \xrightarrow{:\text{next}} \langle \bar{\hat{x}}_1, c_1 \rangle$ $(\bigstar_3)$
- $\pi_{\text{vars}} = \pi_{x_1} \cdot \pi_{\text{nextVar}_1} \cdot \pi_{x_2} \cdot \ldots \cdot \pi_{\text{nextVar}_{n-1}} \cdot \pi_{x_n}$
- $\pi_{\text{clauses}} = \pi_{\text{checkClause}_1} \xrightarrow{:\text{next}} \pi_{\text{checkClause}_2} \xrightarrow{:\text{next}} \ldots \xrightarrow{:\text{next}} \pi_{\text{checkClause}_m}$.

On pose $\pi = \pi_{\text{start}} \cdot \pi_{\text{vars}} \xrightarrow{:\text{next}} \pi_{\text{clauses}} \xrightarrow{:\text{next}} \langle t \rangle$.

On remarque que $\text{src}(\pi) = \langle s \rangle$ et $\text{tgt}(\pi) = \langle t \rangle$. Il reste à montrer que π est acceptée par Q sous Binding Trail.

1. $\lambda(\pi) = \lambda(\pi_{\text{start}}) \cdot \lambda(\pi_{\text{vars}}) \cdot \text{next} \cdot \lambda(\pi_{\text{clauses}}) \cdot \text{next}$
   Or, par construction :
   - $\lambda(\pi_{\text{start}}) = \text{choose} \cdot \text{commit} \cdot \text{back} \cdot \text{next}$
   - $\lambda(\pi_{\text{vars}}) = \text{choose} \cdot \text{commit} \cdot \text{back} \cdot \text{next} \cdot \ldots \cdot \text{choose} \cdot \text{commit} \cdot \text{back}$
   - $\lambda(\pi_{\text{clauses}}) = \text{choose} \cdot \text{commit} \cdot \text{back} \cdot \text{next} \cdot \ldots \cdot \text{choose} \cdot \text{commit} \cdot \text{back}$

   On en déduit $\lambda(\pi) \in \lambda(Q)$.
2. On remarque que chaque nœud source d'une transition choose est aussi le nœud cible de la transition back suivante (voir $\bigstar_1$, $\bigstar_2$, et $\bigstar_3$), donc la variable u dans Q est correctement unifiée.
3. On remarque que π est une trail.

Finalement : $\pi \in [\![Q]\!]_{\mathrm{BT}}(G)$.

($\Leftarrow$) On suppose que (G, s, t) est acceptant pour $\textsc{Tuple Membership}_{\mathrm{BT}}(Q)$. Soit π une walk de $\langle s \rangle$ à $\langle t \rangle$ telle que $\pi \in [\![Q]\!]_{\mathrm{BT}}(G)$.

On rappelle que, par définition de Q :

a. $\lambda(\pi) \in (\text{choose} \cdot \text{commit} \cdot \text{back} \cdot \text{next})^*$
b. chaque nœud source d'une transition choose est aussi le nœud cible de la transition back suivante.

Par définition de G et Q, comme $\text{src}(\pi) = \langle s \rangle$, $\langle s \rangle \xrightarrow{:\text{choose}} \langle s \rangle \xrightarrow{:\text{commit}} \langle s \rangle \xrightarrow{:\text{back}} \langle s \rangle$ est un préfixe de π : on note $\pi_{\text{prefixe}} = \langle s \rangle \xrightarrow{:\text{choose}} \langle s \rangle \xrightarrow{:\text{commit}} \langle s \rangle \xrightarrow{:\text{back}} \langle s \rangle$.
On va prolonger π_{prefixe} par condition nécessaire. On dira qu'un nœud (respectivement chemin) σ est un *prochain nœud par* l (respectivement *prochain chemin par* l) si $\pi_{\text{prefixe}} \xrightarrow{:\text{l}} \sigma$ est aussi un préfixe de π. Dans ce cas, π_{prefixe} prendra la valeur de $\pi_{\text{prefixe}} \xrightarrow{:\text{l}} \sigma$.
On remarque que :

— soit $\langle x_1, c_1 \rangle$ est le prochain nœud par next

— soit $\langle \bar{x}_1, c_1 \rangle$ est le prochain nœud par next.

