Institut National Polytechnique de Grenoble

N^o attribué par la bibliothèque

Thèse

pour obtenir le grade de

Docteur de l'Inpg

Spécialité : "Informatique : Systèmes et Communications"

préparée au laboratoire Leibniz-Imag

dans le cadre de l'Ecole Doctorale
"Mathématiques, Sciences et Technologie de l'Information"

présentée et soutenue publiquement

par

Jean-Christophe Janodet

le 24 janvier 2000

Sur les types de données dans les langages logico-fonctionnels : Réécriture et surréduction des graphes admissibles

---

Directeur de thèse : M. Philippe Jorrand

Jury

Mme Brigitte Plateau	Président
M. Hubert Comon	Rapporteur
M. Michael Rusinowitch	Rapporteur
M. Philippe Jorrand	Directeur de thèse
M. Rachid Echahed	Co-encadrant
M. Didier Bert	Examinateur

A Emmanuelle.

"Trois allumettes une à une allumées dans la nuit,
la première pour voir ton visage tout entier,
la seconde pour voir tes yeux,
la dernière pour voir ta bouche,
et l'obscurité toute entière
pour me rappeler tout cela,
en te serrant dans mes bras."

Paris at night, *Yves Montant & Jacques Prévert.*

Remerciements

Je remercie tout d'abord mon directeur de thèse, M. Rachid Echahed, pour son aide précieuse, et les encouragements appréciés qu'il n'a cessé de me prodiguer pendant toutes les années de mon troisième cycle. C'est lui qui m'a proposé un thème de recherche et c'est avec lui que j'ai appris à chercher, à trouver, à rédiger, à publier.

Je remercie Mme Brigitte Plateau qui m'a fait l'honneur de présider ce jury.

Je remercie M. Hubert Comon et M. Michael Rusinowitch pour l'intérêt qu'ils ont porté à ce travail et pour avoir accepté d'y associer leur grande renommée en étant rapporteurs. J'ai apprécié les discussions scientifiques que nous avons eu par courrier électronique et qui m'ont ouvert, je crois, sur de nouvelles perspectives.

Je remercie M. Philippe Jorrand et M. Didier Bert qui ont été mes directeurs de thèse officiels au sein des laboratoires Leibniz-Imag et Lsr-Imag. Je remercie plus particulièrement Philippe de m'avoir accueilli au sein du laboratoire Leibniz-Imag, et Didier, d'avoir accepté de participer à ce jury.

Je remercie plus généralement toutes les personnes qui ont contribué à la réalisation de ce travail. C'est bien sûr le cas des membres de l'équipe Pmp-Leibniz-Imag, c'est-à-dire, Jérémie Blanc, Rachid Echahed et Wendelin Serwe. C'est aussi le cas de Thierry Boy de la Tour de l'équipe Atinf-Leibniz-Imag et de Frédéric Blanqui du laboratoire Lri d'Orsay. Ils m'ont tous consacré beaucoup de temps, en relisant ce manuscrit, en proposant des corrections et en assistant aux répétitions de la soutenance. La qualité de ce travail doit beaucoup à leurs précieux conseils.

Je remercie pour les mêmes raisons Lalina Coulange, Michael Dierkes, Yves Guerte, Cyrille Lefranc, Nicolas Peirani et Rémy Sanlaville, ainsi que les membres des deux laboratoires qui m'ont accueilli, le personnel administratif et technique de ces laboratoires, le personnel de la médiathèque et les membres des équipes pédagogiques avec qui j'ai enseigné.

Je n'oublierai pas non plus le soutien de mes proches, Béa, Emmanuelle, le docteur Heinrich, Jean-Louis et Marie-Pierre[1], Let', Ludo, Seb et tant d'autres. Je me suis nourri de leur amitié, jour après jour. Je remercie plus particulièrement Did et Seb pour leur intervention pendant le pot de thèse. J'ai été particulièrement sensible à l'hommage qu'ils m'ont tous rendu ce jour-là.

Enfin, j'ai une pensée affectueuse pour mes parents. Ces derniers m'ont toujours encouragé, toujours soutenu pendant ces longues années d'études. Et je suis heureux d'avoir pu lire de la joie, de la fierté dans leurs regards, en ce lundi 24 janvier 2000, lorsque le jury m'a fait docteur en Informatique.

[1] et Camille-Marie ! :-)

Table des matières

Chapitre 1

Introduction

Généralités sur les langages logico-fonctionnels

Les langages logico-fonctionnels [dGL86, Han94] sont des langages de programmation de très haut niveau permettant de définir dans un formalisme unifié des types de données, des fonctions et des prédicats (relations). Ces langages, résultants de l'intégration des langages logiques et des langages fonctionnels, ont de nombreux avantages. Pour illustrer nos dires, nous considérons un exemple simple de programme utilisant les entiers naturels définis à partir des constructeurs `0` et `s` désignants respectivement la constante zéro et la fonction successeur. Notre programme définit une opération et deux prédicats : l'opération d'addition `+` sur les entiers et les prédicats `Pair` et `Impair` qui réussissent si leurs arguments sont des naturels pairs ou impairs. Si nous écrivons ce programme dans un langage logico-fonctionnel comme LPG [BE86, BER94] ou EQLOG [GM86], nous obtenons ceci :

```
0 + x == x
s(x) + y == s(x + y)

Pair(x + x)
Impair(s(x)) <== Pair(x)
```

Par rapport aux langages logiques, on constate que les langages logico-fonctionnels offrent la possibilité de définir des opérations, comme l'addition, alors qu'il faut les coder avec des prédicats, comme `Add`, dans un langage logique pur [SS86] :

```
Add(0,x,x)
Add(s(x),y,s(z)) <== Add(x,y,z)
```

De plus, la possibilité de définir des fonctions permet d'éviter l'utilisation de primitives de contrôle, comme le *cut*, qui font de Prolog un langage logique impur [GM86]. Enfin, du point de vue de la sémantique opérationnelle (c'est-à-dire le mode d'exécution des programmes), les langages logico-fonctionnels sont plus efficaces que les langages logiques : l'usage de fonctions dans les programmes permet des évaluations déterministes d'expressions alors que ces évaluations sont indéterministes (donc coûteuses) dans les langages logiques purs.

Par rapport aux langages fonctionnels, les langages logico-fonctionnels sont, là aussi, plus faciles à utiliser. Dans notre exemple de programme, les prédicats `Pair` et `Impair` sont définis à l'aide de clauses de Horn alors qu'il faudrait les coder avec des fonctions dans un langage fonctionnel pur comme OBJ3 [GKK$^+$87] :

```
pair(0) == true
impair(0) == false
pair(s(x)) == impair(x)
impair(s(x)) == pair(x)
```

De plus, les langages logico-fonctionnels permettent de résoudre des buts, comme par exemple `i + j == s(s(0)) & Pair(i)`, en autorisant l'usage de variables logiques permettant l'inversion de fonctions[1] et l'utilisation de structures de données partielles [Red85]. Ces caractéristiques n'existent pas dans les langages fonctionnels purs.

En outre, les langages logico-fonctionnels jouissent des avantages qui découlent de l'intégration de la programmation logique et fonctionnelle. Dans notre exemple, on constate que la définition de `Pair` se fait à l'aide d'une opération, `+`, que nous avons nous-même définie. De même, on pourrait imaginer une opération définie à l'aide d'un prédicat :

```
pair(x) == true <== Pair(x)
pair(x) == false <== Impair(x)
```

Les langages logico-fonctionnels permettent donc des définitions *mutuellement récursives* de fonctions et de prédicats. Cette caractéristique leur confère une grande souplesse.

Plusieurs langages logico-fonctionnels ont été proposés durant les vingt dernières années. [dGL86] et [Han94] présentent une compilation significative des propositions faites. Parmi les plus célèbres, citons LogLisp [RS82] (le tout premier d'entre eux), SLOG [Fri85], LPG [BE86, BE95], EQLOG [GM86], Life [AN86, AN89], ALF [Han90], K-LEAF [GLMP91], Babel [MR92], Escher [Llo94] et Curry [HAK+99]. A cela, il faut ajouter les langages logiques avec contraintes, comme Prolog3 [Col90], et les langages qui tentent d'intégrer *tous* les paradigmes de programmation, comme Oz [Smo95].

Objectifs de la thèse

L'étude de la mise en œuvre des langages logico-fonctionnels fait encore partie du domaine de la recherche. Mais des progrès importants ont été réalisés ces dernières années. En effet, une sémantique opérationnelle très efficace a été proposée [AEH94a, AEH97a] et plusieurs machines abstraites ont été inventées pour la compilation de ces langages comme par exemple [LK99] (voir le catalogue établi dans [Han94]). Cependant, toutes les propositions faites se restreignent à des calculs basés sur les *termes de premier ordre*. Cette restriction importante permet de manipuler les types abstraits algébriques [EM85] mais elle rend difficile la définition de certaines structures de données modélisées sous la forme de *graphes cycliques*.

Dans cette thèse, nous introduisons les graphes cycliques comme structure de données de base des langages logico-fonctionnels puis nous étudions les conséquences qui en découlent sur la sémantique opérationnelle de ces langages.

L'utilisation de graphes cycliques dans un langage logico-fonctionnel a trois avantages majeurs. Tout d'abord, la syntaxe du langage considéré est plus expressive que celles des langages logico-fonctionnels actuels car les graphes cycliques permettent de représenter les types de données du monde réel. Pour s'en convaincre, nous considérons le graphe de la Figure 1.1. Ce graphe représente une famille composée de deux adultes, Adam et Eve, et de

[1] On parle de *bidirectionalité* dans l'évaluation des arguments d'une fonction ou d'un prédicat.

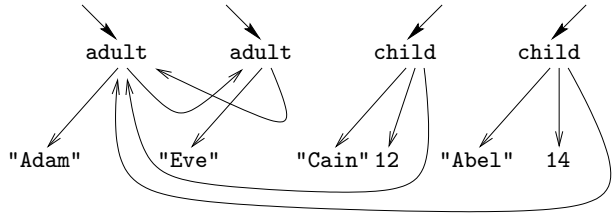


FIG. 1.1:

deux enfants, Caïn, qui a 12 ans, et Abel, qui a 14 ans. Adam et Eve sont mariés ensemble, comme le montrent les deuxièmes flèches qui partent des nœuds `adult` et qui s'entrecroisent. Les troisièmes flèches partant des nœuds `child` indiquent que Adam est le père de Caïn et de Abel. L'ensemble de ces informations peut être *codé* dans un langage logico-fonctionnel actuel ; notre proposition vise à éliminer cette phase de codage en permettant de manipuler les graphes directement dans un langage logico-fonctionnel.

Ensuite, l'utilisation de graphes améliore la représentation interne des données. En effet, s'ils respectent les sémantiques opérationnelles classiques, les langages logico-fonctionnels existants doivent représenter les données sous la forme de termes du premier ordre dans leurs implantations. Or un graphe permet de gagner en place mémoire. Dans la Figure 1.2, le terme (arbre) représentant l'expression `(0 + 0) + (0 + 0)` est un graphe composé de sept nœuds. Le graphe équivalent à ce terme n'est composé que de trois nœuds.

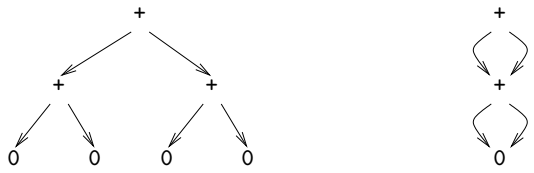


FIG. 1.2:

Enfin, la sémantique opérationnelle des langages logico-fonctionnels modernes [Han94] est souvent basée sur des calculs de *réécriture* et de *surréduction*[2]. Leur utilisation engendre des calculs (des dérivations) qui peuvent être de longueurs exponentielles sur les termes alors qu'elles sont linéaires sur les graphes. Les graphes permettent donc de gagner énormément en temps de calcul. Dans la Figure 1.3, l'évaluation d'une expression nécessite $2^p - 1$ pas de calcul lorsqu'on la représente sous forme de terme et seulement p pas de calcul lorsqu'on la représente sous forme de graphe.

