Dialogue entre la procédure du chase et la réécriture sur les mots

Lily Gallois

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LILLE

Opérée au sein de :
Laboratoire CRIStAL

Ecole Doctorale
Sciences pour l'ingénieur

Spécialité de doctorat :
Informatique

Soutenue publiquement le 19/12/2019, par :
Lily Gallois

Dialogue entre la procédure du chase et la réécriture sur les mots

---

Devant le jury composé de :

Evelyne CONTEJEAN CNRS, Université Paris-Sud	**Rapporteur.e**
Thomas GENET Université de Rennes	**Rapporteur.e**
Rémi GILLERON Université de Lille	**Examinateur.rice**
Marie-Laure MUGNIER Université de Montpellier	**Examinateur.rice**
Sophie TISON Université de Lille	**Directeur.rice de thèse**
Pierre BOURHIS CNRS, Université de Lille	**Co-directeur.rice de thèse**

Abstract

Graphs are a framework that is used to represent data of many different applications : transport, social network, proteins interaction, open data diffusion and its underlying structure. To manage these graphs, the user needs to express queries to extract data and to express properties to enforce integrity constraints or knowledge/ontology used to represent missing data in the graph. In this PhD thesis, we focus on two different questions : how to evaluate efficiently queries over graph and how to complete graphs to satisfy the properties? Our approach to optimize the evaluation of queries is to find a query semantically equal to the initial one but easier to evaluate. We focus at a useful subclass of queries : regular path queries. Indeed, path queries are the core of the classical queries over graphs, for example : "which are the cities reachable from each other by an exclusive bus travel?" In the logical optimisation of queries, one of the core problems is the problem of inclusion of a query into another. This problem is well studied for different classes of queries and paths queries; however, there are few works that took account integrity constraints satisfied by the graphs. In practice, this could lead to a very efficient optimisation : for the path query $x.a^+.y$ and the integrity constraints which impose that the set of edges labelled by a is closed under transitivity, we can prove that under the graphs satisfying this constraints, the queries xa^+y and $x.a.y$ are equivalent. One can remark that it is very efficient to answer this query. Our approach is to reduce the inclusion problem where integrity constraints are expressed by word constraints, to a problem of containment of regular languages by using string rewriting systems following the approach introduced by Grahne and Thomo. After proving that this reduction of Grahne and Thomo is partially wrong for finite graphs, we introduce a restriction over the rewriting systems guaranteeing the correctness of this reduction and the decidability of the inclusion. The key idea of our restriction is to bound the number of "parallel steps" for every derivation by a constant. We prove that checking if a system is uniformly-bounded by k is pspace-complete. The second problem is to complete the graph to satisfy the properties. This problem is well studied in the context of relational databases by diverse variants of a same procedure : the chase. Unfortunately, this procedure does create a possibly infinite model and not a finite one as we wish. The classical approach in this context is to check whenever the chase procedure terminates for every instance. However, this question is undecidable in general. Inspired by our uniform-bounded notion for rewriting systems developed in the previous part and a similar notion recently introduced for some chase procedures, we study different definitions of uniform boundedness for different kind of chases and we compare them to each other. Finally, we establish that the notions of uniform boundedness for the chase and the uniform boundedness for string rewriting systems are equivalent for a subclass of word constraints reinforcing the links between reasoning over graphs and reasoning over rewriting systems.

Résumé

Les graphes sont une manière de représenter les données et leur structure sous-jacente utilisée dans différentes applications : transport, réseaux sociaux, interaction de protéines, ... Pour gérer ces graphes, l'utilisateur a besoin d'interroger les données du graphe et d'exprimer des propriétés représentées par des contraintes satisfaites par le graphe ou des connaissances pour représenter les données manquantes du graphe. Dans cette thèse nous nous intéressons à deux questions particulières : comment évaluer efficacement des requêtes sur des graphes et comment compléter des graphes pour satisfaire les propriétés? Notre approche d'optimisation d'évaluation des requêtes consiste à déterminer d'autres requêtes sémantiquement égales à l'initiale mais plus faciles à évaluer, nous appliquons cette méthode sur les requêtes de chemin régulier (RPQ). Ces dernières sont au cœur des requêtes classiques de graphe comme : "quelles sont les villes reliées par un trajet uniquement en bus?" Pour l'optimisation de requêtes, un des problèmes essentiels est de décider l'inclusion d'une requête dans une autre. Ce problème a été largement étudié pour différentes classes de requêtes et de requêtes de chemin, toutefois peu de travaux ont pris en compte les contraintes d'intégrité satisfaites par les graphes. D'un point de vue pratique, cela peut conduire à des optimisations très efficaces : par exemple la requête xa^+y pour les graphes qui satisfont la contrainte imposant que l'ensemble des arêtes étiquetées par a est close par transitivité est équivalente à la requête $x.a.y$, elle, très facile à évaluer. Notre approche est de réduire le problème d'inclusion dans le cadre des contraintes de mots au problème d'inclusion de langages réguliers modulo les systèmes de réécriture de mots comme proposé par l'approche de Grahne et Thomo. Après avoir prouvé que la réduction de Grahne et Thomo est partiellement fausse dans le cas des graphes finis, nous introduisons une restriction sur les systèmes de réécriture qui assurent la réduction et la décidabilité du problème d'inclusion. L'idée principale de notre restriction et de borner le nombre "d'étapes parallèles" de toute dérivation par une constante donnée. Nous prouvons que tester si un système est uniformément borné par k est pspace-complet. Le second problème consiste à savoir comment compléter efficacement les graphes afin de satisfaire les propriétés données. Ce problème a été largement étudié dans le contexte des bases de données relationnelles pour diverses variantes d'une même procédure : le chase. Malheureusement cette procédure construit des modèles possiblement infinis contrairement à ce que nous souhaitons. L'approche classique pour répondre à ce problème est de vérifier si la procédure du chase termine pour toute instance, cependant cette question est indécidable en générale. En nous inspirant de la notion de borne uniforme des systèmes de réécriture développée précédemment et d'une notion similaire récemment introduite précédemment, nous étudions différentes définitions de borne uniforme pour plusieurs variantes de chase et nous comparons les classes obtenues entre elles. Finalement nous montrons que les notions de borne uniforme pour le chase et de borne uniforme pour les systèmes de réécriture de mots sont équivalentes pour une certaine sous-classe de contrainte de mots renforçant les dialogues entre les raisonnements sur les graphes et les raisonnements sur les systèmes de réécritures.

Table des matières

CHAPITRE 0

Introduction

Les données sont devenues omniprésentes dans notre société. Devant la masse de ces données, les méthodes pour les interroger et la diversité des structures pour les représenter ne cesse d'augmenter.

Le développement rapide des réseaux sociaux, des données des transports en commun, de la représentation des interactions entre protéines, ... ont rendu populaire la notion de base de données graphe.

Une base de données graphe est un modèle pratique pour stocker les données sous forme de nœuds et les mettre en relation par des arcs étiquetés. Dans un réseau social, si Claudie est la mère de Marius, on représente Claudie et Marius comme deux nœuds reliés avec un arc étiqueté par "mère".

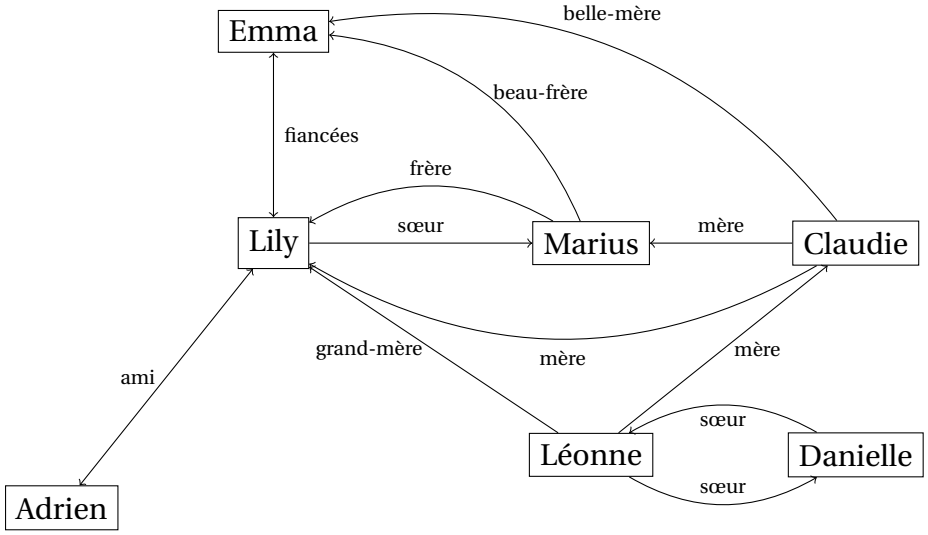


FIGURE 1 – Base de données graphe qui représente un réseau social

Pour utiliser les données, nous devons pouvoir les interroger, dans le contexte des bases de données graphe il existe plusieurs langages comme SparQL, Cypher,... [88] Ces langages utilisent des requêtes particulières qui expriment des propriétés sur les chemins : les Regular Path Queries (RPQ).

Ces requêtes permettent d'interroger le graphe avec des requêtes comme "Quels sont tous les trajets qui peuvent être effectués uniquement en bus ?". Dans la syntaxe de SparQL ces requêtes s'écrivent sous la forme :

SELECT ?x, ?y WHERE {**?x r ?y**}

où **r** est une expression régulière. Cette requête retourne les paires de nœuds du graphe qui sont reliés avec un chemin étiqueté par un mot satisfaisant l'expression

régulière. Par exemple si **r** est l'expression régulière bus$^+$, la requête renvoie toutes les paires de nœuds reliés par un chemin formé d'arêtes toutes étiquetées par "bus".

Malheureusement, les techniques classiques des bases de données relationnelles ne sont pas adaptées pour évaluer de façon optimisée ces requêtes. Une méthode consiste à les optimiser en évaluant une requête plus simple (par exemple une requête acyclique) équivalente à la première.

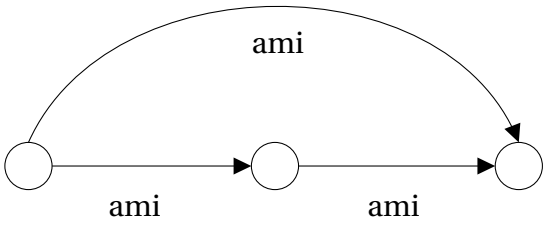
Nous utilisons des connaissances sur nos bases de données graphes pour optimiser plus efficacement les requêtes mais aussi pour compléter des données afin que les modèles satisfassent certaines propriétés. Par exemple, si on considère la clôture transitive par la locution "les amis de mes amis sont mes amis" comme contrainte dans un réseau social, nous pouvons évaluer la requête $x.\text{ami}.y$ au lieu de la requête $x.\text{ami}^+.y$ car il suffit de vérifier avec qui x est directement ami au lieu de rechercher "les amis d'amis". On représente ces connaissances par des contraintes d'intégrité.

Les contraintes d'intégrité les plus largement utilisées sont les "tuple generating dependencies" ou *tgd* en abrégé. Elles sont formées d'une tête et d'un corps et correspondent à des contraintes logiques de la forme "corps implique tête". Par exemple, les deux contraintes suivantes :

$$\forall x, \text{est}(x) \longrightarrow \exists z, \text{est}(z), \text{parent}(z, x)$$

$$\forall x, y, z, \text{fille}(x, y), \text{sœur}(y, z) \rightarrow \text{nièce}(x, z)$$

Les contraintes de mots sont régulièrement utilisées pour l'optimisation de requêtes RPQ. Une contrainte de mots est une expression de la forme $xuy \sqsubseteq xvy$, elle est satisfaite par une base de données graphe si pour tout chemin étiqueté par u entre deux nœuds, il existe un chemin entre ces mêmes nœuds étiqueté par v. Par exemple la contrainte "les amis de mes amis sont mes amis" est une contrainte de mots : dès que deux arcs étiquetés par "ami" se succèdent, il existe un arc étiqueté par "ami" qui relie ces nœuds.



Une des questions clés pour optimiser les requêtes est le problème d'inclusion sous contraintes : on teste si l'ensemble des réponses d'une requête RPQ est inclus dans l'ensemble des réponses d'une requête RPQ pour toute base de données qui satisfait les contraintes.

Il existe des cas où le problème d'inclusion est connu comme étant décidable par exemple si les requêtes sont appliquées à partir d'un nœud particulier (racine) [3, 7]. On souhaite généraliser dans le cas où les requêtes peuvent être appliquées depuis n'importe où.

L'étude de l'inclusion et de l'optimisation de requêtes RPQ sous contraintes de mots a été peu étudiée dans la littérature. Il a été montré que le problème d'inclusion est indécidable en général mais peu de classes de contraintes de mots assurant la décidabilité ont été proposées.

L'approche classique pour résoudre ce problème est de le réduire au problème d'inclusion de langages sous réécriture de mots [57]. À chaque contrainte de mots $xuv \sqsubseteq xvy$ on peut associer la règle de réécriture $u \to v$.

Les règles de réécriture permettent de construire des dérivations à partir d'un mot donné. À partir d'un mot x, si un des facteurs est le membre gauche d'une règle de réécriture, alors on peut remplacer ce facteur par le membre droit de la règle. Par exemple, si on considère la règle $ab \to bb$, le mot aab contient le facteur ab qu'on peut remplacer par bb pour obtenir le nouveau mot abb, à nouveau ce mot présente un facteur ab et on peut donc réécrire le mot abb en bbb, puis la règle ne peut plus être appliquée.

De nombreux problèmes sur les systèmes de réécriture ont été largement étudiés. Par exemple, le problème de terminaison "Est ce qu'il existe une dérivation infinie?" est indécidable. Mais aussi, les problèmes de la préservation des réguliers (auquel nous portons une attention particulière) "Pour tout langage régulier, est ce que l'ensemble des descendants est régulier?et "pour tout langage régulier, est ce que l'ensemble des ancêtres est régulier?" sont indécidables.

Il est possible de réduire le problème d'inclusion RPQ sous contraintes de mots au problème d'inclusion de langages réguliers [57], nous montrons que cette réduction dan le cas où les modèles sont finis n'est valable que sous hypothèse :

Sous cette hypothèse, une RPQ de langage L est incluse dans une RPQ de langage L' sous contraintes de mots si, et seulement si le langage L est inclus dans les ancêtres de L' par le système de réécriture associé aux contraintes de mots.

Ainsi, si les systèmes de réécriture d'une classe préservent les réguliers (pour les ancêtres) alors il suffit de tester l'inclusion de deux langages réguliers.

En général, les requêtes ne sont pas évaluées sur des bases de données qui satisfont toujours les contraintes d'intégrité. On souhaite alors compléter ces bases de données par des faits supplémentaires pour les satisfaire. Cette complétion a été largement utilisée à travers la procédure du *chase* qui comporte de nombreuses variantes (oblivious, semi-oblivious, standard, ...).

Le principe est d'ajouter une donnée lorsque celle-ci est manquante pour une contrainte en ajoutant la tête si le corps est vérifié. Par exemple, considérons la requête $\forall x, y, z, \text{fille}(x, y), \text{sœur}(y, z) \to \text{nièce}(x, z)$, la base de données (figure 1) contient

les faits fille(Claudie, Léonne) et sœur(Léonne, Danielle), on peut donc ajouter le fait nièce(Claudie, Danielle) à notre base de connaissance.

Cette procédure présente un inconvénient majeur : elle ne termine pas toujours. Par exemple si on considère la contrainte $\forall x, \text{est}(x) \longrightarrow \exists z, \text{est}(z), \text{parent}(z, x)$ sur la base de données constituée uniquement de est(Adrien) alors une donnée **NEW** est ajoutée avec les relations est(**NEW**) et parent(**NEW**, Adrien). Mais les contraintes ne sont toujours pas satisfaites : nous devons compléter à nouveau, on ajoute alors une nouvelle donnée **NEW2** et les faits est(**NEW2**) et parent(**NEW2**, **NEW**). Les contraintes ne sont toujours pas satisfaites, et cette procédure continue indéfiniment.

Le problème de la terminaison consiste à décider si la procédure du chase pour des contraintes tgd complète les données manquantes en un nombre fini d'étapes pour toute base de données.

Le problème de la terminaison du chase est indécidable en général, et de nombreuses classes de contraintes ont été proposées pour assurer sa terminaison. Par exemple, la classe des weakly-acyclic, en évitant la récursion, assure la terminaison.

Une des difficultés est de proposer des contraintes qui présentent un bon compromis entre résoudre les problèmes précédents avec une bonne complexité et une grande expressivité.

Notre contribution Nous souhaitons étudier les contraintes de mots dans deux problématiques : le problème d'inclusion de requêtes RPQ sous contraintes et la complétion des bases de données par la procédure du chase pour satisfaire les contraintes. Notre approche est inspirée d'un parallèle entre la complétion d'un automate qui correspond à un pas de réécriture parallèle avec la complétion par l'oblivious-chase. Nous définissons des classes qui ont un bon comportement avec ces complétions. Nous découpons cette étude en trois chapitres :

Au chapitre 3 nous étudions la réduction du problème d'inclusion de RPQ sous contraintes au problème de la préservation des réguliers. Nous montrons que la réduction proposée dans [57] n'est pas valide dans le cas des modèles finis.

Nous proposons alors une condition suffisante qui assure cette réduction : les systèmes doivent être descendant-limités. Un système de réécriture est descendant-limité si tout mot admet un nombre fini de descendants. En particulier, tous les systèmes de réécriture qui terminent sont descendants-limités.

Nous souhaitons donc proposer des classes de systèmes de réécriture descendant-limités qui préservent la régularité qui rendent décidable le problème d'inclusion des requêtes sous contraintes.

Nous montrons que de manière générale :

Le problème d'inclusion de RPQ sous contraintes de mots est indécidable.

Nous montrons ensuite dans ce chapitre un résultat de dureté du problème d'inclusion pour la classe des systèmes de réécriture non-récursifs (NR) :

Le problème d'inclusion de RPQ sous contraintes de mots NR est EXPSPACE-difficile.

Au chapitre 4, nous introduisons la notion de complexité parallèle d'une dérivation. Un pas de réécriture parallèle consiste à modifier deux facteurs (ou plus) simultanément. Par exemple $aabab$ avec la règle $ab \to bb$ peut se réécrire en une étape parallèle $abbbb$, qui lui-même peut se réécrire en $bbbbb$, il n'est pas possible d'obtenir moins de deux étapes parallèles pour réécrire $aabab$ en $bbbbb$.

La réécriture parallèle a été étudiée en lambda calcul et en réécriture des termes [72, 12, 14]. Mais dans ce manuscrit nous nous intéressons uniquement à la réécriture parallèle sur les mots.

Toute dérivation séquentielle peut être transformée par permutation des pas de réécriture en une dérivation de pas de réécriture parallèle de longueur minimale. La complexité parallèle d'une dérivation est le longueur de cette dérivation. Par exemple la dérivation $aabab \to^* bbbbb$ est de complexité parallèle 2.

Nous étendons la notion de complexité parallèle aux systèmes de réécriture : la complexité parallèle d'un système de réécriture R est la plus grande complexité parallèle des dérivations construites par R.

Nous proposons les classes $\mathrm{MB}_{\max}(k)$ des systèmes de réécritures dont la complexité parallèle est $\leq k$ (i.e. toutes les dérivations sont équivalentes à faire moins de k pas parallèles) et $\mathrm{BPar}(k)$ la classe des systèmes où chaque dérivation peut être simulée par une dérivation de complexité parallèle $\leq k$ (tous les descendants sont accessibles par une dérivation de moins de k pas parallèles). Tout système de ces deux classes est descendant-limité et préserve la régularité. Nous montrons que si les contraintes de mots correspondent à un système de réécriture de $\mathrm{MB}_{\max}(k)$ ou $\mathrm{BPar}(k)$ alors le problème d'inclusion de requêtes sous contraintes est EXPSPACE. Nous montrons :

Le problème d'inclusion de RPQ sous contraintes de mots correspondant à un système $\mathrm{MB}_{\max}(k)$ (ou $\mathrm{BPar}(k)$) est EXPSPACE-complet.

Nous proposons un algorithme pour décider si un système de réécriture admet une complexité parallèle $\leq k$:

Décider si un système de réécriture est $\mathrm{MB}_{\max}(k)$ est PSPACE-complet.

Dans le chapitre 5, nous souhaitons comparer la complexité parallèle avec la procédure du chase en base de données.

Nous utilisons la notion de contraintes tgd uniformément bornées :

On attache une profondeur aux faits des bases de données qui mesure le nombre d'étapes nécessaires pour produire ce fait. Si un fait est dans la base de données de départ, sa profondeur vaut 0. Ensuite si une règle tgd $B \rightarrow H$ produit un fait f, alors la profondeur de f est égale à 1 plus le maximum de la profondeur des faits qui ont permis de construire f [35, 34].

Par exemple si le fait ami(Adrien, Vincent) est de profondeur 0 et ami(Lily, Adrien) de profondeur 1 la procédure du chase sous la contrainte "les amis de mes amis sont mes amis" i.e. $\forall x, y, z, \text{ami}(x, y), \text{ami}(y, z) \rightarrow \text{ami}(x, z)$ complète les données. Avec le fait ami(Lily, Vincent) de profondeur égale à $1 + \max\{0, 1\}$ soit 2.

Un ensemble de contraintes tgd est uniformément borné par k si pour toute base de données la procédure du chase ne produit jamais de faits de profondeur $> k$.

On montre que :

> Déterminer si un système de contraintes tgd est uniformément borné par k pour l'oblivious et le semi-oblivious chase est Co-NEXPTIME-complet.

La notion de borne uniforme est étudiée en stratégie breadth-first : on impose que s'il existe un fait f de profondeur k qui peut être produit sur une base de données qui contient déjà un fait de profondeur k, alors on applique la contrainte qui construit f, sinon on applique une contrainte quelconque (si possible) qui construit un fait de profondeur $k+1$.

Nous montrons que (en parallèle à [17]) que si tout fait produit contient au moins une nouvelle valeur (les contraintes sont totales) alors le problème d'être uniformément borné est le même en breadth-first et en général :

> Si les contraintes tgd sont **totales**, on a équivalence entre être uniformément borné par k et être uniformément bornée par k en breadth-first (pour l'oblivious et semi-oblivious)

Nous montrons (aussi en parallèle à [17]) que :

> Décider si des contraintes de mots **totales** sont uniformément bornées est équivalent à décider la terminaison du chase (oblivious et semi-oblivious)

L'objectif final du chapitre est de comparer les classes $\text{MB}_{\text{max}}(k)$ et $\text{BPar}(k)$ avec les systèmes uniformément bornés. On montre en l'occurrence que :

> Si un système de réécriture a une complexité parallèle $\leq k$ alors les contraintes de mots associées sont uniformément bornées par k (oblivious).

De même nous montrons que :

Tout système de réécriture associé à des contraintes de mots uniformément bornées par k est dans BPar(k)

Enfin nous montrons que si les contraintes de mots sont totales on a équivalence entre être uniformément borné par k et avoir une complexité parallèle $\leq k$.

Un système de contraintes de mots **totales** est uniformément borné par k (oblivious) si et seulement si le système de réécriture associé a une complexité parallèle $\leq k$.

Ainsi, dans le cadre des contraintes de mots totales, décider si l'oblivious-chase termine est équivalent à décider s'il existe k tel que le système de réécriture associé est $\text{MB}_{\text{max}}(k)$

Plan de la thèse Nous proposons dans ce paragraphe une courte présentation du plan autour duquel a été articulé ce document. Nous avons fait le choix de découper la thèse en 7 chapitres :

- **Chapitre 0** : Il s'agit de la présente introduction.
- **Chapitre 1** : Nous introduisons sous formes de préliminaires les notions et notations utilisées tout au long du document
- **Chapitre 2** : Dans cette partie nous mettons en relation les travaux de la thèse avec la littérature actuelle.
- **Chapitre 3** : Nous discutons du problème d'inclusion de RPQ sous contraintes de mots : réduction du problème à un problème d'inclusion de langages, indécidabilité du problème, dureté.
- **Chapitre 4** : Nous introduisons la notion de complexité parallèle des systèmes de réécritures, définissons les classes $\text{MB}_{\text{max}}(k)$ et BPar(k) puis proposons un algorithme d'appartenance à $\text{MB}_{\text{max}}(k)$. Nous appliquons ces concepts pour décider l'inclusion de requêtes RPQ sous contraintes de mots.
- **Chapitre 5** : Nous comparons plusieurs classes de systèmes tgd uniformément bornées, puis nous comparons ces classes avec les systèmes de réécriture $\text{MB}_{\text{max}}(k)$ et BPar(k) dans le cadre des contraintes de mots.
- **Chapitre 6** : Conclusion et perspectives.

CHAPITRE 1

Préliminaires

Ce chapitre est à vocation de rappels. Nous découpons le chapitre en plusieurs parties :

- **Ensembles, relations et graphes** : nous rappelons les notions les plus élémentaires sur les relations et donnons quelques notations utiles au document.
- **Langages** : Cette section permet de fixer les notations sur les langages que nous utiliserons ainsi que les automates et les relations rationnelles.
- **Systèmes de réécriture et réécriture parallèle** : Cette section donne quelques rappels sur les systèmes de réécritures de mots, en particulier nous préciserons des notations sur les dérivations très utiles au chapitre 4. Enfin nous expliciterons la notion de pas de réécriture parallèle largement utilisée dans le manuscrit.
- **Base de données graphes, requêtes RPQ et contraintes de mots** : cette section introduit la notion de requêtes régulières de graphe et le problème d'inclusion.
- **Les trois sections suivantes** (schéma relationnel et instance, tuple generating dependencies et Datalog, Chase et dérivations) permettent de rappeler les notions sur la procédure du chase et sa terminaison, les notions de ces sections seront essentielles au chapitre 5.
- **Contraintes de mots** : dans cette dernière section nous expliciterons la notion de contraintes de mots, en tant que contraintes RPQ, mais aussi en tant que tgd. Nous expliciterons comment elles sont associées un système de réécriture. Les contraintes de mots sont présentent tout au long du document, elles permettent le dialogue entre les requêtes RPQ, le chase et les systèmes de réécritures.

Enfin nous supposons le lecteur familier avec la théorie de la complexité : machine de Turing et complexité temporelle et spatiale. Lorsque nous utilisons des entiers, ils sont toujours considérés en unaire.

1.1 Ensembles, relations et graphes

Nous fixons les notation sur les ensembles que nous utilisons dans ce document :
— On dénote par $\#E$ le cardinal de E quand E est un ensemble fini.
— Parfois nous utilisons l'expression classe à la place d'ensemble, bien que ce terme généralise la notion d'ensemble, dans ce manuscrit les classes seront toujours des ensembles.

.

Définition 1

Soient E et F deux ensembles.
— Une relation entre E et F (ou sur $E \times F$) est un sous-ensemble de $E \times F$.
— Une relation sur E est une relation entre E et lui-même.

Une relation d'équivalence est une relation étant :
— réflexive $(\forall x, (x,x) \in R)$
— symétrique $(\forall x, y, (x,y) \in R \implies (y,x) \in R)$
— transitive $(\forall x, y, z, (x,y) \in R, (y,z) \in R \implies (x,z) \in R)$

Une relation d'ordre est une relation étant :
— réflexive $(\forall x, (x,x) \in R)$
— antisymétrique $(\forall x, y, (x,y) \in R, (y,x) \in R \implies x = y)$
— transitive $(\forall x, y, z, (x,y) \in R, (y,z) \in R \implies (x,z) \in R)$

Une relation d'ordre est totale si pour tout $(x,y) \in E^2$ on a soit $(x,y) \in R$, soit $(y,x) \in R$. i.e. tous les éléments sont comparables.

De même une relation d'ordre est dîtes bien fondée si toute partie non vide de E admet un minimum (x est minimal dans $A \subseteq E$ s'il n'existe pas de $y \in A$ tel que $(y,x) \in R$).

Soient E, F et G trois ensembles, si R est une relation entre E et F et si S est une relation entre F et G alors on construit une relation T entre E et G noté $S \circ R$ en posant :

$$(x,z) \in T \text{ si et seulement s'il existe } y \text{ tel que } (x,y) \in R, (y,z) \in S$$

On peut composer une relation R avec elle même. On note alors R^2 au lieu de $R \circ R$. De même on note :
— R^{i+1} pour $R \circ R^i$ (ou de manière égale $R^i \circ R$)
— $R^{\leq k}$ la relation $\cup_{i=0}^{k} R^i$
— R^* la relation $\cup_{i \geq 0} R^i$
— $R^+ = \cup_{i \geq 1} R^i$

De même on peut définir les puissances négatives d'une relation :

Si R est une relation alors la relation inverse de R notée R^{-1} et la relation définie par :

$$(x, y) \in R \text{ si et seulement si } (y, x) \in R^{-1}$$

Nous introduisons la notion de relation compatible avec une loi de monoïde :

Définition 2

Soient E un monoïde de loi $\star$ et R une relation sur E. On dit que R est compatible avec $\star$ si pour tous u, v de E et (x, y) de R on a :

$$(u \star x \star v, u \star y \star v) \in R$$

Pour toute relation R sur un monoïde $(E, \star)$, il existe une unique plus petite relation compatible avec $\star$ contenant R on l'appelle la clôture par congruence de R.

Nous introduisons la notion de graphe :

Définition 3

Un graphe (orienté) est un couple (N, E) où N est un ensemble fini appelé ensemble des nœuds et E est une partie de $N \times N$ appelée ensemble des arêtes de G.

Nous précisons les notions de base sur les graphes utilisées dans ce document :

Définition 4

Soient $G = (N, E)$ et $G' = (N', E')$ deux graphes. On dit que G et G' sont isomorphes s'il existe une bijection φ (appelée isomorphisme de graphe) de N dans N' telle que :

$$\forall (x, y) \in N^2, (x, y) \in E \iff (\varphi(x), \varphi(y)) \in E'$$

L'inverse de G, noté G^{-1} est le graphe défini par (N', E') où $N' = N$ et

$$E' = \{(y, x), (x, y) \in E\}$$

L'application $G \mapsto G^{-1}$ est un isomorphisme de graphe.

Définition 5

Un chemin dans un graphe orienté G est une séquence finie de la forme $(n_i, n_{i+1})_{1 \leq i \leq t}$ d'arêtes de G. On dit qu'un chemin va de n à n' (ou est entre n et n') si $n_1 = n$ et $n_t = n$.

Un chemin est cyclique s'il existe i et j tels que $i \neq j$ et $n_i = n_j$.
Si un chemin n'est pas un cycle, t est appelé la longueur du chemin.
Un graphe orienté est acyclique s'il n'existe pas de cycle dans le graphe.
Dans un graphe acyclique on peut construire une relation d'ordre totale compatible avec la relation du graphe appelée tri topologique.

Définition 6

Soit $G = (N, E)$ un graphe acyclique. Un tri topologique de G est une relation d'ordre totale $\prec$ telle que :

$$\forall x, y \in n, (x, y) \in E \implies x \prec y$$

Un graphe enraciné est un graphe qui contient un nœud, appelé racine, qui n'admet pas d'arête entrante i.e. il n'existe pas d'arête de la forme (x, r) dans E.

Définition 7

Soit G un graphe acyclique et enraciné. On définit la profondeur d'un nœud comme la longueur du plus long chemin de la racine à ce nœud. On note $\text{depth}_G(n)$ la profondeur de n.
Si un tel chemin n'existe pas la profondeur de ce nœud est $+\infty$ par convention.

1.2 Langages

Un alphabet est un ensemble fini non vide. Les éléments d'un alphabet sont appelés lettres.

Définition 8

À tout alphabet Σ on désigne par Σ^* l'ensemble des mots de Σ. C'est l'ensemble des séquences finies d'éléments de Σ. On écrira $a_1 a_2 ... a_n$ au lieu de $(a_1, a_2, ..., a_n)$. On note ϵ le mot vide i.e. la séquence $()$.
On note $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. On munit Σ^* d'une loi de composition interne associative et unitaire définie par :

$$. : (a_1 a_2 ... a_n, b_1 b_2 ... b_m) \mapsto a_1 a_2 ... a_n b_1 b_2 ... b_m$$

On introduit quelques notations usuelles sur les mots.

Définition 9

Soit m un mot de Σ^*, on désigne par $|m|$ la longueur de m et $m[p]$ la p-ième lettre de m.

Définition 10

Un préfixe d'un mot u est un mot v tel qu'il existe w un mot qui vérifie $u = vw$. L'ensemble des préfixes d'un mot u est noté $\text{Pref}(u)$.

Un langage est un ensemble, éventuellement vide ou infini, de mots.

Toutes les notations sur les mots sont étendues aux langages. Par exemple, $\text{Pref}(L)$ désigne l'ensemble des préfixes des mots de L.

Nous introduisons quelques rappels sur la notion d'automate.

Définition 11

Un automate non-déterministe A est un uplet de la forme $(Q, \Sigma, \delta, I, F)$ où Q est un ensemble fini, dit ensemble d'états, Σ un alphabet, δ est une partie de $Q \times \Sigma \times \Sigma$ appelée ensemble des transitions, I et F sont des sous-ensembles de Q appelés respectivement états initiaux et état finaux. Si I est un singleton et δ est une fonction de $Q \times \Sigma \to Q$ alors A est dit déterministe.

Si A est un automate déterministe, on peut étendre δ aux mots avec δ^* de $Q \times \Sigma^*$ dans Q définie par :

— $\delta(q, \epsilon) = q$

— $\delta^*(q, a) = \delta(q, a)$

— $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$

Nous introduisons la notion de mot reconnu par un automate par la notion de run :

Définition 12

Soit $A = (Q, \Sigma, \delta, I, F)$ un automate.

— Un run sur A est une suite d'états et de lettres de Σ de la forme $(q_0, a_0, ..., a_n, q_{n+1})$ tel que pour tout $i \leq n$, $(q_i, a_i, q_{i+1}) \in \delta$

— Un run est acceptant si $q_0 \in I$ et $q_{n+1} \in F$

— Si un run $(q_0, a_0, ..., a_n, q_{n+1})$ est acceptant alors on dit que A reconnaît le mot $a_0...a_n$.

On écrira un run sous la forme : $q_0 a_0 ... a_n q_{n+1}$.

Soit L un langage, on dit que L est reconnu par A si A reconnaît exactement les mots de L. Soit A un automate, le langage reconnu par A est noté $L(A)$. Un langage est régulier si et seulement s'il est reconnu par un automate.

Définition 13

La taille d'un automate noté $|A|$ est l'entier $\#Q$.

On remarquera que puisque δ est une partie de $Q \times \Sigma \times Q$, considérer le nombre de transitions ou le nombre d'états ne modifiera pas les classes de complexités des problèmes que nous considérons.

On définit maintenant la notion de relation rationnelle :

Définition 14

La classe des relations rationnelles **RAT** est la plus petite classe des relations de $\Sigma^* \times \Sigma^*$ telle que :

— **RAT** contient les relations finies et le vide

— **RAT** est close par union, concaténation et étoile [a]

a. on entend ici par étoile, l'étoile du monoïde $(\Sigma^* \times \Sigma^*, .)$.

On a [39] :

Proposition 15

Les relations rationnelles sont closes par composition.

1.3 Système de réécriture et réécriture parallèle

Nous introduisons la notion de système de réécriture, nous sous-entendons dans ce manuscrit que tous les systèmes de réécritures sont sur les mots.

Définition 16

Un système de réécriture est une relation finie sur Σ^*.
On suppose dans ce document que tous les systèmes de réécriture sont **sans mot vide**, donc des relations finies sur Σ^+.

On introduit la notion de taille d'un système :

Définition 17

La taille d'un système de réécriture R est l'entier $|R|$ défini par

$$\sum_{(u,v)\in R} |u| + |v|$$

Nous introduisons la notion de pas de réécriture :

Définition 18

Pour tout système de réécriture R, on dénote par $\to_R$ la clôture par congruence de R, appelée "pas de réécriture".
S'il n'y a pas d'ambiguïté on notera $\to$ au lieu de $\to_R$.

Exemple 19

Si $R = \{(ab, b), (aa, bb)\}$ alors on a :

$$aabbaa \to_R abbaa$$

Par définition de la clôture par congruence : à tout couple de mot (x, y) tel que $x \to_R y$ il existe un unique quadruplet (u, v, μ, μ') de mots tels que $x = \mu u \mu'$, $y = \mu v \mu'$ et $(u, v) \in R$.

Si on dénote par p l'entier $|\mu|+1$ i.e. la position de la première lettre de x utilisée par le pas de réécriture, on précise alors des annotations sur $\rightarrow$:

$$x \overset{p,(u,v)}{\longrightarrow} y$$

.

On étend la relation $\rightarrow$ par sa clôture transitive $\rightarrow^*$.

Définition 20

On appelle dérivation une séquence (potentiellement infinie) de mots $\sigma = (x_0, x_1, ..., x_n, ...)$ telle que pour tout i on a :

$$x_i \rightarrow x_{i+1}$$

On a alors $x_0 \rightarrow^* x_n$.
On écrira parfois abusivement $\sigma = x \rightarrow^* y$ pour dire qu'il existe une dérivation $(x_0, ..., x_n)$ telle que $x_0 = x$ et $x_n = y$.

Si $\sigma = x \rightarrow^* y$ alors x est l'origine de σ notée orig(σ) et y est le but de σ noté aim(σ).

Si la séquence $\sigma = (x_0, ..., x_n, ...)$ est finie, alors l'indice du dernier élément est appelée la longueur de la dérivation.

Comme pour $\rightarrow$ on étend les annotations aux dérivations :

$$\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$$

Nous avons la proposition facile suivante :

Proposition 21

Pour tout système de réécriture R :

$$\rightarrow_R^{-1} = \rightarrow_{R^{-1}}$$

On définit facilement les dérivations sur le système de réécriture inverse :

Définition 22

Soit R un système de réécriture. On considère une dérivation :

$$\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$$

Alors :

$$x_n \xrightarrow{p_{n-1},(v_{n-1},u_{n-1}))} x_{n-1} \ldots \xrightarrow{p_0,(v_0,u_0)} x_0$$

est une dérivation sur R^{-1} appelée dérivation inverse de σ. On note σ^{-1} la dérivation inverse de σ

Toute dérivation sur R^{-1} est l'inverse d'une dérivation sur R
Nous introduisons la notion d'ancêtres (resp. de descendants) d'un langage L.

Définition 23

Soit L un langage et soit R un système de réécriture.

- Les descendants de L par R (noté $\mathrm{Desc}_R(L)$) est l'ensemble des mots m tel qu'il existe $w \in L$ tel que $w \rightarrow^* m$
- Les ancêtres de L par R (noté $\mathrm{Anc}_R(L)$) est l'ensemble des descendants de L par le système R^{-1}.

Nous évoquons quelques problèmes de décisions utiles au document : le problème de terminaison, le problème du mot et le problème de la préservation des réguliers.

Définition 24

Un système de réécriture est terminant s'il n'a pas de dérivation infinie. On note **TERM** la classe des systèmes de réécriture qui terminent.

On a d'après [37]

Théorème 25

Le problème de terminaison est indécidable.

Soit R un système de réécriture, le problème du mot consiste à décider si pour un couple de mots (u, v) il existe une dérivation d'origine u et de but v.

Il est montré dans [86] que :

Théorème 26

Le problème du mot est indécidable.

Le problème de l'arrêt dans les systèmes de réécriture de mot consiste à décider si étant donné un mot u et un système de réécriture R, toutes les dérivations dont l'origine est u sont finies.

Théorème 27

Le problème de l'arrêt est indécidable.

Nous introduisons la classe CF des systèmes de réécriture hors-contexte (context-free) :

Définition 28

Un système de réécriture R est hors-contexte si pour toute règle $(u, v) \in R$ on a $|u| = 1$.

La préservation des réguliers se définit par :

Définition 29

Soit R un système de réécriture.
R préserve les langages réguliers si pour tout langage régulier L, $\mathrm{Desc}_R(L)$ est régulier.
On dénote par **PrREG** la classe des systèmes de réécriture qui préservent les langages réguliers.

Comme R préserve les réguliers si l'image par $\longrightarrow^*$ d'un langage régulier est régulier on a d'après [39] :

Théorème 30

Pour tout système de réécriture R, si $\rightarrow_R^* \in$ **RAT** alors $R \in$ **PrREG**.

Nous introduisons la notion de réécriture parallèle et d'orthogonalité.

Définition 31

Soit R un système de réécriture. Soit $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ une dérivation.
On dit que deux pas de réécritures consécutifs $x_{k-1} \rightarrow x_k$ et $x_k \rightarrow x_{k+1}$ sont orthogonaux si l'ensemble des positions de lettres produits par la première règle est disjoints de l'ensemble des positions des lettres qui vont être utilisées par la seconde règle.

Nous adoptons ici la notation de *multi-step rewriting* issue de [14] : $\multimap$.

Définition 32

Soit R un système de réécriture. On dit qu'un mot x se réécrit parallèlement en y et on note $x \multimap_R y$ si x et y peuvent être décomposés sous la forme $x = \mu_0 u_1 \mu_1 ... \mu_{n-1} u_n \mu_n$ et $y = \mu_0 v_1 \mu_1 ... \mu_{n-1} v_n \mu_n$ tel que pour tout $i \leq n$ on a $(u_i, v_i) \in R$.

Lorsque le contexte est clair, $\multimap$ pourra être adoptée au lieu de $\multimap_R$
Contrairement à la relation $\rightarrow_R$, $\multimap$ est réflexive. De plus on a $\rightarrow_R \subseteq \multimap$.
On en déduit que :

Proposition 33

Pour tout système de réécriture, $\rightarrow_R^* = \multimap^*$.

Les ancêtres d'un langage L sont aussi les mots w tel qu'il existe m dans L avec $w \multimap^* m$.
Enfin, on a pour tout système R : $\multimap_{R^{-1}} = \multimap_R^{-1}$.

1.4 Base de données graphes, requêtes RPQ et contraintes de mots

Nous définissons la notion de base de données graphe, ce sont des graphes étiquetés sur lesquels nous allons interroger nos requêtes. Ils sont supposés finis en général mais nous discuterons sur des modèles infinis chapitre au 3.

Définition 34

On appelle graphe étiqueté par un alphabet Σ, ou **base de données graphe**, un couple (N, E) avec N ensemble fini de nœuds et E est un sous-ensemble de $N \times \Sigma \times N$.

À tout graphe étiqueté (N, E) par Σ on peut associer de manière naturelle un graphe orienté (N', E') en posant :

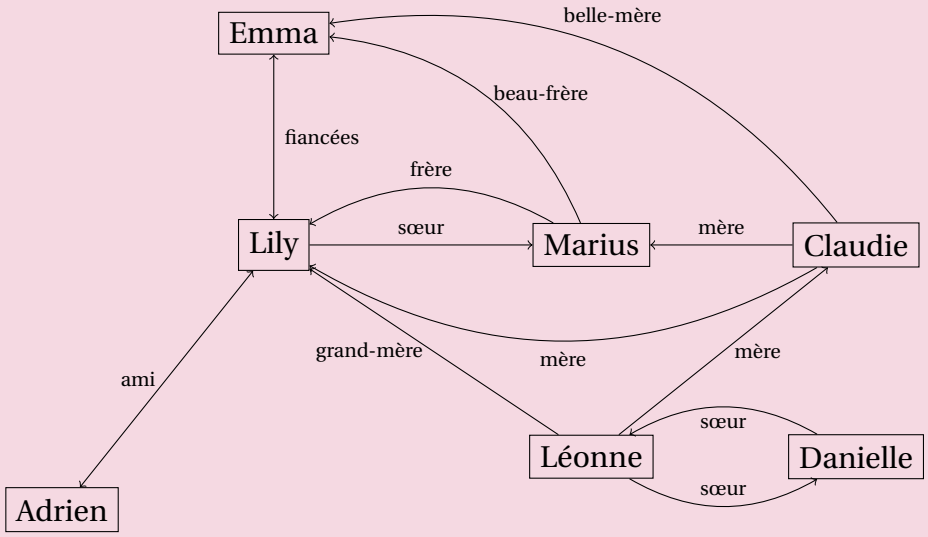
$$N' = N$$

$$E' = \{(x, y), \exists a \in \Sigma, (x, a, y) \in E\}$$

Ainsi on peut étendre les notions sur les graphes aux bases de données graphes comme les chemins ou l'acyclicité. Nous employons parfois le terme graphe ou le terme modèle dans le sens de "base de données graphe".

Exemple 35

Soit Σ = {frère, beau-frère, fiancées, sœur, mère, amis, belle-mère}.
On a par exemple la base de données graphe [a] sur Σ avec les nœuds N = {Emma, Lily, Marius, Claudie, Adrien, Léonne, Danielle} tels que :



a. Toute ressemblance avec des personnes existantes ou ayant existé est purement fortuite.

Nous introduisons les notions utiles pour définir les requêtes RPQ, ce sont les requêtes que nous étudions pour interroger nos modèles.

Définition 36

On associe à tout chemin $(n_i, a_i, n_{i+1})_{i \leq t}$ le mot $a_1 ... a_t$ appelé étiquette du chemin.

Exemple 37

Dans l'exemple 35, le chemin

((Marius,frère,Lily), (Lily,sœur,Marius), (Marius,frère,Lily))

est étiqueté par le mot frère.sœur.frère

Nous définissons des bases de données graphe particulières constituées d'un unique chemin maximal :

Définition 38

Un graphe-mot sur Σ est un graphe $G = (N, E)$ où E est de la forme :

$$\{(n_1, a_1, n_2), (n_2, a_2, n_3), ..., (n_t, a_t, n_{t+1})\}$$

Les nœuds $n_i \in N$ sont distincts et $a_i \in \Sigma$.

Soit $w = a_1...a_t$ un mot de Σ^*, on associe à w un graphe $G = (N, E)$ noté G_w défini par :

— N est un ensemble de la forme $\{n_1, ..., n_{t+1}\}$
— $E = \{(n_1, a_1, n_2), (n_2, a_2, n_3), ..., (n_t, a_t, n_{t+1})\}$

Nous introduisons la notion de requête de chemin régulier RPQ (regular path query).

Définition 39

Une requête RPQ Q est une expression de la forme xLy où L est un langage régulier.

On peut interroger les bases de données graphe avec les requêtes RPQ :

Définition 40

Soit G une base de donnée graphe. La réponse d'une RPQ $Q = xLy$ sur G notée $Q(G)$ est l'ensemble des couples de nœuds (n, n') tels qu'il existe un chemin de n à n' étiqueté par un mot de L.

Exemple 41

Considérons la base de données G de l'exemple 35.
On considère la RPQ

$$Q = x(\text{sœur} \mid \text{frère})^* y$$

On a $Q(G) = \{$
(Lily,Lily),
(Lily,Marius),
(Marius,Lily),
(Marius,Marius),
(Léonne,Léonne),
(Danielle,Léonne),
(Léonne, Danielle),
(Danielle, Danielle) $\}$

On dit que Q est incluse dans Q' et on dénote par $Q \sqsubseteq Q'$ si pour toute base de données graphe G on a $Q(G) \subseteq Q'(G)$.

D'après [24] on a le théorème suivant :

Théorème 42

Soient $Q = xLy$ et $Q' = xL'y$ deux RPQ.

$$Q \sqsubseteq Q' \text{ ssi } L \subseteq L'$$

Décider si $Q \sqsubseteq Q'$ est PSPACE-complet car l'inclusion de deux langages réguliers sous forme d'automates non déterministes est PSPACE-complète.

Nous nous intéressons dans ce document à l'inclusion de requêtes quand les graphes vérifient des contraintes de mots :

Définition 43

On appelle contrainte de mots une inclusion de la forme $xuy \sqsubseteq xvy$ où u et v sont deux mots.

Un ensemble de contraintes de mots $\mathscr{C}$ est toujours considéré fini.

Un graphe étiqueté G satisfait $\mathscr{C}$, on note $G \models \mathscr{C}$, si pour toute contrainte de mots $xuy \sqsubseteq xvy$, on a $U(G) \subseteq V(G)$ où $U = x\{u\}y$ et $V = x\{v\}y$.

Autrement dit pour chaque contrainte $xuy \sqsubseteq xvy$, de $\mathscr{C}$ s'il existe un chemin de n à n' étiqueté par u alors il existe un chemin entre n et n' étiqueté par v.

Enfin on introduit le problème d'inclusion de requêtes RPQ sous contraintes de mots :

Définition 44

Le problème d'inclusion de requêtes RPQ sous contraintes de mots $\mathscr{C}$ dénoté :

$$Q \sqsubseteq_{\mathscr{C}} Q'$$

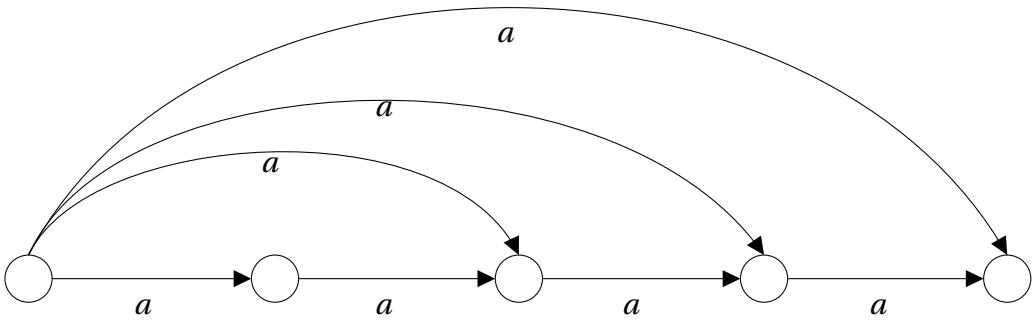
est défini par le problème

$$\forall G, G \models \mathscr{C} \implies (Q(G) \subseteq Q'(G))$$

Exemple 45

On considère $xaay \sqsubseteq xay$ comme constrainte de $\mathscr{C}$.
On a $xa^+y \sqsubseteq_{\mathscr{C}} xay$.

En effet s'il existe un chemin γ étiqueté par a^n avec $n \geq 2$, comme le graphe satisfait la contrainte. Le premier et le troisième nœud du chemin sont reliés par une arête étiqueté par a, donc il existe un chemin étiqueté par a^{n-1}. Par récurrence il existe donc un chemin entre le premier nœud et le dernier nœud de γ étiqueté par a.



1.5 Schéma relationnel et instance

On introduit la notion de schéma relationnel.

Définition 46

Un schéma relationnel S est un couple (r, arity) où r est un ensemble fini appelé ensemble des prédicats, et arity est une fonction de r dans $\mathbb{N}$ appelée fonction d'arité de S.

Si $S = (r, \text{arity})$ alors nous écrivons $A \in S$ au lieu de $A \in r$. Aussi $\#S$ désigne $\#r$.
Soit **Const** un ensemble dénombrable appelé ensemble des constantes.

Définition 47

Une instance est une fonction qui à tout prédicat R de S associe un sous-ensemble de $\mathbf{Const}^{\text{arity}(R)}$.

On l'image de l'instance et l'instance elle-même : si I est une instance on préférera écrire $R(a_1, ..., a_n) \in I$ au lieu de $(a_1, ..., a_n) \in I(R)$. Les éléments des instances sont appelés des faits. Enfin, si a_i est une constante d'un fait f, on écrira de manière abusive $a_i \in f$.

Exemple 48

Soient
- $S = (r = \{A, B\}, \text{arity} = \{A \mapsto 2, B \mapsto 3\})$
- $\{n, n', n'', n'''\} \subseteq \mathbf{Const}$

Alors

$$I = \{A(n, n'''), B(n'', n'', n')\}$$

est une instance (sur S).

Soit I une instance, le domaine de I, noté $\text{Dom}(I)$ est l'ensemble des constantes a telles qu'il existe un fait $f \in I$ avec $a \in f$. Autrement dit c'est l'ensemble des constantes qui apparaissent dans I.

Remarque 49

Il arrivera que l'on confonde la notion de base de données graphes et d'instance lorsque les prédicats sont d'arité 2 (en particulier pour les contraintes de mots) :
- À toute base de données graphe $G = (N, E)$ on associe l'instance $I = \{a(x, y), (x, a, y) \in E\}$. On préférera prendre la notation majuscule $A(x, y)$ au lieu de $a(x, y)$.
- Inversement à toute instance I d'arité 2 on associe une base de données graphe $G = (N, E)$ où $N = \text{Dom}(I)$ et $E = \{(x, a, y), A(x, y) \in I\}$.

Nous définissons la notion d'homomorphisme d'instance.

Définition 50

Soient I et J deux instances. Un homomorphisme de I vers J est une application de $\text{Dom}(I)$ dans $\text{Dom}(J)$ telle que pour tout fait $A(a_1, ..., a_n)$ de I alors $A(h(a_1), ..., h(a_n))$ est un fait de J.

On dénotera par $h(A(a_1, ..., a_n))$ le fait $A(h(a_1), ..., h(a_n))$.
Soit **Var** un ensemble dénombrable appelé ensemble des variables.

Définition 51

À tout prédicat A de S on peut associer un élément de $(\mathbf{Var} \cup \mathbf{Const})^{\text{arity}(R)}$. Si $(x_1, ..., x_n)$ sont les variables et les constantes associées à A alors on écrira $A(x_1, ..., x_n)$, Un tel élément est appelé un atome.

Exemple 52

Supposons que $S = (r, \text{arity})$ et que $A \in r$ avec $\text{arity}(A) = 3$. soient $x_1, x_2, x_3 \in \mathbf{Var} \cup \mathbf{Const}$ alors $A(x_1, x_2, x_3)$ est un atome.

Soit E un ensemble d'atomes. On note $\text{Dom}(E)$ l'ensemble des variables et des constantes qui apparaissent dans E.

On peut étendre la notion d'homomorphisme entre deux instances aux homomorphismes entre un ensemble d'atomes et une instance :

Définition 53

Soient E un ensemble d'atomes et I une instance, un homomorphisme de E dans I est une application de $\mathrm{Dom}(E)$ dans $\mathrm{Dom}(I)$ valant l'identité sur les constantes, et telle que pour tout atome $A(x_1,...,x_n)$ de E on a $A(h(x_1),...h(x_n)) \in I$.

On prendra garde que deux atomes peuvent partager des mêmes variables.

Exemple 54

Considérons deux prédicats A et B de S d'arité respective 2 et 3. On considère x_1, x_2, y_1, y_2 comme des variables distinctes. Les éléments $A(x_1, x_2)$ et $B(y_1, y_2, x_1)$ sont deux atomes. Notons E l'ensemble de ces deux atomes.
Soient a, b, c des constantes distinctes.
Il n'existe pas d'homomorphisme de E dans l'instance $\{A(a,b), B(a,b,c)\}$ car on aurait $h(x_1) = a$ (par A) et $h(x_1) = c$ (par B), contradiction.
Par contre il existe un homomorphisme de E dans l'instance $\{A(a,a), B(a,a,a)\}$.

1.6 Tuple Generating Dependency et Datalog

Soit S un schéma relationnel.