Soit $\langle \bar{\hat{x}}_1, c_1 \rangle \in \{\langle x_1, c_1 \rangle, \langle \bar{x}_1, c_1 \rangle\}$ tel que $\langle \bar{\hat{x}}_1, c_1 \rangle$ est le prochain nœud par next.
On pose $\pi_{x_1} = \langle \bar{\hat{x}}_1, c_1 \rangle \xrightarrow{:\text{choose}} \langle \bar{\hat{x}}_1, c_1 \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_1, c_1 \rangle \xrightarrow{:\text{back}} \langle \bar{\hat{x}}_1, c_1 \rangle \xrightarrow{:\text{next}} \langle \bar{\hat{x}}_1, c_2 \rangle \xrightarrow{:\text{choose}} \langle \bar{\hat{x}}_1, c_2 \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_1, c_2 \rangle \xrightarrow{:\text{back}} \langle \bar{\hat{x}}_1, c_2 \rangle \xrightarrow{:\text{next}} \dots \xrightarrow{:\text{next}} \langle \bar{\hat{x}}_1, c_m \rangle \xrightarrow{:\text{choose}} \langle \bar{\hat{x}}_1, c_m \rangle \xrightarrow{:\text{commit}} \langle \bar{\hat{x}}_1, c_m \rangle \xrightarrow{:\text{back}} \langle \bar{\hat{x}}_1, c_m \rangle$, et on remarque que, pour que la variable dans Q soit correctement unifiée, il faut que π_{x_1} soit le prochain chemin par next, car le nœud source d'une transition choose est aussi le nœud cible de la transition back suivante.
De la même façon, on pose les π_{x_i} pour tout $2 \leq i \leq n$, et on obtient que $\pi_{x_2} \xrightarrow{:\text{next}} \dots \xrightarrow{:\text{next}} \pi_{x_n}$ est le prochain chemin par next.
Par définition de G, $\langle \bar{\hat{x}}_n, c_m \rangle$ n'a qu'une arête sortante étiquetée par next, donc $\langle c_1 \rangle$ est le prochain nœud par next.
Soit $\langle x_\alpha, c_1 \rangle$, $\langle x_\beta, c_1 \rangle$ et $\langle x_\gamma, c_1 \rangle$ les trois nœuds qui ont une arête entrante étiquetée par choose ayant pour source $\langle c_1 \rangle$, et $\langle \hat{x}_{k_1}, c_1 \rangle \in \{\langle x_\alpha, c_1 \rangle, \langle x_\beta, c_1 \rangle, \langle x_\gamma, c_1 \rangle\}$ le prochain nœud par choose. On en déduit que $\langle \tilde{x}_{k_1}, c_1 \rangle \xrightarrow{:\text{back}} \langle c_1 \rangle$ est le prochain chemin par commit.
Comme π est conforme à la sémantique Binding Trail, $\langle \hat{x}_{k_1}, c_1 \rangle$ ne peut pas être un nœud par lequel passe $\pi_{x_{k_1}}$: en effet, si $\pi_{x_{k_1}}$ passait par $\langle \hat{x}_{k_1}, c_1 \rangle$, alors son unique arête sortante étiquetée par commit aurait déjà été consommée et ne serait plus utilisable.
Par définition de G, $\langle c_1 \rangle$ n'a qu'une arête sortante étiquetée par next. On en déduit que $\langle c_2 \rangle$ est le prochain nœud par next.
De la même façon, on obtient qu'il existe des entiers $1 \leq k_2, \dots, k_m \leq n$ tels que $\langle \hat{x}_{k_2}, c_2 \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_{k_2}, c_2 \rangle \xrightarrow{:\text{back}} \langle c_2 \rangle \xrightarrow{:\text{next}} \langle c_3 \rangle \xrightarrow{:\text{choose}} \dots \xrightarrow{:\text{next}} \langle c_m \rangle \xrightarrow{:\text{choose}} \langle \hat{x}_{k_m}, c_m \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_{k_m}, c_m \rangle \xrightarrow{:\text{back}} \langle c_m \rangle \xrightarrow{:\text{next}} \langle t \rangle$ est le prochain chemin par choose, et les $\langle \hat{x}_{k_i}, c_i \rangle$ n'apparaissent nulle part ailleurs dans π_{prefixe}.
On remarque que $\text{src}(\pi_{\text{prefixe}}) = \langle s \rangle$, $\text{tgt}(\pi_{\text{prefixe}}) = \langle t \rangle$, et que, par définition de G, $\langle t \rangle$ n'a aucune arête sortante. On en déduit que $\pi = \pi_{\text{prefixe}} = \langle s \rangle \xrightarrow{:\text{choose}} \langle s \rangle \xrightarrow{:\text{commit}} \langle s \rangle \xrightarrow{:\text{back}} \langle s \rangle \xrightarrow{:\text{next}} \pi_{x_1} \xrightarrow{:\text{next}} \dots \xrightarrow{:\text{next}} \pi_{x_n} \xrightarrow{:\text{next}} \langle c_1 \rangle \xrightarrow{:\text{choose}} \langle \hat{x}_{k_1}, c_1 \rangle \xrightarrow{:\text{commit}} \langle \tilde{x}_{k_1}, c_1 \rangle \xrightarrow{:\text{back}} \langle c_1 \rangle \xrightarrow{:\text{next}} \dots \xrightarrow{:\text{next}} \langle t \rangle$.
On remarque que pour toute variable x_k et pour toute clause $c_i \neq c_j$, $\langle \hat{x}_k, c_i \rangle$ et $\langle \hat{x}_k, c_j \rangle$ sont cohérents : si $\langle \hat{x}_k, c_i \rangle = \langle x_k, c_i \rangle$ alors $\langle \hat{x}_k, c_j \rangle = \langle x_k, c_j \rangle$, et si $\langle \hat{x}_k, c_i \rangle = \langle \bar{x}_k, c_i \rangle$, alors $\langle \hat{x}_k, c_j \rangle = \langle \bar{x}_k, c_j \rangle$. On note respectivement $\hat{x}_k = x_k$ ou $\hat{x}_k = \bar{x}_k$.
On définit la valuation v telle que $v(x_k) = \begin{cases} \top & \text{si } \hat{x}_k = x_k \\ \bot & \text{sinon.} \end{cases}$
Comme π impose que les $\pi_{x_{k_i}}$ ne passent pas par les $\langle \hat{x}_{k_i}, c_j \rangle$, cela impose que v soit une valuation rendant I vraie. En effet, si v ne rendait pas I vraie, alors il existerait une clause c_j dans I de la forme $(\tilde{x}_\alpha \vee \tilde{x}_\beta \vee \tilde{x}_\gamma)$ telle que $v(x_\alpha), v(x_\beta), v(x_\gamma)$ ne rendent pas c_j vraie, donc π passerait deux fois par les nœuds $\langle \hat{x}_\alpha, c_j \rangle, \langle \hat{x}_\beta, c_j \rangle, \langle \hat{x}_\gamma, c_j \rangle$, ce qui est impossible par construction de π. Finalement, I est une instance acceptante pour 3-SAT. □