L'usage de graphes dans un langage logico-fonctionnel permet donc de gagner en expressivité, en place mémoire et en temps de calcul. Les langages impératifs ont été les premiers à permettre de représenter les graphes grâce à la notion de pointeurs. Dans les langages fondés sur le λ-calcul, l'utilisation d'un symbole d'affectation (le `setq` de Lisp ou le `:=` d'Objective Caml) permet aussi de représenter les graphes mais elle fait perdre la déclarativité du langage. D'autres langages fonctionnels plus récents comme CLEAN [PvE93] ou HOPS [Kah94] sont directement implantés à l'aide de graphes ce qui permet de les utiliser dans le langage. Quant aux langages logiques, la manipulation de graphes est rendue possible par l'utilisation de

[2]La *résiduation* [ALN87] constitue un autre pilier de la sémantique opérationnelle de ces langages.

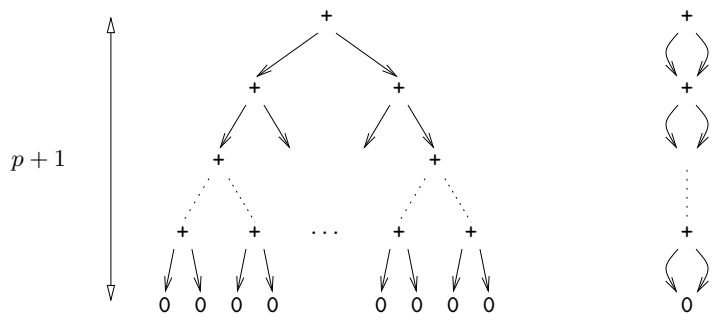


FIG. 1.3:

termes infinis rationnels [Cou83] comme dans le langage Prolog2 [CKvC83] (voir aussi [Llo87, Chapter 6]). Citons enfin le cas de LIFE [AP92] qui utilise les Ψ-termes, des types de graphes souvent utilisés en Intelligence Artificielle [AL88, HdBA93].

De manière plus générale, les systèmes de transformations de graphes connaissent un véritable essor actuellement. Cet engouement s'explique de deux manières. D'une part, la puissance actuelle des ordinateurs nous permet d'utiliser les graphes avec des performances raisonnables [Dör95]. D'autre part, les systèmes de transformations de graphes ont énormément d'applications, que ce soit en génie logiciel (avec le système PROGRESS [ELN+92, Nag96, PRO99] en particulier), dans les langages de programmation de tout type (visuels [BMST99], concurrents [Cor96] et distribués, fonctionnels [PvE93, Pey87], logiques ...), en bases de données, dans les systèmes d'informations, dans les réseaux neuronaux et même en biologie [CER79]. Le lecteur peut se faire une idée plus précise des applications possibles en feuilletant les récapitulatifs classiques des travaux sur le domaine [ET96, EEKR99, KP98, KP99] ainsi que l'ensemble des actes des différents ateliers intitulés "Graph Grammars and Their Applications to Computer Science" [CER79, ENR83, ENRR87, EKR91, ER94].

Notre approche

Dans cette thèse, nous voyons un programme logico-fonctionnel comme un système de réécriture de graphes avec constructeurs. Un tel système comporte deux parties :

- une signature qui regroupe les déclarations des types de données, des constructeurs de ces types et des opérations permettant de calculer sur ces types.
- un ensemble de règles de réécriture de graphes permettant de définir (c'est-à-dire d'implanter) les opérations.

Décrivons le programme permettant de travailler avec la famille de Adam, Eve, Caïn et Abel (cf. Figure 1.4). La signature décrit le type `human` défini à l'aide des constructeurs `adult` et `child`. Un adulte a un nom de type `string` et une épouse de type `human`. Un enfant a un nom de type `string`, un age de type `nat` et un père de type `human`.

Nous spécifions ensuite trois opérations : `spouse`, `father` et `mother`. `spouse` permet de calculer l'époux d'un adulte. `father` et `mother` permettent de retrouver les parents d'un enfant. Ces opérations sont décrites ("implantées") avec des règles de réécriture de graphes (cf. Figure 1.4). La première règle signifie que l'époux d'un adulte marié avec m est m. La seconde règle signifie que le père d'un enfant de père f est f. Enfin, la troisième règle signifie

que la mère d'un enfant est l'épouse du père de cet enfant[3].

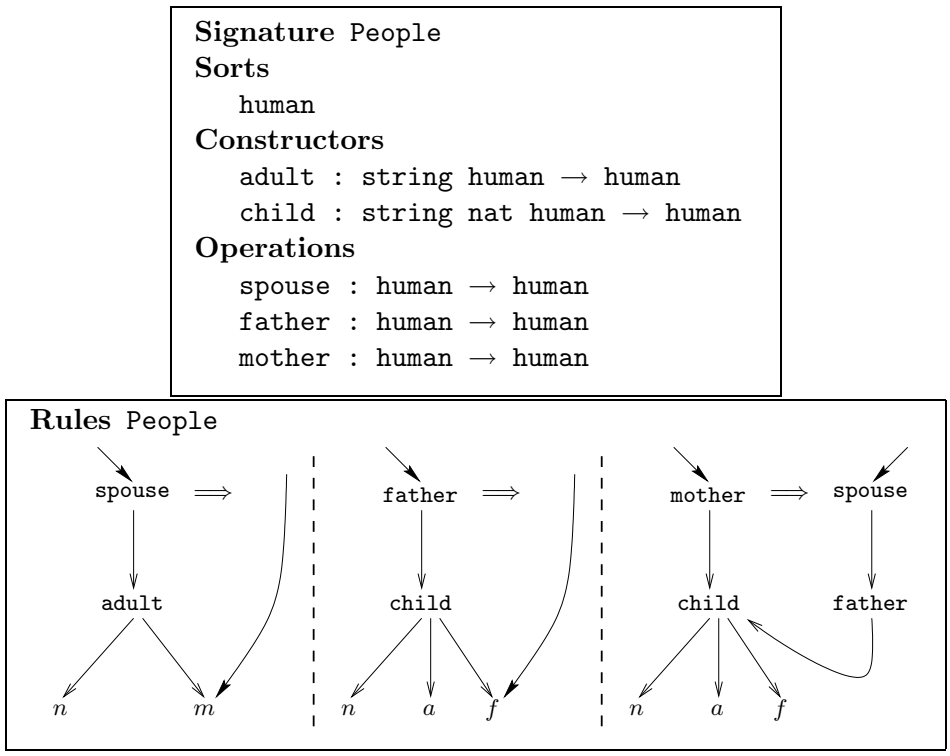


FIG. 1.4:

Concernant l'exécution (sémantique opérationnelle) des langages logico-fonctionnels, elle consiste en la résolution de buts. Un tel calcul [Han94, Ech90] repose, entre autre, sur les relations de réécriture et de surréduction de termes du premier ordre. Dans cette étude, nous remplaçons les termes par des graphes cycliques. Nous nous proposons donc d'étudier les relations de réécriture et de surréduction de graphes cycliques engendrées par les systèmes de réécriture de graphes décrits ci-dessus. La réécriture (ou la surréduction) de graphes cycliques permettent de calculer la "valeur" d'une expression. Dans la Figure 1.5, nous utilisons le programme de la Figure 1.4 pour calculer l'époux de la mère de Abel. L'évaluation par réécriture du graphe cyclique représentant cette expression nous permet de déduire qu'il s'agit d'Adam.

La réécriture des termes du premier ordre fait l'objet de nombreux travaux dont les principaux résultats sont regroupés dans [DJ90, Klo92]. Quant à la réécriture de graphes, plusieurs définitions de ce calcul existent : [SPvE93, ET96, Roz97] constitue une compilation significative des formalismes utilisés. Nos travaux appartiennent à un domaine de recherche connu sous le nom de "Term Graph Rewriting" dans la littérature. En ce qui concerne la réécriture et la surréduction des graphes acycliques, les résultats des études dans ce domaine sont regroupés dans [Plu98a]. La réécriture des graphes cycliques, celle qui nous

[3]Les règles de la Figure 1.4 et, plus généralement, les graphes de nos exemples sont souvent représentées avec des dessins. Néanmoins, il existe des syntaxes permettant de décrire les règles et les graphes de manière textuelle (par exemple [BvEG$^+$87]). De plus, un éditeur de graphes comme daVinci [FW99] permet de dessiner nos graphes puis d'en obtenir automatiquement une expression sous la forme d'un texte (voir aussi l'interface graphique du système PROGRESS [PRO99]).

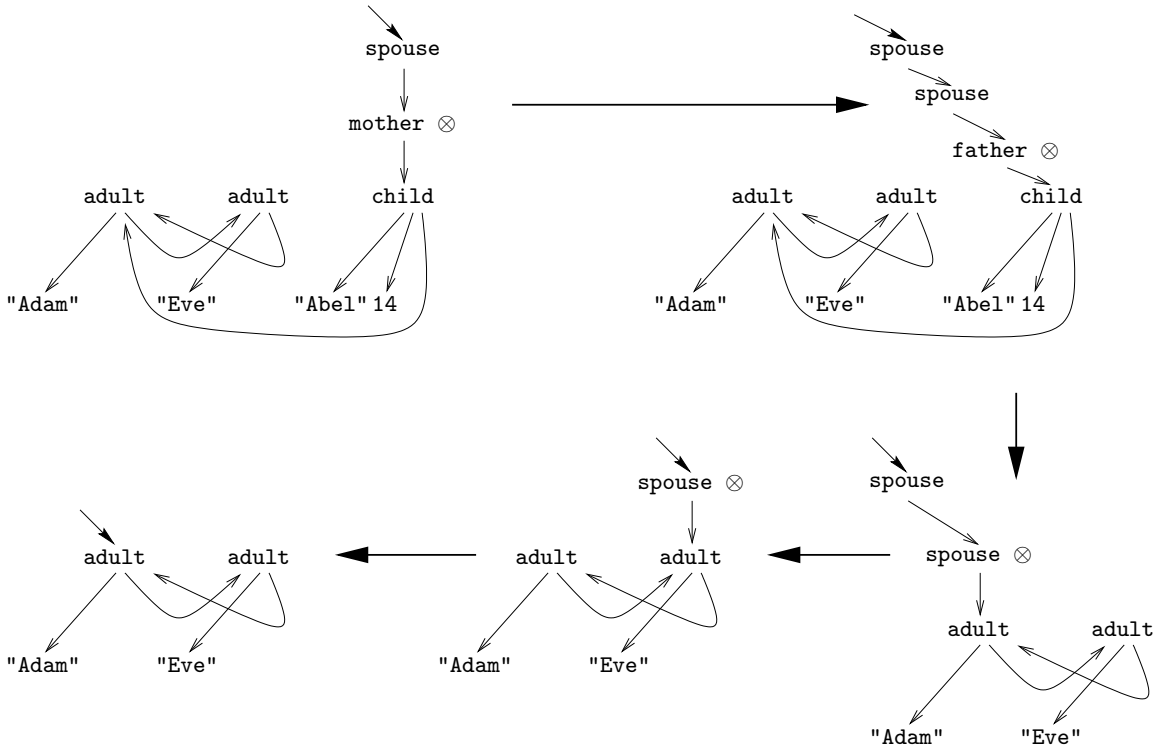


FIG. 1.5:

concerne dans cette étude, fait l'objet de travaux indépendants dont les principales références sont [BvEG+87, FRW89, Cor93, KKSV94, KKSV95]. Quant à la surréduction des graphes cycliques, les seuls travaux que nous connaissions sur ce sujet sont [EJ98a, EJ98b, EJ99c].

Nos résultats

Pour introduire les graphes dans les langages logico-fonctionnels, nous modélisons les programmes sous la forme de systèmes de réécriture de graphes cycliques avec constructeurs (cGRS[4]) et nous étudions les relations de réécriture et de surréduction qu'ils induisent. Ces résultats sont théoriques. Néanmoins, les performances des implantations des langages fonctionnels à base de graphes, comme celles de CLEAN [PvE93], nous laissent penser qu'une implantation d'un langage logico-fonctionnel avec des graphes sera plus performante que les implantations avec des termes.