Définition 55

Une règle tgd [a] (tuple generating dependency) est une formule logique du premier ordre de la forme :

$$\forall x_1, ..., x_n \forall y_1, ..., y_p B(x_1, ..., x_n, y_1, ..., y_p) \rightarrow \exists z_1, ..., z_t H(x_1, ..., x_n, z_1, ..., z_t)$$

Avec

— $B(x_1, ..., x_n, y_1, ..., y_p)$ un ensemble d'atomes dont les variables sont dans $\{x_1, ..., x_n, y_1, ..., y_p\}$

— $H(x_1, ..., x_n, z_1, ..., z_t)$ un ensemble d'atomes dont les variables sont dans $\{x_1, ..., x_n, z_1, ...z_t\}$

a. ou règle existentielle en français, mais nous utiliserons la terminologie tgd dans tout ce document

B est appelé le corps et H est appelée la tête de la règle tgd. On écrira en général $B \rightarrow H$ lorsque le contexte est clair. De même nous omettrons systématiquement les { et } du corps et de la tête ainsi que les quantificateurs existentiels : les variables de la tête n'apparaissant pas dans le corps étant toujours quantifiées de manière existentielle.

Exemple 56

L'expression $r = A(x, y), A(y, z) \rightarrow A(x, z), M(u, x, x)$ est une règle tgd, le corps B est $A(x, y), A(y, z)$ et la tête H est $A(x, y), M(u, x, x)$ la variable u est quantifiée de manière existentielle

On introduit l'ensemble des variables partagées entre la tête et le corps comme étant la frontière, cette notion est utile pour définir l'activation des triggers en semi-oblivious chase (voir définition 70).

Définition 57

La frontière d'une règle tgd dénotée ∂r est l'ensemble des variables qui apparaissent à la fois dans B et H.

Exemple 58

Si on conserve les notations de l'exemple 56 alors $\partial r = \{x, z\}$.

Définition 59

On appelle extension de h, une application h' de $\mathrm{Dom}(B) \cup \mathrm{Dom}(H)$ dans **Const** telle que la restriction de h' à $\mathrm{Dom}(B)$ est h.

Un système de règles tgd est un ensemble fini de règles tgd.

Définition 60

Soit I une instance, et soit $\mathscr{C}$ un système de règles tgd, on dit que I satisfait $\mathscr{C}$ et on note $I \models \mathscr{C}$ si pour toute règle tgd $B \rightarrow H$ de $\mathscr{C}$ on a :

Pour tout homomorphisme h tel que $h(B) \subseteq I$ il existe une extension h' de h telle que $h'(H) \subseteq I$

La classe des systèmes tgd est parfois notée Datalog$^{\exists}$, cette classe peut être vue abusivement comme une généralisation des contraintes logiques Datalog :

Définition 61

Une règle Datalog est une règle tgd telle que la tête est un singleton qui ne contient pas de variable existentielle.
Un programme Datalog est un ensemble fini de règles Datalog.

En général les règles Datalog sont plutôt écrites sous la forme :
$H :- B$
au lieu de $B \rightarrow H$, mais nous utiliserons rarement cette notation dans ce manuscrit.

Exemple 62

On considère le programme Datalog :

1. $\mathbf{ancetre}(X, Y) :- \mathbf{parent}(X, Y)$
2. $\mathbf{ancetre}(X, Y) :- \mathbf{parent}(X, Z), \mathbf{ancetre}(Z, Y)$

Soit r une règle Datalog et I une instance.
On note $r(I)$ l'instance :

$$\{h(H), \text{s'il existe } h \text{ tel que } h(B) \subseteq I\}$$

Appliquer une règle Datalog consiste à ajouter les fait $h(H)$ à l'instance I.

Définition 63

Soit I une instance et P un programme Datalog. On pose :

$$P(I) = I \cup \bigcup_{r \in P} r(I)$$

Exemple 64

— Soit $I = \{\textbf{parent}(\text{Lily}, \text{Claudie}), \textbf{ancetre}(\text{Claudie}, \text{Léonne})\}$
— Soit P le programme

1. $\textbf{ancetre}(X, Y) :- \textbf{parent}(X, Y)$
2. $\textbf{ancetre}(X, Y) :- \textbf{parent}(X, Z), \textbf{ancetre}(Z, Y)$

on a

$$P(I) = I \cup \{\textbf{ancetre}(\text{Lily}, \text{Léonne})\}$$

L'application qui a I associe $P(I)$ est monotone :

$$I \subseteq J \implies P(I) \subseteq P(J)$$

D'après [30], pour toute instance I, il existe un entier k tel que $P^{k+1}(I) = P^k(I)$. L'instance $P^k(I)$ est appelée point fixe de P sur I et le point fixe vérifie $P^k(I) \models P$

Le problème de la borne uniforme est le problème de décision : il existe un entier k tel que pour toute instance $P^k(I) = P^{k+1}(I)$.

On a [41] :

Théorème 65

Le problème de la borne uniforme d'un programme Datalog est indécidable.

Définition 66

Soit P un programme Datalog. L'arité de P est la plus grande arité des prédicats des têtes des règles de P.

Dans [74] est précisé :

Théorème 67

Le problème de la borne uniforme d'un programme Datalog est :
- Indécidable en arité 3
- Décidable en arité 1
- Ouvert en arité 2

1.7 Chase et dérivations

Le chase est une procédure qui permet à partir d'une instance donnée I et d'un système de règles tgd $\mathscr{C}$ de construire (quand elle termine) une instance J contenant I telle que $J \models \mathscr{C}$.

Il y a différentes manières de compléter I : ce sont les variantes du chase. Dans ce document nous en distinguons particulièrement trois : l'oblivious **O**, le semi-oblivious **SO** et le standard **S**.

Soient S un schéma relationel, **Var** un ensemble de variables et **Const** un ensemble de constantes.

Définition 68

Soit I une instance. Un trigger sur I est un couple (r,h) où r est une règle tgd et h une application de $\mathrm{Dom}(B)$ dans **Const**.

Les triggers sont les objets qui permettent de tester si $h(B)$ est une sous-instance de I et d'activer la règle : on construit alors (selon la variante de chase) des nouvelles constantes grâce à une extension de h (voir définition 59).

Exemple 69

Considérons l'instance $I = \{A(a,b)\}$. On considère le trigger (r,h) où $r = A(x,y) \rightarrow A(x,z)$ et h tel que $h(x) = a$ et $h(y) = b$.
Alors h admet une extension telle que $h'(H) \subseteq I$: soit h' telle que $h' = h$ pour x et y et $h'(z) = b$.

Nous définissons la procédure du chase : à partir d'une instance, on peut construire une X-dérivation (où X est une variante de chase).

Définition 70

— Soit X une variante du chase i.e. $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$.

— Soit $\mathscr{C}$ un ensemble de contraintes tgd.

— Soit I_0 une instance. Une X-dérivation qui commence par I_0 est définie récursivement par :

– Initialisation : I_0
– Induction : Si $I_0 \xrightarrow{(r_0,h_0)} I_1 ... I_n$ est une X-dérivation, soit (r_n, h_n) un trigger sur I_n avec $r_n \in \mathscr{C}$.
(r_n, h_n) est X-actif sur I_n si et seulement si :

1. $(X = \mathbf{O})$: $h_n(B_n) \subseteq I_n$ et il n'existe pas de $i < n$ tel que $(r_i, h_i) = (r_n, h_n)$
2. $(X = \mathbf{SO})$: (1) et $\forall i < n, h_i(\partial r_i) \neq h_n(\partial r_n)$ si $r_i = r_n$
3. $(X = \mathbf{S})$: (1), (2) et il n'existe pas d'extension h'_n de h_n telle que $h'_n(H) \subseteq I_n$

Si (r_n, h_n) est X-actif, on considère une extension h'_n de h_n telle que $h'_n(\mathrm{Dom}(H) \setminus \mathrm{Dom}(B))$ ne soit pas dans $\mathrm{Dom}(I_n)$ (autrement dit on associe aux variables existentielles de la tgd des nouvelles constantes), on pose alors $I_{n+1} = I_n \cup h'_n(H)$.
La séquence $I_0 \xrightarrow{(r_0,h_0)} I_1 ... I_n \xrightarrow{(r_n,h_n)} I_{n+1}$ est une X-dérivation qui démarre sur I_0.

Exemple 71

On considère l'instance $I_0 = \{A(a,b)\}$. On considère le trigger (r_0, h_0) sur I_0 avec $r_0 = A(x,y) \to A(x,z)$ et h_0 tel que $h_0(x) = a$ et $h_0(y) = b$.

— Si $X = \mathbf{O}$ (oblivious) alors le trigger vérifie $h_0(B) = \{A(a,b)\} \subseteq I_0$, donc le trigger est actif.

— Si $X = \mathbf{SO}$ (semi-oblivious) alors on a la condition au dessus et il n'existe aucun autre trigger dans la dérivation donc (r_0, h_0) est actif.

— Si $X = \mathbf{S}$ (standard) alors on a les deux conditions au dessus mais il existe une extension h_0' de h_0 en posant $h_0'(z) = b$ telle que $h_0'(H) = A(a,b) \subseteq I_0$, donc le trigger n'est pas actif.

Si le trigger est actif :
On considère une constante n qui n'appartient pas à $\{a,b\}$, on pose alors h_0' l'extension de h_0 définie par $h_0'(z) = n$. On pose alors :

$$I_1 = I_0 \cup h_0'(H) = I_0 \cup \{A(a,n)\} = \{A(a,b), A(a,n)\}.$$

Ainsi $s = I_0 \xrightarrow{(r_0,h_0)} I_1$ est une X-dérivation.

On considère le trigger (r_1, h_1) sur I_1 défini par $r_1 = \{A(x,y) \to A(x,z)\}$ et h_1 tel que $h_1(x) = a$ et $h_1(y) = n$.

— Si $X = \mathbf{O}$ alors le trigger vérifie $h(B) = \{A(a,n)\} \subseteq I_1$ et on a bien $(r_1, h_1) \neq (r_0, h_0)$, donc le trigger est actif.

— Si $X = \mathbf{SO}$ alors on a la condition au dessus et on a $\partial r = \{x\}$, or $h_1(\partial r) = h_0(\partial r)$ et $r_0 = r_1$ donc le trigger (h_1, r_1) n'est pas actif.

— Si $X = \mathbf{S}$ alors comme la condition pour le semi-oblivious n'est pas vérifiée le trigger n'est pas actif.

Si le trigger est actif (donc dans le oblivious) :
On considère une constante n' qui n'appartient pas à $\{a,b,n\}$, on pose alors h_1' l'extension de h_1 définie par $h_1'(z) = n'$. On pose alors : $I_2 = I_1 \cup h_1'(H) = I_1 \cup \{A(a,n')\} = \{A(a,b), A(a,n), A(a,n')\}$.

Ainsi $s = I_0 \xrightarrow{(r_0,h_0)} I_1 \xrightarrow{(r_1,h_1)} I_2$ est une $\mathbf{O}$-dérivation.

On dit qu'une X-dérivation termine si elle est de longueur finie et s'il n'existe pas de trigger actif sur la dernière instance I_n.

Si une dérivation termine elle a construit un modèle qui satisfait les contraintes [56] :

Proposition 72

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soient I_0 une instance et $\mathscr{C}$ un système de règles tgd. Toutes les instances obtenues par toutes les X-dérivations qui terminent démarrant sur I_0 sont isomorphes.
On note alors $\mathscr{C}^\infty(I_0)$ cette instance. On a :

— $I_0 \subseteq \mathscr{C}^\infty(I_0)$
— $\mathscr{C}^\infty(I_0) \models \mathscr{C}$

Un problème important en management des données est de déterminer des classes de tgd assurant la terminaison :

Définition 73

Soit $\mathscr{C}$ un ensemble de règles de tgd, on dit que $\mathscr{C}$ X-termine, ou que le X-chase termine, si toute X-dérivation termine.

On note $CT^X_{\forall\forall}$ la classe des systèmes de règles tgd qui terminent.

De même on note $CT^X_{\forall\exists}$ la classe des systèmes de règles tgd tels que pour toute instance I, il existe une X-dérivation démarrant sur I qui termine.

On a de manière évidente :

Proposition 74

Pour tout $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$

$$CT^X_{\forall\forall} \subseteq CT^X_{\forall\exists}$$

Ces deux classes sont égales dans les cas oblivious et semi-oblivious [79]. C'est faux dans le cas standard. Autrement dit :

Théorème 75

Pour tout $X \in \{\mathbf{O}, \mathbf{SO}\}$,

$$CT^X_{\forall\forall} = CT^X_{\forall\exists}$$

De manière générale décider la terminaison est indécidable [55],[51] :

Théorème 76

Décider si un système $\mathscr{C}$ appartient à $CT^X_{\forall\forall}$ (resp. $CT^X_{\forall\exists}$) est indécidable pout $x \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$.

1.8 Contraintes de mots et traduction

Dans cette section nous introduisons d'autres règles tgd particulières : les contraintes de mots. Nous relions cette notion avec les systèmes de réécritures.

L'expression "contraintes de mots" est déjà utilisée pour la définition 43. Ici nous définissons les contraintes de mots à l'aide de tgd, les définitions sont équivalentes sur les graphes.

Définition 77

Une contrainte de mots est une règle tgd $B \rightarrow H$ telle que B et H sont de la forme :

- B est de la forme $\{A_1(x_1,x_2), A_2(x_2,x_3), ..., A_n(x_n,x_{n+1})\}$
- H est de la forme $\{B_1(y_1,y_2), B_2(y_2,y_3), ..., B_m(y_m,y_{m+1})\}$
- les variables communes de B et H sont x_1 et x_{n+1} avec $x_1 = y_1$ et $x_{n+1} = y_{m+1}$.
- Les variables $x_1, ..., x_{n+1}$ sont distinctes les variables $y_1, ..., y_{m+1}$ sont distinctes.

Exemple 78

Les règles suivantes sont des contraintes de mots

- $A(x,y), A(y,z) \rightarrow A(x,z)$
- $A(x,y), B(y,z) \rightarrow B(x,t), C(t,z)$

Et les règles suivantes n'en sont pas :

- $A(x,y), A(x,z) \rightarrow A(x,z)$
- $B(x,x) \rightarrow A(x,x)$
- $A(x,y) \rightarrow A(x,z), B(x,y)$

Nous allons maintenant relier trois notions :

- Les contraintes de mots (sous formes tgd)
- Les contraintes de mots (sous formes RPQ)
- Les systèmes de réécriture

On commence par introduire une transformation entre les prédicats et les lettres d'un alphabet.

Très souvent, les prédicats sont désignés par des lettres majuscules, nous noterons alors par la minuscule correspondante la transformation en lettre d'un alphabet.

Définition 79

Soit r une contrainte de mot. Soit $M = A_1(x_1, x_2), A_2(x_2, x_3), ..., A_n(x_n, x_{n+1})$ la tête ou le corps de r.
On associe à M le mot $a_1 a_2 ... a_n$.

À chaque règle tgd nous pouvons donc associer deux mots, et donc associer une contrainte de mots (forme RPQ) ou une règle d'un système de réécriture.

Définition 80

À toute contrainte de mots (en tant que tgd) $r = B \rightarrow H$ on associe le mot u à B et v à H on peut donc associer à r :

— La contrainte de mots $xuy \sqsubseteq xvy$
— La règle de réécriture (u, v)

À toute instance I dont tous les prédicats sont d'arité 2 on peut associer une base de données graphes G :

— $N = \text{Dom}(I)$
— $E = \{(x, a, y), A(x, y) \in I\}$

Pour tout ensemble de contraintes de mots $\mathscr{C}$ en tant que tgd, si $\mathscr{C}'$ désigne les contraintes de mots en tant que RPQ on a :

$$I \models \mathscr{C} \text{ si et seulement si } G \models \mathscr{C}'$$

Ainsi, nous préférons confondre les termes de contraintes de mots tgd et RPQ. De même nous confondons une instance d'arité 2 et la base de données graphe associée.

CHAPITRE 2

État de l'art

Le but de ce chapitre est de présenter l'état de l'art sur les résultats en lien avec ce manuscrit. Tout d'abord sont présentés les principaux résultats de réécriture des mots qui nous préoccupe. Nous verrons dans un second temps quelques résultats sur l'inclusion de requêtes sur les bases de données graphes, sans contraintes puis avec contraintes. Dans une dernière partie, nous verrons quelques travaux sur la complétion des bases de données par la procédure du chase sous contraintes tgd : les problèmes de la terminaison et de la borne uniforme.

2.1 Réécriture

Les systèmes de réécritures de mots, ont été définis originellement pour des questions de simplification dans les semi-groupes. Ils sont depuis devenus des objets très utilisés en informatique. Dans ce manuscrit nous utilisons plusieurs problèmes majeurs de la théorie de la réécriture des mots :

- Le problème du mot :

Étant donné un système R et deux mots u et v, peut-on décider si $u \to^* v$?

Le problème du mot est indécidable en général [82].

- Le problème de terminaison :

Est ce que toutes les dérivations sont de longueur finie ?

Le problème de terminaison est indécidable en général [37, 16]. La preuve de la terminaison de certains problèmes est difficile comme le problème de Zantema constitué de l'unique règle $0011 \to 111000$ [47, 33, 49], ou de manière plus célèbre la conjecture de Collatz [1] qui peut-être traduite en un problème de terminaison d'un système de réécriture [90]. De nombreuses méthodes pour assurer la terminaison ont été proposées [70, 36, 10, 89, 32]. Des programmes comme **matchbox**, **torpa**, **Aprove**, **COCCINELLE**, ... ont été développés dans le cadre d'un compétition annuelle de terminaison [1].

- Le problème de la préservation des réguliers :

Est ce que $\mathrm{Desc}_R(L)$ est régulier pour tout langage régulier R ?

Ce problème a été initialement démontré comme indécidable dans le contexte des termes [50, 31], puis a été précisé un peu plus tard dans le cadre des mots [81]. Dans ce document, nous nous intéressons à des classes de systèmes de réécriture qui assurent la préservation des réguliers et nous définissons les classes $\mathrm{MB}_{\max}$ (def.141) et BPar (def.150)comme preservant les réguliers. La classe $\mathrm{MB}_{\min}$ des match-bounded est une notion similaire à la classe des systèmes à complexité parallèle bornée $\mathrm{MB}_{\max}$. Les systèmes inverse match-bounded préservent les réguliers mais ne sont pas descendant-limités (def.88)ce qui ne permet pas de décider l'inclusion de RPQ sous contrainte $\mathrm{MB}_{\min}$.

De nombreuses classes et critères ont été proposés on citera notamment le critère [29], et des classes comme les monadiques [16, 80], les systèmes préfixes/suffixes [83, 15, 28] les deleting [60], les inverses match-bounded [49, 48],...

1. si x est pair on le transforme en $x/2$, sinon en $3x+1$, la conjecture énonce que pour tout x, le nombre 1 est atteint.

Une méthode classique pour préserver la régularité consiste à compléter un automate du langage pour reconnaître les descendants. Une étude de la terminaison de la complétion d'automate dans le contexte de la réécriture des termes peut être trouvée [42, 45, 46, 44].

Comme dit précédemment, nous proposons dans ce manuscrit une complétion similaire à l'oblivious-chase (def.158 et 98). Nous proposons des classes $\mathrm{MB}_{\max}(k)$ et $\mathrm{BPar}(k)$ qui assurent une borne uniforme de la procédure de complétio.

Un système de réécriture est $\mathrm{MB}_{\max}(k)$ (def.141) si toute dérivation peut-être effectuée en un nombre d'étapes de réécriture parallèles $\leq k$:

La réécriture parallèle consiste à pouvoir appliquer plusieurs étapes de réécriture sur des facteurs orthogonaux. L'orthogonalité a été mise en avant en lambda calcul [72] et dans la réécriture des termes [63, 53, 67, 68, 14] notamment comme modèle de calcul. Dans [64, 12] des annotations sont ajoutées afin de garder l'historique de l'évolution de la réécriture, nous proposons ces mêmes annotations pour les systèmes de réécriture (def.139).

2.2 Requêtes de graphes

Une méthode d'optimisation de requêtes consiste à déterminer une requête équivalente plus simple à évaluer. Dans ce sens le problème d'inclusion de requêtes a été étudié.

Des études dans les modèles relationnels ont été initiées, notamment avec les requêtes conjonctives (CQ) et leurs unions (UCQ) largement utilisées dans des langages comme SQL. L'inclusion de requêtes UCQ est NP-complète. Cependant les UCQ ne comportent aucune récursion et ne sont pas adaptées aux bases de données graphes. L'ajout de la récursion aux requêtes UCQ conduit à définir le langage de requêtes Datalog [43]. Cependant l'inclusion de requêtes Datalog est indécidable [85].

Des langages comme SparQL [88, 5] permettent d'interroger de manière plus adaptées les bases de données graphes en utilisant des requêtes de navigation [78, 8] comme les RPQ [73] (def.39)et des extensions comme CRPQ (conjonctions de RPQ), 2RPQ (les graphes peuvent être parcourus dans les deux sens) [25], C2RPQ, UC2RPQ [26] (union de C2RPQ)...

Ces requêtes sont des sous-classes de Datalog [43]. Dans [87] on montre que l'inclusion est PSPACE-complète pour les requêtes RPQ et 2RPQ, et EXPSPACE-complète pour les requêtes UC2RPQ.

Une requête récursive et une non-récursive ne peuvent pas être équivalentes. L'introduction de contraintes d'intégrité sur les modèles permet des optimisations de requêtes plus efficaces particulièrement dans le cadre des requêtes de graphes.

Le problème d'inclusion de requêtes **sous contraintes d'intégrité** de formule :

Est-ce qu'une requête de graphes est incluse dans une autre pour tous les modèles qui satisfont les contraintes ?

Les premières études de l'inclusion sous contraintes ont été menées dans le cadre des graphes enracinés sous contrainte d'inclusion [3, 18, 19, 7]. Une contrainte d'inclusion est la donnée de deux requêtes de chemins p et p' (sous forme d'expression régulière) dont les réponses de p sont incluses dans celles de p'. Dans ce cadre, les requêtes de chemin sont évaluées depuis une racine.

Le problème d'inclusion de requêtes RPQ sous contraintes (d'inclusions) dans les graphes enracinés est EXPTIME [3, 7, 6]. Dans le cas où les p' sont des mots (on dit que les contraintes d'inclusions sont bornées) le problème d'inclusion sous contraintes est PSPACE-complet [4, 7, 6] (la preuve utilise une réduction vers les systèmes de réécriture préfixe [27]).

Dans ce manuscrit nous nous intéressons aux modèles sans racines (def.34). Les premiers travaux ont été initiés par [57] dans le cadre des contraintes de mots (def.43). Une base de données graphes satisfait une contrainte de mots $xuy \sqsubseteq xvy$

si pour tout chemin étiqueté par u dans G, il existe un chemin de même origine et même arrivée étiqueté par v dans G.

Dans [57] est associé aux contraintes de mots $\mathscr{C}$ un système de réécriture R pour montrer :

$$xuy \sqsubseteq_{\mathscr{C}} xvy \text{ ssi } u \rightarrow_R^* v$$

Cette proposition implique que l'inclusion de requêtes RPQ sous contraintes de mots est indécidable. Elle implique aussi que :

$$xLy \sqsubseteq_{\mathscr{C}} xL'y \text{ ssi } L \subseteq \text{Anc}_R(L')$$

Ainsi, si une classe de système de réécriture assure que $\text{Anc}_R(L')$ est régulier pour tout L', alors le problème d'inclusion est décidable. Cependant, ce sera l'objet du chapitre 3, ces deux propositions sont fausses en général.

2.3 Procédure du chase et borne uniforme

La procédure du chase a été initialement définie pour le "problème d'implication" [77, 65] afin de tester les implications de certaines dépendances logiques.

Notamment sont définis comme contraintes :

— les egd (equality generating dependencies)

— les tgd (tuple generating dependencies)

Ces contraintes sont utilisées en data exchange, data integration, data repair, ... (on pourra considérer [9] pour une référence complète).

Étant donné un ensemble $\mathscr{C}$ de contraintes tgd (def.55) et une variante du chase (def.70) comme oblivious [22], semi-oblivious [75], standard [40], core [38],...) quatre problèmes de terminaison sont définis :

1. Soit I une instance, est ce qu'il existe une branche d'exécution qui termine ? (i.e. $\mathscr{C} \in CT^*_{I\exists}$)
2. Soit I une instance, est ce que toute branche d'exécution termine ? (i.e. $\mathscr{C} \in CT^*_{I\forall}$)
3. Pour toute instance, est ce qu'il existe une branche d'exécution qui termine ? (i.e. $\mathscr{C} \in CT^*_{\forall\exists}$)
4. Pour toute instance, est ce que toute branche d'exécution termine ? (i.e. $\mathscr{C} \in CT^*_{\forall\forall}$)

Tous les problèmes de terminaison sont indécidables [75, 38]. Dans notre cadre nous nous intéressons tout particulièrement au dernier problème que nous appelons *le* problème de terminaison.

Dans le cas oblivous ou semi-oblivious les deux premiers problèmes et les deux derniers problèmes sont équivalents [79, 55, 56]. C'est dans ces deux variantes que nous étudierons principalement le problème de terminaison dans cette thèse.

La décidabilité de la terminaison a aussi été étudiée en fonction de l'arité des prédicats [55, 56, 51] :

— Décidable en arité 1

— Indécidable en arité 3

— Ouvert en arité 2

— Le problème en arité 2 est EXPSPACE-difficile.

Il est naturel de proposer des classes qui assurent la terminaison. De nombreuses classes ont été proposées comme la classe des faiblement acyclique (weakly acyclic) [40] et des extensions plus expressives [76, 79, 21, 58, 52, 11].

Les systèmes tgd de profondeur uniformément bornée (def.184) terminent [35, 17, 34] et ont été motivés par l'étude de requêtes BCQ (boolean conjonctive queries) [20, 23, 54, 91, 66]. Ces systèmes seront étudiés au chapitre 5.

Le problème uniforme pour les règles tgd s'énonce :

"Soient X une variante de chase et k un entier. Étant donné une stratégie et des contraintes tgd, est ce que les X-dérivations sont de profondeurs bornées par k?"

La profondeur d'un fait est définie par 1+ la profondeur des faits utilisés pour le produire par la règle tgd [35, 34] (def.191). Des résultats de décision du problème de la borne uniforme ont été étudiés dans [35, 34, 17].

L'étude de la borne uniforme dans le contexte des programmes Datalog est très largement étudiée avec des applications en data management, protocoles réseaux, analyse de programmes, ... [61]. Contrairement à la procédure du chase, les programmes Datalog terminent [30, 86]. Le problème de la borne uniforme a été montré indécidable en général pour Datalog dans [41] même si une seule règle est récursive [2]. Une étude en fonction de l'arité des programmes a été menée [59, 74] : le problème est indécidable en arité 3 et décidable en arité 1 [74]. Le problème reste ouvert en arité 2.

Dans notre cas nous étudions le problème de la borne uniforme dans le contexte de la terminaison. Nous comparons la borne uniforme avec la notion de réécriture parallèlement bornée dans le cadre des contraintes de mots.

CHAPITRE 3

Inclusion de requêtes sous contraintes de mots

Le développement des bases de données graphes (comme le format RDF [71]) pour représenter des données comme des réseaux de transports ou des réseaux sociaux ont conduit le développement de langages spécifiques pour interroger ces données : SparQL, Cypher, ...

Une des fonctionnalités de SParQL sont les property paths [5] : ce sont des requêtes qui retournent l'ensemble des paires de nœuds reliés par une expression régulière. Ces requêtes sont formalisées comme Regular Path Queries (RPQ) [87].

Une méthode d'optimisation de ces requêtes consiste à évaluer une requête plus simple à évaluer équivalente à la première. Des connaissances exprimées comme contraintes de mots permettent de proposer une requête acyclique (facile à évaluer) équivalente à une requête récursive. Un des problèmes majeurs issu de cette optimisation de requêtes RPQ est de décider si deux requêtes RPQ sont équivalentes sous contraintes de mot.

Est ce que l'ensemble des réponses d'une RPQ Q est inclus dans l'ensemble des réponses d'une RPQ Q' pour tout modèle qui satisfait les contraintes de mots ?

Une approche classique de l'étude de ce problème consiste à réduire ce problème à un problème d'inclusion de langages réguliers. Dans [57] une telle réduction est proposée.

Nous montrons dans ce chapitre que cette réduction n'est en fait valable que pour des modèles potentiellement infinis et nous proposons une condition suffisante de réduction dans le cas fini. Nous montrons aussi que le problème d'inclusion de RPQ sous contraintes de mots est indécidable.

La réduction du problème d'inclusion de requêtes sous contraintes amène à proposer des classes qui préservent la régularité (pour les ancêtres), si les systèmes sont descendants-limités : une requête RPQ xLy est incluse dans une requête RPQ $xL'y$ sous contraintes de mots si et seulement si L est inclus dans les ancêtres de L'.

Nous utilisons alors une complétion d'automate (à la manière de l'oblivious-chase) qui simule les ancêtres à un pas de réécriture parallèle : une étape de complétion de A reconnaît les mots qui s'écrivent en un pas parallèle en un mot reconnu par A.

Nous proposons finalement une première classe NR peu expressive sous laquelle le problème d'inclusion de requêtes RPQ est EXPSPACE-complète.

3.1 Résumé

Dans ce chapitre nous nous intéressons au problème d'inclusion de deux requêtes RPQ sous contraintes de mots.

Les requêtes RPQ sont toujours données sous forme d'automate non-déterministe.

PROBLEME : Inclusion de RPQ sous contraintes de mots **INSTANCE** : $Q = xLy$ et $Q' = xL'y$ deux RPQ, $\mathscr{C}$ un ensemble de contraintes de mots **QUESTION** : $Q \sqsubseteq_{\mathscr{C}} Q'$?

Nous utilisons l'approche adoptée par Abiteboul [3] dans les graphes enracinés et par Grahne et Thomo [57] dans des base de données graphe : on associe à tout ensemble de contrainte de mots $\mathscr{C}$ un système de réécriture R pour comparer les inclusions

— $xLy \sqsubseteq_{\mathscr{C}} xL'y$

— $L \subseteq \text{Anc}_R(L')$

Malheureusement nous montrons que ces deux inclusions ne sont pas toujours équivalentes dans le cas des modèles finis (contrairement à ce qui est montré dans [57]).

1. Nous étudions les deux inclusions dans le cas des modèles potentiellement infinis où nous prouvons qu'elles sont équivalentes.
2. Nous montrons que dans le cas fini nous avons juste l'implication
   $L \subseteq \text{Anc}_R(L') \implies xLy \sqsubseteq_{\mathscr{C}} xL'y$
3. Nous proposons une condition suffisante (tout mot admet un nombre fini de descendants) qui assure l'équivalence des deux inclusions dans le cas des modèles finis
4. Nous montrons l'indécidabilité de l'inclusion de requêtes RPQ sous contraintes de mots

On appelle descendant-limité tout système de réécriture tel que tout mot admet un nombre fini de descendants. La majorité des systèmes de ce document (comme les terminants et BPar (def. 150) sont descendant-limités. Cette condition assure l'équivalence des deux inclusions.

Dans la section suivante, nous nous intéressons à décider le problème :

$$L \subseteq \text{Anc}_R(L')$$

Pour cela nous proposons de compléter un automate qui reconnaît L' pour reconnaître les ancêtres de L'. Nous proposons la classe NR (non-récursive) peu expressive

qui assure la terminaison de cette complétion. Nous montrons à l'aide du chapitre suivant que cette inclusion est EXPSPACE nous montrons aussi par cette classe que le problème d'inclusion de requêtes RPQ sous contraintes est EXPSPACE-difficile.

PROBLEME : Inclusion de requêtes RPQ sous contraintes de mots NR **INSTANCE** : $Q = xLy$ et $Q' = xL'y$ deux RPQ, $\mathscr{C}$ un ensemble de contraintes de mots dont le système associé est NR **QUESTION** : $Q \sqsubseteq_{\mathscr{C}} Q'$? **COMPLEXITE** : EXPSPACE-complet

3.2 Etude de la restriction vers les modèles finis

Dans tout ce chapitre est fixé un alphabet Σ.

Dans cette section nous souhaitons comparer l'inclusion de requêtes RPQ en fonction du caractère fini ou non des bases de données graphe :

— $Q \sqsubseteq^{\infty}_{\mathscr{C}} Q'$ est le problème d'inclusion de RPQ sous contraintes de mots avec des modèles éventuellement infinis

— $Q \sqsubseteq_{\mathscr{C}} Q'$ où les modèles sont finis (def. 44)

Comme évoqué, nous proposons une étude de ces inclusions de requêtes via les systèmes de réécriture.

$$xuy \sqsubseteq xvy \text{ devient } (u, v)$$

Nous avons la proposition suivante :

Proposition 81

Soit $\mathscr{C}$ un ensemble de contraintes de mots, R son système de réécriture associé. Pour tous mots u et v on a :

$$\text{Si } u \to^* v \text{ alors } xuy \sqsubseteq^{\infty}_{\mathscr{C}} xvy$$

Démonstration. Supposons que $u \to^*_R v$. Alors il existe des mots $i_0, ..., i_n$ tels que pour tout t on a $i_t \to i_{t+1}$ avec $i_0 = u$ et $i_n = v$. Soit G une base de données graphe éventuellement infinie telle que $G \models \mathscr{C}$, soient x, y deux nœuds de G. Montrons que s'il existe un chemin de x à y étiqueté par i_t alors il existe un chemin de x à y étiqueté par i_{t+1}.

On a $i_t \to i_{t+1}$ donc il existe μ, μ' et $(a, b) \in R$ tels que $i_t = \mu a \mu'$ et $i_{t+1} = \mu b \mu'$. Le chemin de x à y se décompose donc en un chemin de x à un certain nœud x' étiqueté par μ, un chemin de x' à un certain nœud y' étiqueté par a et un chemin de y' à y étiqueté par μ'. Or $(a, b) \in R$ signifie que $xay \sqsubseteq xby \in \mathscr{C}$ donc il existe un chemin de x' à y' étiqueté par b. En concaténant les chemins, il existe un chemin de x à y étiqueté par $\mu b \mu'$ c'est à dire i_{t+1}.

Ainsi par récurrence, s'il existe un chemin de x à y étiqueté par u, il existe un chemin de x à y étiqueté par v. Ainsi $G \models xuy \sqsubseteq xvy$. Puisque c'est vrai pour tout graphe (même infini) on a par définition $xuy \sqsubseteq^{\infty}_{\mathscr{C}} xvy$. □

Nous proposons maintenant d'étudier l'autre implication :

$$\text{Si } xuy \sqsubseteq^{\infty}_{\mathscr{C}} xvy \text{ alors } u \to^* v$$

Pour cela, nous souhaitons construire un modèle satisfaisant les contraintes qui contienne un chemin γ étiqueté par u tel que tout chemin entre l'origine de γ et le but de γ est étiqueté par un descendant de u.

Une première méthode consisterait à considérer le graphe-mot G_u (définition 38), on ajoute alors des nouveaux chemins en appliquant la procédure de l'oblivious-chase.

Une seconde méthode consiste à adapter le modèle de Abiteboul [3] dans les graphes sans racines comme il a été fait dans [57] :

Nous considérons le modèle (éventuellement infini) suivant :

Définition 82

Soient u et v deux mots.
Posons $G = (N, E)$ avec $N = \text{Pref}(\text{Desc}(u))$ (def. 10) et :

$$E = \{(x, a, m), x, m \in N, a \in \Sigma \mid m \rightarrow^* xa\}$$

Nous conservons le même graphe G (fixé par u et v) tout au long des preuves.

Lemme 83

Pour tout nœud a de N. Si b un mot non vide tel que $a \rightarrow_R^* b$,
alors tout préfixe de b est dans N

Démonstration. Soit p un préfixe de b alors il existe r tel que $b = pr$.

Puisque $a \in N$, on a a qui est le préfixe d'un descendant de u, donc il existe r' tel que $u \rightarrow_R^* ar'$. Or $a \rightarrow_R^* b$ donc :

$$u \rightarrow_R^* ar' \rightarrow_R^* br' = prr'$$

Par définition, p est un préfixe d'un descendant de u, donc $p \in N$. □

Lemme 84

Pour tout nœud x de N et tout $w \in \Sigma^*$, si $xw \in N$ alors il existe un chemin de x à xw étiqueté par w.

Démonstration. Puisque xw est un nœud de N et $xw \rightarrow_R^* xw$, tous les préfixes de xw sont dans N d'après le lemme 83.

Soit $x \in N$,

Posons H_n : "Pour tout mot w de longueur n, si $xw \in N$ alors il existe un chemin de x à xw étiqueté par w"

Montrons H_0 : soit w de longueur 0, il existe un chemin de x à xw étiqueté par w.

Supposons H_n vraie, montrons H_{n+1} : soit w de longueur $n+1$ tel que $xw \in N$. On a $w = w'a$ avec a une lettre. Puisque xw' est un préfixe de xw on a $xw' \in N$. De plus, $xw'a \rightarrow_R^* xw'a$ donc $(xw', a, xw'a) \in E$, donc il existe un chemin de xw' à $xw'a$ étiqueté par a. On a $|w'| = n$ donc d'après H_n il existe un chemin de x à xw' étiqueté par w'. En concaténant les deux chemins, il existe un chemin de x à xw étiqueté par $w'a$ et $w'a = w$. □

Montrons le lemme :

Lemme 85

Pour tous $x, m \in N$ et $w \in \Sigma^+$ on a :

$m \rightarrow_R^* xw$ ssi il existe un chemin de x à m étiqueté par w

Démonstration. Soient $x, m \in N$ et $w \in \Sigma^+$ tels que :

$$m \rightarrow_R^* xw$$

Puisque $w \in \Sigma^+$ il existe w' et a avec a une lettre tels que $w = w'a$. On a : $m \rightarrow_R^* xw'a$, on en déduit donc :

1. Par définition $(xw', a, m) \in E$
2. $xw' \in N$ d'après le lemme 83
3. il existe un chemin de x à xw' étiqueté par w' d'après le lemme 84

On en déduit d'après les points 1) et 3) qu'il existe un chemin de x à m étiqueté par $w'a = w$.

Montrons maintenant la réciproque.

Posons H_n : "Pour tous $x, m \in N$, pour tout $w \in \Sigma^+$, si $|w| = n$ et s'il existe un chemin de x à m étiqueté par w alors $m \rightarrow_R^* xw$".

- **Initialisation :** montrons H_1. Soit w de longueur 1 tel qu'il existe un chemin de x à m étiqueté par w. On a $(x, w, m) \in E$ et donc $m \rightarrow_R^* xw$ par définition de E.

- **Hérédité :** supposons H_n montrons H_{n+1}. Soit w un mot de longueur $n+1$ tel qu'il existe un chemin de x à m étiqueté par w. Il existe donc w' et une lettre a tels

que $w = w'a$. Puisque xw' est un préfixe de $xw \in N$ et que $xw \to_R^* xw$ on a d'après le lemme 83 que $xw' \in N$.

Il existe donc un nœud m' tel qu'ils existent un chemin de x à m' étiqueté par w' et une arête (m', a, m) de E. On a w' de longueur n donc $m' \to_R^* xw'$. Et $(m', a, m) \in E$ implique par définition que $m \to_R^* m'a$. Ainsi :

$$m \to_R^* m'a \to_R^* xw'a = xw$$

Ce qui achève le lemme.

□

Lemme 86

Pour tout mot v, si $G \models xuy \sqsubseteq xvy$ alors $u \to_R^* v$.

Démonstration. On a $u \to_R^* u$ donc $\epsilon \in N$. Puisque $u \in N$ il existe un chemin de ϵ à u étiqueté par u (Lemme 84).

Par définition de $G \models xuy \sqsubseteq xvy$ il existe un chemin de ϵ à u étiqueté par v.

D'après le lemme 85, on a $u \to_R^* \epsilon v = v$. □

Montrons maintenant l'équivalence :

$$xuv \sqsubseteq_{\mathscr{C}}^{\infty} xvy \iff u \to_R^* v$$

Supposons $xuv \sqsubseteq_{\mathscr{C}}^{\infty} xvy$. Montrons que $G \models \mathscr{C}$. Soit $xty \sqsubseteq xt'y$ une contrainte de mots de $\mathscr{C}$. On a alors $t \to_R t'$ donc pour tous $x, m \in N$, si $m \to_R^* xt$ on a $m \to_R^* xt'$. D'après le lemme 85, s'il existe un chemin de x à m étiqueté par t alors il existe un chemin de x à m étiqueté par t'. Puisque x et m sont quelconques on a $G \models xty \sqsubseteq xt'y$, donc $G \models \mathscr{C}$. Par définition de $xuv \sqsubseteq_{\mathscr{C}}^{\infty} xvy$ on en déduit que $G \models xuv \sqsubseteq xvy$. D'après le lemme 86, on a $u \to_R^* v$.

Nous obtenons finalement :

Théorème 87

Soient $\mathscr{C}$ un ensemble de contraintes de mots et R le système de réécriture associé. Pour tous mots u et v on a :

$$xuy \sqsubseteq_{\mathscr{C}}^{\infty} xvy \text{ ssi } u \to_R^* v$$

Malheureusement les modèles que nous souhaitons utiliser sont obligatoirement finis et le théorème autorise des modèles infinis.

Nous remarquons que si les descendants sont en nombre finis alors le modèle proposé dont les nœuds sont les préfixes des descendants de u est fini.

On introduit naturellement :

Définition 88

Un système de réécriture R est dit descendant-limité si pour tout mot u de Σ^+ on a :

$$\mathrm{Desc}_R(u) \text{ fini}$$

Cette condition est vérifiée par les systèmes terminant (def. 24) ou les systèmes de réécriture à longueur décroissante [1]. Nous proposerons une classe (BPar) au chapitre 4 qui satisfait aussi cette condition et qui est non comparable aux deux précédentes.

Ainsi nous avons :

Théorème 89

Soient $\mathscr{C}$ un ensemble de contraintes de mots et R le système de réécriture associé.
Si R est descendant-limité alors pour tous mots u et v on a :

$$xuy \sqsubseteq_{\mathscr{C}} xvy \text{ ssi } u \rightarrow_R^* v$$

Ainsi nous pouvons en déduire :

Corollaire 90

Soient $\mathscr{C}$ un ensemble de contraintes de mots et R le système de réécriture associé.
Si R est descendant-limité alors pour toutes RPQ $Q = xLy$ et $Q' = xL'y$ les assertions suivantes sont équivalentes :

1. $Q \sqsubseteq_{\mathscr{C}} Q'$
2. $L \subseteq \mathrm{Anc}_R(L')$

1. Pour tout $(u, v) \in R$ on a $|v| \leq |u|$

Démonstration. Montrons que (1) implique (2). Soit u un mot de L. On considère le graphe G défini par Pref(Desc(u)). Alors entre le nœud ϵ et u il existe un chemin étiqueté par u (d'après le lemme 84), donc il existe $v \in L'$ et un chemin étiqueté par v d'après (1). D'après le lemme 86 on a alors $u \to^* v$. Donc $L \subseteq \mathrm{Anc}_R(L')$.

Inversement, montrons que (2) implique (1). Soit u un mot de L, puisque $L \subseteq \mathrm{Anc}_R(L')$, il existe $v \in L'$ tel que $u \to^* v$. D'après 81 $xLy \sqsubseteq_{\mathscr{C}} xL'y$. □

Nous montrons que :

Théorème 91

La proposition énoncée dans [57] :
Soit $\mathscr{C}$ un ensemble de contraintes de mots et R le système de réécriture associé. Pour tous mots u et v on a :

$$xuy \sqsubseteq_{\mathscr{C}} xvy \text{ ssi } u \to^* v$$

est fausse.

Démonstration. On rappelle avant tout qu'un langage L est récursivement énumérable s'il existe une machine de Turing qui termine et accepte les mots de L. Un langage à la fois récursivement énumérable et co-récursivement énumérable est récursif (i.e. décidable : il existe une machine de Turing terminante qui reconnaît L).

Supposons que la proposition soit vraie. Pour tout u et $\mathscr{C}$ on pose $E(\mathscr{C}, u)$ l'ensemble des v tels que $xuy \sqsubseteq_{\mathscr{C}} xvy$.

On a alors d'après la proposition que $E(\mathscr{C}, u) = \mathrm{Desc}(u)$.

Tester si $v \notin E(\mathscr{C}, u)$ consiste à tester s'il existe un modèle fini G satsifaisant $\mathscr{C}$ tel que $\neg(G \models xuy \sqsubseteq xvy)$, donc $E(\mathscr{C}, u)$ est co-récursivement énumérable.

De plus Desc(u) est récursivement énumérable. Donc s'il était co-récursivement énumérable il serait récursif.

Or tester si $v \in \mathrm{Desc}(u)$ i.e. si $u \to^* v$, est indécidable d'après le problème de l'arrêt (théorème 27), donc il existe des Desc(u) qui sont non récursifs. Contradiction.

□

Nous proposons avec [84] aussi un contre-exemple à la proposition :

Exemple 92

Soit $\mathscr{C}$ l'ensemble des contraintes de mots dont le système de réécriture R traduit est :

1. $a \to bab$
2. $bab \to a$
3. $cab \to db$
4. $bac \to bd$
5. $db \to d$
6. $bd \to d$
7. $dc \to d$
8. $cd \to d$

alors on a :
— $xcacy \sqsubseteq_{\mathscr{C}} xdy$
— $\neg(\{cac\} \subseteq \mathrm{Anc}_R(\{d\}))$

Démonstration. Par une récurrence facile nous avons pour tout n :

$$cac \to^n cb^n ab^n c$$

Montrons maintenant que nous avons $xcacy \sqsubseteq_{\mathscr{C}} xdy$: Soit G un graphe tel que $G \models \mathscr{C}$. Soient x et y deux nœuds tel qu'il existe un chemin de x à y étiqueté par cac.

Le langage L défini par l'ensemble des étiquettes des chemin de x à y est un langage régulier (il suffit de considérer x comme état initial d'un automate, y comme état final, et toutes les arêtes comme des transitions).

L contient cac. Or $G \models \mathscr{C}$ et $cac \to^n cb^n ab^n c$ donc ce langage contient pour tout n le mot $cb^n ab^n c$. Puisque le langage est régulier, il existe k et l avec $k \neq l$ tels que $cb^k ab^l c \in L$.

Supposons sans perdre de généralité que $k > l$ (l'autre cas étant symétrique) :
— Il existe un chemin de x à y étiqueté par $cb^k ab^l c$
— Règle 2 : il existe un chemin de x à y étiqueté par $cb^{k-l} ac$
— Règle 4 : il existe un chemin de x à y étiqueté par $cb^{k-l} d$
— Règle 6 (appliquée $k-l$ fois) : il existe un chemin de x à y étiqueté par $cb^{k-l-(k-l)} d = cd$
— Règle 8 : il existe un chemin de x à y étiqueté par d

Ainsi, si $G \models \mathscr{C}$ alors $xcacy \sqsubseteq xdy$ i.e.

$$xcacy \sqsubseteq_{\mathscr{C}} xdy$$

Montrons maintenant que $\neg(\{cac\} \subseteq \mathbf{Anc}_R(\{d\}))$ **:**

Montrons par récurrence que pour tout n : Si $cac \rightarrow^n u$ alors il existe k_n tel que $u = cb^{k_n}ab^{k_n}c$.

- Initialisation : soit $n = 0$. Si $cac \rightarrow^0 u$ alors $u = cac = cb^0ab^0c$, posons $k_0 = 0$. Il existe k_0 tel que $u = cb^{k_0}ab^{k_0}c$

- Hérédité : supposons la propriété vraie au rang n. Montrons la au rang $n+1$.

Si $cac \rightarrow^{n+1} u$ alors il existe v tel que $cac \rightarrow^n v$ et $v \rightarrow u$. Par hypothèse de récurrence, il existe k_n tel que $v = cb^{k_n}ab^{k_n}c$. Les seuls facteurs de v qui sont des membres gauches de règle de R sont a (règle 1) et bab (règle 2) si $k_n > 0$. Comme $v \rightarrow u$ on a alors $u = cb^{k_n+1}ab^{k_n+1}c$ par la règle 1, il suffit de poser $k_{n+1} = k_n + 1$, ou $u = cb^{k_n+1}ab^{k_n+1}c$ par la règle 2 si $k_n > 0$, il suffit de poser $k_{n+1} = k_n - 1$. Dans tous les cas, il existe k_{n+1} tel que $u = cb^{k_{n+1}}ab^{k_{n+1}}c$.

Ainsi, comme pour tout k on a $d \neq cb^kab^kc$ on a $\neg(cac \rightarrow^* d)$ i.e. $\neg(\{cac\} \subseteq \mathrm{Anc}_R(\{d\}))$.

□

Le problème d'inclusion sous contraintes est indécidable en général, et même dans le cas où les contraintes sont inverse hors-contextes :

Définition 93

Un ensemble de contraintes inverse hors-contexte est un ensemble de contraintes de la forme $\{xwy \sqsubseteq xVy\}$ avec $w \in \Sigma^*$ et $V \in \Sigma$. En particulier ces règles sont des règles Datalog.

On montre le théorème suivant :

Théorème 94

Le problème d'inclusion de RPQ sous contraintes de mots hors-contextes est indécidable :

PROBLEME : Inclusion de requêtes RPQ sous contraintes inverse hors-contextes
INSTANCE : $\mathscr{C}$ un ensemble de contraintes inverse hors-contextes, $Q = xLy$ et $Q' = xvy$ deux requêtes
QUESTION : $Q \sqsubseteq_{\mathscr{C}} Q'$?
COMPLEXITE : indécidable

Démonstration. Décider si une grammaire hors-contexte Γ d'alphabet Δ est universelle i.e. reconnaît le langage Δ^+ est un problème indécidable. Nous allons réduire le problème d'inclusion de requêtes RPQ sous-contraintes hors-contexte au problème d'universalité.

Soit $\Gamma = (N, \Delta, S, P)$ une grammaire hors-contexte : N est l'ensemble des symboles non-terminaux, Δ l'alphabet des terminaux, S l'axiome de départ et $P \subseteq N \times \Sigma^*$ (où $\Sigma = N \cup \Delta$) est l'ensemble des règles de productions .

Nous noterons par $u \Rightarrow v$ si u se dérive en v par la règle de production P.

Le langage reconnu par Γ noté $L(\Gamma)$ est l'ensemble des $m \in \Delta^*$ tels que $S \Rightarrow^* m$. On peut supposer que $L(\Gamma)$ ne contienne pas le mot vide.

À toute règle de production $V \to w$ on associe la contrainte de mots inverse hors-contexte $xwy \sqsubseteq xVy$ et on associe la règle de réécriture $w \to V$. Ainsi à P on associe un ensemble de contraintes de mots $\mathscr{C}$ et un système de réécriture R. On notera que $x \to y$ si et seulement si $x \Rightarrow y$.

Pour tout règle de R de la forme $u \to v$, on a $|u| \geq |v|$ donc pour tout mot u, il n'y a qu'un nombre fini de descendants. Ainsi R est descendant-limité.

D'après le corollaire 90, on a $x\Delta^+ y \sqsubseteq xSy$ si et seulement si on a $\Delta^+ \subseteq \mathrm{Anc}_R(S)$ si et seulement si Γ reconnaît Δ^+. Ce dernier point est indécidable. □

L'objectif de la section suivante est de donner des critères à imposer aux contraintes pour que le problème d'inclusion de requêtes RPQ soit décidable. On cherche à déterminer des classes de systèmes de réécritures qui vérifient ces deux propriétés :

— Les systèmes sont descendant-limités
— Les systèmes préservent les langages réguliers

3.3 Dureté du problème d'inclusion sous contraintes

L'objectif de cette section est de proposer des conditions suffisantes pour que le problème d'inclusion de requêtes sous contraintes de mots soit décidable.

Notre corollaire 90 nous assure que si le système de réécriture associé aux contraintes préserve la régularité et est descendant-limité alors l'inclusion de requêtes RPQ sous contraintes de mots est décidable.

Nous utilisons un algorithme qui construit (quand il termine) un automate qui reconnaît les ancêtres d'un langage régulier en complétant naïvement des transitions pour chaque règle de réécriture.

Pour chaque run étiqueté par r de q à q' tel qu'il existe l avec (l,r) dans R on construit un nouveau run étiqueté par l de q à q'.

Ainsi, si A reconnaît le langage L alors cette transformation permet de reconnaître tous les mots m tel qu'il existe $w \in L$ avec $m \multimap w$. On note $T_R(A)$ l'automate obtenu. On rappelle que $\multimap$ désigne la réécriture parallèle (voir définition 32)

Afin de contrôler un minimum la taille de l'automate $T_R(A)$ nous allons "factoriser" les chemins :

Si plusieurs runs commencent par un même état q et reconnaissent tous r, alors on construit un unique run qui reconnaît l privé de sa dernière lettre, puis de ce nouvel état on relie ce run à tous les états atteints par les run qui reconnaissent r.

De même, nous devons distinguer si l est de longueur 1 ou non : si l est de longueur 1 alors aucun état n'est ajouté.

On commence par définir l'ensemble des états accessibles depuis un état q en suivant un run qui lit un mot $r = b_1...b_m$ fixé.

Afin de ne pas ajouter des runs à des runs dont l'ajout a déjà été effectué auparavant nous annotons les transitions de symboles + et −. Si une transition a été créée alors nous l'annotons +, toutes les autres seront annotés −, ainsi un run n'a pas déjà construit un nouveau run si et seulement si il contient au moins une transition annotée +. Nous dirons qu'un tel run est positif.

Nous donnons deux exemples avec le système $R = \{(bc, ab), (a, cc)\}$ (on associe 1 à la règle (bc, ab) et 2 à la règle (a, cc)) :

- S'il existe un run qui lit ab i.e. de la forme $q_1 a q_2 b q_3$ et qu'une des transitions est positives (on a soit $(q_1, a, q_2, +) \in \delta$ soit $(q_2, b, q_3, +) \in \delta$) alors on ajoute un nouveau run entre q_1 et q_3 qui lit bc. Nous notons le nouvel état $\mathrm{st}(q_1, 1, 2)$ (le second membre est 1 car il s'agit de la règle numéro 1) pour construire le run $q_1 b \mathrm{st}(q_1, 1, 2) c q_3$. Les deux nouvelles transitions sont considérées positives.

- S'il existe un run qui cc donc de la forme $q_1 c q_2 c q_3$ alors on ajoute la transition $(q_1, a, q_3, +)$ à l'automate.

Une fois tous les runs possibles ajoutés, toutes les transitions de l'automate (avant complétion) passent à des transitions négatives. Sinon, par exemple, le run qui lit ab

construirait à chaque étape un nouveau run qui lit bc.

On étend donc la notion d'automate avec des transitions signées :

Définition 95

Un automate signé est un quintuplet $A_s = (Q, \Sigma, \delta_s, I, F)$ où Q est un ensemble d'états, Σ un alphabet, I et F respectivement les états initiaux et finaux et δ_s les transitions signées, i.e. un sous-ensemble de $Q \times \Sigma \times Q \times \{-, +\}$.
Le langage reconnu par A_s est le langage $L(A_s)$ reconnu par l'automate $(Q, \Sigma, \delta, I, F)$ où δ est la projection de δ_s sur ses trois premières composantes.