Remarque 38. Avant d'obtenir le résultat présenté, on a obtenu le même résultat avec

des requêtes plus permissives, qui contenaient des disjonctions — imbriquées ou non — sous l'étoile. On trouvera en Annexe B les requêtes et les gadgets en question.

4.4 Variable hors des répétitions

Les deux théorèmes précédents utilisent de façon cruciale les variables sous les répétitions. On verra dans le Théorème 40 que les requêtes de GPC^0 ont une évaluation en temps polynomial dans la taille de la base de données. En revanche, on montre ici que le problème reste difficile en complexité en requête.

Théorème 39. Tuple Membership$_{\mathrm{BT}}$ *est* ***NP****-difficile pour les expressions de* GPC^0. *C'est déjà vrai pour les expressions de* GPC^{--} *et en complexité en requête : il existe une base de données* $\mathcal{D}$ *pour laquelle* Tuple Membership$_{\mathrm{BT}}(\mathcal{D})$ *est* ***NP****-difficile.*

Preuve. Par réduction depuis 3-SAT.
On pose $\mathcal{D}$ la base de données de la Figure 6.

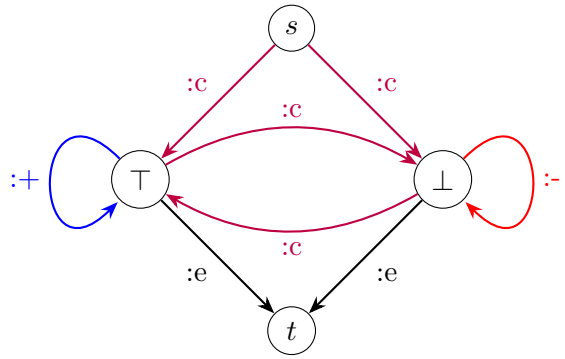


Figure 6 – Base de données utilisée dans la preuve du Théorème 39

Soit $I = c_1 \wedge \ldots \wedge c_m$ une instance de 3-SAT avec m clauses et n variables $x_1, \ldots, x_n$. On note $\tilde{x}_{i,j} \in \{x_i, \neg x_i\}$ l'atome x_i et son signe tels qu'ils apparaissent dans la clause c_j. On pose $Q_{\text{assign}} = (u_1) \xrightarrow{:c} \ldots \xrightarrow{:c} (u_n)$, et pour toute clause $c_i = (\tilde{x}_\alpha \vee \tilde{x}_\beta \vee \tilde{x}_\gamma)$, on pose

$$Q_{x_\alpha,i} = \begin{cases} (u_\alpha) \xrightarrow{:+} (u_\alpha) \xrightarrow{:c} (u_\beta) \xrightarrow{:c} (u_\gamma) & \text{si } \tilde{x}_{\alpha,i} = x_\alpha \\ (u_\alpha) \xrightarrow{:-} (u_\alpha) \xrightarrow{:c} (u_\beta) \xrightarrow{:c} (u_\gamma) & \text{si } \tilde{x}_{\alpha,i} = \neg x_\alpha \end{cases},$$

$$Q_{x_\beta,i} = \begin{cases} (u_\alpha) \xrightarrow{:c} (u_\beta) \xrightarrow{:+} (u_\beta) \xrightarrow{:c} (u_\gamma) & \text{si } \tilde{x}_{\beta,i} = x_\beta \\ (u_\alpha) \xrightarrow{:c} (u_\beta) \xrightarrow{:-} (u_\beta) \xrightarrow{:c} (u_\gamma) & \text{si } \tilde{x}_{\beta,i} = \neg x_\beta \end{cases},$$

$$Q_{x_\gamma,i} = \begin{cases} (u_\alpha) \xrightarrow{:c} (u_\beta) \xrightarrow{:c} (u_\gamma) \xrightarrow{:+} (u_\gamma) & \text{si } \tilde{x}_{\gamma,i} = x_\gamma \\ (u_\alpha) \xrightarrow{:c} (u_\beta) \xrightarrow{:c} (u_\gamma) \xrightarrow{:-} (u_\gamma) & \text{si } \tilde{x}_{\gamma,i} = \neg x_\gamma \end{cases},$$

$$\text{et } Q_i = Q_{x_\alpha,i} | Q_{x_\beta,i} | Q_{x_\gamma,i}.$$

On pose $Q = () \xrightarrow{:c} Q_{\text{assign}} \xrightarrow{:c} Q_1 \xrightarrow{:c} \ldots \xrightarrow{:c} Q_m \xrightarrow{:e} ()$.
On veut montrer que I est une instance acceptante de 3-SAT ssi (Q, s, t) est une instance acceptante de Tuple Membership$_{\mathrm{BT}}(\mathcal{D})$.