Résultats sur la réécriture de graphes cycliques

Une propriété importante de la réécriture concerne la *confluence* : elle exprime le déterminisme des calculs effectués, l'unicité du résultat d'un calcul lorsqu'il termine. Dans le cadre des termes du premier ordre, ce problème a fait l'objet de nombreuses études (par exemple [CR36, New42, KB70, Hue80]). Malheureusement, les résultats de confluence concernant la réécriture des termes ne s'étendent généralement pas à la réécriture des graphes

[4] *Constructor-based Graph Rewriting Systems* en anglais.

cycliques (voir par exemple [KKSV94]). Nous mettons en évidence une classe de graphes cycliques particuliers, les graphes *admissibles*, qui est stable par réécriture. Puis nous montrons que la relation de réécriture engendrée par les systèmes de réécriture *faiblement admissibles* de graphes (WAGRS[5]) est confluente sur la classe des graphes admissibles.

D'autre part, il est important que deux graphes représentant la même information (graphes *bisimilaires* [AK96]) conduisent aux mêmes résultats après une évaluation par réécriture. Nous montrons que la relation de réécriture engendrée par les WAGRS (et par les bWAGRS[6]) est *confluente modulo la bisimilarité* sur la classe des graphes admissibles. Un résultat similaire est établi dans [AKP97, Theorem 6.4] pour le cas des systèmes de réécriture orthogonaux de graphes acycliques.

Ces premiers résultats sur la réécriture font l'objet du Chapitre 3.

Une fois la confluence établie, nous proposons une *stratégie optimale* de réécriture de graphes admissibles, c'est-à-dire un algorithme permettant d'éviter de faire des pas de calcul inutiles ou redondants pendant l'évaluation d'un graphe par réécriture. Les stratégies de réécriture de termes ont fait l'objet de plusieurs études [Klo92, Mid90] ; les plus célèbres sont définies dans [HL91, O'D77, SR93]. Notre stratégie de réécriture de graphes admissibles étend deux stratégies définies sur les termes du premier ordre dans [Ant92].

Lorsque le WAGRS considéré est *inductivement séquentiel* [Ant92], notre stratégie induit des pas de réécriture qui sont *nécessaires*[7] à l'évaluation d'un graphe par réécriture. En conséquence, on calcule effectivement la valeur d'un graphe en utilisant les pas par réécriture calculés par notre stratégie (propriété de $\mathcal{C}$*-hyper-normalisation*). De plus, notre stratégie engendre des dérivations de réécriture qui sont plus courtes que toute autre dérivation de réécriture. Notre stratégie est donc *optimale*.

Lorsque le WAGRS considéré n'est pas inductivement séquentiel, notre stratégie retourne un *ensemble suffisant de rédex*[8], ce qui nous permet de montrer qu'elle est $\mathcal{C}$-hyper-normalisante, comme dans le cas précédent. Pour traiter ces rédex de manière efficace, nous définissons la relation de *réécriture parallèle* de graphes. Cette relation étend la réécriture parallèle de termes [Hue80, O'D77], mais sa définition pose plus de problèmes dans le cadre des graphes que dans le cadre des termes, du fait du partage de sous-structures dans les graphes. Une définition différente de la réécriture parallèle de graphes existe dans [PvE93, Chap. 14] ; elle est étudiée du point de vue de l'implantation.

Enfin, comme l'ensemble des rédex que notre stratégie calcule est un sous-ensemble des plus hauts rédex dans un graphe, nous obtenons comme corollaire que la stratégie qui consiste en la réécriture parallèle de *tous* les plus hauts rédex d'un graphe[9] est elle aussi $\mathcal{C}$-hyper-normalisante. Nous retrouvons donc, dans le cadre des graphes admissibles, un résultat classique établi pour les termes du premier ordre dans [O'D77].

Nous présentons nos travaux sur notre stratégie de réécriture dans le Chapitre 4. L'ensemble de nos résultats sur la réécriture des graphes admissibles ont été publiés dans [EJ99b, EJ97c]. Les preuves se trouvent dans [EJ98c, EJ97b].

---

[5] *Weakly Admissible Graph Rewriting Systems* en anglais.
[6] *Bisimilar Weakly Admissible Graph Rewriting Systems* en anglais.
[7] *needed* en anglais [HL91].
[8] *necessary set of redexes* en anglais [SR93].
[9] *parallel-outermost strategy* en anglais.

Résultats sur la surréduction de graphes cycliques

La surréduction des termes du premier ordre est une relation qui a été introduite pour la première fois dans [Sla74]. Son utilisation pour résoudre des équations a été illustrée dans [Fay79] et [Hul80a]. Une première étude de la surréduction de graphes a été faite dans [Yam93, Kri96, HP96, HP99] dans le cadre des graphes acycliques et dans [EJ99c, EJ98a, EJ99a] dans le cadre des graphes cycliques admissibles.

Pour la définir, nous utilisons des *stratégies de compression de graphes*[10] permettant d'utiliser judicieusement les propriétés de partage de sous-structures dans les graphes afin d'optimiser les calculs [Kri96, HP99, EJ99a]. La relation qu'on obtient, la *surréduction compressante*, préserve l'admissibilité des graphes. Nous définissons la *cohérence* et la *complétude* de cette relation par rapport à la réécriture. Nous montrons que la surréduction compressante est cohérente dans le cas des cGRS. Puis nous montrons que la surréduction compressante *la plus générale* est complète dans le cas des WAGRS mais qu'elle ne l'est pas dans le cas des cGRS. Enfin, nous proposons un contre-exemple montrant que la *surréduction décompressante* de graphes n'est pas complète.

La surréduction compressante des graphes admissibles est traitée dans le Chapitre 5.

Nous développons enfin deux *stratégies de surréduction compressante* de graphes admissibles. De nombreuses stratégies de surréduction existent dans le cadre des termes du premier ordre, par exemple, la surréduction basique [Hul80a, MH92], *innermost* [Fri85], *outermost* [Ech88, Ech92], *outer* [You89], paresseuse [DG89, MKLR90, Red85] ou avec des tests de redondance [BKW92, KB91]. Nous nous intéressons plus particulièrement à la surréduction nécessaire[11] [AEH94a], à la surréduction faiblement nécessaire[12] [AEH97a] et à la surréduction parallèle[13] [AEH97a]. Récemment étudiées, ces stratégies de surréduction de termes sont optimales selon de nombreux critères. Elles peuvent être facilement programmées et sont donc couramment utilisées dans l'implantation des langages logico-fonctionnels.

Notre première stratégie étend la surréduction nécessaire [AEH94a] et la surréduction faiblement nécessaire [AEH97a] de termes aux graphes admissibles. La relation de surréduction compressante qu'elle engendre est cohérente et complète dans le cas des WAGRS. De plus, elle a d'intéressantes propriétés d'optimalités. En effet, les dérivations de surréduction compressante qu'elle engendre sont plus courtes que n'importe quelle dérivation de surréduction non compressante. De plus, nous montrons qu'en nous restreignant à des WAGRS inductivement séquentiels, les substitutions qu'elle calcule sont *disjointes* [AEH94b, EJ98a] et les pas de surréduction qu'elle induit sont *nécessaires*[14]. La seconde stratégie que nous étudions étend la surréduction parallèle [AEH97a] de termes aux graphes admissibles. De même que pour notre première stratégie, la relation de surréduction compressante qu'elle engendre est cohérente et complète.

Une autre stratégie de surréduction de graphes existe dans la littérature, la surréduction basique de graphes acycliques[15] [Kri96, HP99]. La surréduction basique de termes est la première stratégie de surréduction à avoir été étudiée [Hul80a]. Elle ne possède pas les propriétés d'optimalité des trois stratégies qui nous intéressent.

[10] *graph collapsing strategies* en anglais.
[11] *needed narrowing* en anglais.
[12] *weakly needed narrowing* en anglais.
[13] *parallel narrowing* en anglais.
[14] *needed narrowing steps* en anglais [AEH94a].
[15] *basic graph narrowing* en anglais.

Ces résultats sur les stratégies de surréduction compressante de graphes admissibles sont développés dans le Chapitre 6. L'ensemble de nos travaux sur la surréduction des graphes admissibles ont été publiés dans [EJ99c, EJ98a, EJ98b]. Les versions longues de ces papiers sont [EJ99a, EJ97a].

Plan de la thèse

Le Chapitre 2 regroupe l'ensemble des notions de bases qui sont utiles pour définir la réécriture et la surréduction de graphes.

Le reste de cette thèse est divisée en deux. Nous attaquons d'abord l'étude de la relation de réécriture. Sa définition et les résultats de confluence que nous obtenons sont développés dans le Chapitre 3. Nous consacrons le Chapitre 4 aux stratégies de réécriture.

Nous développons dans un deuxième temps notre étude de la surréduction compressante. La définition de cette relation, sa cohérence et sa complétude font l'objet du Chapitre 5. Nous abordons les stratégies de surréduction dans le Chapitre 6.

Pour finir, le Chapitre 7 présente un résumé de notre contribution ainsi que les perspectives de ces travaux.

Chapitre 2

Définitions et préliminaires

Comme mentionné en introduction, nous modélisons les structures de données des langages logico-fonctionnels à l'aide de graphes cycliques. La sémantique opérationnelle de ces langages est basée sur la réécriture et la surréduction de graphes. L'étude de ces calculs fait l'objet des autres parties de cette thèse mais leurs définitions nécessitent plusieurs notions de base qui sont développées dans ce chapitre de préliminaires.

Nous commençons par donner une définition formelle des graphes dans la section suivante. Puis nous explicitons le calcul du remplacement, dans un graphe, d'un sous-graphe par un autre, à l'aide de la notion de redirection de pointeurs (Section 2.2) ; la notion de remplacement pose plus de problème dans le cadre des graphes que dans le cadre des termes du fait du partage de sous-graphes. Dans la Section 2.3, nous développons la notion d'homomorphisme de graphes. Les homomorphismes permettent d'exprimer le filtrage et l'unification des graphes. Nous verrons dans la Section 2.4 comment se calcule l'application d'un homomorphisme sur un graphe. Toutes les notions précédentes sont nécessaires aussi bien pour la réécriture que pour la surréduction de graphes. La relation de surréduction nécessite en plus des substitutions que nous étudions brièvement dans la Section 2.5. Pour finir ce chapitre, nous faisons une liste des préordres et des relations d'égalité (en particulier la *bisimilarité* [AK96]) que nous utiliserons sur les graphes et les substitutions (Section 2.6). Ces préliminaires sont issus de [EJ97b] ; ils nous ont servi pour écrire toutes nos publications.

Toutes les sections de ce chapitre sont importantes pour lire la thèse. Toutefois, la Section 2.5 n'est utile que dans les Chapitres 5 et 6.

2.1 Définition des graphes

Plusieurs notations de graphes existent dans la littérature pour traiter de la réécriture et de la surréduction, par exemple, les systèmes d'équations [AK96], les hypergraphes [Plu98a], les graphes de données[1] [PvE93] ou les formalismes à base de catégories [ET96]. Nos définitions sont proches de celles de [BvEG$^+$87, KKSV94].

Définition 2.1.1 (Signature multi-sortée)
Une *signature multi-sortée* $\Sigma = \langle S, \Omega \rangle$ consiste en un ensemble de sortes S et une famille S-indicée d'ensembles de symboles de fonctions $\Omega = \uplus_{s \in S} \Omega_s$ avec $\Omega_s = \uplus_{w \in S^*} \Omega_{w,s}$. On note

[1] *data graph* en anglais.

$f : s_1 \dots s_n \to s$ tout symbole $f \in \Omega_{s_1 \dots s_n, s}$ et on dit que f est de *sorte* s, d'*arité* $s_1 \dots s_n$, et de *profil* $s_1 \dots s_n \to s$. ◇

Exemple 2.1.2 Soit $\Sigma = \langle S, \Omega \rangle$ la signature multi-sortée telle que $S = \{\zeta_1, \zeta_2\}$ et $\Omega = \{\mathtt{c} : \zeta_1\zeta_2 \to \zeta_2, \mathtt{s} : \zeta_1 \to \zeta_1, \mathtt{a} : \varepsilon \to \zeta_1, \mathtt{f} : \zeta_1\zeta_1 \to \zeta_2, \mathtt{g} : \zeta_1\zeta_1 \to \zeta_1, \mathtt{h} : \zeta_1\zeta_1 \to \zeta_1\}$. □

Nous considérons qu'un graphe est composé de nœuds et de flèches entre ses nœuds. Chaque nœud est étiqueté par un symbole de fonction ou par une variable. Les flèches désignent les successeurs d'un nœud. Ces flèches sont ordonnées car elles représentent les arguments de la fonction qui étiquette le nœud. Soient $\mathcal{N} = \uplus_{s \in S} \mathcal{N}_s$ une famille S-indicée d'ensembles de *nœuds* et $\mathcal{X} = \uplus_{s \in S} \mathcal{X}_s$ une famille S-indicée d'ensembles de *variables*. Nous supposons que $\mathcal{X}$ et $\mathcal{N}$ sont fixés dans toute la suite.