Les transitions dont la quatrième composante est + sont dites positives.

À tout automate $A = (Q, \Sigma, \delta, I, F)$, on associe de façon injective un automate signé A_+ défini par $(Q, \Sigma, \delta_+, I, F)$ avec $\delta_+ = \{(q, a, q', +), (q, a, q') \in \delta\}$.

Définition 96

Soit A_s un automate signé. Un run sur A est une suite d'états et de lettres $q_0 a_0 q_1 ... q_n a_n q_{n+1}$ telle que pour tout i, $(q_i, a_i, q_{i+1}, -)$ ou $(q_i, a_i, q_{i+1}, +)$ est une transition de A (i.e. $\in \delta$).
Un run $q_0 a_0 q_1 ... q_n a_n q_{n+1}$ est dit positif (resp. strictement positif) s'il existe une transition positive dans le run (resp. toutes les transitions du run sont positives).

Nous explicitons maintenant comment nous allons compléter l'automate formellement pour calculer les ancêtres :

Définition 97

Soient $A = (Q, \Sigma, \delta, I, F)$ un automate signé, $q \in Q$ et $a_1 ... a_n$ un mot de Σ^* on pose :

$$\text{Paths}_+(A, q, a_1 ... a_n) = \{q' \in Q, \text{ il existe un run positif } q a_1 q_2 a_2 ... q_n a_n q' \text{ dans } A\}$$

L'idée est alors la suivante : on considère un automate signé A. Pour tous les run positifs qui lisent un r tel que (l, r) est dans R, on construit un run qui lit l.

Mais s'il existe deux runs qui lisent le même r et partent du même état, on construit qu'une seul fois les nouveaux états. Par exemple, avec la règle (cc, ab) si on a $qaq'bq''$ et $qaq'_2bq''_2$ alors on construit un nouvel état $\text{st}(q,k,2)$ et une transition $(q,c,\text{st}(q,k,2))$ puis deux transitions $(\text{st}(q,k,2),c,q'')$ et $(\text{st}(q,k,2),c,q''_2)$ au lieux de créer deux états et deux runs complets.

On a donc 4 cas à gérer :

- si le membre gauche est de longueur 1 alors on ajoute juste une transition sans créer d'état (c'est δ_4).

- sinon alors il y a l'état à créer et à relier depuis q(c'est δ_2). Puis le run entre les états créés (c'est δ_1) et enfin relier le dernier état créé avec toutes les fins de run comme q'' et q''_2 dans l'exemmple (c'est δ_3).

On fait cela en parallèle, d'où les union.

Chaque état nouvellement créé est dénoté par un $\text{st}(q,k,j)$ où q désigne le premier état du run, k désigne le numéro d'une règle de R (la règle (l_k,r_k)) et j la position de la lettre.

Définition 98

Soient $A_s = (Q, \Sigma, \delta_s, I, F)$ un automate signé, R un système de réécriture sur Σ. On définit par $T_R(A)$ l'automate signé $(Q_{new}, \Sigma, \delta_{new}, q_0, F)$ où :

$$Q_{new} = Q \cup \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}_+(A,q,r_k) \neq \emptyset \wedge |l_k| > 1}} \left(\bigcup_{j=2}^{|l_k|} \{\text{st}(q,k,j)\} \right)$$

$$\delta_{new} = \delta_- \cup \delta_1 \cup \delta_2 \cup \delta_3 \cup \delta_4$$

avec :

$$\delta_- = \{(q, x, q', -), \exists s \in \{+,-\}, (q, x, q', s) \in \delta\}$$

$$\delta_1 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}_+(A_s,q,r_k) \neq \emptyset \wedge |l_k| > 2}} \left(\bigcup_{j=2}^{|l_k|-1} \{(\text{st}(q,k,j), l_k[j], \text{st}(q,k,j+1), +)\} \right)$$

$$\delta_2 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}_+(A_s,q,r_k) \neq \emptyset \wedge |l_k| > 1}} \{(q, l_k[1], \text{st}(q,k,2), +)\}$$

$$\delta_3 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ |l_k| > 1}} \left(\bigcup_{q' \in \text{Paths}_+(A_s,q,r_k)} \{\text{st}(q,k,|l_k|), l_k[|l_k|], q', +))\} \right)$$

$$\delta_4 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ |l_k| = 1}} \left(\bigcup_{q' \in \text{Paths}_+(A_s,q,r_k)} \{(q, l_k[|l_k|], q', +))\} \right)$$

On peut appliquer de manière itérative la construction de l'automate pour définir à partir d'un automate A un automate $T_R^i(A)$.

On peut estimer la taille de l'automate obtenu au bout de k étapes :

Théorème 99

Soient R un système de réécriture, $A = (Q, \Sigma, \delta, q_0, F)$ un automate non déterministe signé. On note Q_i l'ensemble des états de $T_R^i(A)$. On a :

$$\#Q_i \leq \#Q \times |R|^i$$

Démonstration. On a tout d'abord l'inégalité $\#R \leq |R|$.

Montrons le résultat par récurrence :

Initialisation :
On a $Q_0 = Q$ et $\#Q \times |R|^0 = \#Q$, ce qui prouve l'initialisation.
Récurrence : Supposons que $\#Q_i \leq \#Q \times |R|^i$.
On a

$$Q_{i+1} = Q_i \cup \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}(A,q,r_k) \neq \emptyset \wedge |l_k| > 1}} \left(\bigcup_{j=2}^{|l_k|} \{st(q,k,j)\} \right)$$

Donc $\#Q_{i+1} \leq \#Q_i + \#Q_i \times \left(\sum_{(l_k,r_k) \in R} (|l_k| - 1) \right) \leq \#Q_i + \#Q_i \times (|R| - \#R) \leq \#Q_i \times |R|$
Par hypothèse de récurrence on a $\#Q_i \leq \#Q \times |R|^i$ donc :

$$\#Q_{i+1} \leq \#Q \times |R|^i \times |R| = \#Q \times |R|^{i+1}$$

Ce qui achève la preuve.

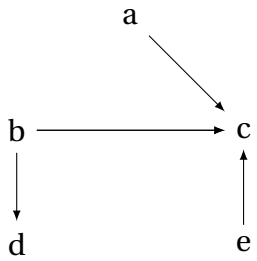
□

Malheureusement, de manière générale, il n'existe pas toujours de k tel que $T_R^{k+1}(A) = T_R^k(A)$ pour tout A. Par exemple : le système $a \to aa$. En effet si on considère les automates A_n qui lisent a^{2^n} en un unique run, alors il faut calculer $T^n(A_n)$ pour terminer la complétion. Donc il n'existe pas de tel k.

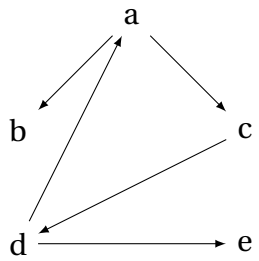
Nous nous intéressons aux systèmes de réécriture dont la complétion termine. Certaines de ces classes seront étudiées au chapitre suivant.

Nous proposons une première classe qui assure la terminaison de la construction : les systèmes NR (non-récursif) puis nous montrons que cette classe peu expressive suffit à montrer la dureté du problème d'inclusion sous-contrainte.

On associe une arête entre les lettres des membres gauches de R et celles des membres droits. Pour définir NR on impose que ce graphe soit acyclique.



$\{abe \to c, bb \to ccd\}$ est NR



$\{aa \to bc, dd \to aea, c \to d\}$ ne l'est pas

Définition 100

Soit R un système de réécriture. Le graphe-lettre de R noté G_R est le graphe (Σ, E) où E contient tous les couples (a,b) tels qu'il existe $(u,v) \in R$ avec a une lettre de u et b une lettre de v.

Définition 101

Un système de réécriture est dit NR si son graphe-lettre est acyclique.

Proposition 102

Soit R un système de réécriture,

$$R \in \text{NR si et seulement si } R^{-1} \in \text{NR}$$

Démonstration. On a clairement $G_{R^{-1}} = G_R^{-1}$, or l'inverse d'un graphe acyclique est acyclique. □

Pour pouvoir utiliser la classe NR dans le contexte de l'inclusion de requêtes RPQ sous contraintes de mots nous devons démontrer que les systèmes NR sont descendant-limités.

Nous montrerons que si un système est dans $\text{MB}_{\max}(k)$ alors la complétion admet un point fixe après k étapes.

Nous montrerons au théorème 142 que les systèmes non-récursifs (NR) ont une complexité parallèle bornée par$|R|$. En particulier les systèmes NR sont terminants, donc descendants-limités et :

Théorème 103

Pour tout système R dans NR on a $T_R^{|R|+1}(A) = T^{|R|}(A)$ pour tout automate A.

Il suffit donc de calculer $T_R^{|R|}(A_s)$ pour reconnaître $\mathrm{Anc}(L')$ avec A un automate qui reconnaît L' et A_s l'automate signé correspondant.

Nous évaluons la complexité de calcul de l'automate des ancêtres :

Proposition 104

Soient R un système de réécriture, A un automate, et k un entier en unaire. Le calcul de $T_R^k(A)$ s'effectue en EXPTIME.

Démonstration. Soit A un automate non déterministe, pour chaque état q et chaque règle (l_k, r_k) de R on souhaite déterminer les q' tels que $q' \in \mathrm{Paths}_+(A, q, r_k)$ pour ajouter le nouveau run $q l_k q'$. L'ajout d'un run est linéaire par rapport à la longueur maximale des $|l_k|$ de R.

Le calcul de $\mathrm{Paths}_+(A, q, m)$ pour un mot m et un état q est un calcul d'accessibilité qui se fait en temps $\#Q^2 \times |m|$.

Donc la construction de $T_R(A)$ s'effectue en $\leq \#Q^2 \times \#R \times (\max_{(l_k,r_k)\in R} |r_k| + \max_{(l_k,r_k)\in R} |l_k|) \leq \#Q^3 \times |R|^2$ car $\max_{(l_k,r_k)\in R} |r_k| + \max_{(l_k,r_k)\in R} |l_k| \leq |R|$ et $\#R \leq |R|$.

D'après la proposition 99, la taille de $T_R^i(A)$ est majorée par $\#Q \times |R|^i$.

Ainsi, on a une complexité totale égale à :

$$\sum_{i=0}^{k} |T_R^i(A)|^3 \times |R|^2 = |R|^2 \times \#Q^3 \times \sum_{i=0}^{k} |R|^{3i}$$

Donc la construction de $T_R^k(A)$ s'effectue en EXPTIME. □

Ainsi on en déduit que :

Théorème 105

On a :

PROBLEME : Inclusion de requêtes RPQ sous contraintes NR **INSTANCE** : $\mathscr{C}$ un système de contraintes de mots NR, $Q = xLy$ et $Q' = xL'y$ deux RPQ **QUESTION** : $Q \sqsubseteq_{\mathscr{C}} Q'$? **COMPLEXITE** : EXPSPACE

Démonstration. Nous savons que tout système NR est terminant donc descendant-limité et d'après le théorème 90 il suffit de tester l'inclusion $L \subseteq \text{Anc}_R(L')$ où R est le système de réécriture associé à $\mathscr{C}$.

Soit A l'automate qui reconnaît L'.

Puisque R est NR, l'automate des ancêtres de tout langage régulier est calculable en EXPTIME d'après 104 et est de taille exponentielle d'après 99.

L'inclusion de deux langages réguliers sous formes d'automates non déterministe est en PSPACE en la taille des automates, ce qui permet de conclure. □

Nous montrons maintenant que le problème d'inclusion de requêtes RPQ sous contraintes issues d'un système de réécriture NR est EXPSPACE-complet.

On rappelle le problème de décision suivant :

PROBLEME : reconnaissance du mot vide par une machine de Turing de mémoire exponentielle **INSTANCE** : M une machine de Turing déterministe de taille m et de ruban de longueur 2^{m^k} avec un unique état final **QUESTION** : M reconnaît ϵ ? **COMPLEXITE** : EXPSPACE-complet

L'objectif est de réduire le problème de reconnaissance du mot vide dans le problème d'inclusion $\neg(L \subseteq \text{Anc}_R(L'))$ où L et L' sont des langages réguliers et R un système de réécriture NR.

Soit \$ le symbole blanc du ruban. On pose $\Sigma_\$ = \Sigma \cup \{\$\}$.

Soit $M = (Q, \Sigma, \delta, q_0, q_f)$ une machine de Turing de taille m où Q est l'ensemble des états de M, Σ est l'alphabet du ruban, δ est une fonction de $Q \times \Sigma_\$$ dans $Q \times \Sigma \times \{\leftarrow, \rightarrow\}$. La machine travaille sur un ruban de taille exponentielle : on peut supposer le ruban de longueur 2^{m^k}, soit 2^n en posant $n = m^k$.

Définition 106

Une configuration est l'ensemble des symboles inscrit sur le ruban, l'état dans lequel se trouve la machine M et la position du ruban désignée par la tête.
On modélise une configuration par un mot de longueur 2^n du langage :

$$\Sigma^*(Q \times \Sigma_\$)\Sigma^*\$^*$$

La position de l'unique lettre de la configuration qui appartient à $Q \times \Sigma_\$$ indique la position de la tête de lecture, la première composante de cette lettre indique l'état dans lequel se trouve la machine M.

La configuration initiale de M est le mot $(q_0, \$)\$^{2^n - y1}$.

Définition 107

Soient C_i et C_{i+1} deux configurations de $\Sigma^*(Q \times \Sigma_\$)\Sigma^*\$^*$.
Soit j la position de la tête de lecture de C_i. Si (q, a) est la j-ième lettre de C_i (c'est l'unique lettre de $Q \times \Sigma_\$$) alors (C_i, C_{i+1}) est une transition valide si et seulement si :

— $C_{i+1}[j] = x, C_{i+1}[j-1] = (q', C_i[j-1])$ et $C_{i+1}[k] = C_i[k]$ pour $k \notin \{j, j-1\}$ si $\delta(q, a) = (q', x, \leftarrow)$ [a]

— $C_{i+1}[j] = x, C_{i+1}[j+1] = (q', C_i[j+1])$ et $C_{i+1}[k] = C_i[k]$ pour $k \notin \{j, j+1\}$ si $\delta(q, a) = (q', x, \rightarrow)$ [b]

a. autrement dit, la machine a écrit x sur a et s'est déplacée à gauche en passant à l'état q'

b. autrement dit, la machine a écrit x sur a et s'est déplacée à droite en passant à l'état q'

Par définition nous avons la proposition suivante :

Proposition 108

M reconnaît ϵ si et seulement s'il existe une séquence finie de configurations $C_0, ..., C_t$ telle que

- C_0 est la configuration initiale
- Pour tout $i < t$, (C_i, C_{i+1}) est valide
- C_t est la configuration $(q_f, \$)\$^{2^n-1}$ [a]

a. On peut supposer que la tête de lecture termine tout à gauche quitte à ajouter des transitions de déplacement de la tête vers la gauche à partir de l'état final qui efface les lettres en \$.

Soit # un symbole qui n'appartient pas à $\Sigma_\$$.

L'idée est de considérer le langage des configurations consécutives qu'une machine de Turing peut parcourir que l'on concatène entre elles par le nouveau symbole #.

On pose alors :

$$L = \#(q_0, \$)\$^*\#(\#\Sigma^*(Q \times \Sigma_\$)\Sigma^*\$^*\#)^*\#(q_f, \$)\$^*\#$$

On introduit un symbole d'erreur *e*.

Nous allons introduire alors deux systèmes de réécritures R_l et R_v.

- Le premier, R_l, permet de tester si tous les facteurs entre deux # qui ne comportent pas de # (a priori les configurations) sont bien de longueur 2^n : on montre qu'il existe une dérivation dont le but est un mot contenant le symbole *e* si et seulement si une des *a priori* configurations n'est pas de la bonne longueur.

- Le second, R_v, permet de tester si deux configurations successives, i.e. un facteur d'un mot de *L* de la forme $\#C\#\#C'\#$, forment un couple valide pour *M* : on montre qu'il existe une dérivation dont le but est un mot contenant le symbole *e* si et seulement si (C, C') n'est pas valide.

On pose alors $R = R_l \cup R_v$ on aura ainsi :

Pour tout mot *u* de *L*, il existe une dérivation d'origine *u* et de but un mot contenant *e* si et seulement si *M* ne reconnaît pas le mot vide.

On pose $W = \Sigma \cup Q \times \Sigma \cup \{\$\} \cup \{\#\}$.

Définition 109

Soit R_l le système de réécriture défini par :
- $a \to l_0$ pour tout $a \in \Sigma_\$ \cup Q \times \Sigma$
- $l_i l_i \to l_{i+1}$ pour $i < n+1$
- $l_{n+1} \to e$
- $\# l_i \to \mathfrak{I}_i$ pour $i \leq n$
- $\mathfrak{I}_i l_k \to \mathfrak{I}_k$ pour $k < i \leq n$
- $\mathfrak{I}_k \# \to e$ pour $k < n$

Lemme 110

Pour tout mot m de L.
Il existe un facteur de m de la forme $\#w\#$ tel que w ne contient pas de # et de longueur $\neq 2^n$ et $< 2^{n+1}$ ou il existe un facteur w de m ne contenant pas de # de longueur 2^{n+1} si et seulement s'il existe une dérivation $m \to^*_{R_l} m'$ avec m' qui contient e.

Démonstration. 1)

Montrons par récurrence sur t que si il existe une séquence finie d'entiers positifs $(i_0, ..., i_t)$ tous distincts telle que $|w| = 2^{i_0} + 2^{i_1} + ... + 2^{i_t}$. alors il existe une dérivation $w \to^* l_{i_0} l_{i_1} ... l_{i_t}$.

On remarquera qu'on a $w \to^* l_0^{|w|}$.

- Initialisation : Si $t = 0$ alors on a $|w| = 2^{i_0}$ alors $w \to^* l_0^{2^{i_0}} \to^* l_1^{2^{i_0-1}} \to^* l_2^{2^{i_0-2}} ... \to^* l_{i_0}$.

- Hérédité : On suppose la propriété vraie pour $t-1$, montrons la pour t. On a $|w| \geq 2^{i_t}$ donc il existe w' et w'' tels que $w = w'w''$ avec $|w'| = 2^{i_t}$, donc $|w''| = 2^{i_0} + ... + 2^{i_t} - 2^{i_t} = 2^{i_0} + ... + 2^{i_{t-1}}$.

On a donc $w' \to^* l_{i_t}$ et par hypothèse de récurrence $w'' \to^* l_{i_0} ... l_{i_{t-1}}$ ainsi, on a $w \to^* l_{i_0} ... l_{i_t}$.

Ce qui termine la récurrence.

2)

Montrons maintenant par récurrence sur t que

Pour tout uplet d'entiers $(i_0, i_1, ..., i_t)$ tel que $n \geq i_0 > i_1 > i_2 ... > i_t$ il existe une dérivation $\# l_{i_0} l_{i_1} ... l_{i_t} \to^* \mathfrak{I}_{i_t}$.

- Initialisation : soit $t = 0$, on considère un uplet (i_0) avec $n \geq i_0 > i_1$ on a $\# l_{i_0} \to \mathfrak{I}_{i_0}$.

- Hérédité : supposons la propriété vraie au rang $t-1$. On considère un uplet $(i_0, i_1, ..., i_t)$ tel que $n \geq i_0 > i_1 > i_2 ... > i_t$. Par hypothèse de récurrence on a

$\#l_{i_0}l_{i_1}...l_{i_t}\# \rightarrow^* \mathfrak{I}_{i_{t-1}}l_{i_t}\# \rightarrow \mathfrak{I}_{i_t}\#$. Ce qui termine la récurrence.

3)

On suppose que $|w| < 2^{n+1}$.

Pour tout entier $x < 2^{n+1}$ il existe un unique uplet $(i_0, i_1, ..., i_t)$ avec $n \geq i_0 > i_1 > i_2 > ... > i_t$ tel que $x = 2^{i_0} + ... + 2^{i_t}$.

On pose $x = |w|$.

Montrons maintenant que :

si $t > 0$ ou $i_0 < n$ alors il existe une dérivation de la forme $\#w\# \rightarrow^* e$.

On suppose que $t = 0$ on a alors $i_0 < n$. On a $x = 2^{i_0}$, donc d'après 1) on a $x \rightarrow l_{i_0}$ donc $\#x\# \rightarrow^* \#l_{i_0}\# \rightarrow \mathfrak{I}_{i_0}\# \rightarrow e$ car $i_0 < n$.

On suppose maintenant que $t > 0$. D'après 1), il existe une dérivation $w \rightarrow^* l_{i_0}...l_{i_t}$. Puis d'après 2) il existe une dérivation $\#l_{i_0}...l_{i_t} \rightarrow^* \mathfrak{I}_{i_t}$, on a $i_t < n$. Donc $\#w\# \rightarrow^* \#l_{i_0}...l_{i_t}\# \rightarrow^* \mathfrak{I}_{i_t}\# \rightarrow e$.

4)

Soit m un mot de L. Supposons qu'il existe un facteur de la forme $\#w\#$ dans m tel que w ne contiennent pas de #, dont la longueur vérifie $\neq 2^n$.

Si la longueur est $< 2^{n+1}$, d'après 3), il existe un unique uplet $(i_0, i_1, ..., i_t)$ avec $n \geq i_0 > i_1 > i_2 > ... > i_t$ tel que $|w| = 2^{i_0} + ... + 2^{i_t}$. Comme $|w| \neq 2^n$, on a soit $t = 0$ et $i_0 = n$, soit $t > 0$. Donc il existe une dérivation $\#w\# \rightarrow^* e$. Or $\#w\#$ facteur de m, donc il existe une dérivation $m \rightarrow^* m'$ avec e lettre de m'.

Supposons maintenant que m admet un facteur w sans # tel que $|w| \geq 2^{n+1}$ alors w se décompose en $w'w''$ avec $|w'| = 2^{n+1}$ donc il existe une dérivation $\#w\# \rightarrow \#l_{n+1}w''\# \rightarrow \#ew''\#$.

5)

Réciproquement, supposons qu'il existe une dérivation de la forme $m \rightarrow^* m'$ avec e lettre de m'.

Comme le système est non-récursif (i.e. $\in NR$), il existe un facteur u de m tel que $u \rightarrow^* e$.

e ne peut être produit que par des règles de la forme

1. $l_{n+1} \rightarrow e$
2. $\#l_i\# \rightarrow e$
3. $\mathfrak{I}_k\# \rightarrow e$

Premier cas : $l_{n+1} \rightarrow e$:

le symbole l_{n+1} apparaît dans la dérivation alors comme l'unique règle de production de l_{i+1} est $l_i l_i \rightarrow l_{i+1}$ pour tout i. Aucune lettre l_i n'appartient à m, donc pour avoir l_{n+1} il faut 2^{n+1} lettres consécutives l_0 puisqu'aucune lettre de u n'est réécrite en autre chose que l_0 et que u est l'origine de la dérivation, on en déduit que $u \rightarrow^* l_0^{2^{n+1}}$, donc de longueur 2^{n+1}.

Second cas : $\mathfrak{I}_k\# \rightarrow e$:

Les seules règles qui produisent $\mathfrak{I}_k$ sont de la forme :

— $\#l_i \to \mathfrak{I}_i$
— $\mathfrak{I}_i l_k \to \mathfrak{I}_k$

Il existe $t \geq 0$ tel que la seconde règle a été appliquée exactement t fois pour produire successivement $\mathfrak{I}_{i_0}, ...\mathfrak{I}_{i_t}$ avec $i_t = k$.

Par récurrence sur j entre t et 0, montrons qu'il existe un mot de la dérivation qui admet le facteur $\mathfrak{I}_{i_j} l_{i_{j+1}} ... l_{i_t}\#$ et on a $i_j > i_{j+1} ... > i_t$ et $i_{j+1} < n$.

- Initialisation : Si $j = t$ alors on a $\mathfrak{I}_{i_t}\# \to e$, et cette règle est appliquée si $k < n$ donc la propriété est vraie.

- Hérédité : Soit j entre 1 et t tel qu'il existe un mot de la dérivation qui admet le facteur $\mathfrak{I}_{i_j} l_{i_{j+1}} ... l_{i_t}\#$ et on a $n > i_j > i_{j+1} ... > i_t$. Alors $\mathfrak{I}_{i_j}$ est produit soit par une règle de la forme $\#l_i \to \mathfrak{I}_i$ soit de la forme $\mathfrak{I}_i l_k \to \mathfrak{I}_k$. La première est exclue car il y a exactement t fois application de la première règle.

Ainsi, la règle $\mathfrak{I}_i l_k \to \mathfrak{I}_k$ a été appliquée pour produire $\mathfrak{I}_{i_j}$, on pose i_{j-1} l'entier i, comme la règle a été appliquée on a $i_{j-1} > i_j$ et $i_{j-1} < n$. On a alors :

$$\mathfrak{I}_{i_{j-1}} l_{i_j} l_{i_{j+1}} ... l_{i_t}\# \to \mathfrak{I}_{i_j} l_{i_{j+1}} ... l_{i_t}\#$$

Ce qui achève la récurrence.

Il existe donc $i_0, ..., i_t$ tels que $i_0 > i_1 > ... > i_t$, $i_1 < n$ et un mot de la dérivation qui contient un facteur de la forme $\mathfrak{I}_{i_0} l_{i_1} ... l_{i_t}\#$. Or la seconde règle a été appliquée t fois, il reste donc la première règle.

La règle $\#l_{i_0} \to \mathfrak{I}_{i_0}$ a donc été appliquée, on a donc $i_0 \leq n$, de plus :

Il existe un facteur de la forme $\#l_{i_0} l_{i_1} ... l_{i_t}\#$ dans la dérivation et on a $n \geq i_0 > i_1 > ... > i_t$. Or aucune des lettres l_{i_j} n'apparaît dans u, et pour produire un l_{i_j} il faut un facteur de la forme $l_0^{2^{i_j}}$. De même, les l_0 ne sont pas dans u, et les lettres de u ne peuvent devenir que des l_0 donc u est de la forme $\#w\#$ avec $|w| = \sum_{j=0}^{t} 2^{i_j} < 2^{n+1}$ qui ne contient pas de #..

Comme u est facteur de m, le lemme est prouvé.

□

Définition 111

Soit R_v le système de réécriture défini par :
- $a \to L_0$ pour tout $a \in W$
- $L_i L_i \to L_{i+1}$ pour $i < n$
- $(q,a) b L_n c d \to e$ pour tout $a, b, c \in \Sigma_\$$, $d \in Q \times \Sigma_\$$, $q \in Q$ tels que $x \neq c$ ou $d \neq (q', b)$ où $\delta(q,a) = (q', x, \to)$
- $a (q,b) L_n c d \to e$ pour tout $a, b, d \in \Sigma_\$$, $c \in Q \times \Sigma_\$$, $q \in Q$ tels que $x \neq d$ ou $c \neq (q', a)$ où $\delta(q,b) = (q', x, \leftarrow)$
- $a L_c d \to e$ pour tout $a, c \in \Sigma_\$$ avec $a \neq c$

Lemme 112

Soient C et C' deux configurations :
(C, C') n'est pas valide si et seulement s'il existe une dérivation de la forme $\#C\#\#C'\# \rightarrow_{R_v}^{*} m$ avec e lettre de m.

Démonstration. Tout d'abord, on remarquera que les seules règles de productions de L_0 sont depuis les lettres a de W. De plus, chacune de ces lettres n'est jamais produite par aucune règle. La seule règle qui produit L_{i+1} est la règle $L_iL_i \rightarrow L_{i+1}$. Le système étant NR, il existe une dérivation $\mu w\mu' \rightarrow \mu L_n\mu'$ si et seulement si $|w| = 2^n$.

Ensuite puisque C et C' sont des configurations on a $|C| = |C'| = 2^n$, et il existe une unique lettre de C (resp. C') qui appartient à $Q \times \Sigma$, on note j sa position.

On pose $m = \#C\#\#C'\#$. Par définition on a $m[j+1] \in Q \times \Sigma$. Et on a pour tout i entre 1 et 2^n, $m[i+1] = C[i]$ et $m[i+2^n+1] = C'[i]$.

On pose $(q', x, f) = \delta(m[j+1])$ $(= \delta(C[j]))$.

On suppose que $f = \leftarrow$ alors par définition :

(C, C') est non valide si et seulement l'un des trois conditions est vérifiées

1. $\exists k \notin \{j-1, j\}$, $C[k] \neq C'[k]$
2. $C'[j] \neq x$
3. $C'[j-1] \neq (q', C[j-1])$

Ces conditions sont respectivement équivalentes aux conditions

1. $\exists k \notin \{j-1, j\}$ avec k entre 1 et 2^n tel que $m[k+1] \neq m[k+2^n+1]$
2. $m[j+2^n+1] \neq x$
3. $m[j+2^n] \neq (q', m[j])$

Implication :

Supposons que (C, C') n'est pas valide.

- Premier cas : $\exists k \notin \{j-1, j\}$ avec k entre 1 et 2^n tel que $m[k+1] \neq m[k+2^n+1]$ et tel que $m[k+2^n+1] \in \Sigma_\$$.

Par hypothèse comme l'unique lettre de C qui appartiennent à $Q \times \Sigma_\$$ est celle de position j, alors la seule lettre de m dont la position est entre 2 et 2^n+1 qui appartiennent à $Q \times \Sigma_\$$ est $m[j+1]$, donc $m[k+1]$ appartient à $\Sigma_\$$.

On a $m = \mu awc\mu'$ avec $\mu = m[1]...m[k]$, $a = m[k+1]$, $w = m[k+2]...m[k+2^n]$, $c = m[k+2^n+1]$, $\mu' = m[k+2^n+2]...m[|m|]$.

On a $|w| = 2^n$ donc $m \rightarrow^* \mu aL_nc\mu'$.

Par hypothèse on a $m[k+1] \neq m[k+2^n+1]$ donc $a \neq c$ donc $a \neq c$ donc $aL_nc \rightarrow e$.

Ainsi $m \rightarrow^* \mu e\mu'$.

- Deuxième cas : $\exists k \notin \{j-1, j\}$ avec k entre 1 et 2^n tel que $m[k+1] \neq m[k+2^n+1]$ et tel que $m[k+2^n+1] \in Q \times \Sigma_\$$

Puisqu'il existe une unique lettre de C' de la forme $Q \times \Sigma_\$$, donc une unique position $> 2^n+1$ qui appartienne à $Q \times \Sigma_\$$

Puisque $k \notin \{j-1, j\}$ on a $k \neq j-1$ donc $m[j+2^n] \notin Q \times \Sigma$.

On a $m = \mu a(q,b)wcd\mu'$ avec $\mu = m[1]...m[j-1]$, $a = m[j]$, $(q,b) = m[j+1]$, $w = m[j+2]...m[j+2^n+1]$, $c = m[j+2^n]$, $d = m[j+2^n+1]$ et $\mu' = m[j+2^n+2]...m[|m|]$.

On a donc $a(q,b)L_ncd$ avec $c \notin Q \times \Sigma_\$$ donc $c \neq (q', a)$, donc $a(q,b)L_ncd \to e$.

On a $|w| = 2^n$ donc $m \to^* \mu abL_ncd\mu'$.

Ainsi $m \to^* \mu e\mu'$.

- Troisième cas : $m[j+2^n+1] \neq x$.

On a $m = \mu a(q,b)wcd\mu'$ avec $\mu = m[1]...m[j-1]$, $a = m[j]$, $(q,b) = m[j+1]$, $w = m[j+2]...m[j+2^n-1]$, $c = m[j+2^n]$, $d = m[j+2^n+1]$ et $\mu' = m[j+2^n+2]...m[|m|]$.

On a $|w| = 2^n$ donc $\mu a(q,b)wcd\mu' \to^* \mu a(q,b)L_ncd\mu'$.

Or on a $d \neq x$ donc $a(q,b)L_ncd \to e$.

Ainsi $m \to^* \mu e\mu'$.

- Quatrième cas : $m[j+2^n] \neq (q', m[j])$.

On a $m = \mu a(q,b)wcd\mu'$ avec $\mu = m[1]...m[j-1]$, $a = m[j]$, $(q,b) = m[j+1]$, $w = m[j+2]...m[j+2^n-1]$, $c = m[j+2^n]$, $d = m[j+2^n+1]$ et $\mu' = m[j+2^n+2]...m[|m|]$.

On a $|w| = 2^n$ donc $\mu a(q,b)wcd\mu' \to^* \mu a(q,b)L_ncd\mu'$.

Or on a $c \neq (q', a)$ donc $a(q,b)L_ncd \to e$.

Ainsi $m \to^* \mu e\mu'$.

Réciproque :

Supposons qu'il existe μ et μ' tel que $m \to^* \mu e\mu'$ avec $\mu, \mu' \in W^*$.

Les règles qui produisent e sont de la forme (on rappelle qu'on est dans le cas $f = \leftarrow$) :

— $aL_nc \to e$ avec $a, c \in \Sigma_\$$ et $a \neq c$

— $a(q,b)L_ncd \to e$ avec $x \neq d$ ou $c \neq (q', a)$

- Premier cas : on a $aL_nc \to e$ avec $a, c \in \Sigma_\$$ et $a \neq c$. Les lettres a, c ne sont jamais produites, il existe donc un facteur w tel que $|w| = 2^n$ et

$$awc \to^* aL_nc \to e$$

Notons $k+1$ la position de a. On a $m[k+1] \neq m[k+2^n+1]$ car $|w| = 2^n$. On a k entre 1 et 2^n car $m[1] = \# \notin \Sigma_\$$ et $a \in \Sigma_\$$, et $|C| = 2^n$.

Si $k = j$ alors $a = m[k+1] = m[j+1] \in Q \times \Sigma_\$$, contradiction.

Si $k \neq j-1$, il existe $k \notin \{j-1, j\}$ entre 1 et 2^n tel que $m[k+1] \neq m[k+2^n+1]$, donc $C[k] \neq C'[k]$. Donc (C, C') non valide.

Si $k = j-1$, alors $m[j+2^n] = c \notin Q \times \Sigma_\$$, donc $m[j+2^n] \neq (q', a)$, donc $C'[j-1] \neq (q', C[j-1])$. Donc (C, C') non valide.

- Deuxième cas : on a $a(q,b)L_n cd \to e$ avec $a,b,d \in$ et $c \in Q \times \Sigma_\$$ tels que $x \neq d$ ou $c \neq (q',a)$. Les lettres a,b,c,d ne sont jamais produites, il existe donc un facteur w tel que $|w| = 2^n$ et

$$a(q,b)wcd \to^* a(q,b)bL_n cd \to e$$

Supposons (C,C') valide. Par définition, on a $C'[j] = x$ et $C'[j-1] = (q', C[j-1])$. Or $j \leq 2^n$ donc $m[j+1] = (q,b)$ car c'est l'unique lettre entre les positions 1 et 2^n telle que $m[k+1] \in Q \times \Sigma_\$$.

On a donc $m[j] = a$, comme C est une configuration, on a $m[j+2^n+1] = C'[j] = x$ et $m[j+2^n] = C'[j-1] = (q', m[j]) = (q',a)$.

Or $|w| = 2^n$ donc $m[j+2^n+1] = c$ et $m[j+2^n] = d$, on a donc $x = d$ et $c = (q',a)$, contradiction.

Ainsi (C,C') est non valide.

Les mêmes cas apparaissent pour $f = \to$ et sont similaires à prouver.

□

On pose alors Ω l'ensemble des lettres qui apparaissent dans les systèmes R_l et R_v, i.e. $\Omega = W \cup \bigcup_{i=0}^n \{l_i\} \cup \bigcup_{i=0}^n L_i \cup \bigcup_{i=0}^n \mathfrak{I}_i$. On pose alors L' l'ensemble des mots qui contiennent un symbole e, i.e. $L' = \Omega^* e \Omega^*$.

On pose $R = R_l \cup R_v$.

Théorème 113

M reconnaît ϵ si et seulement si $\neg(L \subseteq \mathrm{Anc}_R(L'))$

Démonstration. Puisque les systèmes R_l et R_v n'utilisent pas de lettres produites en commun en dehors de e. Si $m \to m'$ avec $m' \in L'$ alors soit il existe un facteur w tel que $|w| \neq 2^n$, soit il existe un facteur $\#C\#\#C'\#$ avec C et C' deux configurations telles que (C,C') est non valide.

On remarque que s'il existe w tel que $\#w\#$ est un facteur d'un mot de L sans $\#$ alors w est une configuration si et seulement si $|w| = 2^n$.

Supposons que $L \subseteq \mathrm{Anc}_R(L')$. Pour toute séquence $(C_0,...,C_f)$ avec $C_0 \in (q_0,\$)\$^*$, $C_i \in \Sigma^*(Q \times \Sigma_\$)\Sigma^*\$^*$ pour $0 < i < f$ et $C_f \in (q_f,\$)\$^*)$ alors le mot $m = \#C_0\#\#C_1\#...\#C_f\#$ de L, il existe une dérivation de m dans un mot de L'. Les mots de L' contenant e :

- (Lemme 110) Soit il existe un facteur $\#w\#$ tel que w ne contient pas de $\#$ avec $|w| \neq 2^n$, donc w n'est pas une configuration. Or il existe i tel que $w = C_i$.

Donc il existe i tel que C_i n'est pas une configuration.

- (Lemme 112) Sinon, pour tout i, C_i est une configuration. Il existe donc un facteur $\#C_i\#\#C_{i+1}\#$ tel que (C_i, C_{i+1}) est non valide.

Ainsi, M ne reconnaît pas ϵ.

Réciproquement,

Si M ne reconnaît pas ϵ il existe une séquence finie $(C_0, ..., C_f)$ avec $C_0 \in (q_0, \$)\$^*$, $C_i \in \Sigma^*(Q \times \Sigma_\$)\Sigma^*\$^*$ pour $0 < i < f$ et $C_f \in (q_f, \$)\$^*$) telle que soient l'une d'elle ne soit pas une configuration, soit un couple (C_i, C_{i+1}) n'est pas valide.

Dans le premier cas, d'après le Lemme 110, il existe un facteur w sans # de longueur $\geq 2^{n+1}$ ou entre deux # de longueur $< 2^{n+1}$ et $\neq 2^n$, donc $m = \#C_0...\#C_f\#$ se dérive en un mot de L'

Dans le second cas, d'après le Lemme 112, il existe un facteur $\#C_i\#\#C_{i+1}\#$ qui dérive en e, donc $m = \#C_0...\#C_f\#$ se dérive en un mot de L'.

Ainsi tous les mots de L se dérivent en un mot de L', donc $L \subseteq \text{Anc}_R(L')$.

□

Ainsi :

Corollaire 114

Soit $\mathscr{C}$ un ensemble de contraintes de mots tel que le système de réécriture R est NR.

PROBLEME : Inclusion de requêtes RPQ sous contraintes NR
INSTANCE : $Q = xLy$ et $Q' = xL'y$ deux RPQ, C un ensemble de contraintes de mots dont le système associé est NR
QUESTION : $Q \sqsubseteq_{\mathscr{C}} Q'$?
COMPLEXITE : EXPSPACE-complet

3.4 Conclusion

Dans ce chapitre nous avons étudié le problème de l'inclusion de requêtes RPQ sous contraintes de mots.

Nous avons montré que dans le cas des modèles potentiellement infinis la réduction du problème vers le problème $L \subseteq \text{Anc}_R(L')$ est toujours possible. Dans le cas où nous sommes restreints aux bases de données graphes finies, nous avons montré que cette réduction n'est pas toujours valable et nous avons proposé un contre-exemple. Nous montrons aussi que le problème d'inclusion de RPQ sous contraintes est indécidable.

Nous avons proposé la condition d'être "descendant-limité" i.e. chaque mot a un nombre fini de descendants comme suffisante pour réduire le problème d'inclusion de requêtes sous contraintes de mots au problème d'inclusion de langages modulo un système de réécriture.

Les classes intéressantes sont descendants-limitées et préservent les ancêtres. Nous avons proposé une première classe limitée NR qui satisfait ces propriétés. Finalement, nous avons montré que le problème d'inclusion de requêtes sous ces systèmes est EXPSPACE-complète.

CHAPITRE 4

Complexité parallèle de la réécriture

Nous avons montré au chapitre précédent qu'une méthode pour décider le problème d'inclusion de requêtes RPQ sous contraintes de mots consiste à proposer des classes de systèmes de réécriture qui :

— préservent les réguliers pour les ancêtres

— sont descendants-limités

L'étude de la préservation de la régularité des ancêtres, ou quitte à inverser les systèmes, des descendants est un problème largement étudié [16, 83, 60, 49]. Malheureusement ces classes ne satisfont pas les deux propriétés à la fois : par exemple, le système $a \to ab$ préserve les réguliers (il est $\mathrm{MB}_{\min}(1)$) mais n'est pas descendant-limité.

Nous avons utilisé au chapitre 3 la complétion d'un automate où chaque étape de complétion reconnaît les ancêtres à un pas parallèle, ainsi nous souhaitons borner uniformément le nombre de pas parallèles des systèmes.

Après avoir étudié à l'aide de graphes les interactions entre les pas de réécriture, nous définissons deux classes de systèmes de réécriture : $\mathrm{MB}_{\max}$. Un système de réécriture est $\mathrm{MB}_{\max}(k)$ si toutes les dérivations sont de complexité parallèle $\leq k$. Un système de réécriture est $\mathrm{BPar}(k)$ si pour toute dérivation entre deux mots u et v il existe une dérivation entre ces deux mots dont la complexité parallèle est $\leq k$ i.e. on peut simuler toute dérivation par une de complexité parallèle $\leq k$.

Nous montrons que si les contraintes de mots correspondent à un système d'une de ces classes alors le problème d'inclusion de requêtes RPQ sous ces contraintes est EXPSPACE-complet.

Finalement nous montrons que le problème de décision $R \in \mathrm{MB}_{\max}(k)$? est PSPACE-complet.

4.1 Complexité parallèle

Dans cette section on étudie la structure et la notion de complexité parallèle d'une dérivation. Dans une dérivation, on peut permuter des règles qui n'ont pas de dépendance (on parle d'orthogonalité [14, 53, 67] voir def. 31) sans changer la structure de la dérivation et on peut donc construire une relation d'équivalence : deux dérivations sont dites équivalentes si on peut permuter des règles orthogonales pour transformer l'une en l'autre (cela correspond à la notion de "permutation equivalence" ou "Lévy permutation equivalence" en lambda-calcul [14, 72, 68, 62]).

Nous introduisons alors une mesure du nombre minimal de pas de réécriture parallèle à effectuer pour réaliser une dérivation (à équivalence près) : ce coût en nombre d'étapes parallèles est la complexité parallèle de la dérivation.

L'objectif de cette section est d'étudier les structures des dérivations.

Exemple 115

Soit R le système de réécriture $\{(aa,a),(ab,a)\}$. Considérons la dérivation :

$$\sigma_1 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{4,(ab,a)} aaaa \xrightarrow{3,(aa,a)} aaa \xrightarrow{1,(aa,a)} aa$$

On remarque par exemple que les deux derniers pas de réécriture sont orthogonaux, on peut donc les échanger :

$$aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{4,(ab,a)} aaaa \xrightarrow{1,(aa,a)} aaa \xrightarrow{2,(aa,a)} aa$$

On peut alors échanger le deuxième et le troisième pas puisque les deux sont orthogonaux :

$$aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{1,(aa,a)} aaab \xrightarrow{3,(ab,a)} aaa \xrightarrow{2,(aa,a)} aa$$

Enfin, les deux premiers pas sont orthogonaux on peut donc les échanger :

$$\sigma_2 = aaaaab \xrightarrow{1,(aa,a)} aaaab \xrightarrow{2,(aa,a)} aaab \xrightarrow{3,(ab,a)} aaa \xrightarrow{2,(aa,a)} aa$$

Les dérivations σ_1 et σ_2 ont la même structure, il suffit d'échanger des règles pour passer de l'une à l'autre. Les dérivations sont équivalentes.

On met en avant la structure des dérivations dans les systèmes de réécriture, pour cela on associe à chaque dérivation un graphe : le graphe de dérivation [1] qui

1. Le graphe de dérivation est une abstraction de la dérivation qui ne rend compte que des dépendances

rend compte des dépendances entre les règles de réécriture : si une lettre qui a été produite par le i-ème pas de réécriture est utilisée pour le j-ème pas de réécriture alors on construit une arête entre i et j. Deux dérivations équivalentes, qui ont la même structure, auront leurs graphes isomorphes. Nous avons fait le choix de ne rendre compte que des dépendances, cet affaiblissement de la structure de la dérivation ne permet pas d'obtenir la réciproque : deux graphes peuvent être isomorphes sans que les dérivations soient équivalentes. Le choix d'un hypergraphe ou l'ajout d'annotations sur les arêtes permettrait de rendre cet invariant complet mais n'est pas nécessaire pour ce manuscrit.

Exemple 116

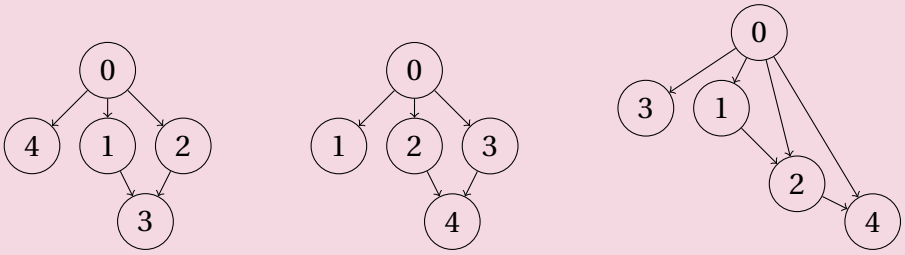
On considère les trois dérivations :

— $\sigma_1 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{4,(ab,a)} aaaa \xrightarrow{3,(aa,a)} aaa \xrightarrow{1,(aa,a)} aa$

— $\sigma_2 = aaaaab \xrightarrow{1,(aa,a)} aaaab \xrightarrow{2,(aa,a)} aaab \xrightarrow{3,(ab,a)} aaa \xrightarrow{2,(aa,a)} aa$

— $\sigma_3 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{3,(aa,a)} aaab \xrightarrow{1,(aa,a)} aab \xrightarrow{2,(ab,a)} aa$

Considérons par exemple la dérivation σ_1. Après le premier pas de réécriture on a $x_1[3] = a$ qui est la lettre produite. Au second pas de réécriture la lettre a en troisième position n'est pas utilisée, donc il n'y a pas d'arête entre 1 et 2. La position de la lettre n'a pas changée non plus (puisque le second pas de réécriture a été appliqué à sa droite). Au troisième pas de réécriture, la lettre $x_2[3] = x_1[3] = a$ est utilisée : c'est la première règle qui a produit cette lettre on a donc une arête entre 1 et 3, etc.
Les graphes des trois dérivations sont :



On voit que les deux premiers graphes sont isomorphes, en effet d'après l'exemple 115, les deux dérivations σ_1 et σ_2 sont équivalentes.
On remarque aussi que le graphe de σ_3 n'est pas isomorphe à l'un des deux autres graphes, on en déduit immédiatement que σ_3 n'est équivalente ni à σ_1 ni à σ_2.

Si plusieurs pas consécutifs sont indépendants (orthogonaux) alors on peut les exécuter de manière simultanée : on dit qu'on a effectué un pas parallèle (def.32).

La complexité parallèle de la dérivation est le nombre minimal de pas parallèles atteignable par permutation de pas de réécriture. Elle est représentée par la profondeur du graphe de la dérivation. Par définition, deux dérivations équivalentes ont la même complexité parallèle.

Exemple 117

On considère la dérivation :

$$\sigma_2 = aaaaab \xrightarrow{1,(aa,a)} aaaab \xrightarrow{2,(aa,a)} aaab \xrightarrow{3,(ab,a)} aaa \xrightarrow{2,(aa,a)} aa$$

Pour la dérivation σ_2, on remarque qu'on peut exécuter les 3 premiers pas en un unique pas parallèle. Il est en revanche impossible d'exécuter le quatrième pas de manière parallèle avec les pas précédents : ainsi la complexité parallèle de cette dérivation est 2.

$$\sigma_4 = aaaaab \multimap aaa \multimap aa$$

On remarque que le graphe de dérivation de σ_2 (donné à l'exemple 116) a ses trois premiers nœuds consécutifs 1, 2 et 3 qui sont orthogonaux et que la profondeur est de 2.
Puisque la dérivation σ_2 est équivalente à σ_1 les deux dérivations ont la même structure et donc la même complexité parallèle.

Enfin dans une dernière partie nous proposons, pour des raisons techniques, de simplifier des preuves en associant à chaque lettre la profondeur du pas de réécriture qui l'a produite : la complexité parallèle apparaît naturellement comme la plus grande valeur qui apparaît. Cette manière d'associer des entiers "profondeurs" a été introduite pour le lambda calcul dans [64, 12, 14] et pour les systèmes $\mathrm{MB}_{\mathrm{min}}$.

Exemple 118

On considère la dérivation :

$$\sigma_1 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{4,(ab,a)} aaaa \xrightarrow{3,(aa,a)} aaa \xrightarrow{1,(aa,a)} aa$$

On étiquette alors les premières lettres par des 0. Puis chaque lettre produite est étiqueté par 1+ le maximum des étiquettes des lettres qui ont été réécrites :

$$\sigma_1 = a_0a_0a_0a_0a_0b_0 \xrightarrow{3,(aa,a)} a_0a_0a_1a_0b_0 \xrightarrow{4,(ab,a)} a_0a_0a_1a_1 \xrightarrow{3,(aa,a)} a_0a_0a_2 \xrightarrow{1,(aa,a)} a_1a_2$$

Nous présentons cette section par ce plan :

1. Étape de réécriture parallèle et dérivation parallèle
2. Orthogonalité et permutation
3. Dérivations équivalentes et complexité parallèle
4. Graphe d'une dérivation et complexité parallèle
5. Dérivations indicées

Finalement, nous introduisons quelques notations techniques utiles pour le chapitre.

On dénote pour un pas de réécriture $r = x \xrightarrow{p,(u,v)} y$:

— l'ensemble $\text{Left}(r) = \{p, p+1, ..., p+|u|-1\}$ comme l'ensemble des positions des lettres réécrites

— $\text{Right}(r) = \{p, p+1, ..., p+|v|-1\}$ comme l'ensemble des positions des lettres produites.

On généralise alors cette notation dans le contexte d'une dérivation.

$$\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 ... \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$$

— $\text{Left}_\sigma(i) = \text{Left}(x_{i-1} \xrightarrow{p_{i-1},(u_{i-1},v_{i-1})} x_i)$ pour i entre 1 et $n-1$

— $\text{Right}_\sigma(i) = \text{Right}(x_{i-1} \xrightarrow{p_{i-1},(u_{i-1},v_{i-1})} x_i)$ pour i entre 1 et $n-1$

— $\text{Right}_\sigma(0) = \{1, ..., |x_0|\}$

Puis nous précisons les notations Right et Left dans le cas d'une dérivation inverse :

si σ est une dérivation de la forme :

$$\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 ... \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$$

alors on considère sa dérivation inverse σ^{-1} :

$$\sigma^{-1} = x_n \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_{n-1} \ldots \overset{p_0,(u_0,v_0)}{\longrightarrow} x_0$$

On a alors :
— $\text{Left}_{\sigma^{-1}}(i) = \text{Left}(x_i \overset{p_{i-1},(v_{i-1},u_{i-1})}{\longrightarrow} x_{i-1}) = \text{Right}_\sigma(i+1)$ pour i entre 1 et $n-1$
— $\text{Right}_{\sigma^{-1}}(i) = \text{Right}(x_i \overset{p_{i-1},(u_{i-1},v_{i-1})}{\longrightarrow} x_{i-1}) = \text{Left}_\sigma(i+1)$ pour i entre 1 et $n-1$
— $\text{Right}_{\sigma^{-1}}(n) = \{1, \ldots, |x_n|\}$

Réécriture parallèle et dérivation parallèle On étend la notion de dérivation (dîte séquentielle) sur la relation $\multimap$ avec la notion de dérivation parallèle. Elles admettent aussi une longueur (le nombre d'étapes parallèles), une origine et un but.

On peut décomposer tout pas de réécriture parallèle en une dérivation séquentielle en effectuant les règles de gauche à droite, successivement. On peut donc associer à toute dérivation parallèle une dérivation séquentielle que nous définissons comme la dérivation canonique associée.

Exemple 119

Le pas de réécriture parallèle

$$aababaabaa \multimap aaaaaa$$

peut être décomposé en la dérivation :

$$aababaabaa \overset{2,(ab,a)}{\longrightarrow} aaaaabaa \overset{5,(ab,a)}{\longrightarrow} aaaaaaaa \overset{6,(aa,a)}{\longrightarrow} aaaaaa$$

Définition 120

Soit R un système de réécriture, soit $\sigma = x_0 \multimap^m x_m$ une dérivation parallèle. Pour tout $t < m$, on peut décomposer x_t et x_{t+1} de la forme $x_t = \mu_0 u_1 \mu_1 \ldots \mu_{n-1} u_n \mu_n$ et $x_{t+1} = \mu_0 v_1 \mu_1 \ldots \mu_{n-1} v_n \mu_n$ tels que pour tout $i \leq n$, $(u_i, v_i) \in R$.
On associe à $x_t \multimap x_{t+1}$ la dérivation $x_t \overset{|\mu_0|,(u_1,v_1)}{\longrightarrow} y_1 \ldots y_{n-1} \overset{|\mu_0 u_1 \ldots \mu_{n-1}|,(u_n,v_n)}{\longrightarrow} x_{t+1}$
Par composition on associe une unique dérivation séquentielle à σ appelée la dérivation canonique associée à σ.

Orthogonalité et permutation de pas de réécriture

Définition 121

Soient R un système de réécriture et $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ une dérivation sur R. On dit que σ est orthogonale en i $(0 < i < n)$ si

$$\text{Left}_\sigma(i+1) \cap \text{Right}_\sigma(i) = \emptyset$$

Lorsqu'une dérivation est orthogonale en i on peut alors permuter les pas de réécriture i et $i+1$ car aucun élément p de $\text{Left}_\sigma(i+1)$ n'est dans $\text{Right}_\sigma(i)$.

$$x_{i-1} = \alpha u_{i-1}\beta u_i \gamma, x_i = \alpha v_{i-1}\beta u_i\gamma, x_{i+1} = \alpha v_{i-1}\beta v_i\gamma \text{ si } p_{i-1} < p_i$$

$$x_{i-1} = \alpha u_i \beta u_{i-1}\gamma, x_i = \alpha u_i\beta v_{i-1}\gamma, x_{i+1} = \alpha v_i\beta v_{i-1}\gamma \text{ sinon.}$$

En posant :
$p'_{i-1} = p_i + |u_{i-1}| - |v_{i-1}|, p'_i = p_{i-1}$ si $p_{i-1} < p_i$
$p'_{i-1} = p_i, p'_i = p_{i-1} + |v_i| - |u_i|$ sinon
on a :
$x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} \cdots x_{i-1} \overset{p'_{i-1},(u_i,v_i)}{\longrightarrow} x'_i \overset{p'_i,(u_{i-1},v_{i-1})}{\longrightarrow} x_{i+1} \overset{p_{i+1},(u_{i+1},v_{i+1})}{\longrightarrow} x_{i+2} \cdots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$
Cela correspond à permuter les deux pas de dérivation.
On dénote par $\text{exch}_{i,i+1}(\sigma)$ la dérivation obtenue après permutation des deux pas de réécriture. Notons qu'on a $x_{i-1} \multimap x_{i+1}$.