($\Rightarrow$) On suppose que I est acceptant pour 3-SAT.
On note Vars l'ensemble des variables $\{x_1, ..., x_n\}$. Soit $v : \text{Vars} \longrightarrow \{\top, \bot\}$ une valuation qui satisfait I. On pose $\nu(u_k) = \begin{cases} \langle \top \rangle & \text{si } v(x_k) = \top \\ \langle \bot \rangle & \text{sinon} \end{cases}$ la valuation de la variable u_k lors de

l'évaluation de Q sur $\mathcal{D}$.
On pose $\pi_{\text{assign}} = \langle s\rangle \xrightarrow{:\text{c}} \nu(u_1) \xrightarrow{:\text{c}} \dots \xrightarrow{:\text{c}} \nu(u_n)$ et, pour toute clause $c_i = (\tilde{x}_\alpha \vee \tilde{x}_\beta \vee \tilde{x}_\gamma)$, on pose

$$\pi_{u_\alpha,i} = \begin{cases} \nu(u_\alpha) \xrightarrow{:+} \nu(u_\alpha) \xrightarrow{:\text{c}} \nu(u_\beta) \xrightarrow{:\text{c}} \nu(u_\gamma) & \text{si } \tilde{x}_{\alpha,i} = x_\alpha \\ \nu(u_\alpha) \xrightarrow{:-} \nu(u_\alpha) \xrightarrow{:\text{c}} \nu(u_\beta) \xrightarrow{:\text{c}} \nu(u_\gamma) & \text{si } \tilde{x}_{\alpha,i} = \neg x_\alpha \end{cases},$$

$$\pi_{u_\beta,i} = \begin{cases} \nu(u_\alpha) \xrightarrow{:\text{c}} \nu(u_\beta) \xrightarrow{:+} \nu(u_\beta) \xrightarrow{:\text{c}} \nu(u_\gamma) & \text{si } \tilde{x}_{\beta,i} = x_\beta \\ \nu(u_\alpha) \xrightarrow{:\text{c}} \nu(u_\beta) \xrightarrow{:-} \nu(u_\beta) \xrightarrow{:\text{c}} \nu(u_\gamma) & \text{si } \tilde{x}_{\beta,i} = \neg x_\beta \end{cases},$$

$$\text{et } \pi_{u_\gamma,i} = \begin{cases} \nu(u_\alpha) \xrightarrow{:\text{c}} \nu(u_\beta) \xrightarrow{:\text{c}} \nu(u_\gamma) \xrightarrow{:+} \nu(u_\gamma) & \text{si } \tilde{x}_{\gamma,i} = x_\gamma \\ \nu(u_\alpha) \xrightarrow{:\text{c}} \nu(u_\beta) \xrightarrow{:\text{c}} \nu(u_\gamma) \xrightarrow{:-} \nu(u_\gamma) & \text{si } \tilde{x}_{\gamma,i} = \neg x_\gamma \end{cases}.$$

Comme I est acceptant pour 3-SAT, pour toute clause $c_i = (\tilde{x}_\alpha \vee \tilde{x}_\beta \vee \tilde{x}_\gamma)$, il existe au moins une variable dans $\{x_\alpha, x_\beta, x_\gamma\}$ dont la valuation rend c_i vraie. Ainsi, il existe au moins une walk $\pi_i \in \{\pi_{u_\alpha,i}, \pi_{u_\beta,i}, \pi_{u_\gamma,i}\}$ telle qu'elle est conforme aux $\nu(u_\alpha), \nu(u_\beta), \nu(u_\gamma)$: par exemple, si $\tilde{x}_{\alpha,i} = \neg x_\alpha$ et $\nu(x_\alpha) = \langle\bot\rangle$, on choisit $\pi_i = \pi_{u_\alpha,i}$, et on remarque qu'on peut effectivement prendre l'arête étiquetée par $-$ qu'on trouve dans $\pi_{u_\alpha,i}$.
Par construction des π_i et par définition de $\mathcal{D}$, pour tout $1 \leq i \leq m$, $\text{src}(\pi_i) \in \{\langle\top\rangle, \langle\bot\rangle\}$ et $\text{tgt}(\pi_i) \in \{\langle\top\rangle, \langle\bot\rangle\}$. Dans tous les cas, on peut construire $\langle s\rangle \xrightarrow{:\text{c}} \pi_1$ et $\pi_m \xrightarrow{:\text{e}} \langle t\rangle$, et pour tout $1 \leq j < m$ on peut construire $\pi_j \xrightarrow{:\text{c}} \pi_{j+1}$.
On peut donc construire $\pi = \langle s\rangle \xrightarrow{:\text{c}} \pi_1 \xrightarrow{:\text{c}} \pi_2 \xrightarrow{:\text{c}} \dots \xrightarrow{:\text{c}} \pi_m \xrightarrow{:\text{e}} \langle t\rangle$.
On remarque que $\text{src}(\pi) = \langle s\rangle$ et $\text{tgt}(\pi) = \langle t\rangle$. Il reste à montrer que π est acceptée par Q sous la sémantique Binding Trail.