Un graphe g est alors un quadruplet $\langle \mathcal{N}_g, \mathcal{L}_g, \mathcal{S}_g, \mathcal{R}\mathrm{oots}_g \rangle$ tel que $\mathcal{N}_g$ soit un sous-ensemble fini de nœuds de $\mathcal{N}$, $\mathcal{L}_g : \mathcal{N}_g \to \Omega \cup \mathcal{X}$ soit une fonction d'étiquetage qui associe à chaque nœud un symbole de fonction ou une variable, $\mathcal{S}_g : \mathcal{N}_g \to \mathcal{N}_g^*$ soit une fonction de successeur qui associe à chaque nœud une chaîne éventuellement vide de nœuds et $\mathcal{R}\mathrm{oots}_g$ soit une chaîne non vide de nœuds distingués qu'on appelle les racines de g. On suppose, de plus, que tous les nœuds de g sont accessibles depuis au moins une racine (condition de *connexité*). On suppose enfin qu'une variable étiquette au plus un nœud de g. Un terme-graphe est un graphe avec une seule racine.

Exemple 2.1.3 Dans la Figure 2.1, nous montrons deux exemples de graphes G et T. Leurs racines sont désignées par des flèches sans antécédent. Le terme-graphe G est le quadruplet tel que (1) $\mathcal{N}_G = \{\mathtt{n1}, \dots, \mathtt{n5}\}$, (2) $\mathcal{R}\mathrm{oot}_G = \mathtt{n1}$, (3) $\mathcal{L}_G$ est définie par $\mathcal{L}_G(\mathtt{n1}) = \mathcal{L}_G(\mathtt{n5}) = \mathtt{c}$, $\mathcal{L}_G(\mathtt{n2}) = \mathtt{g}$, $\mathcal{L}_G(\mathtt{n3}) = \mathtt{s}$ et $\mathcal{L}_G(\mathtt{n4}) = \mathtt{a}$ et (4) $\mathcal{S}_G$ est définie par $\mathcal{S}_G(\mathtt{n1}) = \mathtt{n2.n5}$, $\mathcal{S}_G(\mathtt{n2}) = \mathtt{n3.n3}$, $\mathcal{S}_G(\mathtt{n3}) = \mathtt{n4}$, $\mathcal{S}_G(\mathtt{n4}) = \varepsilon$ et $\mathcal{S}_G(\mathtt{n5}) = \mathtt{n3.n1}$. A droite de la figure, T est un graphe avec deux racines `l1.r1` ; il représente une règle de réécriture (cf. Définition 3.1.9). □

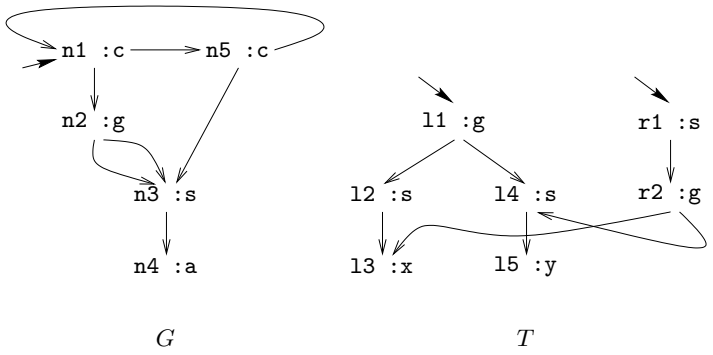


FIG. 2.1:

La définition suivante tient compte du typage :

Définition 2.1.4 (Graphe)
Un *graphe* (enraciné) g sur $\langle \Sigma, \mathcal{N}, \mathcal{X} \rangle$ est un quadruplet $g = \langle \mathcal{N}_g, \mathcal{L}_g, \mathcal{S}_g, \mathcal{R}\mathrm{oots}_g \rangle$ tel que :

- $\mathcal{N}_g$ soit l'ensemble des *nœuds* de g, i.e., $\mathcal{N}_g = \uplus_{s \in S}(\mathcal{N}_g)_s$ avec $(\mathcal{N}_g)_s \subseteq \mathcal{N}_s$.
- La *fonction d'étiquetage des nœuds de* g, $\mathcal{L}_g$, soit une famille S-indicée de fonctions qui associent un symbole de fonction ou une variable à tous les nœuds de g, i.e., $\mathcal{L}_g = \uplus_{s \in S}(\mathcal{L}_g)_s$ avec $(\mathcal{L}_g)_s : (\mathcal{N}_g)_s \to \Omega_s \cup \mathcal{X}_s$.

- La *fonction de successeur des nœuds de* g, $\mathcal{S}_g$, soit une famille S-indicée de fonctions qui associent une chaîne éventuellement vide de nœuds à tous les nœuds de g, i.e., $\mathcal{S}_g = \uplus_{s \in S}(\mathcal{S}_g)_s$ avec $(\mathcal{S}_g)_s : (\mathcal{N}_g)_s \to \mathcal{N}_g^*$.
- $\mathcal{R}\text{oots}_g$ soit une chaîne non vide de nœuds distingués de g qu'on appelle ses *racines* : $\mathcal{R}\text{oots}_g \in \mathcal{N}_g{}^+$.

En outre, nous imposons trois conditions de bonne définition des graphes :

1. Les graphes sont bien typés, c'est-à-dire (1) un nœud n est de même sorte que son étiquette $\mathcal{L}_g(n)$ et (2) ses successeurs $\mathcal{S}_g(n)$ sont compatibles avec l'arité de $\mathcal{L}_g(n)$. Formellement, pour tout nœud $n \in \mathcal{N}_g$:
   - si $(\mathcal{L}_g)_s(n) = f$ avec $f : s_1 \dots s_k \to s$, alors $n \in (\mathcal{N}_g)_s$ et il existe $n_1, \dots, n_k \in \mathcal{N}_g$ tels que $(\mathcal{S}_g)_s(n) = n_1 \dots n_k$ et $n_i \in (\mathcal{N}_g)_{s_i}$ pour tout $i \in 1..k$.
   - si $(\mathcal{L}_g)_s(n) = c$ avec $c \in \Omega_{\varepsilon,s}$ (c est une *constante*), alors $n \in (\mathcal{N}_g)_s$ et $(\mathcal{S}_g)_s(n) = \varepsilon$ (i.e., n n'a pas de successeur).
   - si $(\mathcal{L}_g)_s(n) = x$ avec $x \in \mathcal{X}_s$ (x est une variable), alors $n \in (\mathcal{N}_g)_s$ et $(\mathcal{S}_g)_s(n) = \varepsilon$.
2. On désigne par $\mathcal{V}_g$ l'ensemble des variables de g. Une variable de g n'étiquette qu'un seul nœud de g : pour tout nœud $n_1, n_2 \in \mathcal{N}_g$, pour toute variable $x_1, x_2 \in \mathcal{V}_g$, si $\mathcal{L}_g(n_1) = x_1$ et $\mathcal{L}_g(n_2) = x_2$, alors $x_1 = x_2 \Longrightarrow n_1 = n_2$.
3. *Condition de connexité :* Chaque nœud de g est une racine ou bien est *accessible* depuis une racine (cf. Définition 2.1.5).

Un *terme-graphe* g est un graphe (éventuellement cyclique) avec une seule racine que l'on note $\mathcal{R}\text{oot}_g$. ◇

Nous donnons ci-dessous la définition de chemin dans un graphe [Ken95] ce qui permet de préciser la condition de connexité des graphes. De plus, nous introduisons des notations permettant de comparer les positions relatives de deux nœuds dans un graphe.

Définition 2.1.5 (Chemins)
Soient g un graphe et p et q deux nœuds de g.
On dit que q est le $j^{ème}$ *successeur* de p dans g, noté $q = \mathcal{S}_g(p)_j$ ou simplement $q \in \mathcal{S}_g(p)$, si $\mathcal{S}_g(p) = v_1 \dots v_k$ et $v_j = q$.
Un *chemin* C d'un nœud p vers un nœud q dans un graphe g est une séquence non vide $C = [u_0, i_0, u_1, i_1, \dots, i_{k-1}, u_k]$ alternant des nœuds et des entiers telle que :

1. $u_0 = p$, $u_k = q$ et $k \geq 1$.
2. $u_p \in \mathcal{N}_g$ pour tout $p \in 0..k$.
3. u_{p+1} est le $i_p{}^{\text{ème}}$ successeur de u_p dans g, pour tout $p \in 0..k-1$ (i.e., $u_{p+1} = \mathcal{S}_g(u_p)_{i_p}$).

On note $\mathcal{P}\text{aths}_g(p, q)$ l'ensemble éventuellement vide des chemins de p vers q dans g.
On dit que le nœud q est *accessible* depuis le nœud p dans g, noté $p \prec q$, ssi il existe un chemin de p vers q dans g : $p \prec q \Longleftrightarrow \mathcal{P}\text{aths}_g(p, q) \neq \emptyset$. Nous écrirons $p \preceq q$ ssi $p \prec q$ ou $p = q$.
On dit que les nœuds p et q sont *disjoints* dans g, noté $p \| q$, ssi il n'existe pas de chemin de p vers q ni de q vers p : $p \| q \Longleftrightarrow \neg(p \preceq q \vee q \preceq p) \Longleftrightarrow \mathcal{P}\text{aths}_g(p, q) = \mathcal{P}\text{aths}_g(q, p) = \emptyset$. ◇

Exemple 2.1.6 Dans le graphe T de la Figure 2.1, il est clair que `l1` $\|$ `r1`. Par contre, dans le graphe G de la Figure 2.1, nous avons `n1` $\prec$ `n3`. En effet, il existe un ensemble infini de chemins du nœud `n1` vers le nœud `n3`, par exemple, [`n1`,1,`n2`,1,`n3`], [`n1`,1,`n2`,2,`n3`], [`n1`,2,`n5`,1,`n3`], [`n1`,2,`n5`,2,`n1`,1,`n2`,1,`n3`], ...

On remarque que $\preceq$ n'est pas un ordre partiel, mais juste un pré-ordre : dans un graphe cyclique, il est possible qu'un nœud soit accessible depuis un autre nœud et réciproquement. C'est le cas des nœuds `n1` et `n5` dans le graphe G de la Figure 2.1 : on constate que `n1` $\preceq$ `n5`, `n5` $\preceq$ `n1` et `n1` $\neq$ `n5`.
Pour finir, revenons sur la condition de connexité des graphes. Compte tenu de la définition d'accessibilité, elle s'exprime ainsi : $\forall n \in \mathcal{N}_g, \exists r \in \mathcal{R}\text{oots}_g$ tel que $r \preceq n$. □

La définition formelle des graphes n'est pas pratique pour traiter les exemples. Aussi, nous utilisons une notation linéaire pour décrire les graphes, qui est à peu près la même que celle de [BvEG+87].