Exemple 122

On considère :

$$\sigma = aaaa \overset{3,(aa,a)}{\longrightarrow} aaa \overset{1,(aa,a)}{\longrightarrow} aa$$

On a $\text{Left}_\sigma(2) = \{1,2\}$ et $\text{Right}_\sigma(1) = 3$, ils sont disjoints on peut donc échanger les deux pas pour obtenir :

$$aaaa \overset{1,(aa,a)}{\longrightarrow} aaa \overset{2,(aa,a)}{\longrightarrow} aa$$

On précise la notion d'orthogonalité dans le cas d'une dérivation inverse, si σ^{-1} est la dérivation inverse de :

$$\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$$

alors la dérivation σ^{-1} va être orthogonale en i (on peut échanger les pas $i+1$ et i) si :

$$\text{Left}_{\sigma^{-1}}(i-1) \cap \text{Right}_{\sigma^{-1}}(i) = \emptyset$$

Dérivations équivalentes et complexité parallèle Deux dérivations sont dites équivalentes si on peut passer de l'une à l'autre par permutations de réécritures consécutives orthogonales.

Définition 123

On considère la relation (symétrique) entre deux dérivations σ et σ' définie par : il existe un entier i inférieur à la plus petite longueur des dérivations tel que $\sigma = \text{exch}_{i,i+1}(\sigma')$ (donc σ' est orthogonale en i). soit $\sim$ la clôture réflexive et transitive de cette relation.

On peut alors étendre $\sim$ aux dérivations parallèles en utilisant les dérivations canoniques associées.

Définition 124

Soit R un système de réécriture, soit σ une dérivation, soient σ' et σ'' deux dérivations parallèles. On étend la relation $\sim$ aux dérivations parallèles avec :

- $\sigma \sim \sigma'$ ssi σ est équivalente à la dérivation canonique associée à σ'
- $\sigma' \sim \sigma''$ ssi les dérivations canoniques associées respectivement à σ' et σ'' sont équivalentes

La relation est $\sim$ est stable par inversion [2] :

Proposition 125

Si $\sigma \sim \sigma'$ alors $\sigma^{-1} \sim \sigma'^{-1}$

Démonstration. Nous allons montrer que si σ est orthogonale en i alors σ^{-1} est orthogonale en i.

Dans ce cas si on échange les pas i et $i+1$ dans σ alors on peut échanger les mêmes pas dans σ^{-1}, ainsi par récurrence sur le nombre d'échanges, on a $\sigma^{-1} = \sigma'^{-1}$.

Soit $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ une dérivation. On suppose que σ est orthogonale en i avec i entre 1 et $n-1$.

On a $\text{Left}_{\sigma^{-1}}(i-1) \cap \text{Right}_{\sigma^{-1}}(i) = \text{Right}_{\sigma^{-1}}(i) \cap \text{Left}_{\sigma^{-1}}(i+1) = \emptyset$

Donc σ^{-1} est orthogonale en i. □

2. On inverse tous les pas de réécriture de la dérivation, voir chapitre 1

Comme nous l'avons déjà évoqué la complexité parallèle d'une dérivation représente le nombre minimal d'étapes parallèles nécessaires pour exécuter la dérivation.

Définition 126

Soit R un système de réécriture. Soit σ une dérivation. La complexité parallèle de σ est la plus petite longueur des dérivations parallèles équivalentes à σ.

Graphe d'une dérivation Une lettre qui n'est pas réécrite ne change pas au cours de la dérivation, mais sa position quant a elle est modifiée. Par exemple si a est de position 2 dans bab et que b devient cc alors cette même lettre est de position 3 dans $ccab$. On introduit la relation $\text{Corr}_\sigma(i,j)$ qui à une position p de x_i associe la position p' de x_j correspondant à la même lettre si elle n'a pas été réécrite.

Formellement :

Définition 127

Soit R un système de réécriture et une dérivation $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$. On définit pour tout $i \leq j$ la relation $\text{Corr}_\sigma(i,j)$ sur $\{1,...,|x_i|\} \times \{1,...,|x_j|\}$ définie par induction par :

- $(p,p') \in \text{Corr}_\sigma(i,i)$ ssi $p = p'$
- $(p,p') \in \text{Corr}_\sigma(i,i+1)$ ssi l'une des conditions est vérifiée :
  - $p < p_i$ et $p' = p$
  - $p \geq p_i + |u_i| - 1$ et $p' = p_i - |u_i| + |v_i|$
- $\text{Corr}_\sigma(i,j) = \text{Corr}_\sigma(i,j-1) \circ \text{Corr}_\sigma(j-1,j)$

On peut alors maintenant définir le graphe de dérivation, chaque nœud (sauf la racine) correspond à un pas de dérivation :

Définition 128

Soit R un système de réécriture et une dérivation $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$.
Le graphe de dérivation de σ est défini par $G(\sigma) = (N,E)$ avec :

- $N = \{0,1,...,n\}$
- $(i,j) \in E$ si et seulement s'il existe $p \in \text{Right}_\sigma(i)$ et $p' \in \text{Left}_\sigma(j)$ tels que $(p,p') \in \text{Corr}_\sigma(i,j-1)$.

On prendra garde que 0 est un nœud du graphe avec $\text{Right}_\sigma(0) = \{1, ..., |x_0|\}$.

Exemple 129

On considère la dérivation :
$\sigma_1 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{4,(ab,a)} aaaa \xrightarrow{3,(aa,a)} aaa \xrightarrow{1,(aa,a)} aa$
On a $\text{Right}_\sigma(1) = \{3\}$, $\text{Left}_\sigma(3) = \{3,4\}$ et, comme la lettre en 3-ème position dans x_1 n'a pas changé de position lors du deuxième pas de réécriture, $(3,3) \in \text{Corr}(1,2)$: on a donc une arête entre 1 et 3.

Par définition les graphes de dérivations sont enracinés (par le nœud 0), orientés et acycliques.

Définition 130

Soit R un système de réécriture. Soit σ une dérivation parallèle. Le graphe de dérivation de σ est le graphe $G(\sigma')$ où σ' est la dérivation canonique associée à σ.

Puisque dans un pas $\multimap$ les positions des lettres impliquées dans un pas de réécritures sont disjointes, on en déduit que la dérivation séquentielle associée à un pas de réécriture parallèle admet un graphe de dérivation dont les nœuds sont tous à la même profondeur. Par récurrence, on obtient alors :

Proposition 131

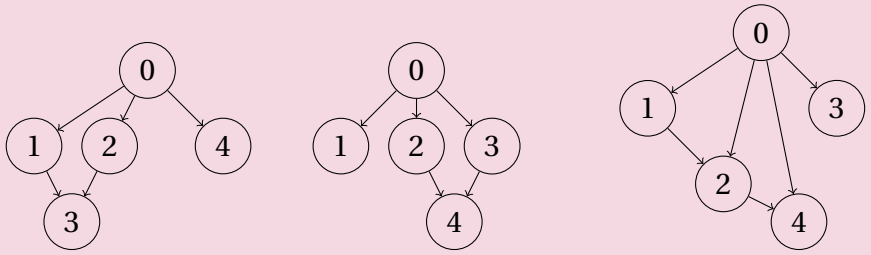
Soit R un système de réécriture. Soit σ une dérivation parallèle. Alors la profondeur du graphe de dérivation de σ est inférieure ou égale à la longueur de σ.

Permuter deux pas orthogonaux d'une dérivation correspond à échanger les nœuds i et $i+1$ du graphe. Si deux dérivations sont équivalentes alors elles ont des graphes de dérivation isomorphes.

Exemple 132

On considère les trois dérivations :
- $\sigma_1 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{4,(ab,a)} aaaa \xrightarrow{3,(aa,a)} aaa \xrightarrow{1,(aa,a)} aa$
- $\sigma_2 = aaaaab \xrightarrow{1,(aa,a)} aaaab \xrightarrow{2,(aa,a)} aaab \xrightarrow{3,(ab,a)} aaa \xrightarrow{2,(aa,a)} aa$
- $\sigma_3 = aaaaab \xrightarrow{3,(aa,a)} aaaab \xrightarrow{3,(aa,a)} aaab \xrightarrow{1,(aa,a)} aab \xrightarrow{2,(ab,a)} aa$

On a $\sigma_2 = (\text{exch}_{1,2} \circ \text{exch}_{2,3} \circ \text{exch}_{3,4})(\sigma_1)$ donc $\sigma_1 \sim \sigma_2$.
De plus les graphes de σ_1 et de σ_3 ne sont pas isomorphes, donc $\sigma_1 \nsim \sigma_3$.



Lemme 133

Soit R un système de réécriture. Soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation orthogonale en i. Soit $\varphi_{i,i+1}$ l'isomorphisme de graphe qui échange les nœuds i et $i+1$. On a :

$$G(\text{exch}_{i,i+1}(\sigma)) = \varphi_{i,i+1}(G(\sigma))$$

Démonstration. On pose $\sigma' = \text{exch}_{i,i+1}(\sigma)$.
Puisque σ est orthogonale en i l'arête $(i, i+1)$ n'est pas dans $G(\sigma)$.
Par définition, σ' est égale à :

$$x_0 \xrightarrow{p_0,(u_0,v_0)} \cdots x_{i-1} \xrightarrow{p'_{i-1},(u_i,v_i)} x'_i \xrightarrow{p'_i,(u_{i-1},v_{i-1})} x_{i+1} \xrightarrow{p_{i+1},(u_{i+1},v_{i+1})} x_{i+2} \cdots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$$

avec
$p'_{i-1} = p_i + |u_{i-1}| - |v_{i-1}|, p'_i = p_{i-1}$, quand $p_i > p_{i-1}$,
$p'_{i-1} = p_i, p'_i = p_{i-1} + |v_i| - |u_i|$ quand $p_i < p_{i-1}$,
On remarque que $p_i \neq p_{i-1}$ car :

$$p_{i-1} \in \text{Right}_\sigma(i),\ p_i \in \text{Left}_\sigma(i+1) \text{ et } \text{Right}_\sigma(i) \cap \text{Left}_\sigma(i+1) = \emptyset.$$

Nous supposons sans perte de généralité que $p_i < p_{i-1}$. L'autre cas ($p_i > p_{i-1}$) étant obtenu par symétrie (par exemple, en considérant les miroirs des mots).
Ainsi nous avons :

$$\mathrm{Corr}_{\sigma'}(i-1,i) = \{(p,p), p < p'_{i-1}\} \cup \{(p, p+|v_i|-|u_i|, p \geq p'_{i-1}+|u_i|\} = \{(p,p), p < p_i\} \cup \{(p, p+|v_i|-|u_i|, p \geq p_i+|u_i|\}$$

et

$$\mathrm{Corr}_{\sigma'}(i,i+1) = \{(p,p), p < p'_i\} \cup \{(p, p+|v_{i-1}|-|u_{i-1}|), p \geq p'_i+|u_{i-1}|\} = \{(p,p), p < p_{i-1}+|v_i|-|u_i|\} \cup \{(p, p+|v_{i-1}|-|u_{i-1}|), p \geq p_{i-1}+|v_i|-|u_i|+|u_{i-1}|\}$$

d'où $\mathrm{Corr}_{\sigma'}(i-1,i+1) = \{(p,p), p < p_i\} \cup \{(p, p+|v_i|-|u_i|, p_{i-1} > p \geq p_i+|u_i|\}$
$\cup \{(p, p+|v_{i-1}|-|u_{i-1}|+|v_i|-|u_i|, p \geq p_{i-1}+|u_{i-1}|\}$.
donc $\mathrm{Corr}_{\sigma'}(i-1,i+1) = \mathrm{Corr}_{\sigma}(i-1,i+1)$.

Ainsi :

— $\mathrm{Left}_\sigma(k) = \mathrm{Left}_{\sigma'}(k)$ et $\mathrm{Right}_\sigma(k) = \mathrm{Right}_{\sigma'}(k)$ si $k < i$ ou $k > i+1$. Les k ème pas des dérivations σ et σ' coïncident si $k < i$ ou $k > i+1$;

— $\mathrm{Left}_{\sigma'}(i) = \mathrm{Left}_\sigma(i+1)$ et $\mathrm{Left}_{\sigma'}(i+1) = \{p+|v_i|-|u_i|, p \in \mathrm{Left}_\sigma(i)\}$. $\mathrm{Right}_{\sigma'}(i) = \mathrm{Right}_\sigma(i+1)$, $\mathrm{Right}_{\sigma'}(i+1) = \{p+|v_i|-|u_i|, p \in \mathrm{Right}_\sigma(i)\}$.

— $\mathrm{Corr}_\sigma(k,l) = \mathrm{Corr}_{\sigma'}(k,l)$, si $l < i$ ou $k > i+1$ ou $k < i < l$. C'est évident si $l < i$ or $k > i+1$ étant donné que les "portions" $[l,k]$ des dérivations σ and σ' sont égales. Si $k < i < l$, c'est vrai car $\mathrm{Corr}_\sigma(k,l) = \mathrm{Corr}_\sigma(k,i-1) \circ \mathrm{Corr}_\sigma(i-1,i+1) \circ \mathrm{Corr}_\sigma(i+1,l)$ (resp. $\mathrm{Corr}_{\sigma'}(k,l) = \mathrm{Corr}_{\sigma'}(k,i-1) \circ \mathrm{Corr}_{\sigma'}(i-1,i+1) \circ \mathrm{Corr}_{\sigma'}(i+1,l)$) et $\mathrm{Corr}_{\sigma'}(i-1,i+1) = \mathrm{Corr}_\sigma(i-1,i+1)$.

Prouvons maintenant que $G(\sigma') = \varphi_{i,i+1}(G(\sigma))$.

Si (k,l) est une arête de $G(\sigma)$, $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l))$ est une arrête $G(\sigma')$.

Soit (k,l) une arête de $G(\sigma)$; par définition, il existe $(q,q') \in \mathrm{Corr}_\sigma(k,l-1)$ tel que $q \in \mathrm{Right}_\sigma(k)$ et $q' \in \mathrm{Left}_\sigma(l)$.

Si k et l sont tous les deux différents de i et $i+1$, $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l)) = (k,l)$; d'après ce qui précède, $(q,q') \in \mathrm{Corr}_{\sigma'}(k,l-1)$, $q \in \mathrm{Right}_{\sigma'}(k)$ and $q' \in \mathrm{Left}_{\sigma'}(l)$ donc $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l))$ est une arête de $G(\sigma')$.

Nous pouvons considérer maintenant toutes les possibilités qui restent :

— $k < l = i$: $(q,q') \in \mathrm{Corr}_\sigma(k,i-1), q \in \mathrm{Right}_\sigma(k), q' \in \mathrm{Left}_\sigma(i)$. Donc $q" = q' + |v_i|-|u_i|$ est dans $\mathrm{Left}_{\sigma'}(i+1)$. D'après ce qui précède on a : $(q,q') \in \mathrm{Corr}_{\sigma'}(k,i-1)$, $(q',q'') \in \mathrm{Corr}_{\sigma'}(i-1,i)$. Donc, $q \in \mathrm{Right}_{\sigma'}(k), (q,q") \in corr_{\sigma'}(k,i), q" \in \mathrm{Left}_{\sigma'}(i+1)$: $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l)) = (k,i+1)$ est une arête de $G(\sigma')$.

— $k < i, l = i+1$: $(q,q') \in \mathrm{Corr}_\sigma(k,i), q \in \mathrm{Right}_\sigma(k), q' \in \mathrm{Left}_\sigma(i+1)$. Donc, $q" = q' \in \mathrm{Left}_{\sigma'}(i)$ et $(q,q")$ appartient $\mathrm{Corr}_{\sigma'}(k,i-1)$. Ainsi, $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l)) = (k,i)$ est une arête de $G(\sigma')$.

— $(i,i+1)$: cas impossible car σ est orthogonale en i.

— $k = i, l > i+1$: alors, $q' \in \mathrm{Left}_\sigma(l)$ donc $q' \in \mathrm{Left}_\sigma(l)$ puisque $l > i+1$. $q" = q + |v_i|-|u_i| \in \mathrm{Right}_{\sigma'}(i+1)$ et $(q",q')$ appartient à $\mathrm{Corr}_{\sigma'}(i+1,l-1)$. Ainsi, $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l)) = (i+1,l)$ est une arête de $G(\sigma')$.

— $k = i+1 < l : (q, q') \in \mathrm{Corr}_\sigma(i+1, l-1), q \in \mathrm{Right}_\sigma(i+1), q' \in \mathrm{Left}_\sigma(l)$; donc, $q' \in \mathrm{Left}_\sigma(l), q \in \mathrm{Right}_{\sigma'}(i), (q", q') \in \mathrm{Corr}_{\sigma'}(i, l-1)$. Ainsi, $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l)) = (i, l)$ est une arête de $G(\sigma')$.

Ainsi, si (k, l) est une arête de $G(\sigma)$ alors $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l))$ est une arête de $G(\sigma')$. Réciproquement comme $\varphi_{i,i+1}$ est involutive, si (k, l) est une arête de $G(\sigma')$ alors $(\varphi_{i,i+1}(k), \varphi_{i,i+1}(l))$ est une arête de $G(\sigma)$.

□

Deux dérivations équivalentes ont des graphes isomorphes. La réciproque est fausse.

Exemple 134

On considère le système de réécriture $R = \{(aa, a)\}$ et les deux dérivations :

$$aaa \xrightarrow{1,(aa,a)} aa$$
$$aaa \xrightarrow{2,(aa,a)} aa$$

Ces deux dérivations ont le même graphe (une unique arête entre 0 et 1) et les deux dérivations ne sont pas équivalentes.

Proposition 135

Soit R un système de réécriture, soit σ une dérivation. Pour tout tri topologique T de $G(\sigma)$ il existe une dérivation σ' équivalente à σ telle que $G(\sigma') = T(G(\sigma))$.

Démonstration. On rappelle qu'un tri topologique est une permutation T des nœuds telle que si $(T(i), T(j))$ une arête de $G(\sigma)$ alors $i < j$. On montre le résultat par récurrence sur le nombre d'inversions de T : s'il n'y a aucune inversion la dérivation σ convient.

Sinon, supposons que la propriété est vraie lorsque nous avons k inversions. Soit T un tri topologique avec $k+1$ inversions.

T contient au moins une inversion, donc il existe i telle que $T(i) > T(i+1)$. Soit $T' = \mathrm{exch}_{i,i+1} \circ T$, c'est un tri topologique et son nombre d'inversions est k, par hypothèse de récurrence il existe une dérivation σ'' équivalente à σ telle que $G(\sigma'') = T'(G(\sigma))$. Comme l'arête $(i+1, i)$ n'est pas une arête de $T(G(\sigma))$ on a $(i, i+1)$ qui n'est pas une arête de $T'(G(\sigma))$ donc la dérivation $\sigma' = exch_{i,i+1}(\sigma'')$ est équivalente à σ par transitivité et vérifie $G(\sigma') = T(G(\sigma))$ car $\mathrm{exch}_{i,i+1}$ est une involution. □

La possibilité de pouvoir échanger deux nœuds indépendants étant établie, nous allons mettre en avant trois étapes qui permettent de montrer que la complexité parallèle d'une dérivation est égale à la profondeur du graphe de dérivation :

1. Si des étapes consécutives sont indépendantes alors on peut les exécuter en une unique étape parallèle (Théorème 136)
2. Il existe un tri topologique T tel que si $\text{depth}_{G(\sigma)}(i) < \text{depth}_{G(\sigma)}(j)$ alors $T(i) < T(j)$ donc il existe une dérivation parallèle de longueur égale à la profondeur de $G(\sigma)$ (Corollaire 137)
3. On en déduit que la profondeur du graphe de dérivation est égale à la complexité parallèle de la dérivation (Corollaire 138)

Théorème 136

Soit R un système de réécriture et une dérivation sur R $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$. Si $\{i+1, i+2, ..., j\}$ est un sous-ensemble indépendant de $G(\sigma)$ alors $x_i \multimap x_j$ et c'est une dérivation équivalente à la sous-dérivation $x_i \rightarrow^* x_j$ de σ.

Démonstration. Soit σ' la dérivation extraite depuis σ entre les indices i et j :

$$x_i \overset{p_i,(u_i,v_i)}{\longrightarrow}_1 x_{i+1} ... \overset{p_{j-1},(u_{j-1},v_{j-1})}{\longrightarrow}_n x_j$$

On remarque que $\text{Left}_\sigma(t) = \text{Left}_{\sigma'}(t-i)$.

De même $\text{Right}_\sigma(t) = \text{Right}_{\sigma'}(t-i)$ et $\text{Corr}_{\sigma'}(x,y) = \text{Corr}_\sigma(x+i, y+i)$.

Par hypothèse sur σ, chaque nœud (sauf 0) de $G(\sigma')$ a une profondeur égale à 1.

En triant σ' topologiquement les nœuds (avec renumérotation) par le tri induit par l'ordre $\leq$ sur les positions, on obtient une dérivation s équivalente à σ' (lemme 135) :

$$s = X_0 \overset{P_0,(U_0,V_0)}{\longrightarrow}_1 X_1 ... \overset{P_{n-1},(U_{N-1},V_{N-1})}{\longrightarrow}_N X_N$$

Avec $N = j-i$, pour chaque nœud t (sauf 0) la profondeur de t dans $G(s)$ est égale à 1 et $P_0 < P_1 < ... < P_{N-1}$.

Si $N = 1$ le résultat est clair.

Sinon soit $t > 0$ et H_t la proposition "il existe $\mu_0, ..., \mu_t$ tels que :

— $X_0 = \mu_0 U_0 \mu_1 ... \mu_{t-1} U_{t-1} \mu_t$

— $X_t = \mu_0 V_0 \mu_1 ... \mu_{t-1} V_{t-1} \mu_t$

— $P_{t-1} = |\mu_0 V_0 \mu_1 ... V_{t-2} \mu_{t-1}| + 1$"

Par définition de $X_0 \xrightarrow{P_0,(U_0,V_0)}_1 X_1$, H_1 est facile. Supposons que H_t est vrai pour un entier $t > 0$ fixé. Par hypothèse sur s, $P_t > P_{t-1}$. Si $P_t < P_{t-1} + |V_{t-1}|$ alors $P_t \in \text{Right}_s(t)$ comme $P_t \in \text{Left}_s(t+1)$ il existe alors une arête $(t, t+1)$, contradiction. Comme $P_t \geq P_{t-1} + |V_{t-1}| = |\mu_0 V_0 \mu_1 ... V_{t-1} \mu_{t-1} V_{t-1}| + 1$ et $X_t \xrightarrow{P_t,(U_t,V_t)}_{t+1} X_{t+1}$ il existe un μ'_t et un μ'_{t+1} tels que :

— $X_0 = \mu_0 U_0 \mu_1 ... \mu_{t-1} U_{t-1} \mu'_t U_t \mu'_{t+1}$

— $X_t = \mu_0 V_0 \mu_1 ... \mu_{t-1} V_{t-1} \mu'_t V_t \mu'_{t+1}$

— $P_t = |\mu_0 V_0 \mu_1 ... V_{t-2} \mu_{t-1} V_{t-1} \mu'_t| + 1$

Dons, H_{t+1} est vrai. H_N implique que $X_0 \multimap X_N$ comme $X_0 = \text{orig}(s) = x_i$ et $X_N = \text{aim}(s) = x_j$ on a $x_i \multimap x_j$.

□

Corollaire 137

Soit R un système de réécriture. Pour toute dérivation σ il existe une dérivation parallèle σ' équivalente de longueur égale à la profondeur de $G(\sigma)$.

Finalement on montre que :

Corollaire 138

La complexité parallèle d'une dérivation est la profondeur de son graphe de dérivation.

Démonstration. Soit σ une dérivation de complexité parallèle k, soit σ' une dérivation parallèle équivalente de longueur égale à la profondeur de $G(\sigma)$ (corollaire 137).

Comme la dérivation σ' est de longueur $\text{depth}(G(\sigma))$, par minimalité on a $k \leq \text{depth}(G(\sigma))$.

Réciproquement, s'il existe une dérivation parallèle σ'' de longueur k équivalente à σ, les graphes de dérivation de σ et σ'' sont isomorphes, donc de même profondeur. Or le graphe d'une dérivation est de profondeur inférieure ou égale à sa longueur (proposition 131), ainsi la complexité parallèle de sigma est $\leq k$. □

Dérivations indicées Une définition syntaxique consiste à donner une étiquette à chaque lettre $x_i[t]$ dans une dérivation σ notée $\text{depth}_\sigma(i,t)$ et appelée profondeur. Cette profondeur correspond à la profondeur de l'indice du pas de réécriture qui a produit la règle dans le pas de réécriture.

Au départ, les lettres sont toutes étiquetées par 0 et si une lettre est issue de l'application d'une règle (u_i, v_i) alors elle reçoit comme étiquette 1 plus le maximum des étiquettes des lettres de u_i. Formellement :

Définition 139

Soit R un système de réécriture. Pour toute dérivation

$$\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$$

on définit une fonction depth_σ qui à (i,t) avec i entre 0 et n et t entre 1 et $|x_i|$ associe :

- si $i = 0$ alors $\text{depth}_\sigma(i,t) = 0$
- si $t < p_{i-1}$ alors $\text{depth}_\sigma(i,t) = \text{depth}_\sigma(i-1,t)$
- si $t \geq p_{i-1} + |v_{i-1}|$ alors $\text{depth}_\sigma(i,t) = \text{depth}_\sigma(i-1, t + |u_{i-1}| - |v_{i-1}|)$
- sinon $\text{depth}_\sigma(i,t) = 1 + \max_{\tau \in \text{Left}_\sigma(i)} \text{depth}_\sigma(i-1,\tau)$

À toute position p du mot x_i, on associe $\text{index}_\sigma(i,p)$ le plus petit entier j tel qu'il existe p' tel que $(p',p) \in \text{Corr}_\sigma(j,i)$. Autrement dit, $\text{index}_\sigma(i,p)$ est l'indice du pas de dérivation qui a produit la lettre $x_i[p]$.

Par minimalité on a $p' \in \text{Right}_\sigma(\text{index}_\sigma(i,p))$. De même $p \in \text{Right}_\sigma(i)$ ssi $\text{index}_\sigma(i,p) = i$. Enfin, on remarquera que si $p \in \text{Left}_\sigma(i+1)$ alors $(\text{index}_\sigma(i,p), i+1)$ est une arête de $G(\sigma)$. Inversement pour toute arête $(j, i+1) \in G(\sigma)$, il existe $p \in \text{Left}_\sigma(i+1)$ tel que $j = \text{index}_\sigma(i,p)$.

Proposition 140

Soit R un système de réécriture, soit $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ une dérivation.
On a pour tout (i,p) avec $i \leq n$ et $p \leq |x_i|$:

$$\text{depth}_\sigma(i,p) = \text{depth}_{G(\sigma)}(\text{index}_\sigma(i,p))$$

Démonstration. Montrons le résultat par récurrence sur i.

Initialisation : si $i = 0$, pour tout p de x_0, le plus petit entier n tel que $\text{Corr}_\sigma(n,i)$ contienne un élément de la forme (p',p) est 0. On a bien pour toute position de x_0 l'égalité $\text{depth}_\sigma(i,p) = 0 = \text{depth}_{G(\sigma)}(0) = \text{depth}_{G(\sigma)}(\text{index}_\sigma(i,p))$.

Hérédité : supposons la propriété vraie au rang i.

Soit p une position de x_{i+1} :

- Si $p \notin \text{Right}_\sigma(i+1)$ alors $\text{index}_\sigma(i+1,p) < i+1$, d'où le résultat par hypothèse de récurrence.

- Si $p \in \text{Right}_\sigma(i+1)$ alors $\text{index}_\sigma(i+1,p) = i+1$.

On a $\text{depth}_\sigma(i+1,p) = 1 + \max_{p' \in \text{Left}_\sigma(i+1)} \text{depth}_\sigma(i,p)$

Pour tout p' de $\text{Left}_\sigma(i+1)$ on pose $j(p') = \text{index}_\sigma(i,p')$, on a $j(p')$ qui vérifie $(j(p'), i+1) \in G(\sigma)$.

Par hypothèse de récurrence on a :

$\text{depth}_\sigma(i+1,p)$
$= 1 + \max_{p' \in \text{Left}_\sigma} \text{depth}_{G(\sigma)}(j(p'))$
$\leq 1 + \max_{j,(j,i+1) \in G(\sigma)} \text{depth}_{G(\sigma}(j)$
$= \text{depth}_{G(\sigma)}(i+1)$.

Inversement, pour tout j tel que $(j, i+1) \in G(\sigma)$, il existe $p'(j) \in \text{Left}_\sigma(i+1)$ tel que $j = \text{index}_\sigma(i, p'(j)) < i+1$.

Par hypothèse de récurrence on a :

$\text{depth}_{G(\sigma)}(i+1)$
$= 1 + \max_{j,(j,i+1) \in G(\sigma)} \text{depth}_{G(\sigma)}(j)$
$= 1 + \max_{j,(j,i+1) \in G(\sigma)} \text{depth}_{G(\sigma)}(\text{index}_\sigma(i, p'(j)))$
$= 1 + \max_{j,(j,i+1) \in G(\sigma)} \text{depth}_\sigma(i, p'(j))$
$\leq 1 + \max_{p' \in \text{Left}_\sigma(i+1)} \text{depth}_\sigma(i, p')$
$= \text{depth}_\sigma(i+1, p)$

Ainsi : $\text{depth}_\sigma(i+1,p) = \text{depth}_{G(\sigma)}(\text{index}_\sigma(i+1,p))$

□

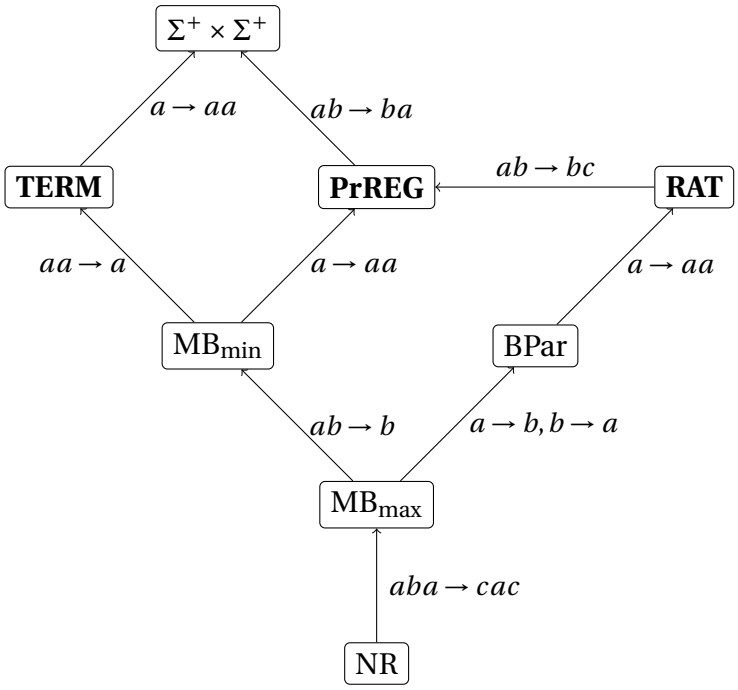
4.2 Systèmes à complexité parallèle bornée

La notion de complexité peut être étendue aux systèmes de réécriture. On s'intéresse aux systèmes dont la complexité parallèle est uniformément bornée par une constante i.e. toutes les dérivations ont une complexité parallèle $\leq k$: ce sont les systèmes $\mathrm{MB}_{\max}(k)$.

On peut relâcher cette condition très forte, en imposant que pour toute dérivation, on puisse trouver une dérivation parallèle de longueur au plus k, avec la même origine et le même but : ce sont les systèmes $\mathrm{BPar}(k)$.

Nous comparerons les classes obtenues avec d'autres classes dont voici un schéma récapitulatif (chaque flèche désigne un exemple qui appartient au but mais pas à l'origine de celle ci).

Nous comparons les classes obtenues avec d'autres classes de système de réécriture :



4.2.1 Classe $\mathrm{MB}_{\max}(k)$

Nous introduisons la notion générale de systèmes à complexité parallèle borné :

Définition 141

Soit k un entier. Soit L un langage. Un système de réécriture est $\mathrm{MB}_{\max}(k, L)$ si pour toute dérivation σ telle que $\mathrm{orig}(\sigma) \in L$ la complexité parallèle de σ est inférieure ou égale à k.

On ne précise pas L si $L = \Sigma^*$: on note $\mathrm{MB}_{\max}(L)$ l'union de tous les $\mathrm{MB}_{\max}(k, L)$.

La classe $\mathrm{MB}_{\max}(k)$ est similaire à la classe $\mathrm{MB}_{\min}(k)$ des match-bounded [49] : en effet dans les deux cas on peut annoter les lettres par un entier (def. 139). Si un facteur est réécrit en un autre, les lettres produites sont annotés de $1 + \max$ des annotations des lettres utilisés dans le cas $\mathrm{MB}_{\max}$ et de $1 + \min$ dans le cas de $\mathrm{MB}_{\min}$. On a :

$$\mathrm{MB}_{\max}(k) \subseteq \mathrm{MB}_{\min}(k)$$

On compare $\mathrm{MB}_{\max}$ avec la classe NR :

Proposition 142

$$\mathrm{NR} \subseteq \mathrm{MB}_{\max}$$

Démonstration. Soit R un système de réécriture NR, soit G son graphe-lettre, il est acyclique donc admet un chemin de longueur maximale M.

Montrons par récurrence que s'il existe une lettre de profondeur k dans une dérivation σ alors il existe un chemin de longueur k dans G, plus formellement, soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation sur R :

S'il existe (i, p) tel que $\mathrm{depth}_\sigma(i, p) = k$, on peut choisir i minimal, alors il existe un chemin de longueur k dans G dont le dernier nœud est $x_i[p]$.

- **Initialisation :** elle est évidente, si $k = 0$, le chemin constitué avec juste la lettre $x_i[p]$ convient (on remarquera d'ailleurs qu'on a $i = 0$ dans ce cas)

- **Hérédité :** Soit (i, p) tel que $\mathrm{depth}_\sigma(i, p) = k + 1$ avec i minimal. Comme i est minimal on a forcément p dans $\mathrm{Right}_\sigma(i)$ sinon il existe p' tel que $\mathrm{depth}_\sigma(i - 1, p') = k + 1$, contradiction avec la minimalité de i. Dans ce cas $\mathrm{depth}_\sigma(i, p) = 1 + \max_{\tau \in [p_{i-1}, p_{i-1}+|u_{i-1}|-1]} \mathrm{depth}_\sigma(i-1, \tau)$, donc il existe p' dans $\mathrm{Left}_\sigma(i)$ tel que $\mathrm{depth}_\sigma(i-1, p') = k$. Par hypothèse de récurrence il existe un chemin de longueur k terminant

par $x_{i-1}[p']$ dans G. Comme $p \in \text{Right}_\sigma(i)$ et $p' \in \text{Left}_\sigma(i)$ il existe une arête entre les lettres $x_{i-1}[p']$ et $x_i[p]$ dans G, donc un chemin de longueur $k+1$ dans G terminant par $x_i[p]$.

Ainsi, comme le graphe est acyclique, les profondeurs des lettres $\text{depth}_\sigma(i,p)$ sont bornées par M pour tout σ et (i,p), donc le système est dans $\text{MB}_{\max}(M)$. □

L'inclusion est stricte :

Exemple 143

Soit $R = \{aba \longrightarrow cac\}$ alors R n'est pas dans NR et $R \in \text{MB}_{\max}(1)$.

Comme NR, la classe $\text{MB}_{\max}(k)$ est stable par inversion :

Proposition 144

Soit R un système de réécriture, si $R \in \text{MB}_{\max}(k)$ alors $R^{-1} \in \text{MB}_{\max}(k)$

Démonstration. En effet, par définition et par la proposition 125, la complexité parallèle d'une dérivation σ sur R est égale à celle de la dérivation σ^{-1} sur R^{-1}. □

Cas hors-contexte Dans le cas des systèmes hors-contexte[3] les max et min se confondent dans les étiquettes :

Proposition 145

$$\text{CF} \cap \text{NR} = \text{CF} \cap \text{MB}_{\max} = \text{CF} \cap \text{MB}_{\min}$$

Démonstration. On a $R \in \text{CF}$ donc les règles (u,v) de R vérifient $|u| = 1$, on a $1 + \max_{1 \le p \le |u|} f(p) = 1 + \min_{1 \le p \le |u|} f(p)$ pour toute fonction p.

Donc $\text{CF} \cap \text{MB}_{\min} = \text{CF} \cap \text{MB}_{\max}$.

Nous avons aussi montré que $\text{NR} \subseteq \text{MB}_{\max}$ donc en particulier $\text{CF} \cap \text{NR} \subseteq \text{MB}_{\max}$, il ne reste que l'autre inclusion.

Soit R un système de réécriture hors-contexte qui n'appartient pas à NR , il existe donc un cycle dans le graphe des lettres :

3. les règles sont de la forme (a,u) où a est une lettre

il existe un entier n et (a_i, l_i, r_i) tel que $(a_{i-1}, l_i a_i r_i)$ est une règle de R pour tout $1 \leq i \leq n$ avec $a_0 = a_n$, on pose $a = a_0$

Donc $s = a \longrightarrow_R^* l_1 ... l_i a_i r_i ... r_1 \longrightarrow_R^* l_1 ... l_n a r_n ... r_1$ est une dérivation dont le graphe contient un chemin de longueur n. Donc pour tout k, $a \longrightarrow_R^* (l_1 ... l_n)^k a (r_n ... r_1^k$ est une dérivation de complexité parallèle $\geq n \times k$, donc R n'est pas dans $\mathrm{MB}_{\max}$.

□

Ainsi, comme on peut décider si un système est NR en temps linéaire on peut décider si un système hors-contexte appartient à $\mathrm{MB}_{\max}$ ou à $\mathrm{MB}_{\min}$ en temps linéaire.

Tout système dans NR est un système de $\mathrm{MB}_{\max}$ dont on peut calculer la complexité parallèle.

Puisque la complétion de l'automate (def.98) correspond aux ancêtres à une étape parallèle on a :

Proposition 146

Si R est un système dans $\mathrm{MB}_{\max}(k)$ alors pour tout automate signé A :

$$T_R^{k+1}(A) = T_R^k(A)$$

Démonstration. Comme pour NR c'est une conséquence directe de la proposition 159 et de la proposition 144. □

Puisque tout système de $\mathrm{MB}_{\max}$ est terminant, il est en particulier descendant-limité.

Le langage reconnu par $T_R^k(A)$ (qui est de taille exponentielle) est $\mathrm{Anc}_R(L(A))$ et l'inclusion de deux langages réguliers est PSPACE donc l'inclusion de deux requêtes RPQ sous contraintes est EXPSPACE.

De plus si R est un système NR alors la longueur du plus long chemin du graphe-lettre est $\leq |R|$ donc $R \in \mathrm{MB}_{\max}(|R|)$: il existe une réduction du problème 114 dans le problème d'inclusion de requêtes RPQ sous contraintes. Ainsi :

Théorème 147

Soit $\mathscr{C}$ un ensemble de contraintes de mots tel que le système de réécriture R est $\mathrm{MB}_{\max}(k)$.

PROBLEME : Inclusion de RPQ sous contraintes de mots à complexité parallèle bornée
INSTANCE : $Q = xLy$ et $Q' = xL'y$ deux RPQ, k un entier et $\mathscr{C}$ des contraintes de mots dont le système associé est dans $\mathrm{MB}_{\max}(k)$
QUESTION : $Q \sqsubseteq_{\mathscr{C}} Q'$?
COMPLEXITE : EXPSPACE-complet

Si k n'est pas connu, nous avons :

Théorème 148

Soit $\mathscr{C}$ un ensemble de contraintes de mots tel que le système de réécriture R est $\mathrm{MB}_{\max}$.

PROBLEME : Inclusion de RPQ sous contraintes de mots à complexité parallèle bornée
INSTANCE : $Q = xLy$ et $Q' = xL'y$ deux RPQ et $\mathscr{C}$ des contraintes de mots dont le système associé est dans $\mathrm{MB}_{\max}$
QUESTION : $Q \sqsubseteq_{\mathscr{C}} Q'$?
COMPLEXITE : décidable

Nous ne savons pas si k est calculable en fonction de R.

4.2.2 Classe BPar(k)

Si un système est dans $\mathrm{MB}_{\max}(k)$ alors pour toute dérivation il existe une dérivation parallèle de longueur inférieure ou égale à k qui lui est équivalente ainsi :

$$\longrightarrow^* = \multimap^{\leq k}$$

.

Cependant la réciproque est fausse :

Exemple 149

Le système $R = \{a \longrightarrow b, b \longrightarrow a\}$ vérifie $\longrightarrow^* = \multimap$ mais n'est pas terminant donc $\notin \mathrm{MB}_{\max}$.

Nous nous intéressons aux systèmes qui satisfont l'équation $\longrightarrow^* = \multimap^k$: c'est une classe qui élargit naturellement $\mathrm{MB}_{\max}$.

Définition 150

Un système de réécriture est dit parallèlement borné par k si

$$\longrightarrow^* = \multimap^{\leq k}$$

On note $\mathrm{BPar}(k)$ la classe des systèmes de réécritures parallèlement bornés par k, de même on note BPar l'union des $\mathrm{BPar}(k)$. Un système de réécriture est dans $\mathrm{BPar}(k)$ si pour toute dérivation il existe une dérivation de complexité parallèle inférieure ou égale à k de même origine et même but.

Comme pour $\mathrm{MB}_{\max}$, on montre facilement que cette classe est stable par inversion :

Proposition 151

Soit R un système de réécriture,

$$R \in \mathrm{BPar} \text{ si et seulement si } R^{-1} \in \mathrm{BPar}$$

BPar préserve la régularité, nous avons d'ailleurs la propriété plus forte suivante (voir théorème 30) :

Proposition 152

Si R est un système de réécriture parallèlement borné alors $\longrightarrow^*$ est une relation rationnelle linéairement bornée [a].

a. il existe une constante K telle que si $x \longrightarrow^* y$ alors $|y| \leq K \times |x|$

Démonstration. Si R est parallèlement borné alors il existe k tel que $\longrightarrow^* = \multimap^{\leq k}$. Or la relation $\multimap$ est égale à $(R \cup \Sigma \times \Sigma)^*$. Comme les relations rationnelles sont stables par union, étoile de Kleene et composition (def. 14 et proposition 15), on en déduit que $\longrightarrow^*$ est rationnelle.

De plus si $u \longrightarrow^* v$ alors $|v| \leq M^k |u|$ avec $M = \max_{(l,r)\in R}(1, \frac{|r|}{|l|})$. □

BPar contient des systèmes non terminant mais inversement le système $ab \to b$ est $\mathrm{MB}_{\min}$ mais n'est pas parallèlement borné. Ces classes sont incomparable.

Théorème 153

Dans le cadre des systèmes hors-contextes nous obtenons :

PROBLEME : Problème d'appartenance à BPar dans le cas hors-contexte
INSTANCE : R un système de réécriture hors-contexte
QUESTION : $R \in$ BPar ?
COMPLEXITE : PTIME

Démonstration. Soit R un système de réécriture hors-contexte et G_R son graphe-lettre associé.

On étiquette les arêtes de G_R par les règles, i.e. si (a,b) est une arête de G_R on associe l'ensemble des règles (u,v) de R telles que a est une lettre de u et b une lettre de v.

On a facilement que si (a,a') est une arête du graphe-lettre alors tous les éléments de l'étiquettes sont de la forme (a,m), pour chaque étiquette donnée il existe un pas de réécriture de la forme $a \to m$, donc par une récurrence facile, pour tout chemin du graphe-lettre entre deux lettres a et b, il existe une dérivation dont a est l'origine et dont le but est de la forme ubv.

On dit qu'un chemin de G_R est expansif s'il existe une arête dont l'étiquette contient un élement de la forme (u,v) avec $|v| > 1$. Si un chemin de a à b est expansif alors il existe une dérivation de la forme $a \to^* ubv$ avec $|uv| > 0$.

Montrons que R est dans BPar si et seulement si G_R ne contient pas de cycle expansif.

- Si G_R contient un cycle expansif, il existe donc une arête dont l'étiquette contient une règle de la forme (b,m) avec $|m| > 1$, b est alors l'origine de l'arête.

Donc il existe une dérivation de la forme $b \to^* ubv$ avec $|uv| > 0$. Donc il existe une dérivation $b \to^* u^n b v^n$ pour tout n, donc R n'est pas linéaire borné, donc R n'est pas BPar.

- Inversement, supposons que G_R ne contienne pas de cycle expansif.
On considère une dérivation de complexité parallèle $> \#\Sigma$:

$$\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$$

D'après la preuve de la proposition 142, il existe un chemin de longueur $> \#\Sigma$ dans G_R dont les lettres apparaissent dans la dérivation.

Comme le nombre de nœuds de G_R est égal à $\#\Sigma$, il existe deux nœuds égaux (i.e. deux lettres) qui appartiennent au chemin. Il existe donc un cycle $a_1, ..., a_f$ inclus dans ce chemin. On a $a_f = a_1$.

Comme R est hors-contexte, il existe des entiers $i_1 < ... < i_f$ tels que $x_{i_j}[p_{i_j}] = a_j$ pour tout j de 1 à f. Pour tout j on a (u_{i_j}, v_{i_j}) qui vérifie $|v_{i_j}| = 1$ sinon le cycle serait expansif par définition.

Puisque $x_{i_f}[p_{i_f}] = x_{i_1}[p_{i_1}]$ on peut retirer les pas correspondants aux indices $i_1, ..., i_p$ de σ pour obtenir σ'. D'après la preuve de la proposition 142, la complexité parallèle de σ' vaut celle de σ moins f. Tant que la complexité est supérieure strictement à $\#\Sigma$ il existe une dérivation de complexité strictement inférieur de même origine et but, donc il existe une dérivation (de même origine et même but) de complexité parallèle $\leq \#\Sigma$, ainsi le système est dans BPar($\#\Sigma$).

□

La classe BPar permet de décider l'inclusion de requêtes RPQ sous contraintes de mots :

Proposition 154

Tout système de réécriture BPar est descendant-limité.

Si un système de réécriture est BPar(k) alors $L(T_R^k(A)) = L(T_R^{i+k}(A))$ (ce sont les ancêtres de $L(A)$). Ainsi :

Théorème 155

Soit $\mathscr{C}$ un ensemble de contraintes de mots tel que le système de réécriture R est $\mathrm{MB}_{\max}(k)$.

PROBLEME : Inclusion de RPQ sous contraintes de mots parallèlement bornées
INSTANCE : $Q = xLy$ et $Q' = xL'y$ deux RPQ, k un entier et $\mathscr{C}$ des contraintes de mots dont le système associé est dans $\mathrm{BPar}(k)$
QUESTION : $Q \sqsubseteq_{\mathscr{C}} Q'$?
COMPLEXITE : EXPSPACE-complet

Soit R un système BPar, nous ne savons pas comment calculer une constante k telle que $R \in \mathrm{BPar}(k)$.

Nous pouvons calculer successivement les langages $L(T_R^i(A))$, jusqu'à ce qu'il existe i tel que $L(T_R^i(A)) = L(T_R^{i+1}(A))$ (puisque $R \in$ BPar, un tel entier existe), le langage obtenu est alors celui des ancêtres. Puisque R est descendant-limité :

Théorème 156

Nous avons :

PROBLEME : Inclusion de RPQ sous contraintes de mots à complexité parallèle
INSTANCE : $Q = xLy$ et $Q' = xL'y$ deux RPQ et $\mathscr{C}$ des contraintes de mots dont le système associé est dans BPar
QUESTION : $Q \sqsubseteq_{\mathscr{C}} Q'$?
COMPLEXITE : décidable

4.3 Décidabilité du problème $R \in \mathrm{MB}_{\max}(k)$?

Le problème de décision $R \in \mathrm{MB}_{\max}$? ou pour un entier k, $R \in \mathrm{MB}_{\max}(k)$? est une question naturelle.

Dans le cas général, pour un entier k donné, on démontre dans cette section qu'on peut décider si un système appartient à $\mathrm{MB}_{\max}(k)$.

Théorème 157

On a :

PROBLEME : Complexité parallèle uniformément bornée par une constante
INSTANCE : R un srs, k un naturel
QUESTION : $R \in \mathrm{MB}_{\max}(k)$?
COMPLEXITE : PSPACE-complet

4.3.1 Premiers algorithmes

Nous présentons deux algorithmes qui étant donné (R,k) décide si $R \in \mathrm{MB}_{\max}(k)$. Le premier est EXPTIME et le second PSPACE.

Avec un automate Une première méthode consiste à détecter si une dérivation de complexité parallèle égale à $k+1$ existe : on a étudié au chapitre 1 et dans la section précédente la complétion d'un automate A (définition 98) qui reconnaît l'ensemble des ancêtres à k étapes parallèles de $L(A)$ i.e. $\{w \in \Sigma^*, \exists m \in L(A), w \xrightarrow{\oplus}{}^{k} m\}$.

Nous adaptons cette même complétion mais pour les descendants, ainsi si A est un automate qui reconnaît L alors $\mathfrak{T}_R^k(A)$ reconnaît les descendants à k étapes parallèles, ainsi, le système appartient à $\mathrm{MB}_{\max}(k)$ si et seulement si $\mathfrak{T}_R^{k+1}(A) = \mathfrak{T}_R^k(A)$.

Définition 158

Soient $A_s = (Q, \Sigma, \delta_s, I, F)$ un automate signé, R un système de réécriture sur Σ. On définit par $\mathfrak{T}_R(A)$ l'automate signé $(Q_{new}, \Sigma, \delta_{new}, q_0, F)$ où :

$$Q_{new} = Q \cup \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}_+(A,q,l_k) \neq \emptyset \wedge |r_k| > 1}} \left(\bigcup_{j=2}^{|r_k|} \{\text{st}(q,k,j)\} \right)$$

$$\delta_{new} = \delta_- \cup \delta_1 \cup \delta_2 \cup \delta_3 \cup \delta_4$$

avec :

$$\delta_- = \{(q, x, q', -), \exists s \in \{+,-\}, (q, x, q', s) \in \delta\}$$

$$\delta_1 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}_+(A_s,q,l_k) \neq \emptyset \wedge |r_k| > 2}} \left(\bigcup_{j=2}^{|r_k|-1} \{(\text{st}(q,k,j), r_k[j], \text{st}(q,k,j+1), +)\} \right)$$

$$\delta_2 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ \text{Paths}_+(A_s,q,l_k) \neq \emptyset \wedge |r_k| > 1}} \{(q, r_k[1], \text{st}(q,k,2), +)\}$$

$$\delta_3 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ |r_k| > 1}} \left(\bigcup_{q' \in \text{Paths}_+(A_s,q,l_k)} \{\text{st}(q,k,|r_k|), r_k[|r_k|], q', +))\} \right)$$

$$\delta_4 = \bigcup_{\substack{q \in Q \\ (l_k, r_k) \in R \\ |r_k| = 1}} \left(\bigcup_{q' \in \text{Paths}_+(A_s,q,l_k)} \{(q, r_k[|r_k|], q', +))\} \right)$$

Si tous les états de A_s sont accessibles (il existe un run d'un état initial à cet état) et co-accessibles (il existe un run de cet état à un état final), alors les états de $\mathfrak{T}_R(A_s)$ sont accessibles et co-accessibles.

On supposera que tous les états de nos automates sont accessibles et co-accessibles.

Lemme 159

Soit R un système de réécriture. Soit A_s un automate signé. Pour tout $k \geq 0$:

Il existe un run de $\mathfrak{T}_R^k(A_s)$ entre $q_0 \in I$ et $q_f \in F$ qui reconnaît x_n si et seulement s'il existe une dérivation $\sigma = x_0 \rightarrow^* x_n$ avec $x_0 \in L(A_s)$, telle que les positions p dont les transitions du run sont positives sont exactement celles qui vérifient $\text{depth}_\sigma(n,p) = k$, les autres positions vérifiant $\text{depth}_\sigma(n,p) < k$.

Démonstration. Montrons l'implication par récurrence sur k.

- Initialisation : Soit $k = 0$. Soit ρ un run positif de A_s. Soit m le mot reconnu par ρ. La dérivation $\sigma = m$ convient.

- Hérédité : Supposons la propriété pour un entier k fixé. On pose $A' = \mathfrak{T}^k(A) = (Q, \Sigma, \delta, I, F)$ et $\mathfrak{T}_R(A') = (Q_{new}, \Sigma, \delta_{new}, I, F)$ avec les notations de la définition 158.

Soit ρ un run positif de $\mathfrak{T}_R(A')$ qui reconnaît m.

On décompose ρ en sous-run de la forme :

1. Les sous-run dont aucun transition n'est positive
2. Les sous-run qui comportent une unique transition positive entre deux états de Q
3. Les sous-run entre deux états q et q' de Q qui comportent au moins deux transitions positives dont tous les états en dehors de q et q' n'appartiennent pas à Q.

On remarquera que par définition de $\mathfrak{T}_R(A')$ si une transition n'est pas positive alors les états de la transition appartiennent à Q. Ainsi cette décomposition est unique et existe pour tout ρ.

1) Dans le premier cas, si aucune transition d'un sous-run ρ' n'est positive, alors par définition puisque toutes les transitions de $\delta_1 \cup \delta_2 \cup \delta_3 \cup \delta_4$ sont positives, les transitions de ρ' sont dans δ_-, soit $(q, a, q', -)$ une des transitions : par la remarque au dessus, $q, q' \in Q$ et par définition de δ_- il existe une transition $(q, a, q', -)$ ou $(q, a, q', +)$ dans δ. Donc il existe un run ρ'' entre les mêmes états que celui de ρ' et qui reconnaît le même mot qui appartient à $L(A')$. Donc il existe une dérivation de $x_0 \rightarrow^* x_n$ avec $x_0 \in L(A_s)$ telle que $\text{depth}_\sigma(n,p) \leq k < k+1$ pour toute position p.

2) Dans le second cas, le sous-run est composé d'une unique transition $(q, r_m, q', +)$ et puisque la transition est positive elle appartient à $\delta_1 \cup \delta_2 \cup \delta_3 \cup \delta_4$, or parmi ces ensembles de transitions, seul δ_4 contient des transitions dont les deux états sont dans Q. Par définition de δ_4 il existe l_m tel que $(l_m, r_m) \in R$ et $q' \in \text{Paths}(A', q, l_m)$ donc il existe un run positif entre q et q' qui reconnaît l_m dans $A' = \mathfrak{T}_R(A_s)$, de plus on rappelle qu'on a $(l_m, r_m) \in R$.