1. $\lambda(\pi) = c \cdot \lambda(\pi_1) \cdot \dots \cdot \lambda(\pi_m) \cdot e$
   Or, pour tout $1 \leq i \leq m$ et pour tout $x \in \{x_\alpha, x_\beta, x_\gamma\}$, $\lambda(\pi_{x,i}) \in \lambda(Q_i)$. Donc par construction des π_i et de Q : $\lambda(\pi) \in \lambda(Q)$.
2. On remarque que, pour tout π_i ($1 \leq i \leq m$), le nœud source et le nœud cible d'une arête étiquetée par $+$ ou par $-$ sont les mêmes. De plus, si une variable $\tilde{x}_\alpha \in \{x_\alpha, \neg x_\alpha\}$ apparaît dans deux clauses $c_i \neq c_j$, $\pi_{u_\alpha,i}$ et $\pi_{u_\alpha,j}$ sont cohérents entre eux : la valeur de $\nu(u_\alpha)$ est la même dans $\pi_{u_\alpha,i}$ et dans $\pi_{u_\alpha,j}$. Donc les variables sont correctement unifiées.
3. Par construction de la requête, une même arête de π ne peut jamais être associée au même motif atomique d'arête de Q, donc π est une binding trail.

Finalement : $\pi \in [\![Q]\!]_{\text{BT}}(\mathcal{D})$.

($\Leftarrow$) On suppose que (Q, s, t) est acceptant pour Tuple Membership$_{\text{BT}}(\mathcal{D})$. Autrement dit, il existe une walk π dans $\mathcal{D}$ allant de $\langle s\rangle$ à $\langle t\rangle$ qui est acceptée par Q sous la sémantique Binding Trail.
On note $\nu(u_i) \in \{\langle\top\rangle, \langle\bot\rangle\}$ la valuation de la variable u_i dans π. Comme (Q, s, t) est acceptant pour Tuple Membership$_{\text{BT}}(\mathcal{D})$, il existe des sous-walks $\pi_1, \dots, \pi_m$ de π telles qu'elles satisfont les $Q_1, \dots, Q_m$. Autrement dit, pour tout $Q_i = Q_{x_\alpha,i}|Q_{x_\beta,i}|Q_{x_\gamma,i}$, les $\nu(u_\alpha), \nu(u_\beta), \nu(u_\gamma)$ sont cohérents et permettent de satisfaire Q_i, donc de satisfaire Q.
Pour tout $1 \leq k \leq n$, on définit la valuation v d'une variable de 3-SAT x_i telle que

$$v(x_k) = \begin{cases} \top & \text{si } \nu(x_k) = \langle\top\rangle \\ \bot & \text{sinon.} \end{cases}$$

Pour tout $i \in [1, m]$, π_i satisfait $Q_i = Q_{x_\alpha,i}|Q_{x_\beta,i}|Q_{x_\gamma,i}$ et la clause c_i est de la forme $(\tilde{x}_\alpha \vee \tilde{x}_\beta \vee \tilde{x}_\gamma)$. Supposons sans perte de généralité que π_i satisfait la clause $Q_{x_\alpha,i}$, et que $\tilde{x}_{\alpha,i} = x_\alpha$ (le cas $\tilde{x}_{\alpha,i} = \neg x_\alpha$ est symétrique). Alors $\nu(u_\alpha) = \langle\top\rangle$ car $Q_{x_\alpha,i}$ contient $\nu(u_\alpha) \xrightarrow{:+} \nu(u_\alpha)$. On en déduit $v(x_\alpha) = \top$. Ainsi, la valuation de $\tilde{x}_\alpha$ rend c_i vraie. On

en déduit que π impose que v soit une valuation rendant I vraie. Finalement, I est une instance acceptante pour 3-SAT. □

5 Résultats positifs

On va proposer une solution à Tuple Membership en temps polynomial dans la taille de la base de données. En revanche, la solution est exponentielle dans la taille de la requête, et ne peut pas être polynomiale d'après le Théorème 39. La solution s'appuie en particulier sur les Définitions 23 et 24.

Théorème 40. Tuple Membership$_{\mathrm{BT}}$ *est* ***PTIME*** *en données pour les expressions de GPC*^{0} *: pour toute requête Q conforme à GPC*^{0}*, il existe un algorithme résolvant* Tuple Membership$_{\mathrm{BT}}(Q)$ *en temps polynomial.*

Intuition de la preuve. Soit $\mathcal{D} = (\Sigma, V, E_d, \mathrm{src}, \mathrm{tgt}, \lambda)$ une base de données. On pose Q une requête conforme à GPC^0 à laquelle on a ajouté les motifs de nœud implicites. On note $|Q|$ le nombre de motifs atomiques de Q. Seules les variables qui apparaissent au moins deux fois, qu'on appellera « variables répétées », nous intéressent : on note v le nombre de variables répétées dans Q, et on remarque $v \leq |Q|$. Il y a donc au plus $(|V| + |E|)^v$ valuations possibles, et $(|V| + |E|)^v \leq (|V| + |E|)^{|Q|}$.
Pour chaque valuation, il suffit de l'encoder dans la base de données, puis d'évaluer la requête sans variable, c'est-à-dire comme une RPQ.

De la même façon, on montre qu'on peut résoudre Query Evaluation$_{\mathrm{BT}}$ avec un algorithme en délai polynomial dans la taille de la base de données ; autrement dit, Query Evaluation$_{\mathrm{BT}}$ est dans **DelayP** en complexité en données.