Définition 2.1.7 (Notation linéaire des graphes)
Dans la grammaire suivante, la variable A (resp. n) parcourt l'ensemble $\Omega \cup \mathcal{X}$ (resp. $\mathcal{N}$). L'expression linéaire d'un graphe est définie par :

GRAPH ::= NODE | NODE + GRAPH
NODE ::= n :A(NODE,...,NODE) | n

La chaîne des racines d'un graphe décrit avec une telle expression est composée du premier nœud apparaissant dans l'expression et de tous les nœuds apparaissant juste après un +. ◇

Exemple 2.1.8 Dans la Figure 2.1, $G =$ `n1 :c(n2 :g(n3 :s(n4 :a),n3),n5 :c(n3,n1))` et $T =$ `l1 :g(l2 :s(l3 :x),l4 :s(l5 :y)) + r1 :s(r2 :g(l3,l4))`. □

Nous terminons cette section avec la notion de sous-graphe :

Définition 2.1.9 (Sous-graphe)
Soient g un graphe et P une chaîne de nœuds de g. On définit le *sous-graphe de racines* P, noté $g_{|P}$, comme le graphe tel que :

- $\mathcal{N}_{(g_{|P})}$ soit l'ensemble des nœuds accessibles depuis les nœuds de P dans g, i.e., $\mathcal{N}_{(g_{|P})} = P \cup \{n \in \mathcal{N}_g \mid \exists r \in P, \mathcal{P}\text{aths}_g(r, n) \neq \emptyset\}$.
- $\mathcal{L}_{(g_{|P})}$ (resp. $\mathcal{S}_{(g_{|P})}$) soit la restriction de $\mathcal{L}_g$ (resp. $\mathcal{S}_g$) à $\mathcal{N}_{(g_{|P})}$, i.e., pour tout $n \in \mathcal{N}_{(g_{|P})}$, $\mathcal{L}_{(g_{|P})}(n) = \mathcal{L}_g(n)$ et $\mathcal{S}_{(g_{|P})}(n) = \mathcal{S}_g(n)$.
- $\mathcal{R}\text{oots}_{(g_{|P})} = P$.

En d'autres termes, le sous-graphe de racines P d'un graphe g se construit en considérant P comme la chaîne des racines de g et en effaçant tous les nœuds de g qui ne sont pas accessibles depuis ceux de P. ◇

Exemple 2.1.10 Dans la Figure 2.1, $G_{|\texttt{n2}} =$ `n2 :g(n3 :s(n4 :a),n3)` et $T_{|\texttt{l3.l4}} =$ `l3 :x + l4 :s(l5 :y)`. □

2.2 Remplacement

La notion de remplacement d'un sous-graphe (avec une racine) par un autre terme-graphe dans un graphe est essentielle pour la réécriture et la surréduction. Cette notion, classique dans le cadre des termes du premier ordre, pose des problèmes dans le cadre des graphes, du fait du partage de sous-graphes. Nous commençons par expliquer le processus de calcul sur un exemple. Puis nous définissons les notions de somme de graphes et de redirection de pointeurs permettant d'écrire une définition formelle du calcul de remplacement.

La Figure 2.2 montre les différentes étapes du remplacement par D du sous-graphe de G au nœud `n2`. Comme G et D partage les nœuds `n3` et `n4`, nous ne devons pas considérer G

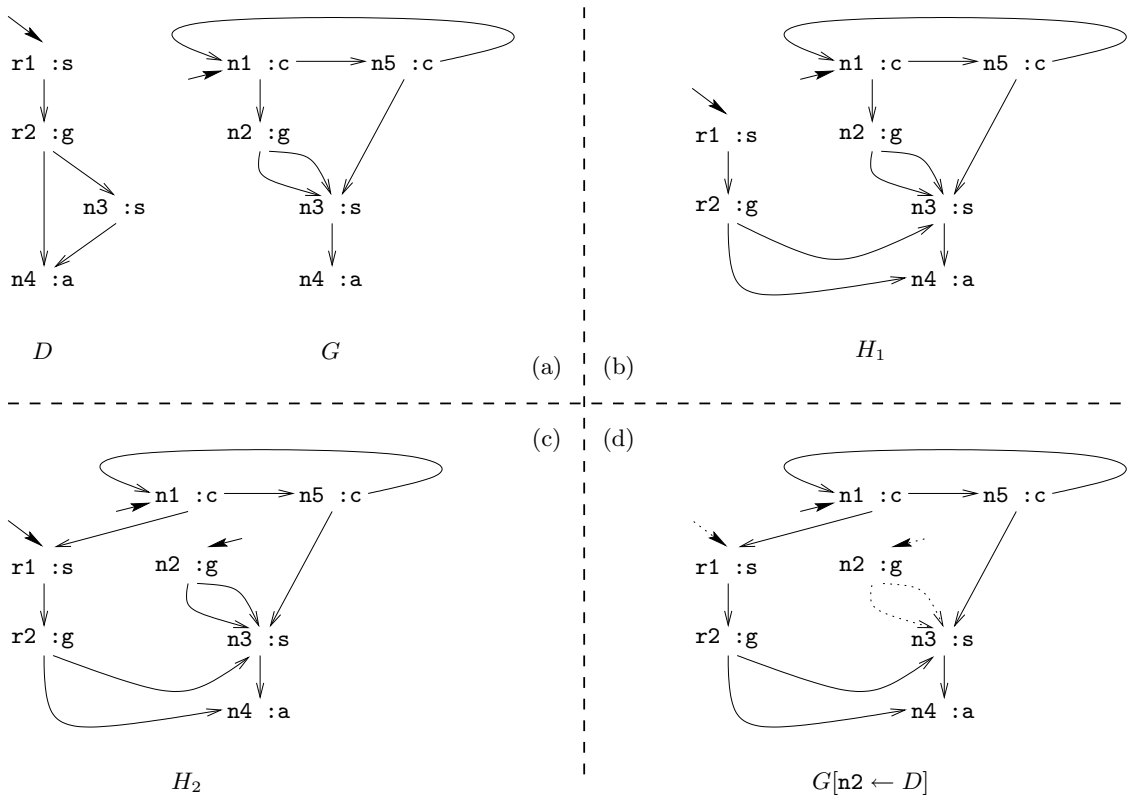

r1 :s
r2 :g
n3 :s
n4 :a
n1 :c
n5 :c
n2 :g
n3 :s
n4 :a
D
G
(a)
(b)
n1 :c
n5 :c
r1 :s
n2 :g
r2 :g
n3 :s
n4 :a
H1
(c)
(d)
n1 :c
n5 :c
r1 :s
n2 :g
r2 :g
n3 :s
n4 :a
H2
n1 :c
n5 :c
r1 :s
n2 :g
r2 :g
n3 :s
n4 :a
G[n2 ← D]

FIG. 2.2:

et D séparément, comme deux graphes indépendants, mais comme un seul graphe avec deux racines (cf. H_1 dans la Figure 2.2). Nous appelons ce calcul *somme des graphes G et D* et nous le notons $G \oplus D$ (i.e., $H_1 = G \oplus D$). Une fois que $G \oplus D$ a été calculé, nous redirigeons toutes les flèches pointant sur `n2` vers le nœud `r1` (cf. H_2 dans la Figure 2.2). Cette opération s'appelle une *redirection de pointeurs*. On remarque que `n2` est devenu une racine : si ce n'était pas le cas, le graphe obtenu après la redirection de pointeurs ne serait pas connexe (il ne satisferait pas la troisième condition de la Définition 2.1.4). Pour finir, on ignore les nœuds, les flèches et les racines "inintéressants" (ramasse-miette, garbage collection). Le résultat du *remplacement par D du sous-graphe de G au nœud* `n2` se note $G[\texttt{n2} \leftarrow D]$ (cf. Figure 2.2).

Nous donnons maintenant toutes les définitions formelles d'un pas de remplacement. Nous commençons par définir la somme de graphes qui formalise le fait qu'on peut considérer deux graphes comme un seul en faisant l'union de leurs nœuds et la concaténation de leurs racines. Sans précaution, le résultat de ce calcul n'est pas toujours un graphe. C'est la raison pour laquelle nous posons la définition technique suivante :

Définition 2.2.1 (Compatibilité de deux graphes)
Soient g_1 et g_2 deux graphes. On dit que g_1 et g_2 sont *compatibles* ssi :

1. Tous les nœuds qui apparaissent à la fois dans g_1 et dans g_2 ont la même étiquette et les mêmes successeurs : Pour tout $n \in (\mathcal{N}_{g_1} \cap \mathcal{N}_{g_2})$, $\mathcal{L}_{g_1}(n) = \mathcal{L}_{g_2}(n)$ et $\mathcal{S}_{g_1}(n) = \mathcal{S}_{g_2}(n)$.
2. Toute variable apparaissant à la fois dans g_1 et dans g_2 étiquette le même nœud : Pour tout $x \in (\mathcal{V}_{g_1} \cap \mathcal{V}_{g_2})$, si $\mathcal{L}_{g_1}(n) = x$ et $\mathcal{L}_{g_2}(m) = x$, alors $n = m$.

◇

Exemple 2.2.2 Dans la Figure 2.3, les graphes g_1 et g_2 ne sont pas compatibles car le nœud `n1` est étiqueté avec `B` dans g_1 alors qu'il est étiqueté avec `A` dans g_2. Les graphes g_1 et g_3 ne sont pas non plus compatibles car le nœud `n1` n'a pas le même successeur dans g_1 et g_3. Enfin, les graphes g_3 et g_4 ne sont pas compatibles car la variable `X` étiquette deux nœuds distincts dans g_3 et g_4 (i.e., `n3` et `n5`). On remarque que les graphes G et D de la Figure 2.2 sont compatibles. □

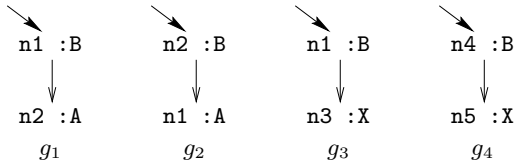


FIG. 2.3:

Définition 2.2.3 (Somme de graphes compatibles)
Étant donnés deux graphes compatibles g_1 et g_2, nous définissons la *somme* $g_1 \oplus g_2$ comme le graphe tel que $\mathcal{N}_{(g_1 \oplus g_2)} = \mathcal{N}_{g_1} \cup \mathcal{N}_{g_2}$, $\mathcal{L}_{(g_1 \oplus g_2)} = \mathcal{L}_{g_1} \cup^2 \mathcal{L}_{g_2}$, $\mathcal{S}_{(g_1 \oplus g_2)} = \mathcal{S}_{g_1} \cup^2 \mathcal{S}_{g_2}$ et $\mathcal{R}\text{oots}_{(g_1 \oplus g_2)} = \mathcal{R}\text{oots}_{g_1}.\mathcal{R}\text{oots}_{g_2}$. ◇

Exemple 2.2.4 Dans la Figure 2.2, $G \oplus D = H_1 =$
`n1 :c(n2 :g(n3 :s(n4 :a),n3),n5 :c(n3,n1)) + r1 :s(r2 :g(n4,n3)).` □

[2] L'*union* $f \cup g$ des fonctions f et g est l'union des relations qui définissent ces fonctions. Dans notre cas, $f \cup g$ reste une fonction (par compatibilité des graphes).

Nous abordons ci-dessous la notion de redirection de pointeurs qui consiste en la substitution d'un nœud par un autre nœud dans un graphe.

Définition 2.2.5 (Redirection de pointeurs)
Soient p et q deux nœuds de même sorte non nécessairement distincts. La *redirection de pointeurs ρ de p vers q* est la fonction $\rho : \mathcal{N} \to \mathcal{N}$ telle que $\rho(p) = q$ et $\rho(n) = n$ pour tout nœud $n \neq p$.
Plus généralement, si $p_1, \ldots, p_n$ et $q_1, \ldots, q_n$ sont des nœuds de mêmes sortes deux à deux, on définit la *redirection de pointeurs (multiple)* ρ de p_1 vers q_1, ..., p_n vers q_n, comme la fonction $\rho : \mathcal{N} \to \mathcal{N}$ telle que $\rho(p_i) = q_i$ pour tout $i \in 1..n$ et $\rho(p) = p$ pour tout nœud p tel que $p \neq p_1$, $p \neq p_2$, ... et $p \neq p_n$.
Soient g un graphe et $\rho = \{p_1 \mapsto q_1, \ldots, p_n \mapsto q_n\}$ une redirection de pointeurs telle que $p_1, \ldots, p_n, q_1, \ldots, q_n$ soient des nœuds de g. On définit le graphe $\rho(g)$ obtenu par *application de la redirection de pointeurs ρ sur le graphe g* comme le graphe tel que tout nœud de g ayant p_i comme successeur soit un nœud de $\rho(g)$ ayant q_i comme successeur :

- $\mathcal{N}_{\rho(g)} = \mathcal{N}_g$ et $\mathcal{L}_{\rho(g)} = \mathcal{L}_g$.
- Pour tout $n \in \mathcal{N}_g$, si $\mathcal{S}_g(n) = n_1 \ldots n_k$, $(k \geq 0)$, alors $\mathcal{S}_{\rho(g)}(n) = \rho(n_1) \ldots \rho(n_k)$.
- Si $\mathcal{R}\text{oots}_g = n_1 \ldots n_k$, alors $\mathcal{R}\text{oots}_{\rho(g)} = \rho(n_1) \ldots \rho(n_k).p_1.p_2 \ldots p_n$. Les nœuds $p_1, \ldots, p_n$ apparaissent dans $\mathcal{R}\text{oots}_{\rho(g)}$ pour que $\rho(g)$ soit un graphe connexe (i.e., pour que $\rho(g)$ satisfasse la troisième condition de la Définition 2.1.4).