Par hypothèse de récurrence, il existe une dérivation $\sigma = x_0 \rightarrow^* x_n = l_m$ avec $x_0 \in L(A_s)$, et puisque le run est positif, il y a une transition positive dans A' donc une des positions p de x_n vérifient $\text{depth}_\sigma(n,p) = k$, les autres vérifient $\text{depth}_\sigma(n,p) \leq k$. Or $(l_m, r_m) \in R$ donc la dérivation $\sigma' = x_0 \rightarrow^* x_n \rightarrow x_{n+1} = r_m$ existe. Et on a $\text{depth}_{\sigma'}(n+1,1) = 1 + \max \text{depth}_\sigma(n,p) = k+1$ et cela correspond à la transition positive d'étiquette $r_m[1] = r_m$.

3) Dans le troisième cas, le sous-run ne possède que des transitions positives et seuls le premier q et le dernier état q' appartiennent à Q. Donc les autres états sont de la forme $\text{st}(q,m,j)$. Puisque chacun de ces états n'ont qu'une unique transition sortante et une unique entrante (i.e. il existe une unique transition telle que l'état soit premier membre, resp. avant dernier membre). Alors le sous-run est de la forme :

$$q a_1 \text{st}(q,m,2) a_2 \text{st}(q,m,3) a_3 ... a_{t-1} \text{st}(q,m,t-1) a_t q'$$

On pose $r_m = a_1 ... a_t$, il existe l_m tel que $(l_m, r_m) \in R$ et il existe un run positif de q à q' qui reconnaît l_m dans $A' = \mathfrak{T}_R^k(A_s)$.

Par hypothèse de récurrence, il existe une dérivation $\sigma = x_0 \rightarrow^* x_n = l_m$ avec $x_0 \in L(A_s)$, et puisque le run est positif, il y a une transition positive dans A' donc une des positions p de x_n vérifient $\text{depth}_\sigma(n,p) = k$, les autres vérifient $\text{depth}_\sigma(n,p) \leq k$. Or $(l_m, r_m) \in R$ donc la dérivation $\sigma' = x_0 \rightarrow^* x_n \rightarrow x_{n+1} = r_m$ existe. Et on a $\text{depth}_{\sigma'}(n+1,p') = 1 + \max \text{depth}_\sigma(n,p) = k+1$ pour tout p' entre 1 et t, et toutes les transitions étiquetés par $r_m[p']$ dans le run sont positives.

Enfin, puisque le run ρ est composé de ces run, par concaténation l'implication de l'hypothèse de récurrence est vraie pour ρ.

Prouvons maintenant la réciproque par récurrence sur k.

- Initialisation : Si $k = 0$ alors la seule dérivation de complexité parallèle égale à 0 est $\sigma = x_0$ alors comme $x_0 \in L(A_s)$ on a bien un run entre un état initial et un état final de $\mathfrak{T}^0(A_s) = A_s$ qui reconnaît x_0.

- Hérédité : Soit $\sigma = x_0 \rightarrow^* x_n$ de complexité parallèle $k+1$. Soit i le plus grand élément tel que x_i est de profondeur k.

Par hypothèse de récurrence il existe un run ρ de $\mathfrak{T}_R^i(A_s)$ entre un état initial q_0 et un état final q_f qui reconnaît x_n.

Puisque $x_i \rightarrow x_{i+1}$ il existe μ, μ', u, v tels que $x_i = \mu u \mu'$, $x_{i+1} = \mu v \mu'$ et $(u,v) \in R$. On considère le sous-run ρ' de ρ qui reconnaît u, on le suppose entre q et q'. u admet au moins une position de profondeur exactement k car x_{i+1} est de profondeur $k+1$.

Soit p une position x_i correspondant à une lettre de u telle que $\text{depth}_\sigma(n,p) = k$, alors la transition correspondante dans $\mathfrak{T}^k(R)$ est positive. Par définition de $\mathfrak{T}_R(\mathfrak{T}_R^k(A_s))$ il existe un run entre q et q' dans $\mathfrak{T}_R(\mathfrak{T}_R^k(A_s))$ qui reconnaît v. Or il existe un run de $\mathfrak{T}_R^k(A_s)$ entre q_0 et q qui reconnaît μ et un autre entre q' et q_f qui reconnaît μ'. Comme $\mathfrak{T}_R^k(A_s) \subseteq T_R^{k+1}(A_s)$ il existe un run de q_0 à q_f qui reconnaît $\mu v \mu' = x_{i+1}$.

Montrons maintenant que pour tout $j > i$ il existe un run dans $\mathfrak{T}_R^{k+1}(A_s)$ entre un état initial et un état final qui reconnaît x_j. L'initialisation a été prouvée avec x_{i+1}. Supposons alors que c'est vrai pour j. Par hypothèse il existe un run ρ qui reconnaît x_j dans $\mathfrak{T}_R^k(A_s)$ entre un état intial q_0 et un état final q_f. On a $x_j \to x_{j+1}$, donc il existe μ, μ', u, v tels que $x_j = \mu u \mu'$, $x_{j+1} = \mu v \mu'$ et $(u, v) \in R$, on considère le sous-run qui reconnaît u. Aucune des positions de u n'est de profondeur $k+1$, sinon on aurait les positions de v de profondeur $k+2$ impossible car x_{j+1} a toutes ses positions valant au plus $k+1$. Notons i la plus grande profondeur des positions de u, par hypothèse de récurrence, comme $i < k+1$, il existe un run dans $\mathfrak{T}_R^i(A_s)$ qui reconnait u, une des transitions est positive, donc il existe un run dans $\mathfrak{T}_R^{i+1}(A_s)$ qui reconnaît v. Or $i+1 \leq k+1$ donc ce run est un run de $\mathfrak{T}_R^{k+1}(A_s)$. Donc il existe un run entre q_0 et q_f qui reconnaît $\mu v \mu'$ donc x_{j+1} dans $\mathfrak{T}_R^{k+1}(A_s)$.

Ce qui conclut la preuve.

□

Soit L un langage régulier, reconnu par un automate A. On note A_s l'automate signé correspondant.

Pour tester si $R \notin \mathrm{MB}_{\max}(k, L)$, il faut et il suffit de construire l'automate $\mathfrak{T}_R^{k+1}(A_s)$ et vérifier si A_s contient une transition positive.

Ainsi, d'après la proposition 104 on a :

INSTANCE : R un système de réécriture, L un langage régulier sous forme d'automate $(Q, \Sigma, \delta, I, F)$, k un entier **QUESTION** : $R \in \mathrm{MB}_{\max}(k, L)$? **COMPLEXITE** : EXPTIME

Avec les ancêtres Une autre méthode consiste à utiliser un algorithme non-déterministe qui construit une dérivation certificat de profondeur $k+1$ ssi $R \notin \mathrm{MB}_{\max}(k)$.

Soit σ une dérivation de complexité parallèle $k+1$ alors il existe un nœud n de profondeur $k+1$ dans le graphe de dérivation $G(\sigma)$. On considère les ancêtres $\mathrm{Anc}^*_{G(\sigma)}(n)$ de n comme l'ensemble des nœuds n' de $G(\sigma)$ tel qu'il existe un chemin de n' jusqu'à n. C'est un sous-graphe de $G(\sigma)$ qui induit une sous-dérivation σ' dont l'origine est un facteur de $\mathrm{orig}(\sigma)$.

Définition 160

Soit R un système de réécriture, σ une dérivation et n un nœud de $G(\sigma) = (N, E)$. On définit par induction :

- Les 1-ancêtres de n, noté $\mathrm{Anc}^1_{G(\sigma)}(n)$ comme les n' tels que $(n', n) \in E$
- Les k-ancêtres de n sont les 1-ancêtres des $(k-1)$-ancêtres de n.
- Les ancêtres de n sont l'union de tous les k-ancêtres de n.

Nous notons $\mathrm{Anc}^*_{G(\sigma)}(n)$ (ou plus simplement $\mathrm{Anc}^*(n)$ si le contexte est clair) le sous-graphe des ancêtres de n.

On défini L comme le max des $\#\mathrm{Left}_\sigma(i)$ pour i de 1 à n.

Proposition 161

Soit R un système de réécriture, soit σ une dérivation et n un nœud de profondeur k dans $G(\sigma)$ alors

$$|\mathrm{Anc}^*_{G(\sigma)}(n)| \leq 1 + \frac{L^k - 1}{L - 1}$$

Démonstration. Montrons que pour toute dérivation σ, pour chaque nœud $i \neq 0$ de $G(\sigma)$ on a $|\mathrm{Anc}^1(i)| \leq |\mathrm{Left}_\sigma(i)|$. Par définition $(j, i) \in E$ ssi il existe q et q' tels que $(q, q') \in \mathrm{Corr}_\sigma(i, j-1)$, $q \in \mathrm{Right}_\sigma(j)$ et $q' \in \mathrm{Left}_\sigma(i)$. Soit $\mathrm{Set} = \{(p', p) \mid \exists j\ (p', p) \in \mathrm{Corr}_\sigma(j, i-1), p' \in \mathrm{Right}_\sigma(j), p \in \mathrm{Left}_\sigma(i)\}$. Supposons que (p'_1, p) et (p'_2, p) appartiennent à Set. Alors, p appartient $\mathrm{Left}_\sigma(i)$ et il existe j_1 et j_2 tels que $(p'_1, p) \in \mathrm{Corr}_\sigma(j_1, i-1)$, $p'_1 \in \mathrm{Right}_\sigma(j_1)$ et $(p'_2, p) \in \mathrm{Corr}_\sigma(j_2, i-1)$, $p'_2 \in \mathrm{Right}_\sigma(j_2)$

Nous pouvons supposer sans perte de généralité que $j_1 \leq j_2$. Comme $(p'_1, p) \in \mathrm{Corr}_\sigma(j_1, i-1)$ par définition de Corr, il existe p'' tel que $(p'_1, p'') \in \mathrm{Corr}_\sigma(j_1, j_2)$ et $(p'', p) \in \mathrm{Corr}_\sigma(j_2, i-1)$. Comme $\mathrm{Corr}_\sigma(j_2, i-1)^{-1}$ est injective à droite (i.e. xRz et yRz implique $x = y$), nous avons $p'' = p'_2$. Mais comme $p'_2 \in \mathrm{Right}_\sigma(j_2)$, si $j_1 < j_2$, il n'existe pas de position x telle que $(x, p_2) \in \mathrm{Corr}_\sigma(j_1, j_2)$. Ainsi, $j_1 = j_2$. Comme, $\mathrm{Corr}_\sigma(j_1, i-1)^{-1}$ est injective à droite, si (p'_1, p) et (p'_2, p) sont dans Set alors $p'_1 = p'_2$. Alors le cardinal de Set, et donc de $\#\mathrm{Anc}^1(i)$ est majoré par $|\mathrm{Left}_\sigma(i)|$.

Ainsi le nombre de nœuds de $\mathrm{Anc}^*(i)$ de profondeur $t > 0$ dans $G(\sigma)$ est majoré par L^{k-t} et donc, si i est de profondeur k $|\mathrm{Anc}^*(i)| \leq 1 + \sum_{t=1}^{k} L^{k-t} = 1 + \frac{L^k - 1}{L-1}$. □

Le sous-graphe $\mathrm{Anc}^*_{G(\sigma)}(n)$ d'une dérivation de profondeur k est de taille bornée par $1 + \frac{L^k-1}{L-1}$ (la borne peut-être atteinte). Il est possible de construire un algorithme Co-NEXPTIME qui décide $R \in \mathrm{MB}_{\max}(k)$.

4.3.2 Un algorithme PSPACE

Nous concevons dans cette section un algorithme PSPACE pour décider R appartient à $\mathrm{MB}_{\max}(k)$ et ainsi montrer que :

Théorème 162

On a :

PROBLEME : Complexité parallèle uniformément bornée par une constante
INSTANCE : R un srs, k un naturel
QUESTION : $R \in \mathrm{MB}_{\max}(k)$?
COMPLEXITE : PSPACE-complet

Pour simplifier les preuves, nous enrichissons l'alphabet et transformons le système de réécriture R en un autre système de réécriture R' (voir proposition170) pour lequel les graphes de dérivation ont une structure plus régulière [4] tel que $R \in \mathrm{MB}_{\max}(k+1)$ si et seulement si $R' \in \mathrm{MB}_{\max}(2k+3, \Sigma_0^*)$.

L'objectif est de garder en mémoire uniquement les lettres dont nous avons besoin au fur et à mesure en les ajoutant si nécessaire : on conserve à chaque pas de réécriture un mot de taille polynomiale en la taille du système. Par exemple avec le système $R = \{(aa, a)\}$ on construit la dérivation la dérivation $\sigma : a_0^{2^{k+1}} \to^* a_{k+1}$:

$$\epsilon \overset{\text{on ajoute}}{\longrightarrow} a_0 a_0 \to a_1 \overset{\text{on ajoute}}{\longrightarrow} a_1 a_0 a_0 \to a_1 a_1 \to a_2 \overset{\text{on ajoute}}{\longrightarrow} a_2 a_0 a_0 \to \ldots \to a_{k+1}$$

Proposition 163

Soit R un système de réécriture, soit σ une dérivation et i un nœud de $G(\sigma)$. Il existe une dérivation σ' telle que $G(\sigma') \cong \mathrm{Anc}^*_{G(\sigma)}(i)$.

Démonstration. Par définition de $\mathrm{Anc}^*_{G(\sigma)}(i)$ il existe un tri topologique qui énumère $\mathrm{Anc}^*_{G(\sigma)}(i) \setminus \{i\}$, puis i puis les autres noeud de $G(\sigma)$.

On considère la dérivation obtenue par ce tri topologique (proposition 135).

On considère la sous-dérivation σ' dont le dernier pas de réécriture correspond à i. Alors par définition du tri topologique, σ' vérifie $G(\sigma') \cong \mathrm{Anc}^*_{G(\sigma)}(i)$.

□

4. s'il existe une arête (i, j) dans le graphe de dérivation alors la profondeur de j est 1+ celle de i

Nous notons de la même manière $\mathrm{Anc}(i)$ la dérivation obtenue par la proposition 163.

Nous disons aussi qu'une dérivation $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ est conique s'il existe i tel que $\mathrm{Anc}(i) = \sigma$.

Dans une dérivation conique, il est possible qu'une lettre ne soit jamais utilisée, ces lettres n'ont pas besoin d'être conservées dans le mot de départ (i.e. $\mathrm{orig}(\sigma)$).

Définition 164

Soit R un système de réécriture,
soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation conique. Soit p une position de x_k pour un entier k fixé. On dit qu'une position p est utilisable s'il existe un couple (π, k') tel que $\pi \in \mathrm{Left}_\sigma(k'+1)$ et $(p,\pi) \in \mathrm{Corr}_\sigma(k,k')$

Dans une dérivation conique pour chaque k, l'ensemble des positions utilisables de x_k forment un intervalle :

Proposition 165

Soit R un système de réécriture. Soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation. Pour tout k, pour toute position p, p' et p'' de x_k telles que $p < p'' < p'$, si p et p' sont utilisables alors p'' est utilisable.

Démonstration. Premièrement on peut remarquer que comme σ est conique alors le noeud de profondeur maximale est le dernier nœud : n.

Nous montrons le résultat par récurrence descendante :

Initialisation : L'ensemble des positions utilisables de x_{n-1} est $\mathrm{Left}_\sigma(n)$ donc la propriété est vraie au rang $n-1$.

Hérédité : Supposons que la propriété est vraie au rang k avec $k < n$. Montrons qu'elle est vraie au rang $k-1$.

Soit I_r l'ensemble des positions utilisables de x_k. Comme $k < n$ et σ est conique, le noeud k admet au moins un successeur, donc il existe une position utilisable dans $\mathrm{Right}_\sigma(k)$ i.e. $\mathrm{Right}_\sigma(k) \cap I_r \neq \emptyset$.

Posons $I_r^- = \{p \mid \exists p' \in I_r, (p,p') \in \mathrm{Corr}_\sigma(k-1,k)\}$. Comme $\mathrm{Right}_\sigma(k) \cap I_r \neq \emptyset$ alors $J = \mathrm{Right}_\sigma(k) \cup I_r$ est un intervalle qui peut se décomposer en trois intervalles : $J_1, \mathrm{Right}(k)$ et J_2 où J_1 est soit vide soit $\max J_1 = p_{k-1} - 1$ et J_2 soit vide soit $\min J_2 = p_{k-1} + |v_{k-1}| - |u_{k-1}|$.

Soient $J_1^- = \{p \mid \exists p' \in J_1, (p,p') \in \mathrm{Corr}_\sigma(k-1,k)\}$ et $J_2^- = \{p \mid \exists p' \in J_2, (p,p') \in \mathrm{Corr}_\sigma(k-1,k)\}$. Par définition on a $I_r^- = J_1^- \cup J_2^-$, $J_1^- = J_1$ et $p \in J_2^-$ si et seulement si $p + |v_{k-1}| - |u_{k-1}| \in J_2$.

Alors, soit J_1^- est vide ou $\max J_1^- = p_{k-1} + 1$ ou soit J_2^- est vide ou $\min J_2^- = p_{k-1} + |u_{k-1}|$.

Donc, par définition de $\mathrm{Left}_\sigma(k)$, $I_r^- \cup \mathrm{Left}_\sigma(k) = J_1^- \cup \mathrm{Left}_\sigma(k) \cup J_2^-$ est un intervalle. Comme $I_r^- \cup \mathrm{Left}_\sigma(k)$ est l'ensemble des positions utilisables de x_{k-1}, la propriété est vraie au rang $k-1$.

□

Définition 166

Soit σ une dérivation, on dit que σ est économe si toutes les positions de σ sont utilisables.

En particulier, si une dérivation est conique, l'ensemble des positions utilisables de son mot de départ est un intervalle d'après ce qui précède. On peut alors définir une dérivation économe de même graphe d'origine le mot correspondant à cet intervalle. On obtient donc la proposition suivante :

Proposition 167

Pour toute dérivation σ, il existe une dérivation économe σ' de même profondeur.

Systèmes de réécriture échelonnés

Dans cette partie nous transformons les systèmes de réécriture afin de les rendre plus réguliers et ainsi simplifier le problème : on ne peut pas avoir d'arête entre deux pas de réécriture avec un écart de profondeur > 1.

Autrement dit : deux lettres avec des profondeurs différentes ne peuvent être réécrites dans le même pas de réécriture.

Par exemple le système $R = \{aa \to a\}$ n'est pas échelonné car on peut utiliser des lettres de profondeurs différentes : $a_0a_0a_0 \to a_0a_1 \to a_2$.

Nous allons montrer qu'un système R est dans $\mathrm{MB}_{\max}(k)$ si et seulement si le système transformé R_{Lad} est dans $\mathrm{MB}_{\max}(2k+1, \Sigma_0^*)$ où Σ_0 désigne les lettres de profondeur 0.

Définition 168

Un système de réécriture R est dit échelonné s'il existe une fonction de Σ dans $\mathbb{N}$ telle que pour tout $(u,v) \in R$ alors f est constante sur $\{u[p], 1 \leq p \leq |u|\}$ et pour tout p' entre 1 et $|v|$ on a $f(v[p']) = 1 + f(u[1])$. On note alors (R,f) un système échelonné.

Comme nous l'avons évoqué, les systèmes échelonnés étiquettent les lettres par une valeur qui correspond à la profondeur :

Proposition 169

Soit (R,f) un système de réécriture échelonné.
Soit $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ une dérivation telle que $x_0 \in f^{-1}(0)$.
Pour tout $k \leq n$ et p entre 1 et $|x_k|$ on a :

$$f(x_k[p]) = \text{depth}_\sigma(k,p)$$

Démonstration. Montrons le résultat par récurrence sur k :
Initialisation : si $k = 0$ alors la propriété est vraie car pour tout p de $\{1,...,|x_0|\}$, $x_0[p]$ appartient à $\Sigma_0 = f^{-1}(0)$.
Hérédité : Supposons la propriété vraie pour tout entier $\leq k$. Soit p une position de x_{k+1}.

Si p n'est pas dans $\text{Right}_\sigma(k+1)$, la lettre $x_k[p]$ n'est pas réécrite par le pas $k+1$ donc sa profondeur est la même et la propriété est vérifiée.

Sinon, p est un élément de $\text{Right}_\sigma(k+1)$ donc $\text{index}_\sigma(k+1,p) = k+1$ et $\text{depth}_\sigma(k+1,p) = \text{depth}_{G(\sigma)}(k+1) = 1 + \max_{i/(i,k+1)\in E} \text{depth}_{G(\sigma)}(i)$. On a $(i,k+1) \in E$ si et seulement s'il existe $(q',q) \in \text{Corr}_\sigma(i,k)$ tel que $q' \in \text{Right}_\sigma(i)$ et $q \in \text{Left}_\sigma(k+1)$.

Ainsi , $\text{depth}_\sigma(k+1,p) = 1 + \max_{q\in\text{Left}_\sigma(k+1)} \text{depth}_{G(\sigma)}(\text{index}^k_\sigma(q))$
$= 1 + \max_{q\in\text{Left}_\sigma(k+1)} \text{depth}_\sigma(k,q)$.

Anisi, par hypothèse de récurrence $\text{depth}_\sigma(k+1,p) = 1 + \max_{q\in\text{Left}_\sigma(k+1)} f(x_k[q])$.

Comme R est échelonné, f est constante sur $\text{Left}_\sigma(k+1)$ et $f(x_{k+1}[p]) = 1 + f(x_k[q])$ pour tout élément p de $\text{Right}_\sigma(k+1)$ et q dans $\text{Left}_\sigma(k+1)$.

Ainsi on a :

$$\text{depth}_\sigma(k+1,p) = f(x_{k+1}[p])$$

□

Théorème 170

Pour tout entier k, pour tout système de réécriture R il existe un système de réécriture échelonné (R_{Lad}, f) tel que :

$$R \in \text{MB}_{\max}(k) \Leftrightarrow R_{\text{Lad}} \in \text{MB}_{\max}(2k+1, \Sigma_0^*)$$

Démonstration. À tout entier n, on associe des annotations aux lettres de Σ par niveau pour obtenir un nouvel alphabet $\Sigma_{\text{ext},n}$:

Pour tout i tel que $0 \leq i \leq n$ on associe à toute lettre a de Σ les quatre lettres distinctes

— a_i
— $\bar{a}_i$
— $\bar{\bar{a}}_i$
— $\tilde{a}_i$

On étendra ces notations par homomorphisme (par exemple x_0 avec $x = aab$ désigne $a_0 a_0 b_0$).

On associe alors à R le système de réécriture $R_{\text{Lad}}(n)$ défini par :

$$R_{\text{Lad}}(n) = \begin{cases} \bar{x}_h \tilde{a}_h \bar{y}_h \rightarrow r_{h+1}, \forall h < n, x, y \in \Sigma^*, a \in \Sigma \text{ s.t. } (xay, r) \in R \\ a_h \rightarrow \bar{a}_h, \forall h \leq n, a \in \Sigma \\ a_h \rightarrow \tilde{a}_h, \forall h \leq n, a \in \Sigma \\ \bar{a}_h \rightarrow \bar{\bar{a}}_{h+1}, \forall h < n, a \in \Sigma \\ \bar{\bar{a}}_h \rightarrow \bar{a}_h, \forall h \leq n, a \in \Sigma \end{cases} \tag{4.1}$$

Le système $R_{\text{Lad}}(n)$ est échelonné en posant : $f(a_t) = f(\bar{\bar{a}}_t) = 2t$ et $f(\bar{a}_t) = f(\tilde{a}_t) = 2t+1$. On pose alors $\Sigma_i = f^{-1}(i)$.

Montrons maintenant que :

$$R \in \text{MB}_{\max}(k) \text{ si et seulement si } R_{\text{Lad}}(k) \in \text{MB}_{\max}(2k+1, \Sigma_0^*)$$

Nous commençons par montrer par récurrence sur n la propriété H_n :

H_n : Pour toute dérivation sur R de longueur n, il existe une dérivation σ' sur $R_{\text{Lad}}(H)$ où H est la complexité parallèle de σ telle que :

— $\text{orig}(\sigma') = \text{orig}(\sigma)_0$
— $|\text{aim}(\sigma')| = |\text{aim}(\sigma)|$
— $\forall p, \text{aim}(\sigma')[p] = \text{aim}(\sigma)[p]_t$ avec $t = \text{depth}_\sigma(n, p)$

Initialisation : on a $n = 0$, il suffit de considérer σ' définie par $\text{orig}(\sigma') = \text{orig}(\sigma)_0$.
Hérédité : Supposons H_n vraie pour un entier fixé n.
Soit $s \longrightarrow^n g \longrightarrow d$ une dérivation de longueur $n+1$. Par hypothèse de récurrence, comme la dérivation $\sigma = s \longrightarrow^n g$ est de longueur n, il existe une dérivation $\sigma' = S \longrightarrow^*_{R_{\text{Lad}}} G$ telle que $\text{orig}(\sigma') = \text{orig}(\sigma)_0$, $|G| = |g|$ et $G[p] = g[p]_t$ avec $t = \text{depth}_\sigma(n, p)$ pour toute position de G.

Comme $g \longrightarrow_R d$ il existe une règle $(l, r) \in R$ et deux mots m et m' tels que $g = mlm'$ et $d = mrm'$. Soit p une position dont la profondeur est maximale dans l pour $s \longrightarrow^n_R g$, notons h cette profondeur. et a cette lettre. On a $g = mxaym'$ et par hypothèse de récurrence, G est de la forme MXa_hYM' avec M, X, Y, M' tels que

base$(M) = m$, base$(M') = m'$, base$(X) = x$, base$(Y) = y$ et la profondeur de chaque lettre de XY est majorée par h.

Alors, pour toute lettre s_t qui apparait dans X ou Y nous avons :

$$s_t \underset{s_h \to \bar{s_h}}{\longrightarrow} \left(\underset{\bar{s}_h \to \bar{\bar{s}}_{h+1}}{\longrightarrow} \cdot \underset{\bar{\bar{s}}_{h+1} \to \bar{s}_{h+1}}{\longrightarrow} \right)^{h-t} \bar{s}_h$$

Donc $G \longrightarrow^*_{R_{\text{Lad}}} M\bar{x}_h a_h \bar{y}_h M' \longrightarrow_{R_{\text{Lad}}} M\bar{x}_h a_h \bar{y}_h M' \longrightarrow_{R_{\text{Lad}}} M r_{h+1} M'$ comme $xay = l$.
Soit $D = M r_{h+1} M'$, nous avons base$(D) = mrm' = d$. Si p n'est pas une position de r_{h+1} alors $D[p] = G[p] = g[p]_t = d[p]_t$ pour $t = \text{depth}_\sigma(g[p])$.
Si p est une position de r_{h+1} alors $\text{depth}_\sigma[n, p] = 1 + h$ et $D[p] = d[p]_{h+1}$. Ce qui termine la récurrence.

Nous pouvons maintenant prouver que si $R_{\text{Lad}}(n) \in \text{MB}_{\max}(2k+1, \Sigma_0^*)$ alors $R \in \text{MB}_{\max}(k)$. Si on suppose le contraire, il existe une dérivation de profondeur $k+1$ sur R qui correspond à une dérivation sur $R_{\text{Lad}}(n+1)$ avec une lettre étiquetée par $k+1$, donc de profondeur $2k+2$ d'après la proposition ci-dessus, ainsi la dérivation devrait être de profondeur au moins $2k+2$.

Montrons finalement que si $R \in \text{MB}_{\max}(n)$ alors $R_{\text{Lad}}(n) \in \text{MB}_{\max}(2n+1, \Sigma_0^*)$.

Nous définissons $\widehat{\text{base}}$ (resp. ind) qui à une des lettres de la forme $a_t, \bar{a}_t, \tilde{a}_t, \bar{\tilde{a}}_t$ associe a (resp. t). Nous étendons ces applications en tant que morphisme de monoïde.

Montrons par récurrence que :

P_n : Pour toute dérivation σ' de longueur n sur R_{Lad} telle que orig$(\sigma') \in \Sigma_0^*$, il existe une dérivation σ sur R telle que :

— $\widehat{\text{base}}(\text{aim}(\sigma')) = \text{aim}(\sigma)$
— $\forall p, \text{depth}_\sigma(|\sigma|, p) = \text{ind}(\text{aim}(\sigma')[p])$

Initialisation : On choisit σ avec orig$(\sigma) = \widehat{\text{base}}(\text{orig}(\sigma'))$

Hérédité : Soit $\sigma' = S \longrightarrow^n_{R_{\text{Lad}}} G \longrightarrow^1_{R_{\text{Lad}}} D$ une dérivation de longueur $n+1$.

Par hypothèse de récurrence, il existe $\sigma = s \longrightarrow^* g$ sur R telle que $\widehat{\text{base}}(G) = g$ et $\text{depth}_\sigma[|\sigma|, p] = \text{ind}(G[p])$ pour tout position p de G. Comme $G \longrightarrow_{R_{\text{Lad}}} D$ il existe M, M', L, R tels que $MLM' = G, MRM' = D, L \to R \in R_{\text{Lad}}$

— Si $L \to R$ est de la forme $\bar{x}_h \tilde{a}_h \bar{y}_h \to r_{h+1}$ alors $\widehat{\text{base}}(L) \to \widehat{\text{base}}(R) \in R$, nous avons $\widehat{\text{base}}(R) = r$.
Soit $d = mrm'$ avec $m = \widehat{\text{base}}(M)$ $m' = \widehat{\text{base}}(M')$ et $l = \widehat{\text{base}}(L) = xay$. Comme $g = \widehat{\text{base}}(G) = \widehat{\text{base}}(M)\widehat{\text{base}}(L)\widehat{\text{base}}(M') = mlm'$, $g \longrightarrow_R d$. Soit σ'' construite

depuis σ en ajoutant ce dernier pas de réécriture. $\widehat{\text{base}}(\text{aim}(\sigma')) = \text{aim}(\sigma'')$.
Prouvons que pour chaque position p de $\text{aim}(\sigma'')$,
$\text{depth}_{\sigma''}[|\sigma''|, p] = \text{ind}(\text{aim}(\sigma')[p])$

Si p est une position de m ou m', la lettre correspondante a été réécrite et $\text{depth}_{\sigma''}[|\sigma''|, p] = \text{depth}_{\sigma}[|\sigma|, q]$ avec (q, p) in $Corr_{\sigma''}(|\sigma''| - 1, |\sigma|)$. Par récurrence, $\text{depth}_{\sigma}[|\sigma|, q] = \text{ind}(\text{aim}(\sigma')[q])$ et par construction, (q, p) dans $Corr_{\sigma'}(|\sigma'] - 1, |\sigma'])$, donc $\text{ind}(\text{aim}(\sigma')[q]) = \text{ind}(\text{aim}(\sigma)[p])$.

Si p est une position de r ; alors $\text{depth}_{\sigma''}(|\sigma''|, p)$ est égale à 1 plus la profondeur maximale dans σ des positions des lettres appartenant à l et nous obtenons le résultat par récurrence.

— Les autres cas $s \longrightarrow_R^* g$ sont évidents.

Ainsi, si $R \in \text{MB}_{\max}(n)$ alors $R_{\text{Lad}}(n) \in \text{MB}_{\max}(2n+1, \Sigma_0^*)$: si ce n'est pas le cas, il existe une dérivation sur R_{Lad} de profondeur supérieure ou égale à $2n+2$ dont l'origine est un mot de Σ_0^* ; alors, une lettre indexée par $n+1$ apparait (d'après la proposition ci-dessus), et il existe donc une dérivation de profondeur $n+1$ dans R, contradiction.

□

Dérivations standardes sur les systèmes échelonnés

Dans cette partie nous travaillons sur les dérivations standardes : si deux pas consécutifs sont orthogonaux, alors on exécute d'abord le pas le plus à gauche en premier. Plus formellement :

Définition 171

Soit R un système de réécriture et soit $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ une dérivation. σ est dite standarde si pour tout i on a :

$$\min \text{Right}_\sigma(i) \leq \max \text{Left}_\sigma(i+1)$$

Proposition 172

Soit R un système de réécriture. Pour toute dérivation σ, il existe une dérivation standarde équivalente à σ.

Démonstration. À toute dérivation $\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 \ldots \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$ on associe une séquence finie de longueur n d'entiers par : $I(\sigma) = (p_0, p_1, \ldots, p_{n-1})$.

On considère l'ordre lexicographique $\prec$ sur $\mathbb{N}^n$.

Soit σ une dérivation et E l'ensemble des dérivations équivalentes à σ.

Soit σ' dans E tel que $I(\sigma')$ est minimal pour $\prec$ et soit $I(\sigma') = (p_0, p_1, \ldots, p_{n-1})$.

Si σ' n'est pas standarde, alors il existe i tel que $\max\text{Left}_{\sigma'}(i) < \min\text{Left}_{\sigma'}(i-1)$. Donc $\text{exch}_{i-1,i}(\sigma)$ est dans E et

$I(\text{exch}_{i-1,i}(\sigma')) = (p_0, ..., p_{i-2}, p_i, p_{i-1} + |v_i| - |u_i|, p_{i+1}, ..., p_n)$.

Comme $\max\text{Left}_{\sigma'}(i) < \min\text{Right}_{\sigma'}(i-1)$ nous avons $p_i < p_{i-1}$, ainsi $I(\text{exch}_{i-1,i}(\sigma')) \prec I(\sigma)$, ce qui contredit la minimalité de $I(\sigma')$. standarde équivalente à σ.

□

Dans le cas des systèmes échelonnés, pour tout mot d'une dérivation standarde les profondeurs des lettres dont les positions sont utilisables sont de gauche à droite.

Cette propriété est fausse en général :

La dérivation $gad \to gbd \to f$ est standard mais le g de gbd est de profondeur 0 donc strictement inférieure à celle du b qui est de profondeur 1 et pourtant les positions 1 et 2 correspondantes à g sont utilisables.

Proposition 173

Soit R un système de réécriture échelonné. Soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation standarde. Pour tout k, pour toute position p et p' de x_k telles que $p < p'$ et p et p' sont utilisables on a

$$\text{depth}_\sigma(k,p) \geq \text{depth}_\sigma(k,p')$$

Démonstration. Soit $(N,E) = G(\sigma)$.

Montrons d'abord le lemme suivant par récurrence sur k :

Lemme 174

Soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation standarde
Soit $k \leq n$ et soit p une position de x_k telle que $p < \min\text{Right}_\sigma(k)$. Si il existe j et π tels que $(p,\pi) \in \text{Corr}_\sigma(k,j)$, pour tout t tel que $k \leq t \leq j$ on a $p < \min\text{Right}_\sigma(t)$ et $(p,p) \in \text{Corr}_\sigma(k,j)$ (donc, $\pi = p$).

Si $t = k$, le lemme est vrai par hypothèse. Supposons que la propriété soit vraie pour t avec $k \leq t < j$. Alors, $p < \max\text{Right}_\sigma(t)$ et $(p,p) \in \text{Corr}_\sigma(k,t)$.

Si $(t, t+1)$ n'est pas une arête de E, comme la dérivation est standarde on a $\max \text{Right}_\sigma(t) < \min \text{Left}_\sigma(t+1)$. Alors $p < \min \text{Right}_\sigma(t+1)$ et $(p,p) \in \text{Corr}_\sigma(t, t+1)$, et donc est appartient à $\text{Corr}_\sigma(k, t+1)$ par hypothèse de récurrence.

Supposons maintenant que $(t, t+1)$ est une arête de E. Comme $(p,p) \in \text{Corr}_\sigma(k,t)$ et $(p,\pi) \in \text{Corr}_\sigma(k,j)$, $(p,\pi) \in \text{Corr}_\sigma(t,j)$: comme $t+1 \leq j$, $p \notin \text{Left}(t+1)$, par hypothèse de récurrence $p < \min \text{Right}_\sigma(t)$, et comme $(t, t+1)$ est une arête $\text{Right}_\sigma(t) \cap \text{Left}_\sigma(t+1)$ est non vide : donc $p < \min \text{Left}_\sigma(t+1)$ et alors $p < \min \text{Right}_\sigma(t+1)$, (p,p) appartient à $\text{Corr}_\sigma(t, t+1)$,et donc à $\text{Corr}_\sigma(k, t+1)$.

Nous pouvons maintenant prouver la proposition par récurrence :

Initialisation : La propriété est clairement vraie pour x_0.

Hérédité : Supposons la propriété vraie pour x_i. Comme $x_i \longrightarrow x_{i+1}$ il existe α_i, β_i et $(u_i, v_i) \in R$ tels que $x_i = \alpha_i u_i \beta_i$ et $x_{i+1} = \alpha_i v_i \beta_i$. Soient p et p' deux positions ables de x_{i+1} avec $p < p'$.

Si p, p' sont des positions de α_i ou de β_i, par hypothèse de récurrence sur x_i la propriété est vraie.

Si p, p' sont des positions de v_i, elles ont même profondeur, la propriété est vraie.

Si p est une position de v_i et p' est une position de β_i,

nous avons $\text{depth}_\sigma(i+1, p) > \text{depth}_\sigma(i, p)$ et $\text{depth}_\sigma(i, p') = \text{depth}_\sigma(i+1, p')$, donc $\text{depth}_\sigma(i+1, p) > \text{depth}_\sigma(i+1, p')$ par hypothèse de récurrence.

Seul le cas où p est une position de α_i et p' est une position de v_i reste donc à étudier.

Soit $B = \{p \mid \exists p' \in \text{Right}_\sigma(i+1), p < p', p \text{ utilisable }, \text{depth}_\sigma(i+1, p') > \text{depth}_\sigma(i+1, p)\}$. Si B est vide, la preuve est terminée. Sinon, soit $p = \max B$. Comme p est utilisable il existe π et j tels que $\pi \in \text{Left}_\sigma(j+1)$ et on a $(p,\pi) \in \text{Corr}_\sigma(i,j)$. Comme $p' \in \text{Right}_\sigma(i+1)$, et $\text{depth}_\sigma(i+1, p) < \text{depth}_\sigma(i+1, p')$ pour un p' dans $\text{Right}_\sigma(i)$, p n'appartient pas à $\text{Right}_\sigma(i)$ et donc $p < \min \text{Right}_\sigma(i)$. D'après le lemme, on a $p < \min \text{Right}_\sigma(j)$. Comme σ est standarde , on a $\text{Left}_\sigma(j+1) \cap \text{Right}_\sigma(j) \neq \emptyset$, et donc $p < \max \text{Left}_\sigma(j+1)$.

Prouvons maintenant que pour tout t, $i \leq t \leq j$, soit $p+1$ n'est pas utilisable soit $\text{depth}_\sigma(t, p) < \text{depth}_\sigma(t, p+1)$. Comme p est maximal dans B, la propriété est vraie pour $t = i$. Supposons qu'elle soit vraie pour t $(t < j)$. Si $p+1$ n'est pas utilisable, c'est terminé. Sinon, au pas $t+1$, si $p+1$ n'a pas été utilisé, comme $p < \min \text{Left}_\sigma(t+1)$, $p+1 < \min \text{Left}_\sigma(t+1)$ et $\text{depth}_\sigma(t+1, p+1) = \text{depth}_\sigma[p+1, t]$. Si $p+1$ est réécrit, comme $p < \min \text{Left}_\sigma(t+1)$, $p+1 = \min \text{Left}_\sigma(t+1) = \min \text{Right}_\sigma(t+1)$ et $\text{depth}_\sigma(t+1, p+1) > \text{depth}_\sigma(t, p+1)$. Dans les deux cas, $\text{depth}_\sigma(t+1, p) < \text{depth}_\sigma(t+1, p+1)$.

Ainsi, à l'étape j, soit $p+1$ n'est pas utilisable, soit $\text{depth}_\sigma(j, p+1) > \text{depth}_\sigma(j, p+1)$. Comme R est échelonné et $p \in \text{Left}_\sigma(j+1)$, $p+1 \notin \text{Left}_\sigma(j+1)$. Ainsi, nous obtenons une contradiction et B est vide.

□

Ainsi, par la proposition 173 et 165, si σ est conique l'ensemble L_i des positions

utilisables de profondeur i de x_k est un intervalle. Nous montrons maintenant que la longueur de cet intervalle est $\leq L$ où L désigne le maximum des $|l|$ pour $(l,r) \in R$.

Proposition 175

Soit R un système de réécriture échelonné. Soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation standarde et conique. Pour tout $k < n$ et pour tout $i \leq \text{depth}_{G(\sigma)}(k+1)$ l'ensemble L_i des positions des lettres utilisables de x_k de profondeur i est un intervalle de longueur $< L$ avec $L = \max_{(l,r)\in R} |l|$.

Démonstration. Premièrement L_i est un intervalle d'après 173 et 165.

Comme σ est conique, seul n n'admet pas de successeur dans le graphe de dérivation de σ. Puisque $k < n$ alors k admet un successeur, i.e. il existe une position utilisable P dans $\text{Right}_\sigma(k)$. Par la proposition 173 comme P vérifie $\text{depth}_\sigma(k,P) < i$ on a pour tout $p \in L_i$, $p < \min \text{Right}_\sigma(k)$.

Nous montrons qu'il existe un j tel que que le minimum de L_i est $\min \text{Left}_\sigma(j+1)$: soit p ce minimum, comme p est utilisable il existe un couple (π, j) tel que $\pi \in \text{Left}_\sigma(j+1)$ et $(p,\pi) \in \text{Corr}_\sigma(k,j)$. Or $p < \min \text{Right}_\sigma(k)$ donc d'après le Lemme 174 on a pour tout t entre k et j l'inégalité $p \leq \min \text{Right}_\sigma(t)$ et on a $\pi = p$ donc $p \in \text{Left}_\sigma(j+1)$. Montrons que ce p est bien le minimum de $\text{Left}_\sigma(j+1)$, c'est un intervalle d'entiers, donc si p n'est pas le minimum alors $p-1$ appartient à $\text{Left}_\sigma(j+1)$. Dans ce cas $p-1$ est utilisable et comme R est échelonné et que p et $p-1$ appartiennent au même Left on a $\text{depth}_\sigma(j,p) = \text{depth}_\sigma(j,p-1)$. On a montré que $p < \min \text{Right}_\sigma(t)$ pour tout t entre k et j, il en est donc de même pour $p-1$ et donc $(p-1,p-1) \in \text{Corr}_\sigma(k,j)$, donc la profondeur ne change pas entre les mots x_k et x_j on a : $\text{depth}_\sigma(k,p) = \text{depth}_\sigma(k,p-1)$. $p-1$ est donc une position de x_k utilisable de profondeur i, donc appartient à L_i, contradiction avec la minimalité de L_i atteinte en p. Ainsi, $p = \min \text{Left}_\sigma(j+1)$.

Nous montrons que les positions de L_i ne sont jamais utilisées en k et j par récurrence, on sait le résultat déjà vrai pour p. Soit $q \in L_i$. Pour $t = k$ on a bien $(q,q) \in \text{Corr}(k,t)$. Supposons la propriété vraie au rang t pour $t < j$, montrons la au rang $t+1$. Supposons que $q \in \text{Left}_\sigma(t+1)$, puisque $(q,q) \in \text{Corr}(k,t)$ on a $\text{depth}_\sigma(t,q) = \text{depth}_\sigma(t,k) = i$, donc toute position q' de $\text{Right}_\sigma(t+1)$ vérifie $\text{depth}_\sigma(t+1,q') = i+1$, or on a $p < \min \text{Right}_\sigma(t+1)$ donc $p < q'$. Or $\text{depth}_\sigma(t+1,p) = i < \text{depth}_\sigma(t+1,q')$ contradiction avec la proposition 173. Ainsi on a $q \notin \text{Left}_\sigma(t+1)$ donc $q < \min \text{Left}_\sigma(t+1)$ donc $(q,q) \in \text{Corr}_\sigma(t,t+1)$ et ainsi $(q,q) \in \text{Corr}_\sigma(k,t+1)$. On obtient donc que pour toute position q de L_i on a $q \notin \text{Right}_\sigma(j)$.

On a montré que $\min L_i = \min \text{Left}_\sigma(j+1) \leq \max L_i < \min \text{Right}_\sigma(j)$, or comme σ est standarde on a $\min \text{Right}_\sigma(j) \leq \max \text{Left}_\sigma(j+1)$. Donc L_i strictement inclus dans $\text{Left}_\sigma(j+1)$ donc $\#L_i < L$. □

Algorithme

Nous avons montré que pour toute dérivation il en existe une conique et standard de même profondeur. Nous avons aussi montré que nous pouvions nous restreindre au problème de décision $R \in \mathrm{MB}_{\max}(2k+1, f^{-1}(0))$ pour un entier k et un système échelonné (R, f).

Dans une dérivation standarde sur un système échelonné, les positions utilisables de même profondeur dans un mot forment un intervalle.

De plus si la dérivation est conique, on peut borner la taille de ces intervalles (pour une profondeur non nulle) par la taille du système R et toutes les lettres qui ne sont pas utilisables forment un préfixe du mot. Plus formellement :

Théorème 176

Soit (R, f) un système de réécriture échelonné, soit $\sigma = x_0 \xrightarrow{p_0,(u_0,v_0)} x_1 \ldots \xrightarrow{p_{n-1},(u_{n-1},v_{n-1})} x_n$ une dérivation conique, économe et standard telle que $\mathrm{orig}(\sigma) \in f^{-1}(0)$.
Pour tout $k \leq n$, en notant m la profondeur de x_k il existe des mots $d_0^k, d_1^k, \ldots, d_m^k$ et $f_1^k, f_2^k, \ldots f_m^k$ tels que :

- $x_k = d_0^k d_1^k \ldots d_m^k f_m^k f_{m-1}^k \ldots f_1^k f_0^k$
- Les positions des d_i^k ne sont pas utilisables
- pour $k < n$, si $f_m^k f_{m-1}^k \ldots f_i^k$ est vide, $d_i^k \ldots d_m^k$ est vide.
- Les positions utilisables sont celles de $f_m^k f_{m-1}^k \ldots f_1^k f_0^k$ si f_0^k est non vide, sinon celle d'un préfixe de $f_m^k f_{m-1}^k \ldots f_1^k f_0^k$
- Les positions de d_i^k et de f_i^k sont de profondeur i
- La longueur de f_i^k pour $i > 0$ est $< L + M$ avec $L = \max_{(l,r)\in R} |l|$ et $M = \max_{(l,r)\in R} |r|$

Démonstration. Nous montrons le théorème par récurrence sur k.
Initialisation : Soit f_0^0 égal à x_0 et donc les f_i^0 pour i>0 et les d_i^0 égaux au mot vide : toutes les positions de f_0 sont utilisables car la dérivation est économe et pour toute position p de x_0 nous avons $\mathrm{depth}_\sigma(0, p) = 0$.

Hérédité : Supposons la propriété vraie au rang k. Nous avons $x_k \xrightarrow{p_k,(u_k,v_k)} x_{k+1}$. Soit $m = \max\{\mathrm{depth}(i), i \leq k\}$.

Comme R est échelonné et comme $\mathrm{Left}_\sigma(k+1)$ est un intervalle nous avons u_k facteur de f_i^k or d_i^k, pour un certain i. Comme toute les positions de u_k sont utilisables alors u_k est un facteur de f_i^k. Si $f_i^k = \alpha u_k \beta$ alors après le pas de réécriture nous avons $\alpha v_k \beta$ facteur de x_{k+1} mais toutes les positions de α sont utilisables et

toutes les lettres de α ont une profondeur strictement inférieure à toutes les lettres de v_k, ce qui est impossible, donc $\alpha = \epsilon$.

- Si $f_m^k f_{m-1}^k ... f_{i+1}^k$ est non vide : posons $d_t^{k+1} = d_t^k$ pour tout t dans $0, 1, ...m$, $f_t^{k+1} = f_t^k$ pour tout t dans $0, 1, ...m$ différent de i et $i+1$, $f_i^{k+1} = \beta$ et $f_{i+1}^{k+1} = f_{i+1}^k v_k$; alors, $x_{k+1} = d_0^{k+1} \dots d_m^{k+1} f_m^{k+1} \dots f_0^{k+1}$.
Comme la dérivation est conique, les positions de x_k utilisables sont un intervalle qui ont pour préfixe $f_m^k f_{m-1}^k ... f_{i-1}^k = f_m^{k+1} f_{m-1}^{k+1} ... f_{i-1}^{k+1}$ non vide, donc forment un préfixe de $f_m^{k+1} f_{m-1}^{k+1} ... f_0^{k+1}$. Les positions de d_i^{k+1} et de f_i^{k+1} sont bien de profondeur i. Si $f_m^{k+1} f_{m-1}^{k+1} \dots f^{k+1}$ est vide, $f_m^k f_{m-1}^k \dots f_j^k$ l'était, donc $d_j^{k+1} ... d_m^{k+1} = d_j^k ... d_m^k$ l'est aussi.

Il reste donc à montrer que la longueur de f_{i+1}^{k+1}, seule à avoir augmenté, est bien inférieure ou égale à $L + M$.

Comme L_{i+1} est l'ensemble des positions de f_{i+1}^k nous avons $|f_{i+1}^k| < L$ d'après la proposition 175 (on a bien i qui supérieur à la profondeur de $k+1$) donc $|f_{i+1}^{k+1}| = |f_{i+1}^k v_k| < L + M$.

- Si $f_m^k f_{m-1}^k ... f_{i+1}^k$ est vide, alors $d_i^k ... d_m^k$ l'est aussi : on peut factoriser v_k en ww' où seules les positions de w' sont utilisables. On remarquera que comme la dérivation est conique, w' est nécessairement non vide, sauf éventuellement pour le dernier pas. Posons $d_t^{k+1} = d_t^k$ pour tout t dans $0, 1, ...m$ sauf $i+1$, $d_{i+1}^{k+1} = w$, $f_t^{k+1} = f_t^k$ pour tout t dans $0, 1, ...m$ différent de i et $i+1$, $f_{i+1}^{k+1} = w'$ et $f_i^{k+1} = \beta$; alors $x_{k+1} = d_0^{k+1} \dots d_i^{k+1} d_{i+1}^{k+1} f_{i+1}^{k+1} f_m^{k+1} \dots f_0^{k+1}$. Les propriétés sur les positions utilisables et les profondeurs sont vraies par définition, la dernière propriété sur les longueurs est immédiate puisque $f_{i+1}^{k+1} \leq M$ et $|f_j^{k+1}| \leq L + M$ si j différent de $i+1$.

□

Nous proposons maintenant un algorithme non déterministe PSPACE qui considère deux entrées :

— Un système de réécriture échelonné
— Un entier k

et qui renvoie Vrai s'il existe une dérivation de profondeur $k+1$ et Faux sinon, autrement dit qui teste si $R \notin \mathrm{MB}_{\max}(k)$.

On rappelle que tout système de réécriture peut-être transformé en temps polynomial en un système de réécriture échelonné.

Cet algorithme non déterministe garde en mémoire un tableau de mots $[f_h, ..., f_1]$. Qui contient les positions des lettres utilisables lors du cours de la dérivation dont chaque f_i contient les lettres de profondeur i.

input : un système de réécriture échelonné R, un entier k

$L \leftarrow \max_{(u,v)\in R} |u|$;
$C \leftarrow (L^{k+1}-1)/(L-1)$;
$h \leftarrow 0$;
while $(C > 0)$ *and* $(h < k+1)$ **do**
  Choices $\leftarrow \{(i,v) \mid 1 \le i \le h, (u,v) \in R, f_i = u f_i', |f_{i+1}| < L\}$;
  if $\max_{1\le h} f_i < L$ **then**
    Choices $\leftarrow$ Choices $\cup \{(0,v), (u,v) \in R\}$;
  if Choices *is empty* **then**
    return False ;
  else
    choose : $(i,v) \in$ Choices ;
    if $i > 0$ **then**
      $f_i \leftarrow f_i'$;
  if $(i == h)$ **then**
    choose : v', v'' s.t. $v = v''v'$;
    $f_{i+1} \leftarrow v'$;
    $h \leftarrow h+1$;
  else
    $f_{i+1} \leftarrow f_{i+1} v$;
  $C \leftarrow C-1$;
return $(h == k+1)$;

Nous utilisons le théorème 176 pour montrer trois points :

1. S'il existe une exécution de l'algorithme qui renvoie vrai alors il existe une dérivation de profondeur $k+1$
2. S'il existe une dérivation de profondeur $k+1$, alors il existe une exécution de l'algorithme qui renvoie vrai.
3. L'algorithme est PSPACE.

Premier point :

Nous montrons par invariance qu'à la fin du t-ième passage dans la boucle While : Il existe une dérivation de profondeur h de but x_t avec $f_h...f_1$ facteur de x_t et pour tout i f_i est non vide et toutes les lettres de f_i sont de profondeurs i où $[f_h, ..., f_1]$ est la mémoire.

- Initialisation : au 0-ième passage dans la boucle, la mémoire est [], on considère tout mot x_0, c'est une dérivation de profondeur 0 dont le but admet $f_h...f_1 = \epsilon$ comme facteur. au départ, on considère tout mot x_0, c'est une dérivation de profondeur 0 dont le but admet $f_h...f_1 = \epsilon$ comme facteur.

- Hérédité : supposons que nous sommes au t-ième passage de la boucle while. On suppose que nous entrons dans la boucle while, on a donc $h < k+1$ et $C > 0$. Par hérédité, il existe une dérivation σ_t de profondeur h de la forme $x_0 \to x_t = \mu f_h ... f_1 \mu'$ où $[f_h, ..., f_1]$ est la mémoire de plus chaque lettre de f_i est de profondeur i pour tout i.

À la fin de la boucle on a Choices non vide sinon la branche d'exécution renvoie faux, contrairement à notre hypothèse initiale. Donc il existe (i, v) dans Choices. Par construction de Choices, (i, v) vérifie l'existence d'un u tel que $(u, v) \in R$ et i entre 0 et h.

Si i est strictement entre 0 et h alors il existe f_i' tel que $f_i = u f_i'$: on a $x_t = \mu f_h ... f_1 \mu'$ on pose $x_{t+1} = \mu g_h g_{i+1} g_i g_{i-1} ... g_1 \mu'$ avec $g_j = f_j$ pour j différent de i et $i+1$, $g_i = f_i'$ et $g_{i+1} = f_{i+1} v$. Puisque i est strictement entre 0 et h, la mémoire de l'algorithme à la fin du $t+1$-ième passage dans la boucle est $[g_h, ..., g_1]$. De plus on a $x_t = \mu f_h ... f_1 \mu' = \mu f_h ... f_{i+1} u f_i' ... f_1 \mu' \to_R \mu f_h ... f_{i+1} v f_i' ... f_1 \mu' = \mu g_h g_{i+1} g_i g_{i-1} ... g_1 \mu' = x_{t+1}$. La profondeur des lettres de g_j est celle de f_j pour j différent de i et $i+1$, la profondeur des lettres de g_i est celle de f_i car $g_i = f_i'$ est un facteur de f_i et la profondeur des lettres de g_{i+1} sont celles de f_{i+1} de profondeur $i+1$ et de v qui sont de profondeur égale à $1+$ la profondeur des lettres de u dont $1+i$ car u facteur de f_i. La dérivation $x_0 \to x_{t+1}$ est de profondeur h car $i < h$ et les lettres des mots μ et μ' n'ont pas changé, ainsi cette dérivation vérifie les hypothèses demandées au rang $t+1$ par la récurrence.