Théorème 41. Query Evaluation$_{\mathrm{BT}}$ *est* ***DelayP*** *en données pour les expressions de GPC*^{0} *: pour toute requête Q conforme à GPC*^{0}*, il existe un algorithme résolvant* Query Evaluation$_{\mathrm{BT}}(Q)$ *avec un précalcul polynomial et un délai entre deux solutions polynomial aussi.*

6 Conclusion

La sémantique Binding Trail a de bonnes propriétés lorsque les requêtes évaluées sont des RPQs, avec une énumération des résultats avec un précalcul et un délai polynomial. Ses bonnes propriétés sont perdues lorsqu'on ne considère plus des RPQs, mais soit des RPQs avec l'extension classique de répétition arbitraire, soit avec des requêtes d'un fragment de GQL, même très réduit, simplement avec la présence de variables répétées.

Bibliographie

[1] International Organization for Standardization, "GQL," https://www.iso.org/standard/76120.html, 2024, standard ISO/IEC CD 39075.

[2] W. Martens, M. Niewerth, and T. Popp, "A trichotomy for regular trail queries," *Logical Methods in Computer Science*, vol. 19, 2023.

[3] C. David, N. Francis, and V. Marsault, "Run-Based Semantics for RPQs," in *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, 8 2023, pp. 178–187. [Online]. Available : https://doi.org/10.24963/kr.2023/18

[4] I. F. Cruz, A. O. Mendelzon, and P. T. Wood, "A graphical query language supporting recursion," *ACM SIGMOD Record*, vol. 16, no. 3, pp. 323–330, 1987.

[5] N. Francis, A. Gheerbrant, P. Guagliardo, L. Libkin, V. Marsault, W. Martens, F. Murlak, L. Peterfreund, A. Rogova, and D. Vrgoc, "Gpc : A pattern calculus for property graphs," in *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '23. New York, NY, USA : Association for Computing Machinery, 2023, p. 241–250. [Online]. Available : https://doi.org/10.1145/3584372.3588662

[6] N. Francis, A. Gheerbrant, P. Guagliardo, L. Libkin, V. Marsault, W. Martens, F. Murlak, L. Peterfreund, A. Rogova, and D. Vrgoč, "A researcher's digest of gql," in *ICDT*, ser. LIPIcs, F. Geerts and B. Vandevoort, Eds., vol. 255. Dagstuhl, Germany : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 03 2023, invited talk. [Online]. Available : https://drops.dagstuhl.de/opus/volltexte/2023/17743

[7] S. Perifel, *Complexité algorithmique*. Ellipses, 2014.

[8] M. Y. Vardi, "The complexity of relational query languages (extended abstract)," in *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '82. New York, NY, USA : Association for Computing Machinery, 1982, p. 137–146. [Online]. Available : https://doi.org/10.1145/800070.802186

[9] Y. Strozecki, "Enumeration complexity : Incremental time, delay and space," Ph.D. dissertation, Université de Versailles Saint-Quentin-en-Yvelines, 2021.

[10] J. Bang-Jensen and G. Z. Gutin, *Digraphs : Theory, Algorithms and Applications*, ser. Springer Monographs in Mathematics. London : Springer London, 2009.

Annexe A Vue d'ensemble des gadgets du Théorème 37

On construit la vue d'ensemble en supposant $x_1 = \bot$, $x_2 = \top$, $x_n = \bot$, $c_1 = (\tilde{x}_\alpha \vee x_2 \vee \tilde{x}_\gamma)$ (avec $\tilde{x}_\alpha \in \{x_\alpha, \neg x_\alpha\}$ et $\tilde{x}_\gamma \in \{x_\gamma, \neg x_\gamma\}$, $1 \leq \alpha, \gamma \leq m$), et $v(x_2) \vDash c_1$.