◇

Exemple 2.2.6 Soit $\rho = \{\texttt{n2}\mapsto\texttt{r1}\}$. Dans la Figure 2.2, $\rho(H_1) = H_2 =$ `n1 :c(r1 :s(r2 :g(n4 :a),n3 :s(n4)),n5 :c(n3,n1)) + r1 + n2 :g(n3,n3)`. Un exemple de redirection de pointeurs multiple est donné dans la Figure 4.10 (page 75) où le graphe H_1 (resp. H_2) est obtenu à partir de H en appliquant la redirection de pointeurs $\rho = \{\texttt{p2} \mapsto \texttt{r1}, \texttt{p4} \mapsto \texttt{p2}\}$ (resp. $\rho = \{\texttt{p2} \mapsto \texttt{r1}, \texttt{p4} \mapsto \texttt{r1}\}$). □

Nous sommes maintenant prêts pour donner la définition d'un pas de remplacement.

Définition 2.2.7 (Remplacement)
Soient g un graphe, u un terme-graphe tel que g et u soient compatibles et p un nœud de même sorte que $\mathcal{R}\text{oot}_u$. Si p est un nœud de g, nous définissons le *remplacement par u du sous-graphe de racine p dans g*, noté $g[p \leftarrow u]$, en trois étapes :

1. Soient $H = g \oplus u$ et $\rho = \{p \mapsto \mathcal{R}\text{oot}_u\}$ la redirection de pointeurs de p vers $\mathcal{R}\text{oot}_u$.
2. Soient $H' = \rho(H)$ et $Q = \rho(\mathcal{R}\text{oots}_g)$.
3. $g[p \leftarrow u] = {H'}_{|Q}$.

Si p n'est pas un nœud de g, alors nous posons $g[p \leftarrow u] = g$. ◇

Exemple 2.2.8 Nous avons détaillé dans la Figure 2.2 le remplacement par $D =$ `r1 :s(r2 :g(n4 :a,n3 :s(n4)))` du sous-graphe de racine `n2` dans le graphe $G =$ `n1 :c(n2 :g(n3 :s(n4 :a),n3),n5 :c(n3,n1))`. Le graphe résultat est $G[\texttt{n2} \leftarrow D] =$ `n1 :c(r1 :s(r2 :g(n4 :a,n3 :s(n4))),n5 :c(n3,n1))`. Notons qu'avec la définition précédente, si n est un nœud d'un graphe g, alors $g[n \leftarrow g_{|n}] = g$. □

Pour terminer cette section, nous prévenons le lecteur que le calcul de $g[p \leftarrow u]$ peut avoir des résultats surprenants dans certains cas :

Exemple 2.2.9 Soient $g = \texttt{p1 :A}$ et $u = \texttt{m1 :B(p1 :A,m2 :A)}$. Calculons $g[\texttt{p1} \leftarrow u]$. Tout d'abord, g et u sont compatibles et $g \oplus u = \texttt{p1 :A + m1 :B(p1,m2 :A)}$. Soit $\rho = \{\texttt{p1} \mapsto \texttt{m1}\}$. Nous obtenons $\rho(g \oplus u) = \texttt{m1 :B(m1,m2 :A) + m1 + p1 :A}$ et $\rho(\texttt{p1}) = \texttt{m1}$. Nous concluons donc que $g[\texttt{p1} \leftarrow u] = \texttt{m1 :B(m1,m2 :A)}$. Or p1 est la racine de g. Donc le lecteur pourrait s'attendre à obtenir $g[\texttt{p1} \leftarrow u] = u$. Il n'en est rien : u est modifié dans $g[\texttt{p1} \leftarrow u]$ parce que p1 est un nœud commun à g et u. □

2.3 Homomorphismes

Nous développons ci-dessous la notion d'homomorphisme de graphes. Les homomorphismes jouent, dans le cadre de la réécriture de graphes, un rôle similaire à celui des substitutions, dans le cadre de la réécriture de termes. En effet, ils permettent d'exprimer les notions fondamentales de filtrage et d'unification.

Un homomorphisme $h : g_1 \to g_2$ allant d'un graphe g_1 vers un graphe g_2 est une application des nœuds de g_1 vers les nœuds de g_2 qui (1) préserve les racines de g_1 : $\mathcal{R}\text{oots}_{g_2} = h(\mathcal{R}\text{oots}_{g_1})$ et (2) préserve l'étiquette et les successeurs des nœuds de g_1 qui ne sont pas étiquetés par des variables : $\mathcal{L}_{g_2}(h(n)) = \mathcal{L}_{g_1}(n)$ et $\mathcal{S}_{g_2}(h(n)) = h(\mathcal{S}_{g_1}(n))$ pour tout $n \in \mathcal{N}_{g_1}$ tel que $\mathcal{L}_{g_1}(n) \notin \mathcal{X}$.

Exemple 2.3.1 Considérons les graphes G et L de la Figure 2.4. Soit μ l'application allant de $\mathcal{N}_L$ vers $\mathcal{N}_{(G_{|\texttt{n2}})}$ telle que $\mu(\texttt{l1}) = \texttt{n2}$, $\mu(\texttt{l2}) = \mu(\texttt{l4}) = \texttt{n3}$ et $\mu(\texttt{l3}) = \mu(\texttt{l5}) = \texttt{n4}$. μ est un homomorphisme de L vers $G_{|\texttt{n2}}$. □

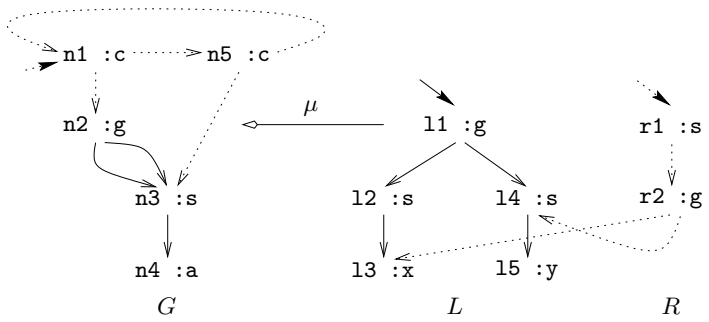


FIG. 2.4:

La définition que nous donnons ci-dessous tient compte du typage :

Définition 2.3.2 (Homomorphisme)
Un homomorphisme (enraciné) $h : g_1 \to g_2$ allant d'un graphe g_1 vers un graphe g_2 est une application $h = \uplus_{s \in S} h_s$ avec $h_s : (\mathcal{N}_{g_1})_s \to (\mathcal{N}_{g_2})_s$ telle que :

1. h préserve la valeur de la fonction d'étiquetage et de la fonction successeur de tous les nœuds de g_1 qui ne sont pas étiquetés par des variables : Pour tout nœud $n \in (\mathcal{N}_{g_1})_s$ tel que $(\mathcal{L}_{g_1})_s(n) \notin \mathcal{X}_s$,
   - $(\mathcal{L}_{g_2})_s(h_s(n)) = (\mathcal{L}_{g_1})_s(n)$.
   - si $(\mathcal{S}_{g_1})_s(n) = n_1 \dots n_k$, $k \geq 0$, avec $n_i \in (\mathcal{N}_{g_1})_{s_i}$ pour tout $i \in 1..k$, alors $(\mathcal{S}_{g_2})_s(h_s(n)) = h_{s_1}(n_1) \dots h_{s_k}(n_k)$.

2. h associe un sous-graphe de g_2 aux nœuds de g_1 qui sont étiquetés par des variables : Pour tout nœud $n \in (\mathcal{N}_{g_1})_s$ tel que $(\mathcal{L}_{g_1})_s(n) \in \mathcal{X}_s$, $h_s(n) \in (\mathcal{N}_{g_2})_s$.
3. Les images des racines de g_1 sont les racines de g_2 : si $\mathcal{R}\text{oots}_{g_1} = n_1 \dots n_k$, $k \geq 0$, avec $n_i \in (\mathcal{N}_{g_1})_{s_i}$ pour tout $i \in 1..k$, alors $\mathcal{R}\text{oots}_{g_2} = h_{s_1}(n_1) \dots h_{s_k}(n_k)$.

◇

Nous rappelons maintenant quelques définitions classiques sur les homomorphismes. Soient $h_1 : g_1 \to g_2$ et $h_2 : g_2 \to g_3$ deux homomorphismes. La *composition* de h_1 et h_2, notée $h_2 \circ h_1$, est une application $h : \mathcal{N}_{g_1} \to \mathcal{N}_{g_3}$ telle que $h(n) = h_2(h_1(n))$ pour tout $n \in \mathcal{N}_{g_1}$. Il est clair que $h_2 \circ h_1$ est un homomorphisme de g_1 vers g_3. De plus, il est clair que la fonction identité sur $\mathcal{N}_g$ est un homomorphisme de g vers lui-même, pour tout graphe g.

Un homomorphisme $h : g_1 \to g_2$ est *injectif au nœud* $n \in \mathcal{N}_{g_1}$ ssi pour tout $n' \in \mathcal{N}_{g_1}$, $n' \neq n \Longrightarrow h(n') \neq h(n)$. h est *injectif* ssi h est injectif au nœud n, pour tout $n \in \mathcal{N}_{g_1}$. h est *surjectif* ssi pour tout $q \in \mathcal{N}_{g_2}$, il existe $p \in \mathcal{N}_{g_1}$ tel que $h(p) = q$. Un *isomorphisme* est un homomorphisme injectif et surjectif.

Définition 2.3.3 (Restriction et Image)
Soient g_1 et g_2 deux graphes, $h : g_1 \to g_2$ un homomorphisme, P une chaîne de nœuds de g_1 et g un sous-graphe de g_1.
Par *restriction de h à P*, notée $h_{|P}$, nous entendons la restriction de h à $\mathcal{N}_{(g_{1|P})}$. $h_{|P}$ reste un homomorphisme allant du graphe $g_{1|P}$ vers le graphe $g_{2|h(P)}$.
Nous définissons l'*image de g par h* en posant $h(g) = g_{2|h(\mathcal{R}\text{oots}_g)}$. ◇

Exemple 2.3.4 Dans la Figure 2.4, $\mu_{|\texttt{l3.l4}}$ est l'homomorphisme allant du graphe `l3 :x + l4 :s(l5 :y)` vers le graphe `n4 :a + n3 :s(n4)`. On a $\mu($`l3 :x + l4 :s(l5 :y)`$) =$ `n4 :a + n3 :s(n4)`. □

Nous donnons ci-dessous la définition du filtrage de graphes. Informellement, un graphe l filtre un graphe g si on peut projeter l sur g avec un homomorphisme.

Définition 2.3.5 (Filtrage)
Etant donné un graphe g, un terme-graphe l et un nœud n de g, on dit que l *filtre g au nœud n* ssi il existe un homomorphisme $h : l \to g_{|n}$ (tel que $h(l) = g_{|n}$). h est appelé *filtre de l sur $g_{|n}$*. ◇

Exemple 2.3.6 Dans la Figure 2.4, μ est un filtre de L sur G au nœud `n2`. □

Nous définissons maintenant l'unification des termes-graphes. L'unification des termes du premier ordre est un problème classique [Rob65]. De nombreux travaux ont été faits (voir par exemple [Kni89, JK91, BS94]) et plusieurs algorithmes ont été proposés, par exemple [Hue76, MM82]. Le problème de l'unification des termes-graphes se ramène, en fait, à celui de l'unification des termes infinis rationnels (puisqu'on obtient un terme rationnel en "dépliant" un graphe (voir Figure 2.9 page 35)). Un algorithme d'unification des termes rationnels est proposé dans [Col82]. Notons que [HP94] développe une étude du problème de l'unification des hypergraphes d'ordre quelconque. Notre définition s'inspire de [AK96, Definition 5.4]. Elle sera complétée dans le Chapitre 5.