Lorsque la branche d'exécution termine, puisqu'elle renvoie vrai, le booléen $h = k+1$ est vrai, comme il existe une dérivation de profondeur h, elle est de profondeur $k+1$.

Second point : Supposons qu'il existe une dérivation de profondeur $k+1$, nous montrons qu'il existe une branche d'exécution qui renvoie vrai.

Pour toute dérivation il existe une dérivation conique, économe et standarde de même profondeur. Il existe donc une telle dérivation de profondeur $k+1$, nous la dénotons par :

$$\sigma = x_0 \overset{p_0,(u_0,v_0)}{\longrightarrow} x_1 ... \overset{p_{n-1},(u_{n-1},v_{n-1})}{\longrightarrow} x_n$$

Comme la dérivation est conique, les seules lettres de profondeur $k+1$ sont celles de x_n.

Comme le système de réécriture est échelonné, le théorème 176 s'applique, on a donc pour tout k :

$$x_k = d_0^k ... d_m^k f_m^k ... f_0^k$$

avec les lettres des d_i^k et des f_i^k de profondeur i.

Montrons par récurrence sur $t \leq n$ qu'il existe une branche d'exécution telle qu'à la fin du t-ième passage dans la boucle la mémoire contient $[f_h^t, ..., f_1^t]$ pour $t \leq n$ et h vaut la profondeur de x_t.

- Initialisation : Si la dérivation est de profondeur nulle, alors $k+1 = 0$, ainsi le test $h < k+1$ n'est pas réalisé et nous ne rentrons pas dans la boucle. On a la mémoire

qui est bien vide. Et $h = 0$.

- Hérédité : Supposons qu'il existe une branche d'exécution telle qu'à la fin du t-ième passage dans la boucle la mémoire contient $[f_h^t, ..., f_1^t]$ avec $t < n$ et h vaut la profondeur de x_t.

D'après la proposition 161 la longueur n de la dérivation vérifie $n \leq \frac{L^{k+1}-1}{L-1}$ donc comme $t < n$ on a $C > 0$. De plus $t < n$ donc la profondeur de x_t est $< k+1$, donc $h < k+1$. On entre dans la boucle.

On a $x_t = d_0^t ... d_h^t f_h^t ... f_0^t \to x_{t+1}$ donc d'après la preuve du théorème 176, il existe un indice i entre 0 et h tel que u_k soit un facteur de f_k^t, il existe donc β tel que $f_k^t = u_k \beta$.

i) On suppose que $i \neq 0$. Pour tout $j > i$, d'après la proposition 175 l'ensemble des lettres aux positions utilisables de profondeurs de x_t i.e. exactement les lettres de f_i^t est un intervalle de longueur $< L$. Par définition puisque $(u_k, v_k) \in R$ et que la mémoire contient $[f_h^t, ..., f_1^t]$ on a (i, u_k) qui appartient à Choices.

ii) On suppose que $i = 0$. Alors par la proposition 175, pour tout $i > 0$ on a f_i^t de longueur $< L$, or la mémoire au début de la boucle contient $[f_h^t, ..., f_1^t]$ donc $\max f_i^t < L$, donc $(0, v_k)$ est un élément de Choices.

Nous choisissons l'élément présenté aux point i) et ii), la mémoire contient alors $[f_h^t, ..., f_{i+1}^t v, \beta, f_{i-1}^t, ... f_1^t]$ si $i > 0$ et $[f_h^t, ..., f_2^t, f_1^t v]$ or par définition du pas de réécriture et du théorème 176

a) si $i < h$ on a $f_j^{t+1} = f_j^t$ pour j différent de i et $i+1$, $f_{i+1}^{t+1} = f_{i+1}^t v$ et $f_i^{t+1} = \beta$. Donc à la fin du $t+1$-ième passage dans la boucle while, la mémoire contient $[f_h^{t+1}, ..., f_1^{t+1}]$. Et h n'augmente pas car $i < h$. On a x_{t+1} de même profondeur que x_t donc h.

b) si $i = h$, on considère v_k factorisé en ww' où les positions utilisables sont celles de w', on a $f_j^{t+1} = f_j^t$ pour j différent de h et $h+1$, $f_{h+1}^{t+1} = w'$ et $f_h^{t+1} = \beta$. On choisit alors $v'' = w$ et $v' = w'$. À la fin du $t+1$-ième passage dans la boucle while, la mémoire contient $[f_{h+1}^{t+1}, ..., f_1^{t+1}]$. Comme $i = h$, on a h qui est incrémenté de 1. Or x_{t+1} est de profondeur $+1$ par rapport à x_t, donc $h+1$.

Ainsi la récurrence est démontrée.

Au n-ième passage dans la boucle, la branche d'exécution admet h qui vaut la profondeur de x_n, donc $k+1$ donc il n'y a plus de passage dans la boucle while, et la branche renvoi $h = k+1$, donc vrai.

Troisième point : Montrons que l'algorithme est PSPACE. La mémoire totale contient h, C, Choices et la mémoire $[f_{h+1}, ..., f_1]$ après t passages dans la boucle. Comme C vaut $\frac{L^{k+1}-1}{L-1}$ il est de taille polynomiale, de même Choices est borné en taille par $\#R \times (k+1)$ donc est de taille polynomiale, il suffit de montrer que les mots f_j sont de longueur $< L + M$ ainsi la mémoire totale est polynomiale.

Montrons ce résultat par récurrence sur t.

- Initialisation : après 0 passage dans la boucle, la mémoire est vide.

- Hérédité : Supposons qu'après t passages dans la boucle on a $|f_j| < L + M$ pour

tout j de 1 à h. À la fin du $t+1$ passage dans la boucle la mémoire est de l'une des formes suivantes :

— $[v', f'_h, f_{h-1}, ..., f_1]$ avec $|v'| < M$ car facteur de v et $|f'_h| < L+M$ car facteur de f_h de longueur $< L+M$ par récurrence

— $[f_h, ..., f_{i+1} v, f'_i, f_{i-1}, ... f_1]$ avec $|v| < M$ et $|f'_i| < L+M$ car facteur de f_i de longueur $< L+M$ par récurrence

Ce qui achève la récurrence.

Ainsi la mémoire générale est polynomiale.

4.3.3 Le problème $R \in \mathrm{MB}_{\max}(k)$ est PSPACE-difficile

Le but de cette section est de montrer la partie dureté du théorème 157. Nous utilisons pour cela une réduction polynomiale depuis le problème du vide de l'intersection d'automates :

Décider si l'intersection de N automates non déterministes est vide est PSPACE-complet.

Il est facile de voir que la restriction aux automates qui ne reconnaissent pas le mot vide et qui ont un unique état initial et un unique état final, reste PSPACE-complète, nous ferons cette hypothèse par la suite.

Nous considérerons des lettres de la forme (a, n) avec n entier et noterons a^n au lieu de (a, n).

On étend cette notation aux mots : si $m = a_1 ... a_k$ alors $m^n = a_1^n ... a_k^n$ [5].

On considère N automates non déterministes $(Q_n, \Sigma, \delta_n, i_n, f_n)$ (pour n de 1 à N) tels que tous les ensembles Q_n sont deux à deux disjoints.

Nous réduisons le problème du vide de l'intersection de ces automates vers le problème :

$$R \in \mathrm{MB}_{\max}(2N + 2p + 3)$$

où $p = \lceil \log_2(\prod_{n=1}^{N} |Q_n|) \rceil$

avec le système de réécriture R défini par :

5. Jusqu'à la fin du chapitre aucune confusion avec la puissance d'un mot dans le monoïde $(\Sigma^*, .)$ n'est possible.

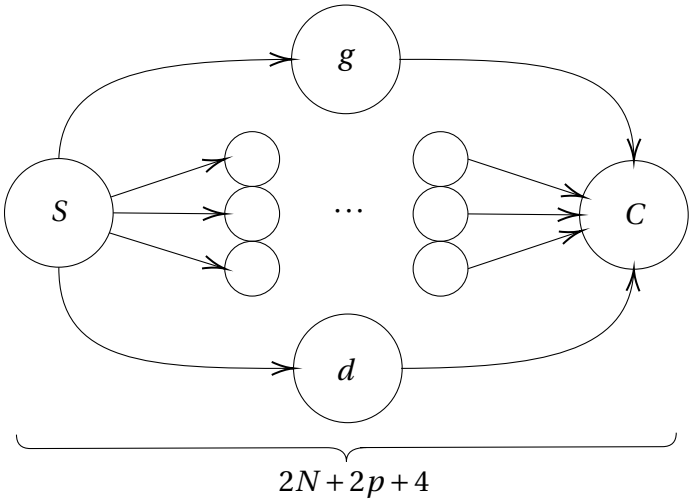


FIGURE 4.2 – Graphe-lettre du système R

$$R = \begin{cases} S \rightarrow g i_N ... i_1 G_0 d \\ G_t \rightarrow G_{t+1} G_{t+1}, \forall t \in [0, p-1] \\ G_t \rightarrow G_{t+1}, \forall t \in [0, p-1] \\ G_p \rightarrow a^0, \forall a \in \Sigma \\ a^i \rightarrow q a_V^i q', \forall (q, a, q') \in \delta_i, \forall i \in [0, N-1] \\ q q a_V^i \rightarrow a^{i+1}, \forall q \in Q_i, \forall i \in [0, N-1] \\ a^N \rightarrow D_p, \forall a \in \Sigma \\ D_{t+1} D_{t+1} \rightarrow D_t, \forall t \in [0, p-1] \\ D_{t+1} \rightarrow D_t, \forall t \in [0, p-1] \\ g D_0 f_N f_{N-1} ... f_1 d \rightarrow C \end{cases} \tag{4.2}$$

Premièrement, on peut remarquer que l'intersection des N automates est non vide si et seulement si il y a un mot non vide de longueur au plus$\prod_{n=0}^{N} |Q_i|$ donc au plus 2^p reconnu par chacun des automates.

— Les 4 premières lignes du système R ci-dessus permettent de générer tous les mots de longueur $\leq 2^p$
— Les 2 lignes suivantes permettent de vérifier l'existence d'un run du mot engendré dans chaque automate à partir de l'état initial
— Les dernières lignes permettent de vérifier que chaque état final est bien atteint

Lemme 177

Pour toute dérivation σ sur R telle que orig(σ) = S et aim(σ) = C la complexité parallèle de σ est $2N+2p+4$.

Démonstration. On considère une telle dérivation.

Soit f définie par $f(S)=0, f(G_t)=t+1, f(a_i)=p+2+2i, f(a_V^i)=f(a^i)+1, f(D_k)=p+3+2N+p-k, f(C)=2p+2N+4, f(g)=f(d)=1,\ f(q)=p+3+2i$ pour les états q de l'automate A_i.

Il est facile de vérifier que pour chaque règle il y a une lettre dans le membre droit dont la valeur est exactement 1+ le maximum des valeurs des lettres du membre gauche.

Par récurrence, il est facile de vérifier que, pour toute lettre x qui n'est pas associée à un état, dans une dérivation d'origine S, $f(x)$ correspond à la profondeur de l'index de la lettre ; donc, pour une dérivation telle que orig(σ) = S et aim(σ) = C, la complexité parallèle vaut $f(C)$ donc $2p+2N+4$.

□

On prouve dans le lemme suivant que tout mot de longueur $\leq 2^p$ peut être engendré par le système :

Lemme 178

Pour tout mot m non vide de longueur $\leq 2^p$ il existe une dérivation de la forme $S \longrightarrow^* g i_N \cdots i_0 m^0 d$.

Démonstration. Soit H_i défini par :

Pour tout mot m non vide de longueur majorée par 2^i il existe une dérivation de la forme :

$$G_{p-i} \longrightarrow^* m^0$$

Initialisation : Pour toute lettre a de Σ on a $G_p \to a^0$ donc H_0 est vraie.

Hérédité : Supposons H_i pour un i fixé. Soit m un mot non vide de longueur majorée par 2^{i+1}.

Si m est de longueur majorée par 2^i, d'après H_i on a une dérivation de la forme $G_{p-i} \longrightarrow_R^* m^0$, or $G_{p-i-1} \to G_{p-i}$ est une règle de R par composition on obtient H_{i+1}.

Sinon, m peut être décomposé comme $m_1 m_2$ avec les longueurs de m_1 et m_2 majorées par 2^i et non nulles. D'après H_i il existe deux dérivations de la forme : $G_{p-i} \longrightarrow^* m_1^0$ et $G_{p-i} \longrightarrow^* m_2^0$, donc $G_{p-i}G_{p-i} \longrightarrow^* m_1^0 m_2^0$. Or $G_{p-i-1} \to G_{p-i}G_{p-i}$ est une règle du système. Par composition H_{i+1} est prouvée.

Enfin comme $S \longrightarrow g i_N \cdots i_0 G_0 d$ est une règle du système, par composition on a prouvé le lemme.

□

De même, nous avons :

Lemme 179

Pour tout mot non vide m tel que $|m| \leq 2^p$ il existe une dérivation de la forme :

$$g m^N f_N \cdots f_0 \longrightarrow_R^* C$$

Le prochain lemme démontre que le système "simule" les runs de l'automate :

Lemme 180

Si A_n reconnait le mot m alors il existe une dérivation de la forme :

$$i_n m^n \longrightarrow_R^* m^{n+1} f_n$$

Démonstration. Soit $m = a_1 \cdots a_k$ un mot.

Comme m est reconnu par A_n il existe un run $q_0 a_1 q_1 ... q_{k-1} a_k q_k$ de A_n tel que $q_0 = i_n$ (l'état initial de A_n) et $q_k = f_n$ (l'état final de A_n).

Pour tout t, la règle de réécriture $a^n \longrightarrow q_{t-1}(a_t^n)_V q_t$ appartient à R donc :

$$i_n m^n \longrightarrow_R^* i_n q_0 (a_1^n)_V q_1 q_1 (a_2^n)_V ... q_{k-1} q_{k-1} (a_k^n)_V q_k$$

De même pour tout t, la règle $q_{t-1} q_{t-1} (a_t^n)_V \longrightarrow a^{n+1}$ appartient à R (on a $q_0 = i_n$) :

$$i_n q_0 (a_1^n)_V q_1 q_1 (a_2^n)_V ... q_{k-1} q_{k-1} (a_k^n)_V q_k \longrightarrow_R^* a_1^{n+1} \cdots a_k^{n+1} q_k$$

Finalement, comme $q_k = f_n$ on a

$$i_n m^n \longrightarrow_R^* m^{n+1} f_n$$

□

Ainsi on obtient :

Proposition 181

Si $\bigcap_{n=1}^{N} L(A_n)$ est non vide alors $R \notin \mathrm{MB}_{\max}(2N+2p+3)$.

Démonstration. D'après les lemmes ci-dessus, s'il existe un mot non vide m de longueur au plus 2^p reconnu par les N automates alors il existe une dérivation de la forme $S \rightarrow^* C$. La complexité parallèle d'une telle dérivation est $2N+2p+4$ donc $R \notin \mathrm{MB}_{\max}(2N+2P+3)$.

□

Proposition 182

Si $R \notin \mathrm{MB}_{\max}(2N+2p+3)$ alors $\bigcap_{n=1}^{N} L(A_n)$ est non vide.

Démonstration. Supposons que R ne soit pas dans $\mathrm{MB}_{\max}(2N+2P+3)$. Il existe donc une dérivation de complexité parallèle $2N+2P+4$, on peut supposer cette dérivation comme étant économe et conique.

Considérons le graphe des lettres de R : les seuls chemins de longueur $2N+2P+4$ sont des chemins d'origine S et C. Or, un chemin dans le graphe de dérivation correspond à un chemin dans le graphe de lettres. Comme il y a unique règle de R dont le membre gauche est S et une unique règle dont le membre droit est C, la dérivation est de la forme $uSv \longrightarrow ugi_N...i_0G_0dv \longrightarrow^l \alpha gD_0f_Nf_{N-1}...f_1d\beta \longrightarrow \alpha C\beta$ avec un chemin de S à C dans le graphe de dérivation. De plus, on peut remarquer que la règle $gD_0f_Nf_{N-1}...f_1d \rightarrow C$ est la seule règle produisant C et également la seule règle avec g ou d en membre gauche et que C n'apparaît dans aucun membre gauche du système. Donc, la dérivation (qu'on a choisie conique et économe) est de la forme $S \longrightarrow gi_N...i_0G_0d \longrightarrow^l gD_0f_Nf_{N-1}...f_1d \longrightarrow C$ avec la sous-dérivation $gi_N...i_0G_0d \longrightarrow^l gD_0f_Nf_{N-1}...f_1d$ de complexité parallèle égale à $2N+2p+2$.

On réutilise la fonction f définie par $f(S)=0, f(G_t)=t+1, f(a_i)=p+2+2i, f(a_V^i)=f(a_i)+1, f(D_k)=p+3+2N+p-k, f(C)=2p+2N+4, f(g)=f(d)=1, f(q)=p+3+2i$ pour les états q de l'automate A_i.

Nous pouvons remarquer que dans toute dérivation de la forme $S \longrightarrow_R^* C$ on a :

— Pour tout i, la lettre correspondante à l'état initial de A_i peut seulement être produite à la profondeur 1 et être consommée à la profondeur $p+4+2i$

— Pour tout i, la lettre correspondante à l'état final de A_i peut seulement être produite à la profondeur $p+3+2i$ et être consommée à la profondeur $p+4+2i$.

— g et d peuvent seulement être produite à la profondeur 1 et consommée à la profondeur $p+4+2i$.
— Toute autre lettre x est produite à la profondeur $f(x)$ et consommée à la profondeur $f(x)+1$.

La dérivation $S \longrightarrow g i_N ... i_1 G_0 d \longrightarrow^* g D_0 f_N f_{N-1} ... f_1 d$ est équivalente à une dérivation constituée de $2N+2p+3$ pas parallèles. Notons u_k le mot obtenu après k étapes parallèles dans cette dérivation. Comme le but de la dérivation est $g D_0 f_N f_{N-1} ... f_1 d$, d'après ce qui précède, il existe un mot non vide m de Σ^* de longueur au plus 2^p tel que :

— $u_{p+2} = g i_N ... i_1 m^0 d$
— $u_{p+2+2k} = g i_N ... i_{k+1} m^k q_k ... q_1 d$ qui assure l'existe d'un run de i_k à q_k dans A_k étiqueté par m .
— $u_{2p+3+2N} = g D_p^{|m|} q_N ... q_1 d$

De plus, comme $g D_p^{|m|} q_N ... q_1 d \longrightarrow C$, nous avons $q_i = f_i$ pour tout i. Ainsi le run de m est terminant pour tout A_i, donc m appartient à $L(A_i)$ pour tout i.

□

Ainsi, le système $R \in \mathrm{MB}_{\max}(2N+2p+3)$ si et seulement l'intersection des N langages reconnus par les automates A_i est vide. Le nombre de règles de R vaut $4p$ plus le double du nombre de règles des automates +4. Toutes les règles exceptées la première et la dernière, sont de taille au plus 4. Les première et dernière sont de taille au plus $N+4$. Ainsi la taille de R est bornée linéairement par la taille des automates.

Nous avions déjà montré que le problème était PSPACE, nous venons de montrer qu'il est PSPACE-difficile, ainsi :

Théorème 183

On obtient :

PROBLEME : Problème d'appartenance à $\mathrm{MB}_{\max}(k)$
INSTANCE : R un srs, k un naturel
QUESTION : $R \in \mathrm{MB}_{\max}(k)$?
COMPLEXITE : PSPACE-complet

4.4 Conclusion

Dans ce chapitre nous avons mis en avant la structure des dérivations. Nous avons proposé une représentation graphique des interactions entre les pas de réécriture d'une dérivation pour étudier les possibilités de parallélisation des pas de réécriture.

Nous avons au chapitre précédent montré qu'une étape de complétion d'automate correspondait au calcul à un pas parallèle des ancêtres. Nous avons défini la classe des systèmes de réécritures où la complexité parallèle est uniformément bornée pour toutes les dérivations : $\mathrm{MB}_{\max}$. Nous avons proposé de pouvoir simuler toute dérivation entre deux mots u et v par une dérivation de complexité parallèle bornée par une constante : BPar.

Les deux classes sont stables par inversion puis nous avons montré que la relation $\to^*$ d'un système de réécriture BPar est une relation rationnelle et donc préserve les réguliers.

Pour ces deux classes nous montrons que le problème d'inclusion de requêtes RPQ est EXPSPACE-complet.

Finalement, nous avons proposé un algorithme qui décide en PSPACE si un système de réécriture est $\mathrm{MB}_{\max}(k)$ et avons montré que ce problème est PSPACE-difficile.

CHAPITRE 5

Classe de règles tgd bornées

Nous souhaitons interroger des bases de données qui satisfont certaines propriétés exprimées comme contraintes d'intégrité. Malheureusement, en général ces contraintes ne sont pas satisfaites.

La procédure du chase permet de compléter une instance afin de satisfaire des contraintes tgd. Le chase admet plusieurs variantes (on pourra citer notamment l'oblivious, le semi-oblivious, le standard, le core, ...) mais dans ce chapitre nous portons une attention particulière aux variantes oblivious et semi-oblivious et évoquons quelques résultats dans le cas standard.

Un des problèmes majeurs de la procédure du chase est sa terminaison : on souhaite décider si pour toute instance le chase termine i.e. construit une nouvelle instance qui satisfait les contraintes en un nombre fini d'étapes.

De manière générale, ce problème est indécidable [56, 55]. De nombreuses classes assurant la terminaison du chase ont été proposées dont les systèmes weakly-acyclic [40] et de nombreuses extensions. Dans ce chapitre nous portons attention aux systèmes uniformément bornés [35, 34, 17]. Ces classes généralisent les études menées sur les programmes Datalog [43, 30, 74].

Nous proposons dans ce chapitre, de comparer les classes de tgd uniformément bornées en fonction de la variante du chase et d'une stratégie choisie.

Les systèmes uniformément bornés consistent à mesurer "l'empilement" des faits produits et de les borner uniformément par une constante. Cette mesure est à mettre en relation avec l'empilement des transitions lors de la complétion des automates (chapitre 4). La complétion d'automates ne correspond pas exactement à la complétion par l'oblivious-chase.

Chaque étape de complétion d'un automate correspond à un pas parallèle des descendants (ou ancêtres). Nous comparons le fait de limiter uniformément la complexité parallèle avec les contraintes de mots qui sont uniformément bornées au tant que tgd. Nous montrons que sous certaines conditions ces notions sont équivalentes.

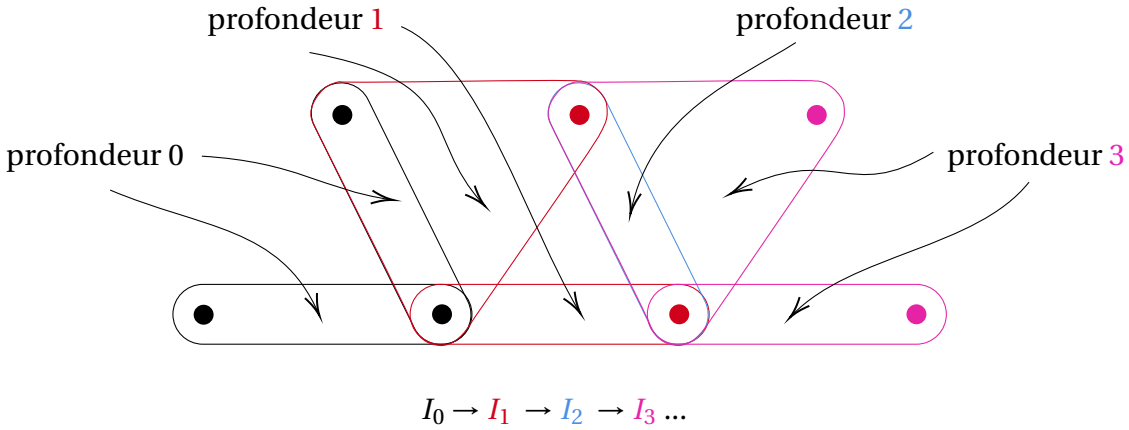
5.1 Systèmes de règles tgd bornés

5.1.1 Résumé

On se fixe une variante $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$.

Nous proposons un résumé des résultats prouvés dans les trois sous-sections de cette section :

Sous-section 5.1.2 : Définitions générales On considère la profondeur [20, 35, 17] d'une X-dérivation comme la profondeur maximale des faits qui la compose (voir définition 191) : à chaque fois qu'un trigger produit des nouveaux faits, la profondeur de ceux-ci est alors 1+ le maximum des faits qui ont permis d'activer le trigger. Les faits de l'instance initiale ont une profondeur nulle.



Nous proposons dans cette section d'étudier le problème de la borne uniforme : Etant donné une variante de chase, une stratégie (breadth-first par exemple), est ce que toute X-dérivation (ou au moins une) sur toute instance est de profondeur $\leq k$?

Plusieurs manières de privilégier certaines X-dérivations à borner sont proposées pour étudier la terminaison ou la profondeur (par exemple les dérivations où les règles Datalog sont prioritairement appliquées [69]).

Ici nous choisissons d'étudier deux cas : celui où toutes les X-dérivations sont bornées et celui de la stratégie breadth-first consistant à appliquer les triggers "étage par étage" : une fois un trigger appliqué, on applique les triggers de même profondeur jusqu'à ce qu'on puisse passer à la profondeur suivante.

Le choix des X-dérivations breadth-first (BF en abrégé) est motivé dans [35, 17] dans le cas oblivious et semi-oblivious ($X \in \{\mathbf{O}, \mathbf{SO}\}$) :

— Confluence : on peut permuter deux triggers orthogonaux sans désactiver un autre trigger de la dérivation (voir proposition 196)

— Profondeur minimale : pour toute X-dérivation terminante de profondeur k, il existe une X-dérivation BF de pronfondeur $\leq k$ (voir proposition 208)

Cette propriété de confluence est fausse dans le cas standard.

Nous proposons de borner les systèmes de quatre manières différentes :

Définition 184

Soit $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$. Soit k un entier. Soit $\mathscr{C}$ un système de règles tgd. Si (respectivement) on a pour toute instance I :

1. toute X-dérivation sur I est de profondeur au plus k
2. toute X-dérivation BF sur I est de profondeur au plus k
3. il existe une X-dérivation sur I de profondeur au plus k
4. il existe une X-dérivation BF sur I de profondeur au plus k

alors le système $\mathscr{C}$ est (respectivement) dit :

1. borné par k et on note $\mathscr{C} \in \beta^k(\text{all}, X)$
2. BF-borné par k et on note $\mathscr{C} \in \beta^k(\text{BF}, X)$
3. faiblement borné par k et on note $\mathscr{C} \in \beta^k(\text{weak}, X)$
4. faiblement BF borné par k et on note $\mathscr{C} \in \beta^k(\text{wBF}, X)$

On dénote par $\beta(\text{all}, X)$ l'union de tous les $\beta^k(\text{all}, X)$ pour k entier naturel. On définit de la même manière $\beta(\text{BF}, X), \beta(\text{weak}, X)$ et $\beta(\text{wBF}, X)$.

Par définition pour toute variante X nous avons les inclusions :

$$\beta^k(\text{all}, X) \subseteq \beta^k(\text{BF}, X) \subseteq \beta^k(\text{wBF}, X) \subseteq \beta^k(\text{weak}, X)$$

Nous montrons que dans le cas oblivious et semi-oblivious être borné (pour l'une des définitions) assure la terminaison. Par contre, si un système tgd termine alors il n'est pas toujours borné (exemple 203).

Dans le cas standard, la propriété d'être uniformément bornée n'assure pas la terminaison (exemple 201).

Nous montrons que les inclusions $\beta^k(\text{all}, X) \subseteq \beta^k(\text{BF}, X) \subseteq \beta^k(\text{wBF}, X) \subseteq \beta^k(\text{weak}, X)$ sont strictes.

Sous-section 5.1.3 : Comparaison des classes dans le cas oblivious et semi-oblivious Nous comparons les classes obtenues dans la sous-section précédentes entre elles i.e. $\beta^k(\text{all}, X), \beta^k(\text{BF}, X), \beta^k(\text{wBF}, X)$ et $\beta^k(\text{weak}, X)$.

Premièrement, nous montrons que toutes les X-dérivations BF en oblivious et semi-oblivious ont la même profondeur.

Ce qui permet d'affirmer que :

Théorème 185

Soient $X \in \{\mathbf{O}, \mathbf{SO}\}$ et k un entier,

$$\beta^k(\text{BF}, X) = \beta^k(\text{wBF}, X)$$

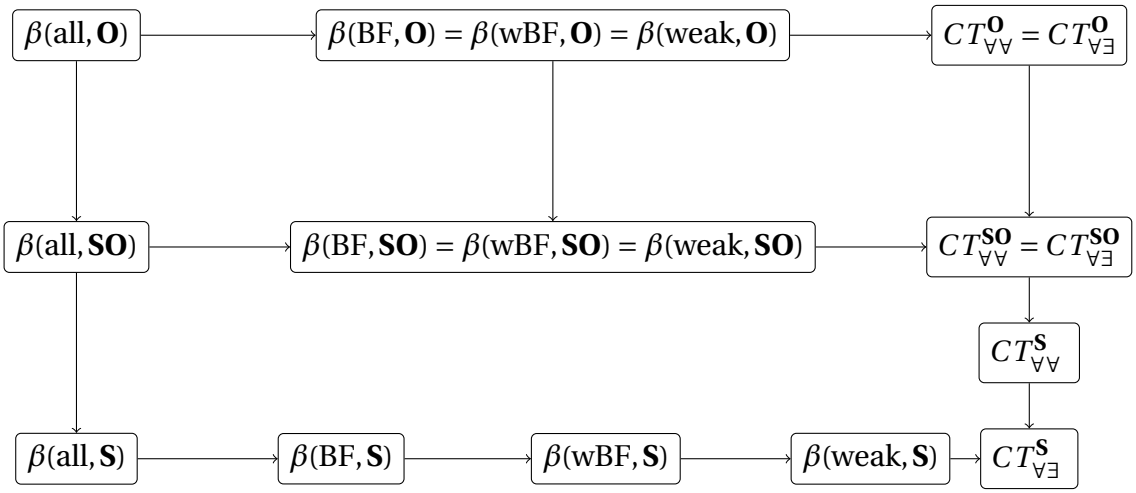
De plus nous montrons :

Théorème 186

Soient $X \in \{\mathbf{O}, \mathbf{SO}\}$ et k un entier,

$$\beta^k(\text{wBF}, X) = \beta^k(\text{weak}, X)$$

Ainsi, nous pouvons résumer les inclusions par le schéma [1] :



Nous montrons ensuite que les inclusions sont strictes.

Les systèmes tgd uniformément bornés en oblivious et semi-oblivious assurent la terminaison, nous montrons que les $\beta^k(\text{all}, X)$,... ne sont pas comparables avec la classe des weakly-acyclic [40].

1. $CT^X_{\forall\forall}$ désigne la classe des systèmes qui terminent sur toute instance

Sous-section 5.1.4 : Problèmes de décision Nous avons montré que les classes $\beta^k(\text{all}, X)$ et $\beta^k(\text{BF}, X)$ assurent la terminaison en oblivious et semi-oblivious (les autres classes $\beta^k(\text{wBF}, X)$ et $\beta^k(\text{weak}, X)$ sont égales à $\beta^k(\text{BF}, X)$). Il est intéressant de pouvoir décider si un système est dans la classe $\beta^k(\text{all}, X)$ ou la classe $\beta^k(\text{BF}, X)$

Nous explicitons les résultats de [35, 17] pour prouver que pour $X \in \{\mathbf{O}, \mathbf{SO}\}$ nous avons les deux résultats de complexités suivants :

PROBLEME : Borne uniforme tgd
INSTANCE : $\mathscr{C}$ des règles tgd, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{all}, X)$?
COMPLEXITE : Co-NEXPTIME

PROBLEME : Borne uniforme tgd dans le cas BF
INSTANCE : $\mathscr{C}$ des règles tgd, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{BF}, X)$?
COMPLEXITE : 2EXPTIME

Enfin, dans le cas Datalog, nous montrons que pour tout X dans $\{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$:

PROBLEME : Borne uniforme Datalog dans le cas BF
INSTANCE : $\mathscr{C}$ un problème Datalog, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{BF}, X)$?
COMPLEXITE : Co-NEXPTIME-complet

Ce qui permet de montrer que les problèmes de décision $\mathscr{C} \in \beta^k(\text{all}, X)$? et $\mathscr{C} \in \beta^k(\text{BF}, X)$? sont Co-NEXPTIME-difficiles.

Donc Co-NEXPTIME-complet pour $\mathscr{C} \in \beta^k(\text{all}, X)$?.

5.1.2 Définitions générales

Dans tout ce chapitre est fixé un schéma relationnel S, un ensemble dénombrable de variables **Var** et un ensemble dénombrable de constantes **Const**.

On pourra se référer au chapitre 1 pour des rappels techniques sur les notions de triggers, de tgd et les variantes de chase.

On rappelle que pour toute règle $r : B \to H$, B désigne le corps de r et H sa tête.

Premièrement, nous avons la proposition évidente suivante :

Proposition 187

Si deux instances I et I' sont isomorphes par φ alors les triggers actifs de I' sont les $(r, \varphi \circ h)$ où (r, h) est un trigger actif de I.

Quand un trigger est activé sur une instance alors d'autres peuvent devenir désactivés : autrement dit un trigger actif sur I ne l'est pas forcément sur $I \cup h'(H)$ que nous noterons parfois $I + (r,h)$.

Définition 188

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soit I une instance. On considère l'ensemble Actif(I) des triggers actifs sur I.
On dit que deux triggers actifs sont équivalents si l'un désactive l'autre, i.e $(r,h) \sim (R,H)$ ssi (R,H) n'appartient pas à Actif($I + (r,h)$). C'est une relation d'équivalence.
L'ensemble des (R,H) tels que $(r,h) \sim (R,H)$ est noté $[(r,h)]_I$.
On étend cette notation aux non actifs : si (r,h) est non actif alors $[(r,h)]_I = \emptyset$.

Aucun trigger ne peut être désactivé pour l'oblivious chase on a $[(r,h)]_I = \{(r,h)\}$ pour tout trigger.

De même, un trigger désactivera un autre s'ils ont les mêmes règles et mêmes frontières dans le semi-oblivious chase donc $(R,H) \in [(r,h)]_I$ ssi $R = r$ et $h(\partial r) = H(\partial R)$ pour tout trigger.

Nous associons aux faits et aux nœuds d'une instance des entiers qui représentent la profondeur.

Dans une X-dérivation, les faits et nœuds sont indicés par 0 s'ils appartiennent à l'instance de départ. Puis, on affecte à tout fait créé 1+ la plus grande profondeur des faits qui ont permit de le produire.

Les faits et les nœuds des instances sont donc affectées par des entiers :

Définition 189

Une instance indicée est un triplet $(I, \text{erank}_I, \text{vrank}_I)$ où I est une instance, erank_I est une fonction des faits de I vers $\mathbb{N}$ et vrank_I est une fonction des nœuds de I vers $\mathbb{N}$.

Nous étendons les isomorphismes d'instances aux instances indicées :

Définition 190

Soient $(I, \text{erank}_I, \text{vrank}_I)$ et $(J, \text{erank}_J, \text{vrank}_J)$ deux instances indicées. Un isomorphisme entre ces deux instances indicées est un isomorphisme φ entre I et J tel que :

— pour tout fait f de I, $\text{erank}_J(\varphi(f)) = \text{erank}_I(f)$

— pour tout nœud n de I, $\text{erank}_J(\varphi(n)) = \text{erank}_I(n)$

Nous définissons la profondeur des faits dans une dérivation :

Définition 191

Soit $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$, une X-dérivation indicée est un quadruplet $(s, \text{erank}_s, \text{vrank}_s, \text{rank}_s)$ où s est une X-dérivation $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ et les fonctions erank_s, vrank_s et rank_s (appelées profondeur) sont définies par induction par :

I. Pour tout fait f et toute valeur n de I_0 on pose $\text{erank}_s(f) = 0$ et $\text{vrank}_s(n) = 0$.

II. Pour tout $i > 0$, soit $k = 1 + \max_{f \in h_i(B_i)} \text{erank}_s(f)$ où B_i est la tête de r_i, on pose :

i) pour tout fait f de $I_{i+1} \setminus I_i$, $\text{erank}_s(f) = k$

ii) pour toute valeur n de $I_{i+1} \setminus I_i$, $\text{vrank}_s(n) = k$

iii) $\text{rank}_s(r_i, h_i) = k$

Afin de ne pas alourdir les notations nous parlerons simplement de X-dérivation au lieu de X-dérivation indicée. Nous écrirons s au lieu du quadruplet $(s, \text{erank}_s, \text{vrank}_s, \text{rank}_s)$. De même pour les instances indicées nous omettrons les fonctions erank_I et vrank_I.

Dans une X-dérivation indicée, toutes les instances I_i sont indicées.

Nous introduisons maintenant la profondeur d'une X-dérivation comme le fait le plus profond de la X-dérivation :

Définition 192

Soient $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$, $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une X-dérivation.
La profondeur de s est la plus grande valeur de erank_s sur $(\cup_i I_i)$.

Une X-dérivation BF est une X-dérivation telle que les triggers sont de profondeurs croissantes et on ne peut pas appliquer un trigger actif de profondeur k s'il existe un trigger actif de profondeur $< k$.

Définition 193

Une X-dérivation $s : I_0 \overset{(r_0,h_0)}{\longrightarrow} I_1 \ldots$ est BF (breadth-first) si elle vérifie les deux conditions suivantes :

i) les profondeurs des triggers sont dans l'ordre croissant des indices i.e. si $i > j$ alors $\text{rank}_s(r_i, h_i) \geq \text{rank}_s(r_j, h_j)$

ii) pour tout i tel que $\text{rank}_s(r_{i+1}, h_{i+1}) = \text{rank}_s(r_i, h_i)+1$, alors pour tout trigger actif (r, h) sur I_i il existe $j > i$ tel que $(r_j, h_j) = (r, h)$ et $\text{rank}_s(r_j, h_j) > \text{rank}_s(r_i, h_i)$

Nous pouvons alors définir les quatre classes de systèmes tgd suivants (voir définition 184) :

— $\mathscr{C} \in \beta^k(\text{all}, X)$ ssi toute X-dérivation est de profondeur $\leq k$.

— $\mathscr{C} \in \beta^k(\text{BF}, X)$ ssi toute X-dérivation BF est de profondeur $\leq k$.

— $\mathscr{C} \in \beta^k(\text{wBF}, X)$ ssi pour toute instance, il existe une X-dérivation BF sur I de profondeur $\leq k$.

— $\mathscr{C} \in \beta^k(\text{weak}, X)$ ssi pour toute instance, il existe une X-dérivation sur I de profondeur $\leq k$.

On définit alors les classes $\beta(\text{all}, X)$, $\beta(\text{BF}, X)$, $\beta(\text{wBF}, X)$ et $\beta(\text{weak}, X)$ comme les unions des classes ci-dessus.

Nous illustrons un exemple de X-dérivation qui n'est pas BF et un exemple de X-dérivation qui est BF :

Exemple 194

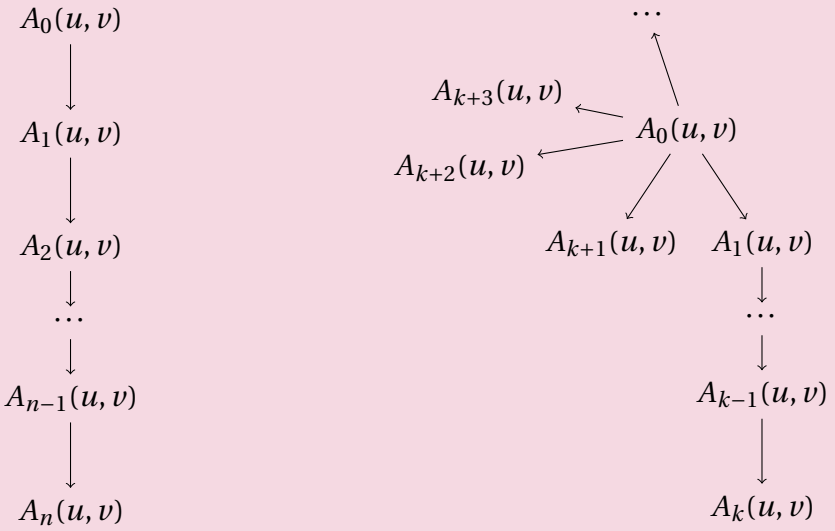
Soient k et n deux entiers tels que $1 \leq k < n$, on définit le système $\mathscr{C}(k,n)$ par :

— $A_i(x,y) \to A_{i+1}(x,y)$ pour i de 0 à n

— $A_0(x,y) \to A_i(x,y)$ pour i de $k+1$ à n

Sur la partie gauche la dérivation n'est pas BF, même si les profondeurs sont dans l'ordre croissant : il existe une règle de profondeur 1 qui n'a pas été appliquée.

La partie droite correspond à une dérivation BF sur la même instance.



De façon analogue aux systèmes de réécriture dans le chapitre précédent nous trions par profondeur les triggers en échangeant des triggers consécutifs orthogonaux i.e. les faits produits par le premier ne sont pas utilisés par le second. Ce tri permet d'appliquer en une étape parallèle l'ensemble de tous les triggers orthogonaux. En particulier, les triggers de même profondeur sont orthogonaux.

Définition 195

X désigne **O** ou **SO**. Soient $\mathscr{C}$ un ensemble de règles tgd, et une X-dérivation $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ... I_i \xrightarrow{(r_i,h_i)} I_{i+1} \xrightarrow{(r_{i+1},h_{i+1})} I_{i+2} ...$, deux triggers consécutifs (r_i,h_i) et (r_{i+1},h_{i+1}) sont orthogonaux ssi $(I_{i+1} \setminus I_i) \cap h_{i+1}(B_{i+1}) = \emptyset$

Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$ une X-dérivation. On dira aussi que s est orthogonale en i si les triggers (r_i,h_i) et (r_{i+1},h_{i+1}) sont orthogonaux.

Proposition 196

Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$ une X-dérivation. Si s est orthogonale en i alors les triggers (r_i, h_i) et (r_{i+1}, h_{i+1}) peuvent être échangés. On note $\text{exch}_{i,i+1}(s)$ la X-dérivation obtenue.

Démonstration. Il suffit de montrer que le trigger (r_{i+1}, h_{i+1}) est actif sur I_i, que le trigger (r_i, h_i) est actif sur $I_i \cup h'_{i+1}(H_{i+1})$ et que l'instance obtenue après avoir appliqué les deux triggers échangés est bien I_{i+2}.

- Première étape ((r_{i+1}, h_{i+1}) est actif sur I_i) : supposons que $X = \mathbf{SO}$, comme ((r_{i+1}, h_{i+1}) est actif sur I_{i+1} il n'existe pas de triggers (r_j, h_j) avec $j \leq i$ tel que $r_j = r_i$ et $h_j(\partial r_j) = h_{i+1}(\partial r_{i+1})$. Donc c'est vrai pour $j < i$. Pour tout X de $\{\mathbf{O}, \mathbf{SO}\}$, puisque (r_{i+1}, h_{i+1}) est actif sur I_{i+1}, $h_{i+1}(B_{i+1}) \subseteq I_{i+1}$. Comme (r_i, h_i) et (r_{i+1}, h_{i+1}) sont orthogonaux, par définition, $(I_{i+1} \setminus I_i) \cap h_{i+1}(B_{i+1}) = \emptyset$. Donc, $h_{i+1}(B_{i+1}) \subseteq I_i$. Donc (r_{i+1}, h_{i+1}) est actif sur I_i.

- Seconde étape ((r_i, h_i) est actif sur $I'_{i+1} = I_i \cup h'_{i+1}(H_{i+1})$) : On a $h_i(B_i) \subseteq I_i \subseteq I'_{i+1}$ et dans le cas semi-oblivious les triggers (r_j, h_j) pour $j < i$ n'ont pas le même couple (frontière,règle) que (r_i, h_i) car (r_i, h_i) est actif sur I_i, donc actif sur I'_{i+1}.

Troisième étape : on remarque qu'après échange on a bien $I_{i+2} = I'_{i+1} \cup h'_i(H_i)$. □

Si la profondeur du second trigger est inférieure ou égale à la profondeur du premier alors les triggers sont orthogonaux (nous pouvons donc échanger les triggers pour les mettre en profondeur croissante) :

Lemme 197

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soit $\mathscr{C}$ un ensemble de règles tgd.
Soit I une instance indicée, on considère la X-dérivation :

$$s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$$

Si $\text{rank}(r_i, h_i) \geq \text{rank}(r_{i+1}, h_{i+1})$ alors s est orthogonale en i.

Démonstration. On pose $a = \text{rank}(r_i, h_i)$ et $b = \text{rank}(r_{i+1}, h_{i+1})$, on a $a \geq b$.

Montrons que :

$$(I_{i+1} \setminus I_i) \cap h_{i+1}(B_{i+1}) = \emptyset$$

Soit f un fait de $(I_{i+1} \setminus I_i) \cap h_{i+1}(B_{i+1})$, comme $f \in I_{i+1} \setminus I_i$, on a par définition $\text{erank}_s(f) = \text{rank}_s(r_i, h_i) = a$.

Or $f \in h_{i+1}(B_{i+1})$ donc $\text{rank}_s(r_{i+1}, h_{i+1}) \geq 1 + \text{erank}_s(f) = a + 1 > b$ or $\text{rank}(r_{i+1}, h_{i+1}) = b$ on a $b > b$.

Contradiction. □

Nous proposons d'appliquer par induction tous les triggers de profondeur i pour obtenir une instance $\mathscr{C}^i(I)$.

Les instances $\mathscr{C}^i(I)$ sont unique à isomorphisme près (la construction dépend de l'ordre dans lequel les triggers de même profondeur ont été appliqués). Nous montrons que cet isomorphisme est un isomorphisme d'instances indicées.

Lemme 198

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soit $\mathscr{C}$ un ensemble de règles tgd.
Soient I et I' deux instances indicées isomorphes, soit φ l'isomorphisme (d'instances indicées) entre I et I'. On considère :

$$I \xrightarrow{(r,h)} I_f$$

Alors pour tout $(R, g)) \in [(r, h)]_I$ on a :

$$I' \xrightarrow{(R,\varphi\circ g)} I'_f$$

Et, I'_f et I_f sont des instances indicées isomorphes.

Démonstration. Si $(R, g) \in [(r, h)]_I$ alors $r = R$ et on obtient :

$$I \xrightarrow{(R,g)} I_f$$

De même comme I et I' sont isomorphes alors on obtient :

$$I' \xrightarrow{(R,\varphi\circ g)} I'_f$$

On pose pour toute valeur a de I_f :

— $\psi(a) = \varphi(a)$ si $a \in I$

— $\psi(a) = g'(a)$ si $a \notin I$

Soit f' un fait de I'_f, on note $f = \psi^{-1}(f')$.

- Si f' est dans I' alors $\text{erank}(f') = \text{erank}(f)$ car $\psi(f) = \varphi(f)$.

- Si f' est un fait de $I'_f \setminus I'$, f' est de la forme $A(\psi(x_1), ..., \psi(x_n))$ avec un $\psi(x_i)$ dans $I'_f \setminus I'$ donc $\text{erank}(f') = \text{vrank}(\psi(x_i)) = \text{rank}(r', \varphi \circ g) = \text{rank}(r, h)$ donc égal à $\text{erank}(f)$.

Ainsi ψ conserve les profondeurs, donc ψ est un isomorphisme d'instances indicées entre I_f et I'_f.

□

Ainsi nous pouvons construire à isomorphisme d'instances indicées près l'instance $\mathscr{C}^k(I)$:

Définition 199

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$.
Soient $\mathscr{C}$ un système de règles tgd, et I une instance. On construit par récurrence :

- Soit $I_0 = I$
- S'il existe un trigger (r,h) de profondeur 1 sur I_i on pose $I_{i+1} = I_i \cup h'(H)$

Puisque tous les triggers sont orthogonaux (car de même profondeurs) les instances obtenues ne dépendent pas de l'ordre d'application des triggers.
Soit n tel qu'il n'existe pas de trigger (r,h) de profondeur 1 sur I_n. On note alors $\mathscr{C}(I)$ au lieu I_n, cette instance est définie de manière unique à isomorphisme d'instance indicée près (proposition 198).
On pose alors $\mathscr{C}^0(I) = I$ puis $\mathscr{C}^{k+1}(I) = \mathscr{C}(\mathscr{C}^k(I))$.

Pour toute X-dérivation BF

$$s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$$

Pour tout k inférieur à la profondeur de s, il existe un indice i tel que $I_i = \mathscr{C}^k(I_0)$.

On pourra remarquer qu'un fait f est de profondeur k dans s si et seulement si $f \in \mathscr{C}^k(I_0) \setminus \mathscr{C}^{k-1}(I_0)$. Pour toute instance I, toutes les X-dérivations BF sur I ont même profondeur, c'est l'entier k tel que $\mathscr{C}^{k+1}(I) = \mathscr{C}^k(I)$.

S'il existe un entier k tel que $\mathscr{C}^k(I) = \mathscr{C}^{k+1}(I)$ alors la X-dérivation BF termine et $\mathscr{C}^k(I) = \mathscr{C}^\infty(I)$ (voir définition 72)

Nous montrons que les systèmes tgd uniformément bornés dans les variantes oblivious et semi-oblivious assurent la terminaison :

Proposition 200

Pour tout $X \in \{\mathbf{O}, \mathbf{SO}\}$ et tout entier k on a :

$$\beta^k(\text{all}, X) \subseteq \beta^k(\text{BF}, X) \subseteq \beta^k(\text{wBF}, X) \subseteq \beta^k(\text{weak}, X) \subseteq CT^X_{\forall\forall}$$

Démonstration. Les trois premières inclusions évidente, montrons la dernière.

Soit $\mathscr{C} \in \beta^k(\text{weak}, X)$.

Il est connu que $CT^X_{\forall\exists} = CT^X_{\forall\forall}$ pour $X = \mathbf{O}$ ou $X = \mathbf{SO}$ [56, 55], il suffit donc de montrer que pour toute instance il existe une X-dérivation qui termine.

Soit I une instance, comme $\mathscr{C} \in \beta^k(\text{weak}, X)$ il existe une X-dérivation sur I de profondeur au plus k, donc terminante. Ainsi $\mathscr{C} \in CT^X_{\forall\forall}$.

Les autres inclusions sont évidentes par définition.

□

Dans le cas **S**, la propriété d'être uniformément borné n'assure pas la terminaison :

Exemple 201

Soit $\mathscr{C} = \{R(x,y) \rightarrow R(x,x); R(x,y) \rightarrow R(y,y); R(x,y) \rightarrow T(x,y); T(x,y) \rightarrow R(y,z)\}$ alors $\mathscr{C}$ n'est pas terminant sur l'instance $I = R(a,b)$ (donc $\mathscr{C} \notin CT^{\mathbf{S}}_{\forall\forall}$) et $\mathscr{C} \in \beta^3(\text{BF}, \mathbf{S})$.

Démonstration. On présente un sketch de preuve.

On note les règles :

$1 : R(x,y) -> R(x,x)$

$2 : R(x,y) -> R(y,y)$

$3 : R(x,y) -> T(x,y)$

$4 : T(x,y) -> R(y,z)$

1) Il existe une S-dérivation infinie :

$R(a,b)$ par la règle 3 donne $T(a,b)$

$T(a,b)$ donne $R(b, NEW1)$

Puis on recommence sur $R(b, NEW1)$

cela donne $R(NEW1, NEW2)$... etc.

2) Les dérivations BF sont de profondeur 3 :

Soit I une instance.

On est en BF.

On note les profondeurs en indice des prédicats :

- Profondeur 1 :

On a (éventuellement) des $R_1(x,x)$ et $R_1(y,y)$ si on a $R_0(x,y)$ (règles 2 et 3)

On a $T_1(x,y)$ si on a $R_0(x,y)$ (règle 3)

On a $R_1(y,z)$ si on a $T_0(x,y)$ (règle 4)

Profondeur 2 : On a $R_2(x,x)$ et $R_2(y,y)$ si on a $R_1(x,y)$ donc si on a $T_0(x,y)$ (règles 2 et 3)

On a $T_2(x,y)$ si on a $R_1(x,y)$ (règle 3)

RIEN par la règle 4 (si on a $T_1(x,y)$ alors on a $R_0(x,y)$ donc on ne créé pas de $R_2(x,y)$)

Profondeur 3 :

Rien pour les règles 1 et 2 car tous les $R_2(x,y)$ vérifient $x = y$.

Par la règle 3 on produit $T_3(x,x)$ si on a $R_2(x,x)$

Rien pour la règle 4 car si $T_2(x,y)$ alors $R_1(x,y)$, et donc $R_2(y,y)$ (donc on construit pas $R_3(y,\star)$).

Profondeur 4 :

Pas de R produits à la profondeur 3, donc pas de règles 1 2 ou 3 applicable.

Et la règle 4 ne produit rien non plus car si on a $T_3(x,y)$ alors $x = y$ et $R_2(x,x)$.

□

Nous proposons des systèmes qui rendent strictes les inclusions $\beta(\text{all}, X) \subseteq \beta(\text{BF}, X)$ et $\beta(\text{wBF}, X) \subseteq CT^X_{\forall\forall}$. On aussi les inclusions $\beta(\text{all}, \mathbf{O}) \subseteq \beta(\text{all}, \mathbf{SO}) \subseteq \beta(\text{all}, \mathbf{S})$

Pour la première inclusion :

Exemple 202

Soit $\mathscr{C}$ défini par :

$$\{A(x,y), A(y,z) \to A(x,z); A(x,y), A(u,z) \to A(x,z)\}$$

On a $\mathscr{C} \in \beta^1(\text{BF}, \mathbf{O})$ et $\mathscr{C} \notin \beta(\text{all}, \mathbf{S})$, donc pour tout $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$ on a l'inclusion $\beta^k(\text{all}, X) \subseteq \beta^k(\text{BF}, X)$ stricte.

Démonstration. On découpe $\mathscr{C}$ en deux sous-ensembles de règles :

$$P = \{A(x,y), A(y,z) \to A(x,z)\}$$

$$Q = \{A(x,y), A(u,z) \to A(x,z)\}$$

Pour toute instance I on a $P(I) \subseteq Q(I)$.

Q construit le produit cartésien (tout couple de nœuds (x,y) vérifie : soit un des nœuds est isolé, soit $A(x,y) \in Q(I)$) donc $Q^2(I) = Q(I)$.