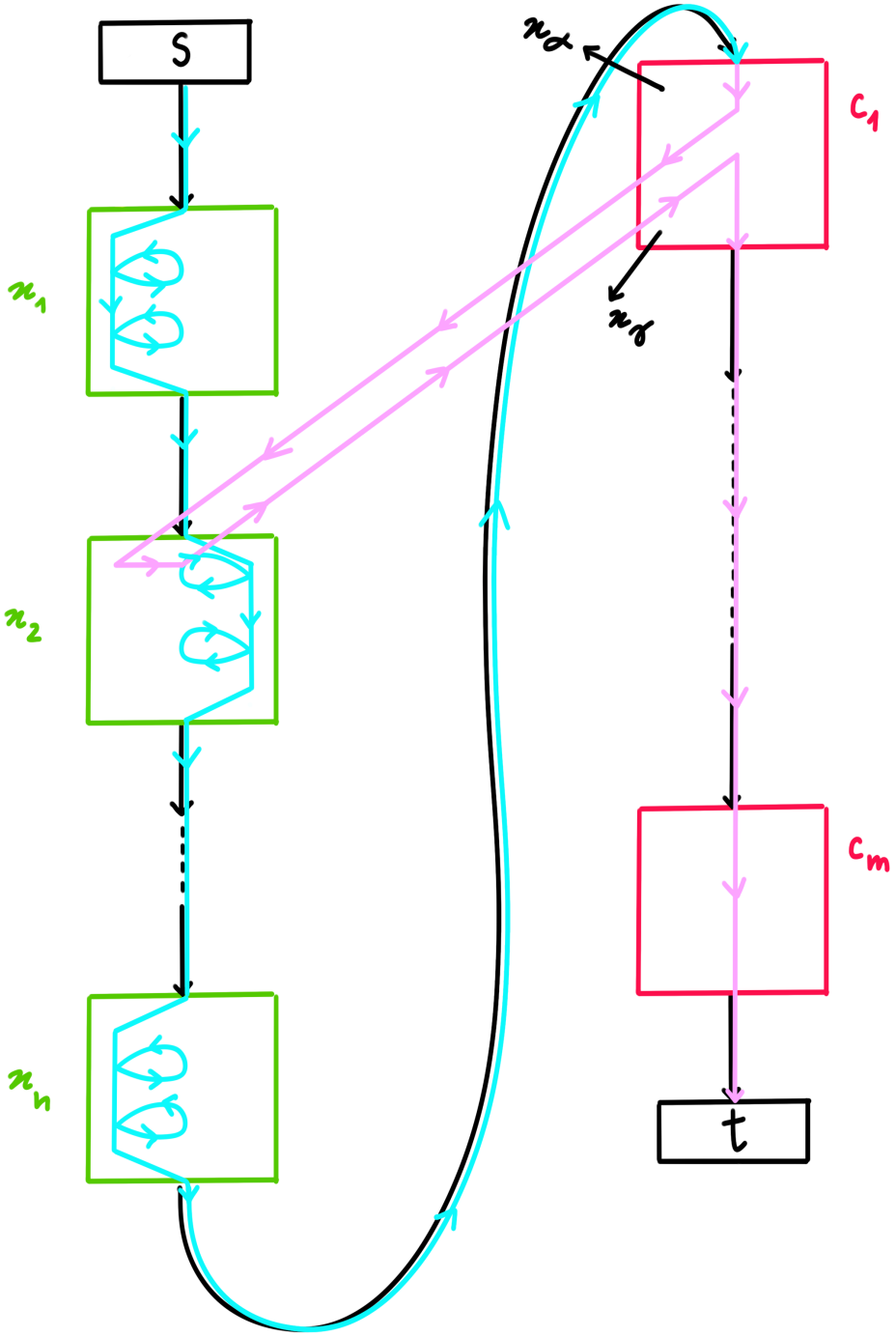


FIGURE 7 – Vue d'ensemble des gadgets du Théorème 37

Annexe B Variantes du Théorème 37

B.1 Avec unions imbriquées

$$
\begin{aligned}
Q'_{\text{nested}} = () &\xrightarrow{\text{:start}} (\\
&\quad (u) \xrightarrow{\text{:forbid}} ()\\
&\quad |(u) \xrightarrow{\text{:nextVar}} ()\\
&\quad |(\\
&\qquad (u)\\
&\qquad |(u) \xrightarrow{\text{:check}} ()\\
&\quad) \xrightarrow{\text{:witIn}} (\\
&\qquad () \xrightarrow{\text{:witOut}} (u) \xrightarrow{\text{:continue}} ()\\
&\qquad |() \xrightarrow{\text{:nextCheck}} (u) \xrightarrow{\text{:nextCheck}} ()\\
&\quad)\\
&)^*
\end{aligned}
$$

$s \xrightarrow{\text{:start}} \tilde{x}_{1,in}$

(a) Départ

Pour tout $1 \leq i < n$:

$\tilde{x}_{i,out} \xrightarrow{\text{:nextVar}} \tilde{x}_{i+1,in}$

(b) Passage à la variable suivante

$\tilde{x}_{n,out} \xrightarrow{\text{:nextVar}} c_1$

(c) Début de l'évaluation

Pour tout $1 \leq i < m$:

$c_i \xrightarrow{\text{:nextCheck}} c_{i+1}$

$c_m \xrightarrow{\text{:nextCheck}} t$

(d) Passage à la clause suivante

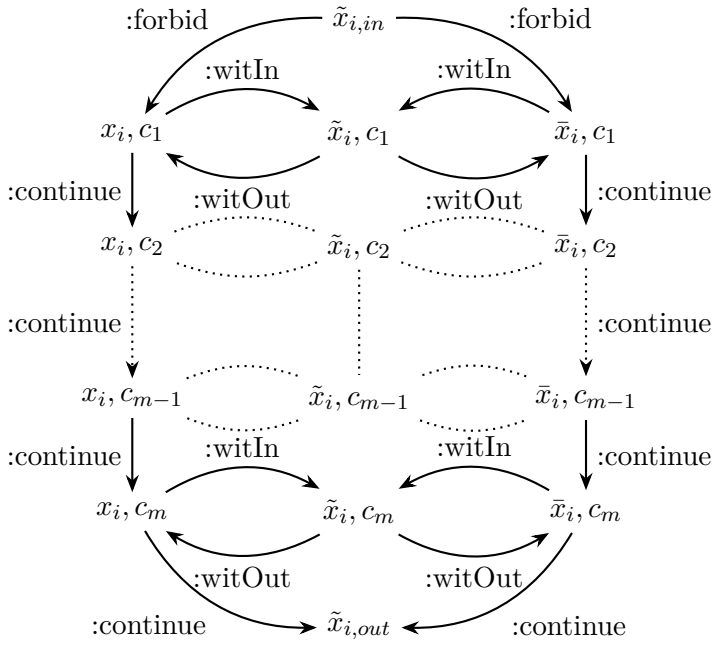
Pour tout sommet $\hat{x}_i, c_j = x_i, c_j$ si x_i apparaît dans la clause c_j, ou $\hat{x}_i, c_j = \bar{x}_i, c_j$ si $\neg x_i$ apparaît dans la clause c_j :