Définition 2.3.7 (Unification)
Soient g_1 et g_2 deux termes-graphes compatibles.

- Deux termes-graphes compatibles g_1 et g_2 sont *unifiables* ssi il existe un homomorphisme $h : L \to M$ où L et M sont deux graphes tels que (1) $g_1 \oplus g_2$ soit un sous-graphe de L et (2) $h(g_1) = h(g_2)$. Un tel homomorphisme $h : L \to M$ est appelé *unificateur* de g_1 et g_2.
- On dit qu'un homomorphisme $h : L \to M$ est un *unificateur le plus général* de g_1 et g_2 ssi L et M sont deux graphes tels que (1) $L = g_1 \oplus g_2$, (2) $h(g_1) = h(g_2)$ et (3) pour tout unificateur $h' : L' \to M'$ de g_1 et g_2, il existe un homomorphisme $\phi : h(g_1 \oplus g_2) \to h'(g_1 \oplus g_2)$.

◇

Exemple 2.3.8 Soient L_1 = `n1 :f(n2 :a,n3 :x)` et L_2 = `m1 :f(m2 :y,m3 :s(m4 :a))` deux termes-graphes compatibles. On constate dans la Figure 2.5 qu'il existe un graphe M = `p1 :f(p2 :a,p3 :s(p4 :a))` et deux homomorphismes $h_1 : L_1 \to M$ et $h_2 : L_2 \to M$. Par conséquent, il existe un graphe $g = M \oplus M$ = `p1 :f(p2 :a,p3 :s(p4 :a)) + p1` et un homomorphisme $\upsilon : (L_1 \oplus L_2) \to g$ tels que $\upsilon(L_1) = \upsilon(L_2)$. Autrement dit, (1) L_1 et L_2 sont unifiables et (2) $\upsilon : (L_1 \oplus L_2) \to g$ est un unificateur.

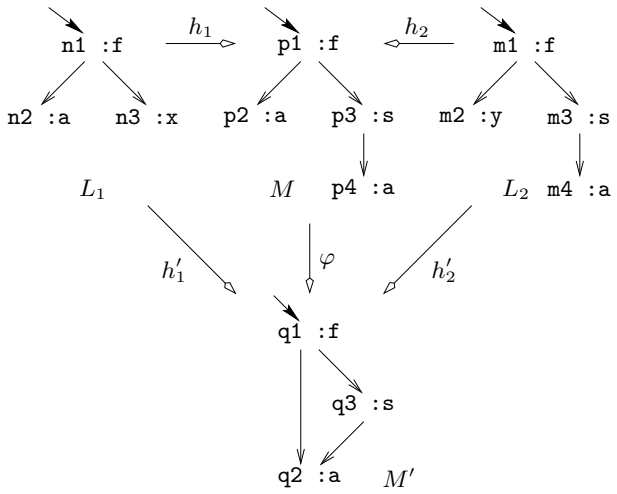


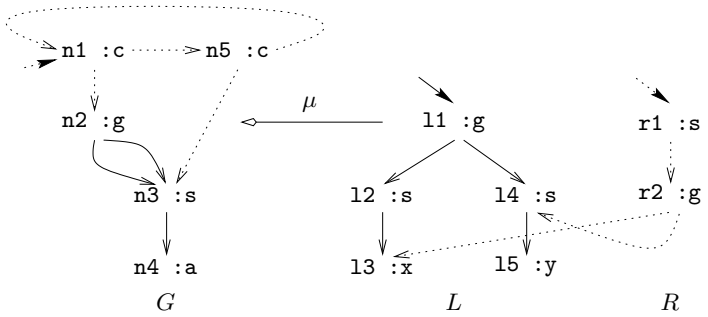
FIG. 2.5:

Considérons maintenant le graphe $g' = M' \oplus M'$ = `q1 :f(q2 :a,q3 :s(q2)) + q1` et l'homomorphisme $\upsilon' : (L_1 \oplus L_2) \to g'$ (cf. Figure 2.5). Comme $\upsilon'(L_1) = \upsilon'(L_2)$, on déduit que υ' est aussi un unificateur de L_1 et L_2. On remarque aussi qu'il existe un homomorphisme $\phi : g \to g'$. Donc υ' est un unificateur "moins général" que υ. En fait, M est un arbre, donc on déduit que υ est un unificateur *le plus général* de L_1 et L_2. □

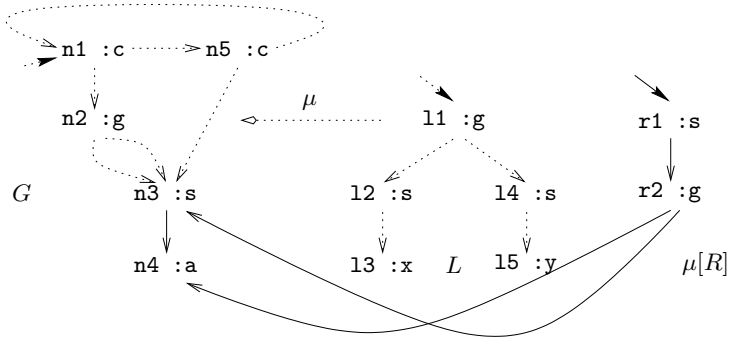
2.4 Application d'un homomorphisme sur un graphe

Nous nous intéressons maintenant à la notion d'*application* d'un homomorphisme $h : g_1 \to g_2$ sur un graphe g, que l'on note $h[g]$. Au niveau des termes, ce calcul correspond à l'application $\sigma(t)$ de l'extension d'une substitution σ sur un terme t. Calculer $h[g]$ consiste à substituer, dans g, tous les sous-graphes communs entre g et g_1 par les sous-graphes correspondants dans g_2. On réalise ceci avec une redirection de pointeurs.

Exemple 2.4.1 Considérons les graphes G, L et R de la Figure 2.6 (a). Nous avons vu dans l'Exemple 2.3.1 qu'il existe un homomorphisme $\mu : L \to G_{|\texttt{n2}}$. Calculer $\mu[R]$ consiste à rem-



(a) Filtrage de L sur $G_{|\texttt{n2}}$ avec l'homomorphisme μ



(b) Evaluation de $\mu[R]$

FIG. 2.6:

placer tous les sous-graphes communs entre R et L par leurs sous-graphes correspondants dans $G_{|\texttt{n2}}$. Ici, R et L partagent les sous-graphes `l3 :x` et `l4 :s(l5 :y)` dont les images par μ sont respectivement `n4 :a` et `n3 :s(n4 :a)`. Aussi, $\mu[R]$ = `r1 :s(r2 :g(n4 :a,n3 :s(n4)))` (cf. Figure 2.6 (b)). □

Pour donner une définition formelle de l'application d'un homomorphisme sur un graphe, il faut prendre des précautions. Tout d'abord, il peut arriver que $h[g]$ ne soit pas un graphe :

Exemple 2.4.2 Soient g = `p1 :A(n1 :B(n2 :X),m1 :C(p2 :Y))` et $h : g_1 \to g_2$ l'homomorphisme tel que g_1 = `n1 :B(n2 :X)` et g_2 = `m1 :B(m2 :Y)`. Calculer $h[g]$ consiste à remplacer dans g le sous-graphe `n1 :B(n2 :X)` par `m1 :B(m2 :Y)`. Nous obtenons $h[g]$ = `p1 :A(m1 :B(m2 :Y),m1 :C(p2 :Y))`. $h[g]$ n'est pas un graphe puisque le nœud `m1` a deux étiquettes disctinctes `B` et `C`. De plus, la variable `Y` est l'étiquette de deux nœuds différents `m2` et `p2`. □

Ensuite, nous avons vu dans l'Exemple 2.2.9 que l'évaluation d'un remplacement pouvait avoir des effets de bord surprenants. Il en est de même pour le calcul de $h[g]$:

Exemple 2.4.3 Soient $g_1 =$ `n :X` et $g_2 =$ `p :B(n :X)` deux graphes. Il existe un homomorphisme $h : g_1 \to g_2$. Calculons $h[g_1]$: le seul sous-graphe que partage g_1 avec lui-même est lui-même, donc $h[g_1] = g_1[\texttt{n} \leftarrow g_2]$. Le lecteur peut vérifier que $g_1[\texttt{n} \leftarrow g_2] =$ `p :B(p)`. Ici, h est un homomorphisme allant de g_1 vers g_2, mais $h[g_1] \neq g_2$. □

Enfin, la définition du calcul de $h[g]$ avec des remplacements peut présenter un autre inconvénient : le résultat du remplacement de plusieurs sous-graphes par d'autres peut dépendre de l'ordre dans lequel on effectue ces remplacements.

Exemple 2.4.4 Soient $g_1 =$ `n1 :X + n2 :Y`, $g_2 =$ `m1 :B(n2 :Y) + m2 :A` et $h : g_1 \to g_2$ l'homomorphisme tel que $h($`n1`$) =$ `m1` et $h($`n2`$) =$ `m2`. Soit $g =$ `p1 :C(n1 :X,n2 :Y)`. Les sous-graphes que partagent g et g_1 sont `n1 :X` et `n2 :Y`. Le lecteur peut remplacer `n1 :X` par `m1 :B(n2 :Y)` dans un premier temps, puis `n2 :Y` par `m2 :A` dans un deuxième temps. Le graphe $h[g]$ serait alors `p1 :C(m1 :B(m2 :A),m2)`. Le lecteur peut aussi faire l'inverse, c'est-à-dire remplacer `n2 :Y` par `m2 :A` dans un premier temps puis `n1 :X` par `m1 :B(n2 :Y)` dans un deuxième temps. Le graphe $h[g]$ serait alors `p1 :C(m1 :B(n2 :Y),m2 :A)`. Ces deux résultats sont clairement différents. □

Après avoir relevé plusieurs problèmes posés par l'application d'un homomorphisme sur un graphe, nous en donnons une définition formelle. Nous commençons avec la notion de *frontière* entre deux graphes qui définit l'ensemble des nœuds où nous devrons ensuite faire des redirections de pointeurs.

Définition 2.4.5 (Frontière)
Soient g et g_1 deux graphes compatibles. On appelle *frontière de g avec g_1* l'ensemble

$$\mathcal{B}(g, g_1) = \{p \in (\mathcal{N}_g \cap \mathcal{N}_{g_1}) \mid p \in \mathcal{R}\text{oots}_g \text{ ou } \exists q \in (\mathcal{N}_g - \mathcal{N}_{g_1}), p \in \mathcal{S}_g(q)\}$$

◇

Exemple 2.4.6 Dans la Figure 2.6 (a), $\mathcal{B}(R, L) = \{$`13`$,$`14`$\}$. □

Pour éviter les problèmes présentés dans les exemples précédents, nous introduisons la notion de compatibilité d'un graphe avec un homomorphisme :

Définition 2.4.7 (Compatibilité d'un graphe avec un homomorphisme)
Soient g un graphe et $h : g_1 \to g_2$ un homomorphisme.
On dit que g *est compatible avec h* ssi :

1. g, g_1 et g_2 sont compatibles deux à deux.
2. $\mathcal{B}(g, g_1) \cap \mathcal{N}_{g_2} = \emptyset$.