On en déduit que :

$$\mathscr{C}^2(I) = (P \cup Q)^2(I) \subseteq (P \cup Q)(I) = R(I) \subseteq R^2(I)$$

On a donc $\mathscr{C} \in \beta^1(\text{BF}, \mathbf{O})$.

Pour tout n, soit $I_n = \{A(a_1, a_2), ..., A(a_n, a_{n+1})\}$, en appliquant successivement les triggers (P, h_i) avec h_i tel que :

$$h_i(x) = a_1, h_i(y) = a_i, h_i(z) = a_{i+1}$$

Alors le fait $A(a_1, a_{i+1}) \in I_{i+1}$ pour $i = 0, ..., n$. De plus, le fait $A(a_1, a_{i+1})$ est de profondeur $i+1$. Ainsi pour tout k, il existe une **S**-dérivation de profondeur $> k$, donc $\mathscr{C} \notin \beta(\text{all}, \mathbf{S})$. □

Pour la seconde inclusion :

Exemple 203

Soit $\mathscr{C}$ le système défini par :

$$\mathscr{C} = \{A(x, y), A(y, z) \rightarrow A(x, z)\}$$

La règle $\mathscr{C} \in CT^{\mathbf{O}}_{\forall\forall}$ et il existe une **SO**-dérivation BF de profondeur m sur l'instance $I = \{A(a_1, a_2), A(a_2, a_3), ..., A(a_n, a_{n+1})\}$ où $n = 2^m$ donc $\mathscr{C} \notin \beta(\text{BF}, \mathbf{SO})$, donc $\mathscr{C} \notin \beta(\text{weak}, \mathbf{SO})$ (d'après les théorèmes 207 et 209).

Démonstration. Par récurrence que pour tout k entre 0 et $m-1$ on a :

$$\{A(a_i, a_{i+2^k}), i \in \{1, ..., n-2^k\}\} \subseteq \mathscr{C}^k(I) \setminus \mathscr{C}^{k-1}(I)$$

avec $\mathscr{C}^{-1}(I) = \emptyset$. On a alors $A(a_1, a_{n+1}) \in \mathscr{C}^m(I) \setminus \mathscr{C}^{m-1}(I)$. □

Nous avons montré que dans le cas standard, si un système est unformément borné alors ce système ne termine pas toujours (exemple 201). Nous obtenons cependant par la preuve de la proposition 200 le résultat suivant :

Proposition 204

Pour tout entier k on a :

$$\beta^k(\text{all}, \mathbf{S}) \subseteq \beta^k(\text{BF}, \mathbf{S}) \subseteq \beta^k(\text{wBF}, \mathbf{S}) \subseteq \beta^k(\text{weak}, \mathbf{S}) \subseteq CT^{\mathbf{S}}_{\forall\exists}$$

Nous proposons deux exemples qui montrent que les deux inclusions $\beta(\text{BF}, \mathbf{S}) \subseteq \beta(\text{wBF}, \mathbf{S})$ et $\beta(\text{wBF}, \mathbf{S}) \subseteq \beta(\text{weak}, \mathbf{S})$ sont strictes (nous verrons que nous avons des égalités en oblivious et semi-oblivious).

Pour la première inclusion :

Exemple 205

On considère le système :

$$\mathscr{C} = \{A(x,y) \to B(x,y); A(x,y) \to B(x,z); B(x,y) \to A(x,y)\}$$

Alors le système vérifie $\mathscr{C} \notin \beta^1(\text{BF}, \mathbf{S})$ et $\mathscr{C} \in \beta^1(\text{wBF}, \mathbf{S})$.

Démonstration. Nous esquissons une ébauche de preuve :

$\mathscr{C} \notin \beta^1(\text{BF}, \mathbf{S})$: soit $I = \{A(a,b)\}$. Alors en appliquant la seconde règle, puis la troisième règle, on construit un fait de profondeur 2.

$\mathscr{C} \in \beta^1(\text{wBF}, \mathbf{S})$: soit I une instance. On applique la première règle sur tous les faits de la forme $A(\star,\star) \in I$ pour obtenir I', alors aucun trigger n'est actif sur I' et la dérivation est de profondeur 1 car aucun fait étiqueté par un A n'est de profondeur plus grande que 0 dans I'.

□

Pour l'inclusion $\beta^k(\text{wBF}, \mathbf{S}) \subseteq \beta^k(\text{weak}, \mathbf{S})$:

Exemple 206

On considère le système $\mathscr{C}$:

$$\mathscr{C} = \{A(x,y) \to A(y,z); A(x,y) \to B(y,z); B(x,y) \to A(x,x)\}$$

$\mathscr{C}$ n'admet pas de $\mathbf{S}$-dérivation BF terminante sur $\{A(a,b)\}$ donc $\mathscr{C} \notin \beta(\text{wBF}, \mathbf{S})$. De plus on a $\mathscr{C} \in \beta^2(\text{weak}, \mathbf{S})$.

5.1.3 Comparaison des classes en O et SO

Nous avons motivé le choix de se restreindre aux cas **O** et **SO** : les classes assurent la terminaison et les permutations de triggers sont possibles.

Nous montrons que nous avons équivalence entre :

— Toutes les BF sont de profondeur $\leq k$

— Il existe une BF de profondeur $\leq k$

— Il existe une dérivation de profondeur $\leq k$

Avant tout nous rappelons que toutes les X-dérivations BF terminantes ont la même profondeur ainsi :

Théorème 207

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Pour tout k on a $\beta^k(\text{BF}, X) = \beta^k(\text{wBF}, X)$.

Démonstration. Soit $\mathscr{C}$ dans $\beta^k(\text{wBF}, X)$. Pour tout I il existe une dérivation BF s de profondeur $k' \leq k$. Cette dérivation termine. Tout autre dérivation terminante sur I a donc la même profondeur que s. Donc $\mathscr{C} \in \beta^k(\text{BF}, X)$.

□

Pour montrer l'égalité $\beta^k(\text{wBF}, X) = \beta^k(\text{weak}, X)$, nous montrons la proposition suivante issue de [35, 34] :

Proposition 208

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Pour tout k, pour toute X-dérivation terminante de profondeur k, il existe une X-dérivation BF de profondeur k' avec $k' \leq k$.

Démonstration. Dans le cas **O** : si une **O**-dérivation s est terminante et de profondeur k alors on peut réordonner les triggers par profondeurs croissantes pour obtenir une **O**-dérivation BF de même profondeur.

On donne une ébauche de preuve dans le cas **SO** :

Soit s une **SO**-dérivation de profondeur k démarrant sur une instance I. On construit par récurrence sur i une **SO**-dérivation BF b_i de profondeur $\leq i$. On pose pour l'initialisation :

— b_0 est la dérivation I (aucun trigger n'est appliqué)

— s_0 est la dérivation s

On pose pour la récurrence :

Soit I_i la dernière instance de la dérivation b_{i-1}. On considère l'ensemble des triggers T de s_{i-1} tels que :

$(r,h) \in T$ ssi il existe $(r,H) \in \text{Actif}(I_i)$ et $(r,h) \sim (r,H)$

- Si T est vide alors on pose $b = b_{i-1}$ sinon :

T est partitionné par les classes $[(r,h)]_{I_i}$, soit T' un système de représentants de ces classes, alors tous les triggers de T' sont de même profondeurs.

On construit b_i en appliquant tous les triggers de T' sur I_i dans la dérivation b_i, et s_i est obtenu depuis s_{i-1} en retirant tous les triggers de T' à s_{i-1}.

Pour tout i, la **SO**-dérivation $b_i \circ s_i$ est bien définie et il y a (à relation d'équivalence $\sim$ près) bijection entre les triggers de s et ceux de $b_i \circ s_i$. De plus b_i est de profondeur $\leq i$ et elle est BF car tous les triggers sont successivement appliqués et ont tous la même profondeur (à chaque étape).

Enfin comme s est de profondeur k, alors s_k ne contient plus aucun trigger. Donc T est vide et on pose $b = b_k$

□

Ainsi :

Théorème 209

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$, pour tout k, $\beta^k(\text{wBF}, X) = \beta^k(\text{weak}, X)$.

Nous allons maintenant comparer les classes en fonction de X.

Nous avons tout d'abord la proposition évidente suivante :

Proposition 210

$$\beta^k(\text{all}, \mathbf{O}) \subseteq \beta^k(\text{all}, \mathbf{SO}) \subseteq \beta^k(\text{all}, \mathbf{S})$$

Nous démontrons maintenant l'inclusion dans le cas oblivious et semi-oblivious pour les autres classes :

Proposition 211

Pour tout Y de $\{\text{BF}, \text{weak}, \text{wBF}\}$

$$\beta^k(Y, \mathbf{O}) \subseteq \beta^k(Y, \mathbf{SO})$$

ébauche de preuve. Le résultat suffit pour $Y = \text{BF}$ puisque nous avons pour tout $X \in \{\mathbf{O}, \mathbf{SO}\}$, $\beta^k(\text{BF}, X) = \beta^k(\text{wBF}, X) = \beta^k(\text{weak}, X)$.

Si s est une $\mathbf{O}$-dérivation de profondeur k sur I, alors en retirant tous les triggers T qui partagent leur frontière avec un trigger déjà appliqué dans la dérivation puis tous les triggers qui dépendent d'un trigger de T (les descendants), on peut construire une $\mathbf{SO}$-dérivation sur I de profondeur inférieur ou égale à k.

D'après la proposition 208, il existe donc une $\mathbf{SO}$-dérivation BF sur I de profondeur inférieure donc de profondeur $\leq k$. □

Nous montrons que l'inclusion est stricte par cet exemple :

Exemple 212

On considère le système $\mathscr{C}$ défini par :

$$\mathscr{C} = \{A(x, y) \rightarrow A(x, z)\}$$

Alors $\mathscr{C} \in \beta^1(\text{all}, \mathbf{SO})$ et $\mathscr{C} \notin CT^{\mathbf{O}}_{\forall\forall}$ donc d'après la proposition 200 :

$$\mathscr{C} \in \beta^1(Y, \mathbf{SO}) \text{ et } \mathscr{C} \notin \beta(Y, \mathbf{O})$$

Démonstration. Il n'y a qu'une unique règle r dans $\mathscr{C}$.

Dans le cas $\mathbf{SO}$: Soit I une instance, soit (r, h) un trigger actif sur I, on ajoute un fait de la forme $A(a, n)$ avec a un nœud de I et n un nouveau nœud. Pour tout homomorphisme h_b tel que $h_b(x) = a$ le trigger (r, h_b) ne peut pas être actif car il possède la même frontière que h ($h(\partial r) = h_b(\partial r)$).

Dans le cas $\mathbf{O}$: Soit $I = \{A(a_0, a_1)\}$, pour tout i le trigger (r, h_i) avec $h_i(x) = a_i$ et $h_i(y) = a_{i+1}$ est actif sur $\{A(a_0, a_1), ..., A(a_i, a_{i+1})\}$, donc par récurrence sur i, il existe une $\mathbf{O}$-dérivation non terminante sur I.

□

Les classes de systèmes bornés dans le cas oblivious et semi-oblivious assurent la terminaison. On propose de comparer ces classes avec les systèmes WA [2] qui assurent eux aussi la terminaison :

Proposition 213

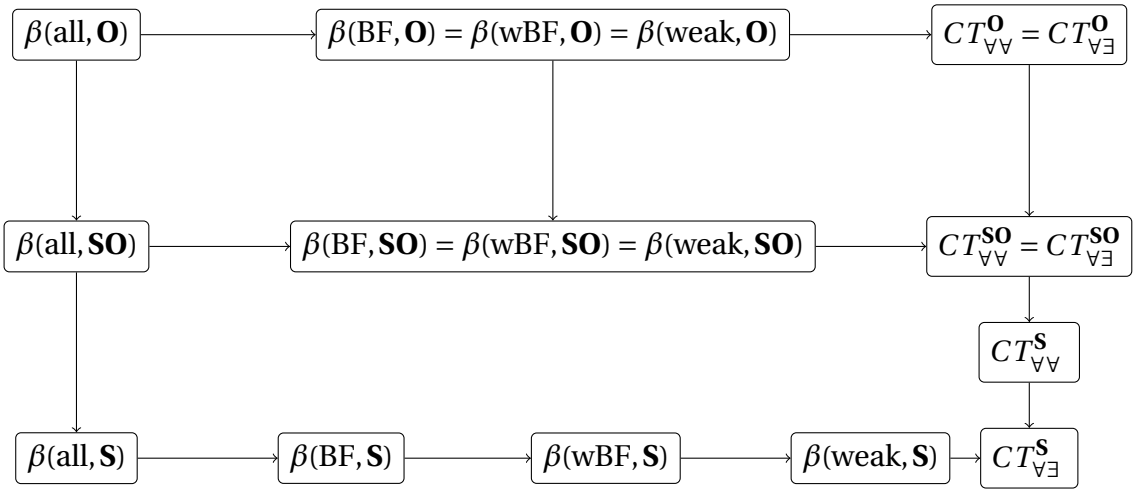
Les classes WA et $\beta(Y, X)$ sont non comparables pour tout $Y \in \{\text{all}, \text{BF}, \text{wBF}, \text{weak}\}$ et tout $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$.

2. weakly acyclic

Démonstration. La règle tgd $A(x,y), B(y,z), C(z,t) \rightarrow D(a,u), A(u,v), E(v,t)$ admet des **S**-dérivations BF toutes de profondeur 1 (car sa traduction est $abc \rightarrow dae \in \text{MB}_{\text{max}}(1)$ par le théorème 245) mais n'est pas faiblement acyclique.
Inversement, la règle $A(x,y), A(y,z) \rightarrow A(x,z)$ est WA mais pour tout k il existe une **S**-dérivation BF de profondeur k sur l'instance $\{A(a_1,a_2), ..., A(a_{2^k}, a_{2^{k+1}})\}$.

□

Nous obtenons ainsi le schéma :



5.1.4 Problèmes de décision

On se place toujours dans le cas où $X \in \{\mathbf{O}, \mathbf{SO}\}$.

Dans cette sous-section nous démontrons les deux résultats de complexité annoncé dans le résumé et dans [35, 17, 34] :

PROBLEME : Borne uniforme tgd **INSTANCE** : $\mathscr{C}$ un système de règles tgd, k un naturel **QUESTION** : $\mathscr{C} \in \beta^k(\text{all}, X)$? **COMPLEXITE** : Co-NEXPTIME-complet

et

PROBLEME : Borne uniforme tgd dans le cas BF **INSTANCE** : $\mathscr{C}$ un système de règles tgd, k un naturel **QUESTION** : $\mathscr{C} \in \beta^k(\text{BF}, X)$? **COMPLEXITE** : 2EXPTIME

Nous définissons les ancêtres d'un fait, ce sont les faits qui sont nécessaires pour le produire.

Définition 214

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une X-dérivation. Soit f un fait et soit i l'unique entier tel que $f \in I_{i+1} \setminus I_i$.

- Les 0-ancêtres de f sont $\{f\}$.
- Les ancêtres directs de f sont les faits de $h_i(B_i)$
- Si $k > 0$. Les k-ancêtres de f sont l'union des ancêtres directs des $(k-1)$-ancêtres de f.
- Les ancêtres de f sont l'union de tous les k-ancêtres de f.

On dénote alors $\text{Anc}^s(f)$ les ancêtres de f, puis nous dénotons $\text{Anc}^s_0(f)$ les ancêtres de f qui appartiennent à l'instance de départ i.e. $\text{Anc}^s_0(f) = \text{Anc}^s(f) \cap I_0$.

Nous allons montrer les étapes suivantes :

1. Le nombre d'instances (à isomorphisme près) ayant au plus t faits est majoré par $(\#S^t(at)^a)^t$ où a est l'arité maximale des prédicats de S.
2. Pour toute X-dérivation et tout fait f de profondeur k, le nombre de faits de $\text{Anc}^s_0(f)$ est majoré par b^k où b est le plus grand nombre de prédicats des corps des règles de $\mathscr{C}$.

3. Pour tout k on a $\#\mathscr{C}^k(I) \leq \#\mathscr{C}^{\sum_{i=0}^{k} b^i} \times \#I^{b^k}$.

On rappelle alors le résultat de [35, 34] :

Proposition 215

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Pour toute X-dérivation s (resp. BF) et tout fait f de profondeur k dans s, il existe une X-dérivation (resp. BF) sur $\mathrm{Anc}_0^s(f)$ telle que f soit de profondeur k.

Nous précisons ce résultat dans le cas non BF pour construire une X-dérivation sur $\mathrm{Anc}_0^s(f)$ de longueur exponentielle.

Proposition 216

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Pour toute X-dérivation s et tout fait f de profondeur k dans s, il existe une X-dérivation sur $\mathrm{Anc}_0^s(f)$ de longueur $\leq \frac{b^{k+1}-1}{b-1}$ telle que f soit de profondeur k.

Démonstration. On considère s, pour chaque fait ϕ des ancêtres de f on associe i_ϕ l'unique indice i tel que $\phi \in I_{i+1} \setminus I_i$. On considère le tuple τ des indices i_ϕ dans l'ordre croissant.

On construit la X-dérivation $\mathrm{Anc}(s)$ sur $\mathrm{Anc}_0^s(f)$ par induction en montrant que les triggers (r_i, h_i) avec i dansτ sont actifs. L'initialisation est claire. Supposons que l'on a construit l'instance I_i avec $i \in \tau$, soit j le prochain indice de τ. Puisque s est une $\mathbf{O}$-dérivation et τ est trié dans l'ordre croissant, le trigger (r_j, h_j) est actif sur I_i.

Le fait f est aussi de profondeur k dans $\mathrm{Anc}(s)$.

La longueur de $\mathrm{Anc}(s)$ est égale au nombre d'ancêtres de f, donc $\sum_{i=0}^{k} b^i = \frac{b^{k+1}-1}{b-1}$. □

Nous avons le résultat de dénombrement suivant :

Proposition 217

Le nombre d'instances (à isomorphisme près) ayant au plus t faits est $\leq (\#S^t(at)^a)^t$ où a est l'arité maximale des prédicats de S.

Démonstration. Chaque fait est associé à un prédicat A et au plus a constantes ou variables, l'instance est donc constituée d'au plus $a \times t$ constantes ou variables.

Pour chaque fait, on associe un couple $(A, tuple)$ où A est un prédicat de S et $tuple$ un tuple de arity(A) éléments de constantes et variables parmi $a \times t$ éléments.

Ainsi il y a au plus $\#S(a \times t)^a$ possibilités pour chaque fait.

Puisque les instances contiennent au plus t faits le nombre de possibilités pour de telles instances est donc inférieur à $\leq (\#S^t(at)^a)^t$. □

Proposition 218

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soit s une X-dérivation. Soit f un fait de profondeur k on a :

$$\#\mathrm{Anc}_0^s(f) \leq b^k$$

Démonstration. Par récurrence immédiate, les faits de I_0 qui sont les ancêtres de f sont exactement les $(k+1)$-ancêtres de f. Par une autre récurrence, les 0-ancêtres sont constitués de 1 fait, et les $(k+1)$-ancêtres sont d'au plus b fois le nombre de k-ancêtres donc $b \times b^{k-1} = b^k$. □

Proposition 219

Soit I une instance, le nombre de triggers (r, h) tels que $h(B) \subseteq I$ est majoré par $\leq \#\mathscr{C} \times \#I^b$

Démonstration. Soit I une instance, soit $r = B \to H$ une règle tgd. Pour chaque atome de B on associe un fait de I par un homomorphisme.

Le nombre de triggers (r, h) pour une règle fixée r sur I est donc majoré par $\#I^{\#B}$. D'où la proposition. □

Proposition 220

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Pour tout k et toute instance I on a :

$$\#\mathscr{C}^k(I) \leq \#\mathscr{C}^{\sum_{i=0}^k b^i} \times \#I^{b^k}$$

Démonstration. Les triggers sont tous de même profondeur, donc orthogonaux, on peut supposer donc que tous les triggers sont actifs. On obtient $\#\mathscr{C}(I) \leq \#\mathscr{C} \times \#I^b$, le résultat s'en déduit par une récurrence facile. □

Pour construire une dérivation, on garde la dernière instance indicée obtenue en mémoire, et un dictionnaire qui contient tous les couples (r,h) des triggers qui ont déjà été appliqués. La mémoire est donc un couple (T,T') formé d'un dictionnaire d'instances et d'un dictionnaire de triggers.

Soit (T,T') fixé. soit I le dernier élément de T. soit (R,H) tels que $H(B) \subseteq I$

i) si $X = \mathbf{O}$ et (R,H) n'appartient pas au dictionnaire T' alors on applique le trigger qui est actif pour **O**.

ii) si $X = \mathbf{SO}$ et $H(\partial R)$ n'appartient à aucun $h(\partial r)$ des (r,h) du dictionnaire alors on applique le trigger qui est actif pour **SO**.

Ainsi, on obtient une nouvelle instance I' qu'on ajoute à T, puis on ajoute (R,H) au dictionnaire T'.

Théorème 221

Nous avons :

PROBLEME : Borne uniforme tgd
INSTANCE : $\mathscr{C}$ des règles tgd, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{all}, X)$?
COMPLEXITE : Co-NEXPTIME

Démonstration. S'il existe une X-dérivation de profondeur $k+1$, alors il existe un fait f de profondeur $k+1$, donc il existe d'après la proposition 215 une dérivation de profondeur $k+1$ sur $\text{Anc}_0^s(f)$, cette instance est de taille $\leq b^{k+1}$ d'après 218. Donc il existe une X-dérivation de profondeur $k+1$ sur les instances de tailles $\leq b^{k+1}$.

On vérifie étant donné deux dictionnaires (T,T') qu'il s'agit d'une X-dérivation en vérifiant que chaque trigger de T' est actif sur les instances de T, cela revient à parcourir T et T' de longueur $\leq \frac{b^{k+2}-1}{b-1}$ d'après 216. □

Théorème 222

PROBLEME : Borne uniforme tgd dans le cas BF
INSTANCE : $\mathscr{C}$ des règles tgd, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{BF}, X)$?
COMPLEXITE : 2EXPTIME

Démonstration.
D'après les propositions 218 et 215 il suffit de vérifier les X-dérivations sur toutes les

instances de taille $\leq b^k$. Il y a au plus $(\#S^{b^k}(ab^k)^a)^{b^k}$ telles instances de cette taille d'après la proposition 217.

La longueur d'une X-dérivation BF de profondeur k est majorée par la taille de $\mathscr{C}^k(I)$ (avec I l'instance de départ), donc $\leq \#\mathscr{C}^{\Sigma_{i=0}^k b^i} \times \#I^{b^k}$ d'après la proposition 220.

On construit donc parmi toutes les instances de taille $\leq b^k$ toutes les X-dérivations sous la forme (T, T') par récurrence : on choisit un trigger pour la dernière instance de T et on vérifie s'il est actif (la longueur de T' est au pire de taille doublement exponentielle) et on l'applique.

Enfin, parmi toutes les X-dérivations on vérifie si une des instances a un fait de profondeur $> k$. Cet algorithme est 2EXPTIME. □

Enfin on montre la dureté du problème en considérant le cas Datalog :

Théorème 223

Pour tout $X \in \{\mathbf{O}, \mathbf{SO}, \mathbf{S}\}$:

PROBLEME : Borne uniforme Datalog dans le cas BF
INSTANCE : $\mathscr{C}$ un programme Datalog, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{BF}, X)$?
COMPLEXITE : Co-NEXPTIME-complet

reuve issue de [17]. . Soit Q_1 (resp. soit Q_2) une requête Datalog booléenne non-récursive, P_0 (resp. P_0^2) le prédicat 0-aire distingué. La requête Q_1 (resp.Q_2) est bornée par son nombre d'atomes k_1 (resp. k_2). Soit $p = \max(k_1, k_2) + 2$. On ajoute à Q_1 (resp. Q_2) des règles de la forme $P_{i-1} \to P_i$ (resp. $P_0^2 \to P_i$) pour i de 1 à p pour obtenir Q_1' (resp. Q_2').

On va montrer que $Q_1' \cup Q_2'$ est dans $\beta^{p-1}(\text{BF}, X)$ si et seulement si Q_1 est inclus dans Q_2.

Supposons que Q_1 est inclus dans Q_2. Soit I une instance. S'il existe une dérivation démarrant sur I qui construit P_0 dans une instance J, alors il existe une dérivation démarrant sur I qui construit P_0^2 avec une profondeur d'au plus k_2, donc avec les règles $P_0^2 \to P_i$ on a $Q_2(J)$ qui contient tous les P_i avec une profondeur au plus $k_2 + 1$. Donc la X-dérivation BF $(Q_1' \cup Q_2')^*(I)$ est de profondeur majorée par $\max(k_1, k_2+1) \leq p-1$. Si une telle dérivation n'existe pas, on a P_0 qui n'appartient pas à $(Q_1' \cup Q_2')^*(I)$, donc la dérivation est de profondeur $\max(k_1, k_2+1) \leq p-1$ (on utilise seulement des règles de Q_1 et Q_2').

Inversement, supposons que Q_1 n'est pas inclus dans Q_2, alors par définition, il existe une instance I telle que $P_0^2 \notin Q_2^*(I)$ et $P_0 \in Q_1^*(I)$. On peut appliquer successivement des triggers actifs dont la règle est de la forme $P_{i-1} \rightarrow P_i$, c'est une dérivation de profondeur $\geq p$.

Ainsi par réduction depuis le problème d'inclusion de requêtes Datalog booléennes non-récursives connu comme Co-NEXPTIME-difficile [13] la dureté est prouvée.

□

5.2 Systèmes de règles tgd totales en oblivious

5.2.1 Résumé

Lors de l'exécution de l'oblivious-chase, la présence d'une variable existentielle dans un atome de la tête assure qu'un nouveau fait est ajouté à l'instance. Ainsi en assurant la présence de variables existentielles dans tous les atomes de la tête, chaque atome de la tête ajoute un nouveau fait : on a alors $I_{i+1} \setminus I_i = h'_i(H_i)$.

Nous appelons par "total" [3] un ensemble de règles tgd dont tout fait de la tête de toutes les règles contient au moins une variable existentielle.

Par exemple les systèmes suivants ne sont pas totaux :

— $\{A(x,y,z) \rightarrow B(x,y)\}$

— $\{A(x,y) \rightarrow B(x,z); C(x) \rightarrow D(x,x)\}$

— Les exemples 202 et 203

Nous dirons parfois règle totale, ou système total quand toutes les règles sont totales.

Nous proposons le plan suivant pour la section, découpée en 3 sous-sections :

Sous-section 1 : Nous avons montré dans la section précédente (les théorèmes 207 et 209) que :

$$\beta^k(\text{all}, X) \subsetneq \beta^k(\text{BF}, X) = \beta^k(\text{wBF}, X) = \beta^k(\text{weak}, X)$$

Si $\beta(Y,X)$ est une classe de tgd, nous précisons par une lettre t pour signifier que nous considérons sa restriction aux systèmes totaux : $\beta(Y,X)_t$.

Lorsque nous sommes dans le cas **total** et **oblivious**, l'exemple 202 n'est plus un contre-exemple et montrons que :

$$\beta^k(\text{all}, \mathbf{O})_t = \beta^k(\text{BF}, \mathbf{O})_t = \beta^k(\text{wBF}, \mathbf{O})_t = \beta(\text{weak}, \mathbf{O})_t$$

Cette propriété est en revanche fausse dans le cas semi-oblivious. En effet :

Exemple 224

On considère le système total défini par :

$$\mathscr{C} = \{A(x,y,u), A(y,t,v) \rightarrow A(x,t,w); A(x,y,u), A(z,t,v) \rightarrow A(x,t,w)\}$$

Alors $C \in \beta^2(\text{BF}, \mathbf{SO})$ et $C \notin \beta(\text{all}, \mathbf{SO})$.

3. L'équivalent anglais utilisé dans [17] est "fully-existential".

Démonstration.
On note

$P = \{A(x,y,u), A(y,t,v) \to A(x,t,w)\}$ et $Q = \{A(x,y,u), A(z,t,v) \to A(x,t,w)\}$.

Soit I une instance, on note :

— Pour tout $S \in \{P,Q\}$, $F_i^S(I)$ l'ensemble des $h(\partial r)$ pour chaque trigger actif (r,h) sur $\mathscr{C}^i(I)$ tel que $r = S$

— G_I l'ensemble des x tel qu'il existe y,u tels que $A(x,y,u) \in I$

— D_I l'ensemble des y tel qu'il existe x,u tels que $A(x,y,u) \in I$

Première propriété :

Les ensembles G_I et D_I sont invariants par $\mathscr{C}$:

Si $x \in G_{\mathscr{C}(I)}$ alors il existe t,w tels que $A(x,t,w) \in \mathscr{C}(I)$, donc soit $A(x,t,w) \in I$ et c'est terminé, soit :

— D'après P, il existe y,u,v tels que $A(x,y,u) \in I$ et $A(y,t,v) \in I$

— D'après Q, il existe y,z,u,v tels que $A(x,y,u) \in I$ et $A(z,t,v) \in I$

Dans les deux cas on a $x \in G_I$ et $y \in D_I$.

De même si $x \in G_I$ et $y \in D_I$ alors d'après Q il existe u tel que $A(x,y,u) \in \mathscr{C}(I)$.

Seconde propriété :

On a $F_2^Q = F_1^Q$

Soit $(x,t) \in F_2^Q$, il existe un trigger (r,h) avec $r = Q$ actif sur $\mathscr{C}^2(I)$ donc d'après h il existe y,u,v tels que $A(x,y,u)$ et $A(y,t,v)$ soient dans $\mathscr{C}^2(I)$. Donc $x \in G_{\mathscr{C}^2(I)} = G_{\mathscr{C}(I)}$, $y \in G_{\mathscr{C}^2(I)} \cap D_{\mathscr{C}^2(I)} = G_{\mathscr{C}(I)} \cap D_I$ et $t \in D_{\mathscr{C}^2(I)} = D_I$ donc :

il existe u', v' tels que $A(x,y,u') \in \mathscr{C}(I)$ et $A(y,t,v') \in \mathscr{C}(I)$, donc $(x,t) \in F_1^Q$

Ainsi $F_2^Q = F_1^Q$.

Conclusion :

On a $F_1^P = G_I \times D_I$ donc $F_1^P = F_2^P$ et $F_2^Q = F_1^Q$. Ainsi $\mathscr{C}^3(I) = \mathscr{C}^2(I)$.

□

Sous-section 2 : Nous montrons une propriété intéressante en oblivious dans le cas total :

Être borné est équivalent à être terminant

Autrement dit nous montrons que :

$$\beta(\text{all}, \mathbf{O})_t = CT^{\mathbf{O}}_{\forall\forall,t}$$

Ce résultat a été généralisé dans le cas non-total avec une autre notion de profondeur [17].

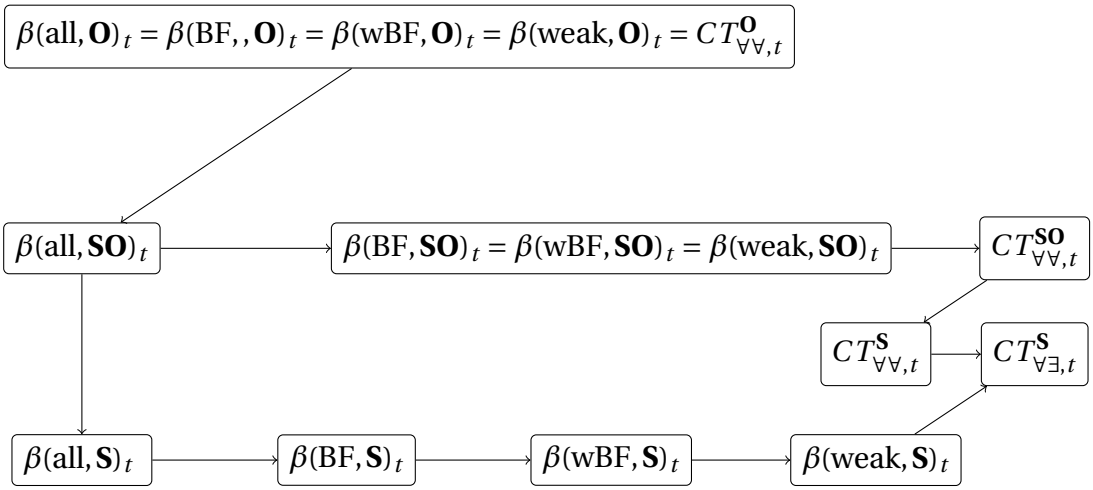
Avec notre notion de profondeur on rappelle que l'équivalence "termine = borné" n'a pas lieu dans le cas non-total. Par exemple le système $A(x,y), A(y,z) \to A(x,z)$ (exemple 203) termine mais il existe des dérivations aussi profondes que souhaité.

Nous n'avons pas non plus cette égalité pour les autres variantes de chase.

Exemple 225

Soit $\mathscr{C} = \{A(x,y,u), A(y,z,v) \longrightarrow A(x,z,w)\}$
- $\mathscr{C}$ est un système total
- $\mathscr{C} \in CT^{\mathbf{SO}}_{\forall\forall}$
- $\mathscr{C} \notin \beta(\text{wBF}, \mathbf{SO}) = \beta(\text{BF}, \mathbf{SO}) = \beta(\text{all}, \mathbf{SO})$

Ainsi, pour résumer nous aurons :



Puisque $\beta^k(\text{BF}, \mathbf{O})_t = \beta^k(\text{all}, \mathbf{O})_t$ nous obtenons d'après le théorème 221 :

Théorème 226

Nous avons :

PROBLEME : Borne uniforme tgd dans le cas BF et total
INSTANCE : un système total $\mathscr{C}$, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\text{BF}, \mathbf{O})$?
COMPLEXITE : Co-NEXPTIME

5.2.2 Equivalence BF - all

Nous allons transformer toute **O**-dérivation en une dérivation BF de même profondeur. Dans le cas total, permuter les triggers ne modifient pas la profondeur. Dans le cas non-total cette propriété n'est pas vérifiée :

Exemple 227

Soit $\mathscr{C}$ le système

$$\{r_{AB} = A(x,y) \rightarrow B(x,y); r_{BC} = B(x,y) \rightarrow C(x,y); r_{AC} = A(x,y) \rightarrow C(x,y)\}$$

- Si on applique successivement r_{AB}, r_{BC} et r_{AC} sur l'instance $A(n,n')$, on obtient une dérivation de profondeur 2) car $C(n,n')$ est de profondeur 2. - Si on échange les deux derniers triggers on obtient une dérivation de profondeur 1 ($C(n,n')$ provient directement de $A(n,n')$ et la règle r_{BC} ne produit pas de fait de profondeur 2).

Lemme 228

Soit $X \in \{\mathbf{O}, \mathbf{SO}\}$. Soit $\mathscr{C}$ un ensemble de règles tgd **totales**, pour toute X-dérivation indicée $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ orthogonale en $i < n$, soit $s' = \text{exch}_{i,i+1}(s)$, pour tout $f \in I_{i+2}$ on a :

$$\text{erank}_{s'}(f) = \text{erank}_s(f)$$

Démonstration. On rappelle que par définition, pour tout i, tout fait f de $h'_i(H_i)$ vérifie $\text{erank}_s(f) = \text{rank}_s(r_i, h_i)$ si et seulement si f n'est pas un fait de I_i. Or dans le cas total, puisque tout atome de H contient au moins une variable existentielle, aucun fait de $h'_i(H_i)$ n'est dans I_i.

On rappelle que $I_{i+2} = I_i \cup h'_i(H_i) \cup h'_{i+1}(H_{i+1})$, les règles étant totales cette union est disjointe. De plus s est orthogonale en i donc $h_{i+1}(B_{i+1}) \subseteq I_i$ donc

$\text{rank}_{s'}(r_{i+1}, h_{i+1}) = \text{rank}_s(r_{i+1}, h_{i+1})$ de même $\text{rank}_s(r_i, h_i) = \text{rank}_{s'}(r_i, h_i)$. Finalement, tout fait f de I_{i+2} est soit dans I_i et alors son erank est inchangé selon s ou s', soit f est dans $h'_i(H_i)$ (et non dans I_i) et donc $\text{erank}_s(f) = \text{rank}_s(r_i, h_i) = \text{rank}_{s'}(r_i, h_i) = \text{erank}_{s'}(f)$., soit f est dans $h'_{i+1}(H_{i+1})$ et on a exactement le même raisonnement. □

Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une $\mathbf{O}$-dérivation de longueur n, on peut réordonner les triggers pour produire une $\mathbf{O}$-dérivation s' dont les triggers sont de profondeur croissante et dont la dernière instance est I_n. Les faits de I_n vérifient $\text{erank}_{s'}(f) = \text{erank}_s(f)$.

Puisque toute $\mathbf{O}$-dérivation terminante dont les triggers sont de profondeur croissante est BF [35] on a :

Lemme 229

Soit $\mathscr{C}$ des règles tgd **totales**, pour toute **O**-dérivation terminante s de profondeur k, il existe une **O**-dérivation BF de même profondeur sur la même instance que s.

Ainsi :

Théorème 230

Soit $k \geq 0$, soit $\mathscr{C}$ des règles tgd **totales**,

$$\mathscr{C} \in \beta^k(\text{BF}, \mathbf{O}) \text{ si et seulement si } \mathscr{C} \in \beta^k(\text{all}, \mathbf{O})$$

Démonstration. Soit $\mathscr{C}$ dans $\beta^k(\text{BF}, \mathbf{O})$, alors si $\mathscr{C}$ n'est pas dans $\beta^k(\text{all}, \mathbf{O})$ il existe une **O**-dérivation de profondeur $k+1$, d'après le lemme 229, il existe une **O**-dérivation BF de profondeur $k+1$, contradiction par hypothèse sur $\mathscr{C}$. □

5.2.3 Équivalence terminant - borné

De manière générale dans une X-dérivation les profondeurs des faits sont toujours supérieurs ou égaux aux profondeurs des nœuds qui composent les faits. Dans le cas total nous avons toujours une égalité :

Lemme 231

Soient $\mathscr{C}$ un système de règles tgd **totales** et une **O**-dérivation :

$$s : I_0 \xrightarrow{(r_0, h_0)} I_1 \ldots$$

On a pour tout i :

$$\forall A(n_1, \ldots, n_t) \in I_i, \text{erank}_s(A(n_1, \ldots, n_t)) = \max_{1 \leq u \leq t} \text{vrank}_s(n_u)$$

Démonstration. Nous montrons le résultat par récurrence sur i. Le cas initial est facile, toutes les profondeurs valent 0. Supposons la propriété vraie pour l'instance I_i et montrons qu'elle est vraie dans I_{i+1} :

Comme r_i est une règle totale alors une des variables de H_i est existentielle donc il existe v tel que n_v appartienne à $I_{i+1} \setminus I_i$. On a donc par définition de vrank$_s$:

$$\max_{1 \leq u \leq t} \text{vrank}_s(n_u) = \text{erank}_s(A(n_1, ..., n_t))$$

□

Nous allons maintenant dans cette sous-section montrer que dans le cas des règles totales, tout système **O**-terminant est borné. Pour cela nous allons adapter le résultat de [75] qui assure la terminaison du système si toutes les **O**-dérivations terminent sur l'instance critique.

Définition 232

Soit a une constante, l'instance critique U_a est l'ensemble des faits $R(a, ..., a)$ où $R \in S$.

Idée de construction de la projection :

Pour toute **O**-dérivation s sur une instance I nous construisons par récurrence une nouvelle **O**-dérivation $\Pi_a(s)$ sur l'instance critique U_a de même profondeur :

pour chaque nouvelle instance I'_i définie (par hypothèse de récurrence) on construit un morphisme de I_{i+1} vers I'_{i+1} permettant de construire un trigger (r_i, g_i) entre I'_i et I'_{i+1}. Parfois ces triggers existent déjà parmi ceux qui ont été construits par récurrence, on le supprime donc car il serait inactif, les triggers que nous ne supprimons pas sont dit "acceptés". Le trigger (r_k, g_k) accepté est actif sur I'_i et vérifie $I'_k = I'_i \cup g'_k(H_k)$ avec i l'indice (du dernier trigger) accepté $< k$. Ainsi on obtient bien une **O**-dérivation sur U_a.

$$\begin{array}{ccccc} I_0 & \xrightarrow{(h_0, r_0)} & I_1 & \xrightarrow{(h_1, r_1)} & \cdots \\ \downarrow{\scriptstyle \varphi_0 = \Pi_a} & & \downarrow{\scriptstyle \varphi_1} & & \\ I'_0 = U_a & \xrightarrow{(g_0, r_0)} & I'_1 & \xrightarrow{(g_1, r_1)} & \cdots \end{array}$$

Définition 233

On note Π_a l'application qui à toute variable ou constante associe a. On étend Π_a par morphisme aux faits, puis aux instances.

On souhaite étendre la projection Π_a aux dérivations (on rappelle que h' désigne toujours l'extension de h aux variables existentielles) :

Définition 234

Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$ une **O**-dérivation.
On associe une séquence d'instances I'_i, de morphismes d'instances φ_i et de morphismes g_i des variables vers les constantes définies par récurrence par :
- **Initialisation :** On pose $\varphi_0 = \Pi_a$. On construit g_0 comme la composée $\varphi_0 \circ h_0$, puis nous définissons $\varphi_1(n)$ par $\varphi(n)$ si n est dans I_0 et $h'_0(n)$ si $n \in I_1 \setminus I_0$. On pose alors I'_1 comme égal à $I'_0 \cup h'_0(H_0)$. On a alors $I'_1 = \varphi_1(I_1)$. On dit aussi que 0 est accepté.
- **Induction :** Supposons qu'on aie construit le triplet $(I'_i, \varphi_i, g_{i-1})$ pour un $i > 0$. On commence par poser $g_i = \varphi_i \circ h_i$, on a alors $g_i(B_i) \subseteq I'_i$, ensuite pour construire φ_{i+1} et I'_{i+1} on distingue alors deux cas :
Cas 1 : Si pour tout $j < i$, $(r_i, g_i) \neq (r_j, g_j)$ on dit que i est accepté et on pose alors :
  — $\varphi_{i+1}(n) = \varphi_i(n)$ si $n \in I_i$
  — $\varphi_{i+1}(n) = h'_i(n)$ si $n \in I_{i+1} \setminus I_i$
  — $I'_{i+1} = I'_i \cup h'_i(H_i)$
On remarquera qu'on a alors $I'_{i+1} = \varphi_{i+1}(I_{i+1})$
Cas 2 : S'il existe $j < i$ tel que $(r_i, g_i) = (r_j, g_j)$ on pose alors :
  — $\varphi_{i+1}(n) = \varphi_i(n)$ si $n \in I_i$
  — $\varphi_{i+1}(n) = (\varphi_{j+1} \circ h'_j \circ h'^{-1}_i)(n)$ si $n \in I_{i+1} \setminus I_i$
  — $I'_{i+1} = I'_i$
On remarquera qu'on a alors $I'_{i+1} = \varphi_{i+1}(I_{i+1})$.
Enfin :
On pose alors $\Pi_a(s) = I'_0 \xrightarrow{(r_0,g_0)} I'_1 \xrightarrow{(r_1,g_1)} I'_k ...$ pour les indices acceptés (ici k n'est pas forcément égal à 2). L'extension de g_i quand i est accepté est $\varphi_{i+1}(h'_i(I_{i+1} \setminus I_i))$. On a $\Pi_a(s)$ qui est une **O**-dérivation, on l'appelle la a-projetée de s.

Nous allons montrer que la projection de la dérivation sur U_a ne change pas la profondeur des faits.

Lemme 235

Soit $\mathscr{C}$ des règles tgd **totales**, soit a une constante, soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une **O**-dérivation, soit alors $(\Pi_a(s), \varphi_0, \ldots)$ la dérivation a-projetée de s. Alors pour tout i et tout fait f de I_i on a :

$$\text{erank}_{\Pi_a(s)}(\varphi_i(f)) = \text{erank}_s(f)$$

Démonstration. Comme le système est total, d'après le Lemme 231, il suffit de vérifier la propriétés pour les nœuds.

On montre par récurrence sur i. Le cas $i = 0$ est facile (les profondeurs sont nulles). On montre l'induction :

Supposons la propriété vraie au rang i. Soit n un nœud de I_{i+1}, si $n \in I_i$ alors par hypothèse de récurrence la propriété est vérifiée, on suppose alors que $n \in I_{i+1} \setminus I_i$.

On distingue deux cas, si i est accepté ou non :

- si i est accepté alors :

$\text{vrank}_{\Pi_a(s)}\varphi_{i+1}(n) = \text{vrank}_{\Pi_a(s)}h'_i(n) = 1 + \max_{f\in g_i(B)} \text{erank}_{\Pi_a(s)}(f)$

$= 1 + \max_{f\in h_i(B)} \text{erank}_{\Pi_a(s)}\varphi_i(f)$ car $g_i = \varphi_i \circ h_i$.

Donc par hypothèse de récurrence, comme $h_i(B) \subseteq I_i$ on a :

$\text{vrank}_{\Pi_a(s)}\varphi_{i+1}(n) = 1 + \max_{f\in h_i(B)} \text{erank}_s(f) = \text{vrank}_s(n)$ $(n \in I_{i+1} \setminus I_i)$.

- si i n'est pas accepté alors :

Il existe $j < i$ tel que $(r_i, g_i) = (r_j, g_j)$ et j accepté.

On pose $z = h'^{-1}_i(n)$. Par définition on a :

$\text{vrank}_{\Pi_a(s)}\varphi_{i+1}(n) = \text{vrank}_{\Pi_a(s)}\varphi_{i+1}(h'_i(z)) = \text{vrank}_{\Pi_a(s)}(\varphi_{j+1}(h'_j(z))$

Or $\varphi_{j+1}(h'_j(z)) = g'_j(z)$ on a : $\text{vrank}_{\Pi_a(s)}\varphi_{i+1}(n) = 1 + \max_{f\in g_j(B_j)} \text{erank}_{\Pi_a(s)}(f)$

Or $g_j = g_i$ et $r_j = r_i$ donc $\max_{f\in g_j(B_j)} \text{erank}_{\Pi_a(s)}(f) = \max_{f\in g_i(B_i)} \text{erank}_{\Pi_a(s)}(f) = \max_{f\in h_i(B_i)} \text{erank}_{\Pi_a(s)}\varphi_i(f)$.

Par hypothèse de récurrence, $\max_{f\in h_i(B_i)} \text{erank}_{\Pi_a(s)}\varphi_i(f) = \max_{f\in h_i(B_i)} \text{erank}_s\varphi_i(f)$

Donc $\text{vrank}_{\Pi_a(s)}\varphi_{i+1}(n) = 1 + \max_{f\in h_i(B_i)} \text{erank}_s\varphi_i(f) = \text{vrank}_s h'_i(z)$ car z existentielle et donc

$$\text{vrank}_{\Pi_a(s)}\varphi_{i+1}(n) = \text{vrank}_s n$$

□

D'après le lemme ci-dessus, si s est de profondeur k alors il existe un fait de profondeur k dans I_n, donc la fait $\varphi_n(f)$ est de profondeur k dans $\Pi_a(s)$. Inversement s'il existe un fait de profondeur $k+1$ dans I'_n comme il est de la forme $\varphi_n(f)$ il existe un fait de profondeur $k+1$ dans I_n, contradiction.

Ainsi, s et $\Pi_a(s)$ ont même profondeur. On en déduit :

Théorème 236

Soit $\mathscr{C}$ un ensemble de règles tgd **totales**, si $\mathscr{C} \in CT^{\mathbf{O}}_{\forall\forall}$ alors il existe k tel que $\mathscr{C} \in \beta^k(\text{all}, \mathbf{O})$.

Démonstration. Supposons que $\mathscr{C} \in CT^{\mathbf{O}}_{\forall\forall}$ alors toute **O**-dérivation termine sur toute instance.

Les **O**-dérivations terminent sur U_a, puisque les règles sont totales, d'après le lemme 229, il existe pour chaque s une **O**-dérivation BF sur U_a de même profondeur que s.

Or toutes les **O**-dérivations sur une instance donnée ont même profondeur, notons k cette profondeur.

Enfin, pour toute **O**-dérivation s terminante, on a $\Pi_a(s)$ de même profondeur, or $\Pi_a(s)$ est sur U_a donc est de profondeur k.

Ainsi, comme toutes les **O**-dérivations sont terminantes, elles sont de profondeur k. □

5.3 Chase dans le cadre des contraintes de mots

5.3.1 Résumé

Nous découpons cette section en 3 parties :
— Nous introduisons des préliminaires sur la notion de chemin
— Nous comparons les classes dans les contraintes de mots
— Nous les comparons à nouveau dans le cas totale

Dans la première partie nous introduisons la notion de chemin, ce sont des sous-instances dont les faits sont chaînés par des nœuds.

En particulier nous introduisons la notion de k-chemin comme un chemin où tous les faits et tous les nœuds internes sont de profondeur k et les nœuds de départ et d'arrivée du chemin sont quant à eux de profondeur $< k$. Ce sont les plus longs chemins dont tous les faits sont de profondeur k.

Nous montrerons que les k-chemins correspondent aux $h'(H_i)$, ce sont donc les chemins ajoutés par les contraintes de mots.

Nous souhaitons mettre en comparaison la profondeur de la dérivation sur les instances avec la complexité parallèle de celle sur les mots.

Cependant le dialogue n'est pas parfait :

Exemple 237

Soit $\mathscr{C} = \{A(x,y) \to A(x,y)\}$ alors $\mathscr{C} \in \beta^1(\text{all}, \mathbf{O})$ tandis que $a \to a$ n'est pas terminant donc n'appartient pas à MB_{max}.

Ainsi nous allons montrer dans la seconde sous-section que la complexité parallèle d'un système est toujours supérieure ou égale à la borne uniforme des contraintes de mots associées. Nous obtenons donc le schéma :

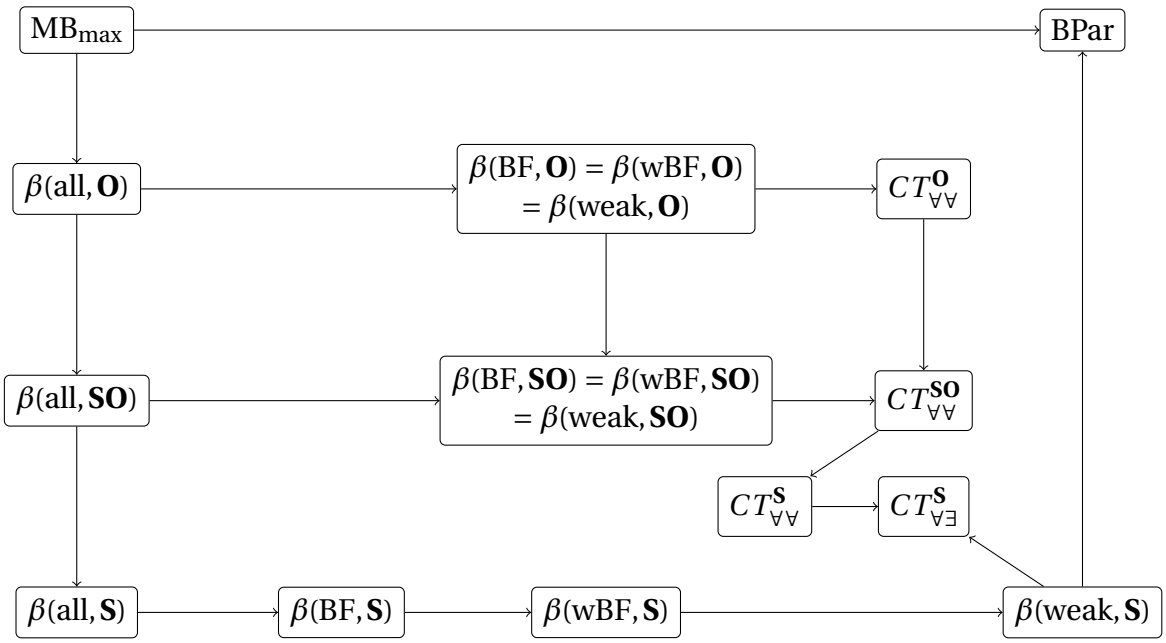
$$\text{MB}_{\text{max}}(k) \xrightarrow{\text{Ex 237}} \beta^k(\text{all}, \mathbf{O}) \longrightarrow \beta^k(\text{weak}, \mathbf{S}) \xrightarrow{\text{Ex 238}} \text{BPar}(k)$$

La dernière inclusion est stricte :

Exemple 238

Soit $\mathscr{C} = \{A(x,z), A(z,y) \to A(x,t), B(t,y); A(x,z), A(z,y) \to B(x,t), B(t,y)\}$. $\mathscr{C}$ n'est pas dans $CT^S_{\forall\exists}$ (voir la preuve de la proposition 248) donc n'est dans aucun des $\beta^k(\text{weak}, \mathbf{S})$ pour tout k. Tandis le système $aa \to ab, aa \to bb$ est dans BPar(2).

Nous avons le schéma :

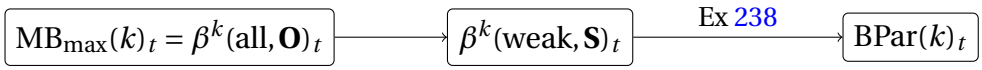


Le contre-exemple $a \to a$ présent le défaut majeur de ne pas toujours ajouter de faits : on propose de contourner en utilisant des règles tgd totales.

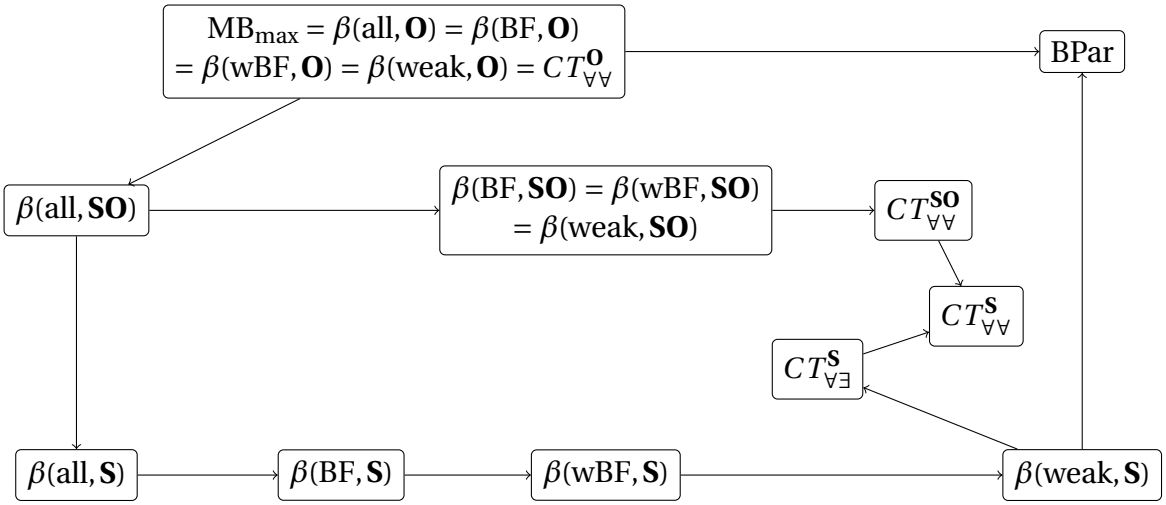
Une contrainte de mot (voir def. 77) $\{A_1(x_1,x_2), A_2(x_2,x_3), ..., A_n(x_n,x_{n+1})\} \to \{B_1(x_1,y_2), B_2(y_2,y_3$ est totale si et seulement on a $y_2 \neq x_{n+1}$ donc $m > 1$. Donc si la règle de réécriture $u \to v$ associée vérifie $|v| > 1$.