$c_j \xrightarrow{\text{:check}} \hat{x}_i, c_j$

$\tilde{x}_i, c_j \xrightarrow{\text{:nextCheck}}$

(e) Évaluation d'une clause

Pour tout $1 \leq i < n$:

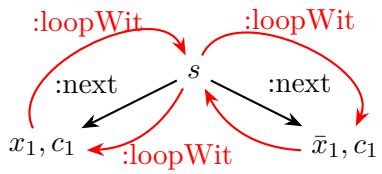


(f) Variable

FIGURE 8 – Gadgets de la réduction avec une requête contenant des disjonctions imbriquées

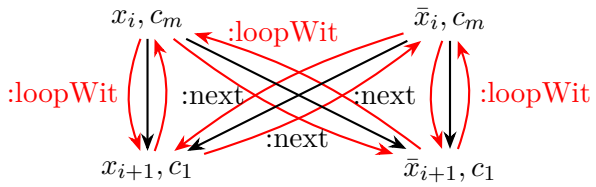
B.2 Avec unions non imbriquées

$$Q' = ((u) \xrightarrow{\text{:next}} (v) \xrightarrow{\text{:witIn}} (\\ () \xrightarrow{\text{:witOut}} (v) \xrightarrow{\text{:loopWit}} (u) \xrightarrow{\text{:loopWit}} (v) \\ |() \xrightarrow{\text{:nextCheck}} (u) \xrightarrow{\text{:loopCheck}} (v) \xrightarrow{\text{:loopCheck}} (u) \xrightarrow{\text{:nextCheck}} () \\ ))^*$$



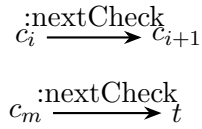
(a) Départ

Pour tout $1 \leq i < n$:

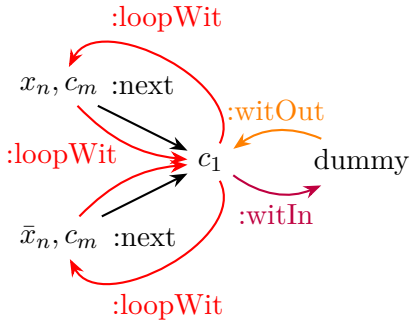


(b) Passage à la variable suivante

Pour tout $1 \leq i < m$:

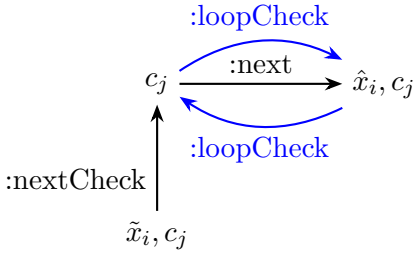


(c) Passage à la clause suivante



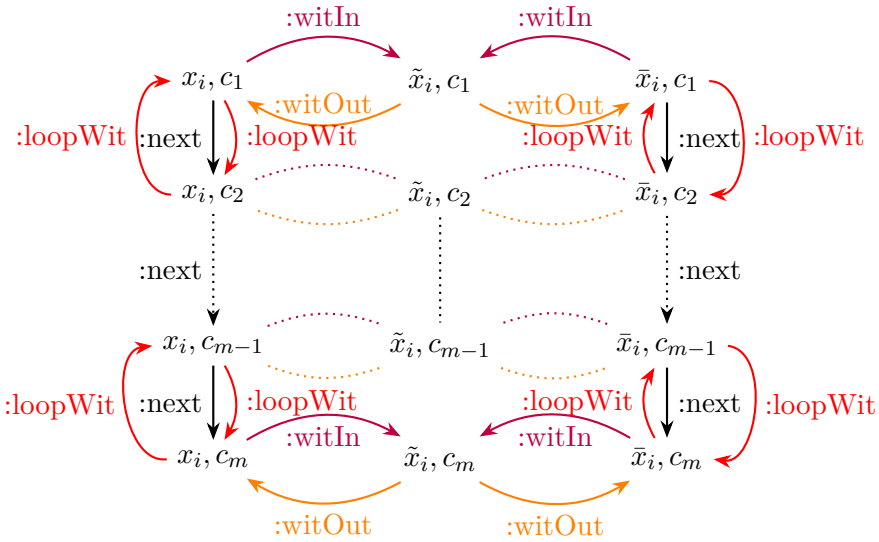
(d) Début de l'évaluation

Pour tout sommet $\hat{x}_i, c_j = x_i, c_j$ si x_i apparaît dans la clause c_j, ou $\hat{x}_i, c_j = \bar{x}_i, c_j$ si $\neg x_i$ apparaît dans la clause c_j :



(e) Évaluation d'une clause

Pour tout $1 \leq i < n$:



(f) Variable

FIGURE 9 – Gadgets de la réduction avec une requête contenant un niveau de disjonction