Nous montrons alors que dans le cas total la complexité parallèle et la borne uniforme coïncident.

Nous avons dans le cas total :



Avec les résultats de la section précédentes nous avons :



Finalement, l'algorithme de décision $R \in \mathrm{MB}_{\max}(k)$? proposé au chapitre 4 permet de décider si des contraintes de mots totales sont uniformément bornées par k.

5.3.2 Préliminaires sur les chemins

Nous introduisons des instances dont le comportement est naturel avec les contraintes de mots : les instances chemins.

Définition 239

Un chemin π est une instance de la forme :

$$\{A_1(n_1, n_2), A_2(n_2, n_3), ..., A_v(n_v, n_{v+1})\}$$

Contrairement à ce qu'on a pu imposer dans les contraintes de mots, les nœuds $n_1, ..., n_{v+1}$ ne sont pas obligatoirement distincts. On identifie n_1 comme étant le premier nœud (ou valeur) de π et n_{v+1} comme le dernier nœud (ou valeur) de π.

De même, les faits sont ordonnés par des indices (ici de 1 à v).

Soit $\pi = \{A_1(n_1, n_2), A_2(n_2, n_3), ..., A_v(n_v, n_{v+1})\}$ un chemin, la numérotation de $f \in \pi$ est l'entier t tel que $A_t(n_t, n_{t+1}) = f$.

Définition 240

Soit I une instance indicée :
Pour tout $k > 0$, un k-chemin de I est un chemin π tel que :

- $\text{vrank}_I(n) < k$, $\text{vrank}_I(n') < k$ où n et n' sont respectivement les premiers et derniers nœuds de π.
- $\forall N \in \pi \setminus \{n, n'\}, \text{vrank}_I(N) = k$
- $\forall A(n, n') \in \pi, \text{erank}_I(A(n, n')) = k$

Finalement nous introduisons un dernier type de chemin :

Définition 241

Soit I une instance indicée. Un chemin ancré de I est un chemin tel que le premier nœud n et le dernier nœud n' vérifient $\text{vrank}_I(n) = \text{vrank}_I(n') = 0$.

5.3.3 Comparaison srs/tgd : cas général

L'objectif de cette section est de prouver :

- $\text{MB}_{\text{max}}(k) \subseteq \beta^k(\text{all}, \mathbf{O})$
- $\beta^k(\text{weak}, \mathbf{S}) \subseteq \text{BPar}(k)$

L'idée générale des preuves consiste à travailler par induction pour construire une dérivation sur les mots à partir d'une **O**-dérivation sur les instances :

pour tout k-chemin π avec $k > 0$ de la dernière instance I_n, il existe un trigger (r, h) tel que

- $h'(H)$ correspond à π
- $h(B)$ est un chemin dont au moins un fait est de profondeur $k-1$.
- On a $u \to v$ où u est le mot associé à B et v le mot associé à H.

On peut alors décomposer $h(B)$ en plusieurs t-chemins et on peut répéter le processus pour construire une dérivation sur les mots. Nous montrerons que cette dérivation a une complexité parallèle égale k.

Ainsi si R est dans $\text{MB}_{\text{max}}(k)$ alors le système de règles tgd associé n'admet pas de **O**-dérivation de profondeur $k+1$, sinon il existerait une dérivation de profondeur $k+1$ sur les mots d'après ce qui précède.

Nous montrons un lemme préliminaire pour les chemins :

Lemme 242

Soient $\mathscr{C}$ un ensemble de contraintes de mots et $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une **O**-dérivation indicée. Soit k un entier > 0. Soit n_f un nœud tel que $\text{vrank}_s(n_f) = k$ et $A_1(n_1, n_f)$, $A_2(n_2, n_f)$ deux faits de profondeur k alors $A_1 = A_2$ et $n_1 = n_2$.

Démonstration. Soit i le plus petit entier tel que n_f est dans $I_{i+1} \setminus I_i$. Alors les faits $A_1(n_1, n_f)$ et $A_2(n_2, n_f)$ appartiennent à $I_{i+1} \setminus I_i$, sinon leurs profondeurs vérifient : $\text{erank}_s(A_j(n_j, n_f)) > \text{vrank}(n_f)$ donc $k > k$ impossible.

On a alors $h_i'^{-1}(A_j(n_j, n_f)) \in H_i$ or $h_i'^{-1}(n_f)$ ne peut pas être en seconde place dans deux atomes car r_i est une contrainte de mots. □

Lemme 243

Soient $\mathscr{C}$ un système de contraintes de mots et $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une **O**-dérivation. Soit k un entier > 0. Pour tout k-chemin π de I_n il existe un entier i et un chemin π' tels que :

— $\text{erank}_s(\pi') \leq k - 1$ (l'égalité est atteinte pour un des faits de π')
— $\pi' = h_i'(B_i)$ et $\pi = h_i'(H_i)$

Démonstration. Soit $A(n, n')$ le fait de π tel que n' est la dernière valeur de π. On va considérer la règle qui a produit ce fait : soit $i = \min\{i \mid A(n, n') \in I_{i+1} \setminus I_i\}$. D'après ce qui a été dit précédemment, puisqu'au moins un fait est produit on a $I_{i+1} \setminus I_i = h_i'(H_i)$ qui est un k'-chemin pour un certain k'. On a $k = k'$ puisque $A(n, n')$ est de profondeur k.

Nous allons prouver que $h_i'(H_i) \subseteq \pi$.
Si $h_i'(H_i)$ contient un unique fait, alors c'est $A(n, n')$. Or $A(n, n') \in \pi$ donc $h_i'(H_i) \subseteq \pi$. Supposons que $h_i'(H_i)$ contiennent au moins deux faits, dont $A(n, n')$. Comme $h_i'(H_i)$ est un k-chemin et que n' est de profondeur $< k$, c'est le dernier nœud de $h_i'(H_i)$, il existe un unique fait contenant n', il s'agit de $A(n, n')$. Puisque il y a au moins deux faits il existe un fait $C(n'', n)$ dans $h_i'(H_i)$ (on remarquera que n est la valeur à gauche de $A(n, n')$). Donc $h_i'^{-1}(n)$ est une variable existentielle car r_i est une contrainte de mot, donc $\text{vrank}_s(n) = k$ par définition de vrank. Comme π est un k-chemin, il existe donc un fait $B(n_0, n)$ de π (on remarquera aussi que n est la première valeur de $A(n, n')$). D'après le lemme 242 les faits sont donc les mêmes, ainsi $C(n'', n) \in \pi$. On peut alors raisonner à nouveau avec $C(n'', n)$ si $h_i'(H_i)$ contient au moins trois faits et par récurrence en déduire que $h_i'(H_i) \subseteq \pi$.

Montrons maintenant que cette inclusion est une égalité : $h'_i(H_i)$ est un chemin contenant au moins un fait, soit n_0 sa première valeur, on a alors $\text{vrank}_s(n_0) < k$ car sinon le fait contenant n_0 dans $h'_i(B_i)$ serait de profondeur au moins k (d'après le Lemme 231) et donc $h'_i(H_i)$ aurait des faits de profondeur $> k$, donc π aussi, contradiction. n_0 est une valeur de π car $h'_i(H_i) \subseteq \pi$, donc n_0 est la première valeur de π. De même les dernières valeurs coïncident donc $h'_i(H_i) = \pi$.

On pose alors $\pi' = H_i(B_i)$, comme r_i est une contrainte de mots c'est un chemin, et comme les faits de $h'_i(H_i)$ sont indicés par k, les faits de $h_i(B_i)$ sont de profondeurs $\leq k-1$ avec égalité pour l'un d'entre eux. Ce qui termine la preuve.

□

Lemme 244

Soient $\mathscr{C}$ un ensemble de contraintes de mots et $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une **O**-dérivation.
Pour tout chemin ancré π_n de I_n il existe un chemin π_0 dans I_0, un entier m et une dérivation $\sigma : x_0 \to^m x_m$ sur le système de réécriture associé R tels que :

- $\text{word}(\pi_0) = x_0$
- $\text{word}(\pi_n) = x_m$
- Pour tout fait f de π_n numéroté par t on a $\text{erank}_s(f) = \text{depth}_\sigma(m, t)$

Démonstration. L'idée est de démontrer le résultat par récurrence sur n. Tout chemin ancré de la dernière instance est obtenu par réécriture depuis un chemin ancré de l'instante précédente. Nous montrons alors que les rangs sont correctement reliés.

Pour le cas où $n = 0$ il n'y a rien à démontrer. On suppose alors la propriété vraie au rang n, on considère donc une **O**-dérivation de longueur $n+1$. On conserve les notations habituelles.

Soit π_{n+1} un chemin ancré de I_{n+1}, s'il est inclus dans I_n il n'y a rien à prouver par hypothèse de récurrence. Sinon π_{n+1} peut donc décomposer en $c_0\pi_1 c_1\pi_2 \ldots c_{a-1}\pi_a c_a$ avec les π_a les chemins dans $I_{n+1} \setminus I_n$. Puisque ces chemins sont dans $I_{n+1} \setminus I_n$ qui est un k-chemin tous les π_j sont égaux à un certain π et correspondent à un k-chemin pour un certain k. On a π qui correspond à $h'_n(H_n)$. Soit π' le chemin $h_n(B_n)$, tous les faits de π' sont de profondeur $\leq k-1$ avec égalité atteinte.

Soit l le mot correspondant à π', r le mot correspondant à π et μ_i les mots correspondant aux c_i. On a $(l, r) \in R$.

On pose π_n le chemin $c_0\pi' c_1 \ldots c_{a-1}\pi' c_a$, ce chemin est est dans I_n. Puisque π_{n+1} est ancré, π_n l'est aussi (ils partagent les mêmes première et dernière valeurs).

Par hypothèse de récurrence il existe un chemin π_0, m et une dérivation $x_0 \longrightarrow^m x_m$ tels que :

— word$(\pi_0) = x_0$
— word$(\pi_n) = x_m$
— Pour tout fait f de π_n numéroté par t , depth$_\sigma(m,t)$ = erank$_s(f)$

Le mot correspondant à π_n est $x_m = \mu_0 l \mu_1 ... \mu_{a-1} l \mu_a$, le mot correspondant à π_{n+1} est $\mu_0 r \mu_1 ... \mu_{a-1} r \mu_a$, on le note y. Puisque $(l,r) \in R$ on a $x_m \multimap y$. Donc il existe un entier m' tel que $x_m \rightarrow^{m'} y$. On pose $m'' = m + m'$. On a $x_0 \rightarrow^{m''} y$. On note σ' la dérivation qui étend σ.

Les deux premières hypothèses de l'énoncé sont vérifiées pour σ'. Vérifions la dernière :

Soit t la numérotation d'un fait f de π_{n+1} :

- Si t est la position d'une lettre d'un μ_j, on a f qui est dans I_n, soit t' tel que $(t',t) \in \text{Corr}_\sigma(n,n+1)$, ainsi t' est la position de la lettre de x_m qui aura la position t dans y, on a depth$_{\sigma'}(m'',t)$ = depth$_{\sigma'}(m,t')$ qui est finalement égale (par hypothèse de récurrence) à erank$_s(f)$ ce qui était voulu.

- Si t est la position d'une lettre d'un des r, on a erank$_s(f) = k$. Pour tous les faits de π' on a un indice inférieur ou égal à $k-1$, avec égalité atteinte pour l'un des faits, donc pour tout t' entre les positions des lettres du mot l correspondant au r dont t est une des positions, depth$_{\sigma'}(m,t') \leq k-1$ avec égalité atteinte pour un t' (par hypothèse de récurrence sur π'). Ainsi, comme $(l,r) \in R$, depth$_{\sigma'}(m'',t) = k$ pour tout t entre la première position et la dernière position du mot r dont t est une position, on a ainsi l'égalité demandée.

Le résultat est démontré.

□

Théorème 245

Soit k un entier, si $R \in \text{MB}_{\max}(k)$ alors le système tgd associé est dans $\beta^k(\text{all}, \mathbf{O})$

Démonstration. Soit R dans MB$_{\max}(k)$, alors soit $s : I_0 \overset{(r_0,h_0)}{\longrightarrow} I_1 ...$ une **O**-dérivation indicée sur une instance I. Soit f un fait, alors il existe un chemin ancré contenant f, donc d'après le Lemme 244, il existe une dérivation $x_0 \longrightarrow^m x_m$ dont les profondeurs des lettres de x_m correspondent aux profondeurs des faits du chemin ancré, les profondeurs sont majorées par k puisque R est dans MB$_{\max}(k)$, donc la profondeur de f est majorée par k, donc $\mathscr{C} \in \beta^k(\text{all}, \mathbf{O})$.

□

Lemme 246

Soient $\mathscr{C}$ un ensemble de contraintes de mots et $\sigma : x_0 \rightarrow_R^n x_n$ une dérivation sur le système de réécriture associé R. Soit I_0 un chemin tel que word$(I_0) = x_0$. Pour toute **O**-dérivation **terminante** $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$ il existe un chemin π dans I_e (où I_e est la dernière instance de s) tel que

— word$(\pi) = x_n$
— n_1 est le premier nœud de I_0
— n_{v+1} est le dernier nœud de I_0

Démonstration. Nous allons démontrer par récurrence sur la longueur de la dérivation σ qu'il existe bien un chemin dans la dernière instance qui vérifie les conditions. Si la dérivation est de longueur nulle alors il suffit de prendre I_0 comme chemin, on a bien I_0 inclus dans toutes les instances de toutes dérivations (terminantes).

On suppose que le résultat est vrai au rang n, soit $\sigma : x_0 \rightarrow^n x_n \rightarrow x_{n+1}$ une dérivation de longueur $n+1$. Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$ une X-dérivation terminante sur I_0 telle que word$(I_0) = x_0$. Soit I_e la dernière instance de s. La sous-dérivation $\sigma' : x_0 \rightarrow^n x_n$ est de longueur n, ainsi par hypothèse de récurrence (on a commencé avec le même mot) il existe un chemin π dans I_e tel que

— word$(\pi) = x_n$
— n_1 est le premier nœud de I_0
— n_{v+1} est le dernier nœud de I_0

Nous devons montrer qu'il existe un chemin π' qui dérive de celui là tel que word$(\pi') = x_{n+1}$.

On a $x_n \rightarrow x_{n+1}$ donc il existe μ, μ' et $(l,r) \in R$ tels que $x_n = \mu l \mu'$ et $x_{n+1} = \mu r \mu'$. Il existe une règle $reg \in \mathscr{C}$ telle que $L(reg) = (l,r)$. De plus, puisque word$(\pi) = x_n$ on peut décomposer π en trois chemins composés $\mu_g b \mu_d$ avec word$(\mu_g) = \mu$, word$(\mu_d) = \mu'$ et word$(b) = l$. Puisque $L(reg) = (l,r)$ il existe un homomorphisme hom tel que $hom(B) = b$, on considère donc le trigger (reg, hom). Puisque la dérivation $s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$ est terminante il existe donc un certaine indice i tel que (reg, hom) a été activé.

Il existe une expression de la forme $h'_i(H_i)$ dans I_e. C'est un chemin car $\mathscr{C}$ est un ensemble de contraintes de mots. De plus le premier nœud de $h'_i(H_i)$ est le premier nœud de $hom(B)$ et le dernier nœud de $hom(B)$ est le dernier nœud de $h'_i(H_i)$. Puisque la règle r_i vérifie $r_i = reg$ on a word$(h'_i(H_i)) = r$ donc par composition avec les chemins μ_g et μ_d (possible vu les premiers et derniers nœuds), il existe un chemin de I_e tel que le premier nœud soit le premier nœud de π et donc de I_0 et de même pour le dernier nœud. C'est ce qu'on voulait démontrer. □

Théorème 247

Soient $\mathscr{C}$ un ensemble de contraintes de mots et k un entier alors :
Si $\mathscr{C}$ est dans $\beta^k(\text{weak}, \mathbf{S})$ alors le système de réécriture R associé est dans BPar(k).

Démonstration. Soit $\sigma : x_0 \to^m x_m$ une dérivation sur R. Soit I_0 le chemin tel que word(I_0) = x_0, comme $\mathscr{C}$ est dans $\beta^k(\text{weak}, \mathbf{S})$ il existe une **S**-dérivation terminante s de profondeur k sur I_0, c'est donc en particulier une **O**-dérivation (qui n'est plus forcément terminante). D'après le Lemme 246, il existe un chemin ancré π dans I_n (le premier nœud et le dernier nœud sont dans I_0) tel que word(π) = x_m, d'après le lemme 244, il existe une dérivation $\sigma' : x_0 \to^m x_m$ dont les profondeurs de x_m sont égaux aux profondeurs de π donc inférieures ou égales à k, ainsi R est dans BPar(k). □

Proposition 248

Les classes $CT^{\mathbf{S}}_{\forall\exists}$ et BPar sont incomparables.

Démonstration. 1) Le système des contraintes de mots associé à $aa \to a$ appartient à $CT^{\mathbf{S}}_{\forall\exists}$ mais n'appartient pas à BPar.

2) Inversement le système $\mathfrak{R} = aa \to bb, aa \to ab$ est dans BPar(2). Soit $\mathscr{C}$ le système de contraintes de mots associé à R.

Prouvons que $\mathscr{C}$ ne termine pas sur l'instance $I = \{A(e,e)\}$:

On pose :

$$P = A(x,y), A(y,z) \to A(x,w), B(w,z)$$

$$Q = A(x,y), A(y,z) \to B(x,w), B(w,z)$$

Soit $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ une **S**-dérivation démarrant sur I.

Notons d'abord en examinant les deux règles que pour tout I_i, si $A(p,q)$ est dans I_i , $p = e$.

Montrons par récurrence que pour tout i :

"il existe un nœud n tel que

$A(e,n) \in I_i$ et qui n'admet pas de c tel que $A(e,c), B(c,n) \in I_i$"

L'initialisation est facile, supposons la propriété vraie au rang i.

On pose $new = h'_i(w)$.

A) Supposons que $r_i = P$. On pose $n = new$, on a alors $h'_i(H_i) = \{A(e,n), B(n,n')\}$ avec $n' \neq n$; comme $n \notin I_i$ le seul fait de I_{i+1} de prédicat A dont le second membre

est n est donc $A(e,n)$, et $B(n,n)$n'est pas I_{i+1}. Donc il n'existe pas de c tel que $A(e,c)$ et $B(c,n)$ soient dans I_{i+1}.

B) Supposons que $r_i = Q$. Par hypothèse de récurrence il existe n tel que $A(e,n) \in I_i$ et qui n'admet pas de c tel que $A(e,c), B(c,n) \in I_i$. Comme $I_i \subseteq I_{i+1}$ on a $A(e,n) \in I_{i+1}$.

Supposons qu'il existe c tel que $A(e,c)$ et $B(c,n)$ soient dans I_{i+1}. Distinguons deux cas :

- Si $B(c,n) \in I_i$ alors comme $h'_i(H_i)$ ne contient aucun fait de prédicat A on a $A(e,c) \in I_i$, contradiction avec l'hypothèse de récurrence.

- Si $B(c,n) \notin I_i$ alors comme $h'_i(H_i) = \{B(e,new), B(new,n)\}$ on a l'une des deux possibilités suivantes :

Soit $B(c,n) = B(e,new)$ et donc $n = new$ or $new \notin I_i$, contradiction.

Soit $B(c,n) = B(new,n)$ et donc $c = new$, or $A(e,c) \in I_i$ (car $h'_i(H_i)$ n'a pas de fait de prédicat A), donc $A(e,new) \in I_i$ donc $new \in I_i$, contradiction.

Montrons maintenant qu'il existe un trigger actif sur I_i pour tout i,

Posons h_i défini par $h(x) = h_i(y) = e$ et $h_i(z) = n$, avec n défini comme ci-dessus. Il n'existe pas de c tel que $A(e,c)$ et $B(c,n)$ soient des faits de I_i, donc il n'existe pas d'extension h' de h telle que $h'(H_i) \subseteq I_i$. Ainsi (P,h_i) est actif sur I_i.

Donc il existe une dérivation non terminante sur $\{A(e,e)\}$.

□

Nous pouvons faire une dernière remarque afin de relier l'étude du chase et la propriété d'être descendant-limité pour les systèmes de réécriture.

Si un système de contraintes de mots termine pour toute instance dans le cas oblivious alors le système de réécriture associé est descendant-limité :

Proposition 249

Soit $\mathscr{C}$ un ensemble de contraintes de mots. Soit R le système de réécriture associé. Si l'oblivious-chase termine alors R est descendant-limité.

Démonstration. Soit u un mot. On considère v un descendant de u, on a $u \to^* v$. D'après le lemme 246, pour toute **O**-dérivation qui démarre sur l'instance associée à G_u (graphe-mot associé a u), il existe un chemin dans la dernière instance (la dérivation termine) étiqueté par v dont le premier nœud est le premier nœud de G_u et de même pour le dernier nœud.

Montrons maintenant que $\mathscr{C}^\infty(G_u)$ est acyclique.

On considère une **O**-dérivation terminante

$$s : I_0 \xrightarrow{(r_0,h_0)} I_1 ...$$

telle que $I_n = \mathscr{C}^\infty(G_u)$.

Montrons par récurrence que pour tout i, on a I_i acyclique.

- **Initialisation :** On a $I_0 = G_u$ acyclique.

- **Hérédité :** Supposons que I_i est acyclique. On a $I_{i+1} = I_i \cup h'_i(H_i)$ où h'_i est une extension de h_i et H_i la tête de r_i. Puisque I_i est acyclique, tout cycle de I_{i+1} contient au moins un fait de $I_{i+1} \setminus I_i$.

Supposons que $I_{i+1} \setminus I_i$ ne contienne pas de nouveaux nœuds. Puisque r_i est une contrainte de mots, s'il n'y a pas de variable existentielle dans H_i alors r_i est de la forme $A_1(x,y) \to B_1(x,y)$, donc si un cycle contient un fait de $I_{i+1} \setminus I_i$ il s'agit de $h_i(B_1(x,y))$, donc il existe aussi un cycle dans I_i en considérant $h_i(A_1(x,y))$ au lieu de cette arête, car ces deux arêtes ont les mêmes extrémités. Donc I_i n'est pas acyclique, contradiction.

Supposons maintenant que $I_{i+1} \setminus I_i$ contienne des nouveaux nœuds. Alors puisque r_i est une contrainte de mots, elle est de la forme

$$A_1(x_1,x_2),...,A_n(x_n,x_{n+1}) \to B_1(y_1,y_2),...,B_m(y_m,y_{m+1})$$

avec toutes variables les y_i qui sont existentielles, $y_1 = x_1$ et $y_{m+1} = x_{n+1}$.

Considérons un cycle dans I_{i+1}, il contient donc un nouveau fait sinon I_i est cyclique.

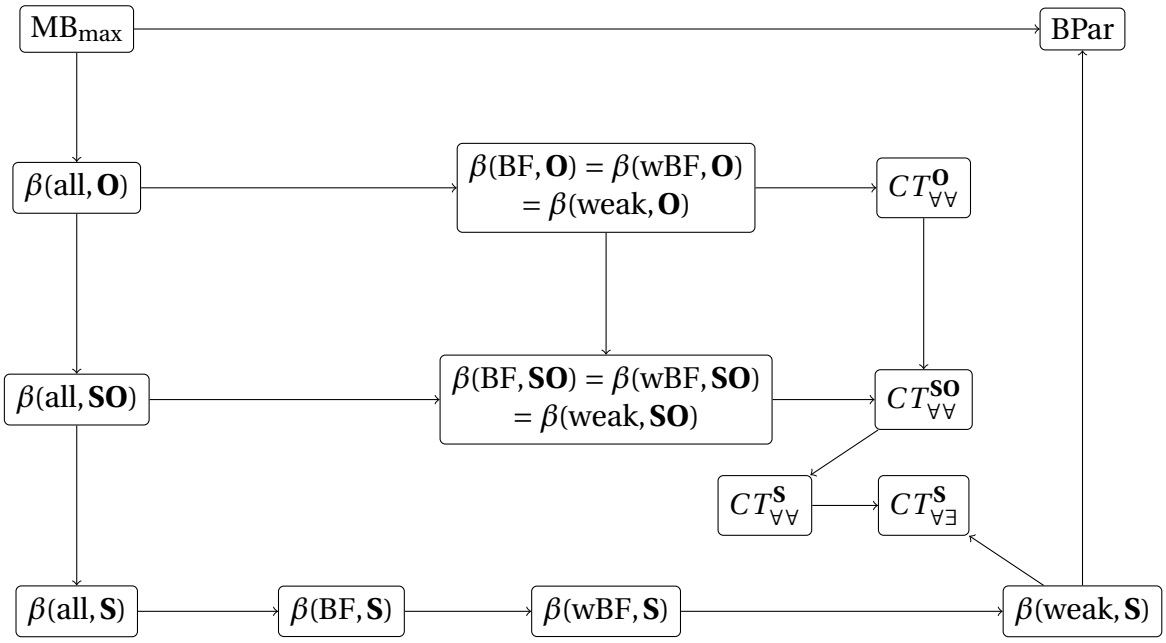
- Si le fait est de la forme $h'_i(B_k(y_k,y_{k+1}))$ avec k entre 2 et m alors : il existe un unique fait dont $h'_i(y_k)$ est membre droit, il s'agit du fait $h'_i(B_{k-1}(y_{k-1},y_k))$, donc ce fait appartient au cycle.

- Si le fait est de la forme $h'_i(B_k(y_k,y_{k+1}))$ avec k entre 1 et $m-1$ alors : il existe un unique fait dont $h'_i(y_{k+1})$ est membre gauche, il s'agit du fait $h'_i(B_{k+1}(y_{k+1},y_{k+1}))$.

Donc, le cycle contient tous les faits de $h'_i(H_i)$. C'est un chemin entre $h'_i(x_1)$ et $h'_i(x_{n+1})$ car $x_1 = y_1$ et $x_{n+1} = y_{m+1}$. Or il existe un chemin dans I_i entre ces mêmes nœuds : c'est le chemin $h_i(B_i)$. Donc en remplaçant $h'_i(H_i)$ par $h_i(B_i)$ il existe un cycle dans I_i. Contradiction. Ainsi I_{i+1} est acyclique.

Puisque $\mathscr{C}^\infty(G_u)$ est acyclique, il existe un nombre fini de chemins entre le premier nœud et le dernier nœud de G_u, donc un nombre fini de descendants pour u. □

Nous concluons comme annoncé dans le résumé par ce schéma récapitulatif. Nous prendrons garde au fait que nous ne savons pas pour certaines inclusions si elles sont, ou non, strictes.



5.3.4 Comparaison srs/tgd : cas total

Nous comparons les classes de la sous-section précédente dans le cas où les règles sont totales.

Lemme 250

Soient $\mathscr{C}$ un ensemble de contraintes de mots **totales** et $\sigma : x_0 \rightarrow^m x_m$ une dérivation sur le système de réécriture associé R. Soit I_0 un chemin tel que $\text{word}(I_0) = x_0$. Pour toute **O**-dérivation **terminante** $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ il existe un chemin π dans I_n tel que

- $\text{word}(\pi) = x_m$
- le premier nœud de π est le premier nœud de I_0
- le dernier nœud de π est le dernier nœud de I_0
- Pour tout fait f de π numéroté par t on a $\text{erank}_s(f) = \text{depth}_\sigma(m,t)$

Démonstration. On montre le résultat par récurrence, on utilise à nouveau les notations de la preuve du Lemme 246.

L'étape d'initialisation est évidente.

On pose $\pi = I_0$, alors π vérifie facilement les conditions demandées. On considère une dérivation $\sigma = x_0 \rightarrow^m x_m \rightarrow x_{m+1}$.

On décompose $x_m = \mu l \mu'$ et $x_{m+1} = \mu r \mu'$ avec $(l,r) \in R$.

La sous dérivation $\sigma' = x_0 \rightarrow^n x_m$ est de longueur m, il existe un indice i tel que $\text{word}(h_i(B_i)) = l$ et $\text{word}(h'_i(H_i)) = r$.

On pose $\pi = \mu_g h'_i(H_i) \mu_d$ avec μ_g et μ_d définis comme au lemme 246.

Nous devons maintenant vérifier la condition sur les profondeurs pour $\mu_g h'_i(H_i) \mu_d$.

On a pour $t \leq |\mu|$ et pour $t > |\mu r|$, $\text{depth}_\sigma(m+1,t) = \text{depth}_\sigma(m,\tau) = \text{depth}_{\sigma'}(m,\tau)$ (avec τ la position de la lettre $x_{m+1}[t]$ dans x_m, i.e. $(\tau,t) \in \text{Corr}_\sigma(m,m+1)$) qui par récurrence correspond à la profondeur du fait de $\mu_g \cup \mu_d$ numéroté par τ.

Il reste donc le cas où t est entre $|\mu|+1$ et $|\mu r|$, c'est-à-dire quand la position t correspond à une lettre qui vient d'être produite dans x_{m+1} de σ. Les faits numérotés par de tels t dans le chemin $\pi' := \mu_g h'_i(H_i) \mu_d$ de I_e sont exactement les faits de $h'_i(H_i)$, or la règle $B_i \rightarrow H_i$ est totale donc $h'_i(H_i) = I_{i+1} \setminus I_i$, donc les faits de f de $h'_i(H_i)$ vérifient $\text{erank}_s(f) = 1 + \max_{g \in h_i(B_i)} \text{erank}_s(g)$.

Les faits g de $h_i(B_i)$ sont exactement les faits de π qui sont numérotés entre $|\mu|+1$ et $|\mu l|$ et vérifient : si g est numéroté par t : $\text{erank}_s(g) = \text{depth}_{\sigma'}(m,t) = \text{depth}_\sigma(m,t)$.

Enfin on sait que $\text{depth}_\sigma(m+1,t) = 1 + \max_{t \in [|\mu|+1, |\mu l|]} \text{depth}_\sigma(m,t)$.

Ainsi les faits f de $h'_i(H_i)$ et donc tous les faits de π' vérifient : si f est numéroté par t dans π' alors $\text{erank}_s(f) = \text{depth}_\sigma(m+1,t)$. Ce qui achève la récurrence.

□

Théorème 251

Pour tous k et système de contraintes de mots **totales** $\mathscr{C}$ dont le système de réécriture R est associé :

$$R \in \mathrm{MB}_{\max}(k) \text{ si et seulement si } \mathscr{C} \in \beta^k(\mathrm{all}, \mathbf{O})$$

Démonstration. On fixe un entier k.

L'implication $\Rightarrow$ est déjà donnée par le théorème 245.

Soit $\mathscr{C}$ un système de contraintes de mots totales dans $\beta^k(\mathrm{all}, \mathbf{O})$ et R le système de réécriture associé. Soit $\sigma = x_0 \to^n x_n$ une dérivation sur R.

D'après le Lemme 250, pour toute dérivation $s : I_0 \xrightarrow{(r_0,h_0)} I_1 \ldots$ terminante telle que $\mathrm{word}(I_0) = x_0$, il existe un chemin π dans l'instance finale tel que $\mathrm{word}(\pi) = x_n$ et tel que $\mathrm{depth}_\sigma(n,t) = \mathrm{erank}_s(f)$ où f est le fait numéroté par t dans π, donc majoré par k car $\mathscr{C} \in \beta^k(\mathrm{all}, \mathbf{O})$. Ainsi $R \in \mathrm{MB}_{\max}(k)$. □

Complexité L'égalité des classes $\mathrm{MB}_{\max}(k)_t$ et $\beta^k(\mathrm{all}, \mathbf{O})_t = \beta^k(\mathrm{BF}, \mathbf{O})_t$ donne la possibilité d'utiliser l'algorithme du second chapitre pour tester si un système est borné par k.

Théorème 252

On a :

PROBLEME : Borne uniforme des contraintes de mots dans le cas BF et total
INSTANCE : $\mathscr{C}$ un systèmes de contraintes de mots totales, k un naturel
QUESTION : $\mathscr{C} \in \beta^k(\mathrm{BF}, \mathbf{O})$?
COMPLEXITE : PSPACE-complet

Démonstration. À tout système de réécriture $\mathfrak{R}$ on associe le système de réécriture $\#\mathfrak{R}$ tel que : pour toute règle $(a_1 \ldots a_n, b_1 \ldots b_m)$ on associe $(\#a_1 \ldots \#a_n, \#b_1 \ldots \#b_m)$ où $\# \notin \Sigma$, cette construction est polynomiale en la taille de $\mathfrak{R}$. Il est facile de vérifier que pour tout k , $\mathfrak{R}$ est $\mathrm{MB}_{\max}(k)$ ssi $\#\mathfrak{R}$ l'est.

Ensuite on associe à $\#\mathfrak{R}$ le système de contraintes de mots totales $\mathscr{C}$ dont la construction est aussi polynomiale.

On a d'après le théorème 251, $\mathscr{C} \in \beta^k(\text{BF}, \mathbf{O})$ si et seulement. $\#\mathfrak{R} \in \text{MB}_{\max}(k)$ donc ssi $\mathfrak{R} \in \text{MB}_{\max}(k)$.

Ainsi on a établi une réduction polynomiale du problème $\mathfrak{R} \in \text{MB}_{\max}(k)$ dans $\mathscr{C} \in \beta^k(\text{BF}, \mathbf{O})$; donc, comme nous avons montré dans le chapitre précédent que le problème $\mathfrak{R} \in \text{MB}_{\max}(k)$ est PSPACE-difficile, le problème $\mathscr{C} \in \beta^k(\text{BF}, \mathbf{O})$ est aussi PSPACE-difficile.

De plus, si $\mathscr{C}$ est un système de règles tgd totales alors d'après le théorème 251, on exécute l'algorithme PSPACE de décision $\mathfrak{R} \in \text{MB}_{\max}(k)$ pour les entrées $\#\mathfrak{R}$ et k pour décider si $\mathscr{C} \in \beta^k(\text{BF}, \mathbf{O})$.

□

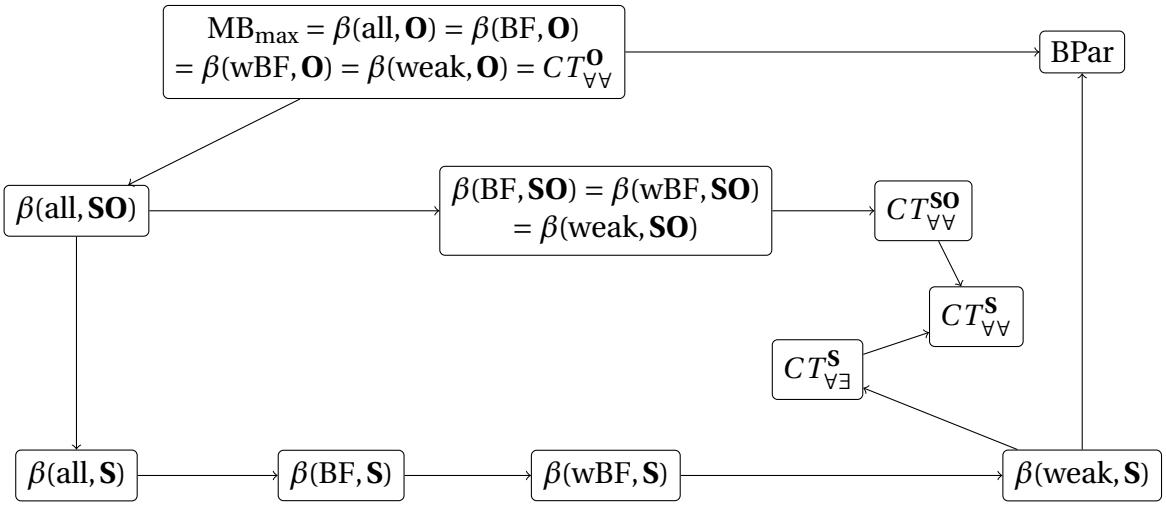
Dans le cas total, le problème de décision $R \in \text{MB}_{\max}$ laissé ouvert est équivalent au problème de décider si un système de contraintes de mots est $\mathbf{O}$-terminant.

La terminaison des règles tgd en arité 2 est un cas connu comme difficile de décision de terminaison [55, 56] :

— en arité 1 la terminaison est connue décidable
— en arité > 2 la terminaison est connue comme indécidable
— en arité 2 le problème est toujours ouvert (aussi en Datalog)

Ainsi la difficulté de problème comme $R \in \text{MB}_{\max}$ et les problèmes de terminaison en arité 2 sont liés et une étude des systèmes de réécritures dans ce contexte peut apporter d'autres éléments de réponses quant à l'étude des bornes ou des terminaisons des règles tgd.

Nous résumons l'ensemble des inclusions obtenues dans le cadre des contraintes de mots totales :



5.4 Conclusion

Dans ce chapitre nous avons mis en lumière la notion de borne uniforme de systèmes tgd. Nous avons comparé l'évolution des profondeurs des faits dans deux cas : le cas total et le cas non-total. Puis nous avons spécialisé cette comparaison dans le cas des contraintes de mots. Nous avons montré qu'il existe un algorithme en Co-NEXPTIME (resp. 2EXPTIME) qui décide si un système de règles tgd est uniformément borné par k (resp. en stratégie breadth-first) en oblivious et semi-oblivious. Nous avons montré que deux problèmes sont équivalents dans le cas total.

Dans le cas total et oblivious, en s'inspirant du critère de terminaison sur l'instance critique on a montré qu'être uniformément borné était équivalent à être terminant.

On s'est intéressé plus particulièrement aux contraintes de mots. La notion de borne uniforme est mise en comparaison avec les profondeurs des lettres dans les systèmes de réécritures : on montre que si un système de réécriture est de complexité parallèle bornée par k alors le système tgd est uniformément borné par k, et que si un système admet pour toute instance une dérivation de profondeur bornée par k alors le système est BPar(k).

Dans le cas des contraintes de mots totales, les profondeurs des faits par la procédure du chase et la profondeur des lettres par les systèmes de réécritures sont égales : les classes $\mathrm{MB}_{\max}(k)$ et "être uniformément borné par k" correspondent et donc décider si un système de règles tgd est uniformément borné par k est PSPACE-complet.

Finalement, on a montré que dans le cas total, le problème de décision $R \in \mathrm{MB}_{\max}$ est équivalent à décider si un système de contraintes de mots termine.

CHAPITRE 6 Conclusion

L'objectif de la thèse est d'établir des liens entre les systèmes de réécriture et la gestion des bases de données graphe. Dans ce cadre, on s'est intéressé à étudier ces liens dans deux thématiques : les requêtes de chemin et la procédure du chase sous les contraintes de mots. Des liens existants ont été proposés [3, 18, 7, 57]. Nous avons revisité cette approche en apportant des réponses plus précises à certaines questions et en corrigeant un résultat erroné de [57].

On s'intéresse à deux problèmes principaux sur les graphes et avons montré des liens avec les systèmes de réécriture pour chacun d'eux :

— L'inclusion de requêtes RPQ sous contraintes de mots
— L'existence d'une borne uniforme sur toutes les instances assurant la terminaison du chase

Inspirés par le parallèle entre la complétion d'automates et la procédure du chase, nous avons identifié des classes de systèmes de réécriture permettant de réduire le problème d'inclusion de requêtes RPQ sous contraintes de mots au problème d'inclusion de langages réguliers.

Ces classes s'appuient sur une notion de complexité parallèle, la première borne uniformément la complétion des automates i.e. après k étapes de complétions l'algorithme termine. Pour la seconde classe, la complétion ne termine pas après k étapes mais le langage reconnu est celui des ancêtres.

Nous avons montré que si les contraintes de mots correspondent à un système d'une de ces classes alors le problème d'inclusion de requêtes RPQ sous contraintes est EXPSPACE-complet. Nous avons aussi montré que décider si un système est $\mathrm{MB}_{\max}(k)$ est PSPACE-complet.

Nous avons regardé à quelles propriétés sur la terminaison du chase les contraintes de mots de ces classes correspondent. Cela nous a conduit à étudier la notion de borne uniforme et nous avons montré que dans le cas total être uniformément borné par k est équivalent à être $\mathrm{MB}_{\max}(k)$.

Nous avons étudié de façon plus générale différentes méthodes de borner uniformément les systèmes tgd et les a vons comparés. En particulier, nous avons montré que dans le cas total elles sont toutes égales et être uniformément borné est équivalent à être terminant. Nous avons étudié la complexité du problème d'être uniformément par k pour ces différentes méthodes.

Perspectives Nous avons mis en lumière des liens forts entre les systèmes de réécriture et les données. Mais le travail de cette thèse n'est qu'une première étape.

Un pas supplémentaire pourrait consister à regarder comment décider plus de propriétés sur les objets que nous avons définis. Est-ce que nous pouvons décider si un système de réécriture est $\mathrm{MB}_{\max}$? Ou si R est $\mathrm{MB}_{\max}$ est-ce que sa complexité est calculable ? Une extension naturelle du graphe-lettre d'un système à un graphe représentant les "overlaps" des membres gauches et droits des règles devrait étendre la classe NR et fournir un critère décidable suffisant (mais non nécéssaire) pour qu'un système soit $\mathrm{MB}_{\max}$.

Nous pouvons aussi chercher à étendre les classes de contraintes considérées à des problèmes représentatifs des bases de données graphes. D'une part nous pouvons par exemple ajouter des contraintes de clôture transitive $aa \to a$ aux classes. D'autre part, nous pouvons considérer $\mathrm{MB}_{\min}$ et tenter de construire une variante de chase dont la terminaison correspond à $\mathrm{MB}_{\min}$ ou pour des sous-classes raisonnables. L'oblivious-chase ne termine pas pour les systèmes $\mathrm{MB}_{\min}$: $ab \to bc$. Dans le cadre de l'inclusion des requêtes RPQ, nous avons imposé d'être descendant-limité, nous pouvons proposer d'autres conditions suffisantes de réduction pour capturer de nouvelles classes.

Un dernier point que nous aimerions aborder consiste à généraliser les contraintes de mots avec des lettres inverses. Nous pouvons traduire ces contraintes en réécriture sur les groupes. Ces propriétés d'inversions pourraient permettre une nouvelle étude du problème d'inclusion de requêtes UC2RPQ où les bases de données graphes peuvent être parcourues dans les deux sens. Ces requêtes sont essentielles dans des langages de requêtes de graphe comme SparQL, Cypher, ...

Bibliographie

[1] Termination portal. http://termination-portal.org/wiki/Home.

[2] Serge Abiteboul. Boundedness is undecidable for datalog programs with a single recursive rule. *Information Processing Letters*, 32(6) :281 – 287, 1989.

[3] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3) :428 – 452, 1999.

[4] Natasha Alechina, Stephane Demri, and Maarten Rijke. A modal perspective on path constraints. *J. Log. Comput.*, 13 :939–956, 01 2003.

[5] Faisal Alkhateeb, Jean-François Baget, and Jérôme Euzenat. Extending sparql with regular expression patterns (for querying rdf). *Web Semantics : Science, Services and Agents on the World Wide Web*, 7(2), 2009.

[6] Yves André, Anne-Cécile Caron, Denis Debarbieux, Yves Roos, and Sophie Tison. Extraction and Implication of Path Constraints. In *29th Symposium on Mathematical Foundations of Computer Science*, volume 3153 of *Lecture Notes in Computer Science*, pages 863–875, Prague, Costa Rica, 2004. Springer.

[7] Yves André, Anne-Cécile Caron, Denis Debarbieux, Yves Roos, and Sophie Tison. Path constraints in semistructured data. *Theoretical Computer Science*, 385(1-3) :11–33, October 2007.

[8] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5) :68 :1–68 :40, September 2017.

[9] M Arenas, P Barceló, Leonid Libkin, and Filip Murlak. Foundations of data exchange. *Foundations of Data Exchange*, pages 1–331, 01 2010.

[10] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1) :133 – 178, 2000.

[11] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables : Walking the decidability line. *Artificial Intelligence*, 175(9) :1620 – 1654, 2011.

[12] H. P. Barendregt. The lambda calculus. its syntax and semantics. studies in logic and foundations of mathematics, vol. 103. north-holland publishing company, amsterdam, new york, and oxford, 1981, xiv 615 pp. *Journal of Symbolic Logic*, 49(1) :301–303, 1984.

[13] Michael Benedikt and Georg Gottlob. The impact of virtual views on containment. *PVLDB*, 3 :297–308, 09 2010.

[14] M. Bezem, J.W. Klop, R. de Vrijer, and Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretica. Cambridge University Press, 2003.

[15] J. Richard BFuchi and Dirk Siefkes. *Finite Automata, Their Algebras and Grammars*. Springer-Verlag, Berlin, Heidelberg, 1990.

[16] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer, 1993.

[17] Pierre Bourhis, Michel Leclère, Marie-Laure Mugnier, Sophie Tison, Federico Ulliana, and Lily Gallois. Oblivious and semi-oblivious boundedness for existential rules. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1581–1587. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[18] Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2) :146 – 193, 2000.

[19] Peter Buneman, Wenfei Fan, and Scott Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. 06 2000.

[20] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14 :57 – 83, 2012. Special Issue on Dealing with the Messiness of the Web of Data.

[21] Marco Calautti, Georg Gottlob, and Andreas Pieris. Chase termination for guarded existential rules. In *Proceedings of the 9th Alberto Mendelzon International Workshop on Foundations of Data Management, Lima, Peru, May 6 - 8, 2015.*, 2015.

[22] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase : Query answering under expressive relational constraints. *J. Artif. Int. Res.*, 48(1) :115–174, October 2013.

[23] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. *Datalog Extensions for Tractable Query Answering over Ontologies*, pages 249–279. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[24] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4) :83–92, 2003.

[25] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing for regular path queries with inverse. In *PODS*, 2000.

[26] Diego Calvanese, Maurizio Lenzerini, and Moshe Vardi. Containment of conjunctive regular path queries with inverse. *Proc. of KR 2000*, 02 2000.

[27] Didier Caucal. On the regular structure of prefix rewritings. Research Report RR-1196, INRIA, 1990. Projet MICAS.

[28] Didier Caucal. On word rewriting systems having a rational derivation. pages 48–62, 03 2000.

[29] Didier Caucal and Dinh Trong Hieu. Regularity and context-freeness over word rewriting systems. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2011.

[30] Ashok K. Chandra and David Harel. Horn clause queries and generalizations. *The Journal of Logic Programming*, 2(1) :1 – 15, 1985.

[31] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

[32] Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. pages 148–162, 09 2007.

[33] Thierry Coquand and Henrik Persson. A proof-theoretical investigation of zantema's problem. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic*, pages 177–188, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[34] Stathis Delivorias. *Chase Variants and Boundedness.* Thèse de doctorat, Université de Montpellier, 2019.

[35] Stathis Delivorias, Michel Leclère, Marie-Laure Mugnier, and Federico Ulliana. On the k-Boundedness for Existential Rules. In *RuleML+RR : Rules and Reasoning*, volume LNCS, pages 48–64, Luxembourg, Luxembourg, September 2018.

[36] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3) :279 – 301, 1982.

[37] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1) :69 – 115, 1987.

[38] Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. In *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, pages 149–158, New York, NY, USA, 2008. ACM.

[39] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1) :47–68, Jan 1965.

[40] Ronald Fagin, Phokion Kolaitis, Renée Miller, and Lucian Popa. Data exchange : Semantics and query answering. *Theoretical Computer Science*, 336, 01 2003.

[41] Haim Gaifman, Harry Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3) :683–713, July 1993.

[42] John P. Gallagher, Mai Ajspur, and Bishoksan Kafle. An optimised algorithm for determinisation and completion of finite tree automata. *CoRR*, abs/1511.03595, 2015.

[43] Hervé Gallaire and Jack Minker. Logic and data bases, symposium on logic and data bases, centre d'études et de recherches de toulouse, 1977. 1978.

[44] Thomas Genet. Termination criteria for tree automata completion. *Journal of Logical and Algebraic Methods in Programming*, 85(1, Part 1) :3 – 33, 2016. Rewriting Logic and its Applications.

[45] Thomas Genet. Automata Completion and Regularity Preservation. Research report, IRISA, Inria Rennes, April 2017.

[46] Thomas Genet. Completeness of Tree Automata Completion. In *FSCD 2018 - 3rd International Conference on Formal Structures for Computation and Deduction*, pages 1–20, Oxford, United Kingdom, July 2018.

[47] Alfons Geser. A solution to zantema's problem. Technical report, Fakultät für Mathematik und Informatik, Universität Passau, 1993.

[48] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. In *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, pages 449–459, 2003.

[49] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Termination proofs for string rewriting systems via inverse match-bounds. *J. Autom. Reasoning*, 34(4) :365–385, 2005.

[50] Rémi Gilleron and Sophie Tison. Regular Tree Languages and Rewrite Systems. *Fundamenta Informaticae*, 24(1/2) :157–176, 1995.

[51] Tomasz Gogacz and Jerzy Marcinkowski. *All–Instances Termination of Chase is Undecidable*, pages 293–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[52] Tomasz Gogacz, Jerzy Marcinkowski, and Andreas Pieris. All-instances restricted chase termination : The guarded case. *CoRR*, abs/1901.03897, 2019.

[53] Joseph Goguen, Claude Kirchner, and José Meseguer. *Concurrent term rewriting as a model of computation*, pages 53–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.

[54] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artificial Intelligence*, 213 :42 – 59, 2014.

[55] Gösta Grahne and Adrian Onet. The data-exchange chase under the microscope. *CoRR*, abs/1407.2279, 2014.

[56] Gösta Grahne and Adrian Onet. Anatomy of the chase. *Fundam. Inform.*, 157(3) :221–270, 2018.

[57] Gösta Grahne and Alex Thomo. Query containment and rewriting using views for regular path queries under constraints. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 111–122, 2003.

[58] André Hernich. Computing universal models under guarded tgds. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 222–235, New York, NY, USA, 2012. ACM.

[59] Gerd G Hillebrand, Paris C Kanellakis, Harry G Mairson, and Moshe Y Vardi. Undecidable boundedness problems for datalog programs. *The Journal of Logic Programming*, 25(2) :163 – 190, 1995.

[60] Dieter Hofbauer and Johannes Waldmann. Deleting string rewriting systems preserve regularity. *Theor. Comput. Sci.*, 327(3) :301–317, 2004.

[61] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. Datalog and emerging applications : An interactive tutorial. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1213–1216, New York, NY, USA, 2011. ACM.

[62] Gérard P. Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, ii. In *Computational Logic-Essays in Honor of Alan Robinson*, pages 415–443, 1991.

[63] Gérard Huet. Confluent reductions : Abstract properties and applications to term rewriting systems : Abstract properties and applications to term rewriting systems. 27 :797–821, 10 1980.

[64] Martin Hyland. A syntactic characterization of the equality in some models for the lambda calculus. *Journal of the London Mathematical Society*, 2(3) :361–370, 1976.

[65] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4) :761–791, September 1984.

[66] Arash Karimi, Heng Zhang, and Jia-Huai You. Restricted chase termination : A hierarchical approach and experimentation. In Christoph Benzmüller, Francesco Ricca, Xavier Parent, and Dumitru Roman, editors, *Rules and Reasoning*, pages 98–114, Cham, 2018. Springer International Publishing.

[67] Claude Kirchner and Patrick Viry. *Implementing parallel rewriting*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[68] Jan Willem Klop and Aart Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, 12(2) :161 – 195, 1991.

[69] Markus Krötzsch, Maximilian Marx, and Sebastian Rudolph. The power of the terminating chase. In Pablo Barceló and Marco Calautti, editors, *Proceedings of the 22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *LIPIcs*, pages 3 :1–3 :17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[70] Dallas Lankford. On proving term rewriting systems are noetherian. 1979.

[71] Ora Lassila, Ralph R. Swick, World Wide, and Web Consortium. Resource description framework (rdf) model and syntax specification, 1998.

[72] Jean-Jacques Levy. *Réduction correctes et optimales dans le λ-calcul*. Thèse de doctorat, Université Paris 7, 1978.

[73] Katja Losemann and Wim Martens. The complexity of regular expressions and property paths in sparql. 38, 11 2013.

[74] Jerzy Marcinkowski. Achilles, turtle, and undecidable boundedness problems for small DATALOG programs. *SIAM J. Comput.*, 29(1) :231–257, 1999.

[75] Bruno Marnette. Generalized schema-mappings : From termination to tractability. pages 13–22, 01 2009.

[76] Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *CoRR*, abs/0906.4228, 2009.

[77] Alberto O. Mendelzon. Database states and their tableaux. *ACM Trans. Database Syst.*, 9 :264–282, 1981.

[78] Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. In Peter M. G. Apers and Gio Wiederhold, editors, *Proceedings of the Fifteenth International Conference on Very Large Data Bases, August 22-25, 1989, Amsterdam, The Netherlands.*, pages 185–193. Morgan Kaufmann, 1989.

[79] Adrian Onet. The chase procedure and its applications in data exchange. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 1–37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[80] Friedrich Otto. *On the property of preserving regularity for string-rewriting systems*, pages 83–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[81] Friedrich Otto. Preserving regularity and related properties of string-rewriting systems. 12 1997.

[82] Emil L. Post. Recursive unsolvability of a problem of thue. *Journal of Symbolic Logic*, 12(1) :1–11, 1947.

[83] J. Richard Büchi. Regular canonical systems. *Archiv für mathematische Logik und Grundlagenforschung*, 6(3) :91–111, Apr 1964.

[84] Sylvain Salvati. Communication privée, 2019.

[85] Oded Shmueli. Equivalence of datalog queries is undecidable. *The Journal of Logic Programming*, 15(3) :231 – 241, 1993.

[86] A. Tarski, A. Mostowski, and R.M. Robinson. *Undecidable theories*. Studies in logic and the foundations of mathematics. North-Holland Pub. Co., 1953.

[87] Moshe Y. Vardi. A theory of regular queries. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 1–9, New York, NY, USA, 2016. ACM.

[88] Nikolay Yakovets, Parke Godfrey, and Jarek Gryz. WAVEGUIDE : evaluating SPARQL property path queries. In Gustavo Alonso, Floris Geerts, Lucian Popa, Pablo Barceló, Jens Teubner, Martín Ugarte, Jan Van den Bussche, and Jan Paredaens, editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 525–528. OpenProceedings.org, 2015.

[89] H. Zantema. Termination of term rewriting by semantic labelling. *Fundam. Inf.*, 24(1-2) :89–105, April 1995.

[90] H. Zantema. Termination of string rewriting proved automatically. *Journal of Automated Reasoning*, 34(2) :105–139, 2005.

[91] Heng Zhang, Yan Zhang, and Jia-Huai You. Existential rule languages with finite chase : Complexity and expressiveness. *CoRR*, abs/1411.5220, 2014.