◇

Exemple 2.4.8 Dans la Figure 2.6 (a), on constate que les graphes R, L et $G_{|\texttt{n2}}$ sont compatibles deux à deux. De plus, nous avons vu que $\mathcal{B}(R, L) = \{$`13`$,$`14`$\}$. Il est clair que $\mathcal{B}(R, L) \cap \mathcal{N}_{(G_{|\texttt{n2}})} = \emptyset$. Donc nous concluons que R est compatible avec l'homomorphisme $\mu : L \to G_{|\texttt{n2}}$. Revenons maintenant sur les exemples problématiques. L'Exemple 2.4.2 est un cas où g n'est pas compatible avec g_2, ce qui viole la première condition de compatibilité de g avec $h : g_1 \to g_2$. Quant aux Exemples 2.4.3 et 2.4.4, ils présentent des cas où $\mathcal{B}(g, g_1) \cap \mathcal{N}_{g_2} \neq \emptyset$, ce qui viole la seconde condition de compatibilité de g avec $h : g_1 \to g_2$. □

Nous sommes prêts pour poser la définition suivante :

Définition 2.4.9 (Application d'un homomorphisme sur un graphe)
Soient g un graphe et $h : g_1 \to g_2$ un homomorphisme tels que g soit compatible avec h. Soit $B = \{p_1, \ldots, p_k\}$ la frontière entre g et g_1. L'*application de h sur g*, notée $h[g]$, est définie par :

$$h[g] = g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$$

◇

Exemple 2.4.10 Nous avons vu que le graphe R et l'homomorphisme $\mu : L \to G_{|\texttt{n2}}$ de la Figure 2.6 étaient compatibles. De plus, $\mathcal{B}(R, L) = \{\texttt{l3}, \texttt{l4}\}$. Donc, par définition, $\mu[R] = R[\texttt{l3} \leftarrow G_{|\mu(\texttt{l3})}][\texttt{l4} \leftarrow G_{|\mu(\texttt{l4})}]$, c'est-à-dire $\mu[R] = R[\texttt{l3} \leftarrow \texttt{n4 :a}][\texttt{l4} \leftarrow \texttt{n3 :s(n4 :a)}] =$ `r1 :s(r2 :g(n4 :a,n3 :s(n4)))`. □

Nous donnons ci-dessous une preuve de bonne construction de $h[g]$:

Proposition 2.4.11 Soient g un graphe et $h : g_1 \to g_2$ un homomorphisme tels que g soit compatible avec h. Alors $h[g]$ est un graphe défini de manière unique. ◇

Preuve : Soit $\{p_1, \ldots, p_k\} = \mathcal{B}(g, g_1)$ la frontière de g par rapport à g_1. Nous commençons par montrer que l'expression $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$ définit bien un graphe. Pour cela, nous prouvons par induction la proposition $\mathcal{P}(i)$ suivante : $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_i \leftarrow g_{2|h(p_i)}]$ *définit un graphe qui est compatible avec* g_2.

$\mathcal{P}(1)$ est évidente. En effet, g et g_2 sont compatibles par hypothèse, donc le remplacement $g[p_1 \leftarrow g_{2|h(p_1)}]$ définit bien un graphe. De plus, $\mathcal{B}(g, g_1) \cap \mathcal{N}_{g_2} = \emptyset$, puisque g est compatible avec h par hypothèse. Comme $p_1 \in \mathcal{B}(g, g_1)$, on en déduit que $g[p_1 \leftarrow g_{2|h(p_1)}]$ est compatible avec g_2 (i.e., g_2 n'est pas modifié par la redirection de pointeurs permettant de calculer $g[p_1 \leftarrow g_{2|h(p_1)}]$).

Soit $i \in 1..k-1$. Supposons que $\mathcal{P}(i)$ soit vraie. Puisque $g_{2|h(p_{i+1})}$ est un sous-graphe de g_2 et que $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_i \leftarrow g_{2|h(p_i)}]$ est un graphe compatible avec g_2, $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_i \leftarrow g_{2|h(p_i)}]$ est compatible avec $g_{2|h(p_{i+1})}$. Donc le remplacement $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_{i+1} \leftarrow g_{2|h(p_{i+1})}]$ est un graphe. De plus, p_{i+1} n'est pas un nœud de g_2 puisque $p_{i+1} \in \mathcal{B}(g, g_1)$ et $\mathcal{B}(g, g_1) \cap \mathcal{N}_{g_2} = \emptyset$. Donc $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_{i+1} \leftarrow g_{2|h(p_{i+1})}]$ est compatible avec g_2.

Nous concluons donc que $\mathcal{P}(i)$ est vraie pour tout $i \in 1..k$. Donc l'expression $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$ définit bien un graphe (qui est compatible avec g_2).

Nous montrons maintenant que $h[g]$ est défini de manière unique, c'est-à-dire que l'ordre dans lequel sont faits les remplacements dans $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$ n'a aucune importance. p_i n'est pas un nœud de $g_{2|h(p_j)}$ pour tout $p_i, p_j \in \mathcal{B}(g, g_1)$ puisque $\mathcal{B}(g, g_1) \cap \mathcal{N}_{g_2} = \emptyset$. Nous verrons (Proposition 3.4.6) qu'étant donné un graphe g, deux termes-graphes u_1 et u_2 et deux nœuds n_1 et n_2 tels que g, u_1 et u_2 soient compatibles, si $n_1 \notin \mathcal{N}_{u_2}$ et $n_2 \notin \mathcal{N}_{u_1}$, alors $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow u_1]$. Ici, p_i n'est pas un nœud de $g_{2|h(p_j)}$ pour tout $p_i, p_j \in \mathcal{B}(g, g_1)$ (puisque $\mathcal{B}(g, g_1) \cap \mathcal{N}_{g_2} = \emptyset$), donc nous concluons que l'ordre dans lequel sont faits les remplacements dans $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$ n'a pas d'importance. □

2.5 Substitutions

Indépendemment des homomorphismes, nous avons besoin des substitutions dans les Chapitres 5 et 6 pour définir le calcul de surréduction, sa cohérence et sa complétude.

Définition 2.5.1 (Substitution)
Une *substitution* σ est une fonction totale d'un ensemble $V \subseteq \mathcal{X}$ de variables vers un ensemble de termes-graphes compatibles deux à deux. On définit le *domaine* de σ, noté $\mathcal{D}\sigma$, comme l'ensemble des variables x telles que $\sigma(x)$ ne soit pas un graphe composé d'un seul nœud étiqueté par la variable x. Par Id, nous désignons n'importe quelle substitution telle

que $\mathcal{D}(Id) = \emptyset$. Nous définissons l'*image* de σ, noté $\mathcal{I}\sigma$, comme l'ensemble des variables du codomaine de σ : $\mathcal{I}\sigma = \bigcup_{x \in \mathcal{D}\sigma} \mathcal{V}_{\sigma(x)}$. On dit que σ est *idempotente* ssi $\mathcal{D}\sigma \cap \mathcal{I}\sigma = \emptyset$. Etant donné un ensemble $V \subseteq \mathcal{X}$ de variables, on définit la *restriction de σ à V*, notée $\sigma_{|V}$, comme la substitution de domaine $V \cap \mathcal{D}\sigma$ telle que $\sigma_{|V}(x) = \sigma(x)$ pour tout $x \in V \cap \mathcal{D}\sigma$. ◇

Exemple 2.5.2 Soit σ la fonction telle que $\sigma(\mathtt{U}) = \mathtt{n1\ :A}$, $\sigma(\mathtt{V}) = \mathtt{n2\ :B(n1\ :A)}$, $\sigma(\mathtt{W}) = \mathtt{n3\ :C(n4\ :X)}$ et $\sigma(\mathtt{Z}) = \mathtt{n5\ :Z}$. σ est une substitution de domaine $\mathcal{D}\sigma = \{\mathtt{U}, \mathtt{V}, \mathtt{W}\}$ et d'image $\mathcal{I}\sigma = \{\mathtt{X}\}$. La restriction de σ à $\{\mathtt{U}, \mathtt{V}\}$ est la substitution $\sigma_{|\{\mathtt{U},\mathtt{V}\}} = \{\mathtt{U} \mapsto \mathtt{n1\ :A}, \mathtt{V} \mapsto \mathtt{n2\ :B(n1\ :A)}\}$. □

Nous nous intéressons maintenant à l'*application* d'une substitution sur un graphe. En fait, une substitution est une sorte d'homomorphisme qui affecte les "variables" d'un graphe, c'est-à-dire des sous-graphes réduits à un nœud étiqueté par une variable, avec des termes-graphes. Aussi, nous définissons l'application d'une substitution σ sur un graphe g comme l'application d'un homomorphisme particulier noté $h_{\sigma,g}$, i.e., $\sigma(g) = h_{\sigma,g}[g]$.

Définition 2.5.3 (Homomorphisme engendré par une substitution et un graphe)
Nous définissons l'*homomorphisme engendré par une substitution σ et un graphe g* comme l'homomorphisme $h_{\sigma,g} : l_{\sigma,g} \to r_{\sigma,g}$ tel que :

1. Soit $V = x_1 \dots x_p$ une chaîne des variables de $\mathcal{V}_g \cap \mathcal{D}\sigma = \{x_1, \dots, x_p\}$.
2. Soit $N = n_1 \dots n_p$ la chaîne des nœuds de g telle que $\mathcal{L}_g(n_i) = x_i$ pour tout $i \in 1..p$.
3. $l_{\sigma,g}$ est le graphe $g_{|N}$.
4. $r_{\sigma,g}$ est le graphe $\oplus_{x \in V} \sigma(x)$.
5. Enfin, $h_{\sigma,g}(n) = \mathcal{R}\mathrm{oot}_{\sigma(x)}$ pour tout $n \in N$ tel que $\mathcal{L}_g(n) = x$.

◇

Exemple 2.5.4 Soient $\sigma = \{\mathtt{U} \mapsto \mathtt{n1\ :A}, \mathtt{V} \mapsto \mathtt{n2\ :B(n1\ :A)}, \mathtt{W} \mapsto \mathtt{n3\ :C(n4\ :X)}\}$ une substitution et $g = \mathtt{p1\ :D(n1\ :A,p2\ :U,p3\ :V)}$ un graphe. Nous voulons calculer $\sigma(g)$. L'homomorphisme $h_{\sigma,g} : l_{\sigma,g} \to r_{\sigma,g}$ est représenté dans la Figure 2.7 et défini comme suit :

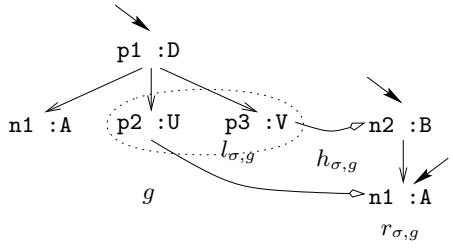


FIG. 2.7:

1. Soient $V = \mathtt{U.V}$ et $N = \mathtt{p2.p3}$.
2. $l_{\sigma,g} = g_{|N} = \mathtt{p2\ :U\ +\ p3\ :V}$.
3. $r_{\sigma,g} = \oplus_{x \in V} \sigma(x) = \mathtt{n1\ :A\ +\ n2\ :B(n1)}$.
4. $h_{\sigma,g}(\mathtt{p2}) = \mathtt{n1}$ et $h_{\sigma,g}(\mathtt{p3}) = \mathtt{n2}$.

□

Comme l'application d'un homomorphisme sur un graphe, l'application d'une substitution sur un graphe est sujette à caution. Aussi, nous introduisons la notion suivante :

Définition 2.5.5 (Compatibilité d'un graphe avec une substitution)
Soient g un graphe et σ une substitution. On dit que g *est compatible avec* σ ssi :

1. σ est idempotente : $\mathcal{D}\sigma \cap \mathcal{I}\sigma = \emptyset$.
2. g est compatible avec $\sigma(x)$ pour tout $x \in \mathcal{D}\sigma$.

◇

Exemple 2.5.6 Considérons la substitution σ et le graphe g de l'Exemple 2.5.4. $\mathcal{D}\sigma \cap \mathcal{I}\sigma = \emptyset$ et g est compatible avec $\sigma(U)$, $\sigma(V)$ et $\sigma(W)$, donc g est compatible avec σ. □

La première condition de la définition précédente peut surprendre le lecteur. En effet, il n'est pas nécessaire qu'une substitution soit idempotente pour l'appliquer sur un terme du premier ordre. Néanmoins, elle est nécessaire dans le cadre des graphes pour éviter les problèmes de mauvaise définition. De plus, elle n'est pas gênante en pratique puisqu'elle est sous-jacente à la plupart des résultats établis sur la surréduction de termes du premier ordre (voir par exemple [MH92]).

Définition 2.5.7 (Application d'une substitution sur un graphe)
Soient g un graphe et σ une substitution tels que g soit compatible avec σ. L'*application de* σ *sur* g, notée $\sigma(g)$, est définie par $\sigma(g) = h_{\sigma,g}[g]$. ◇

Exemple 2.5.8 Considérons de nouveau la substitution σ et le graphe g de l'Exemple 2.5.4. Avec la définition de l'homomorphisme $h_{\sigma,g} : l_{\sigma,g} \to r_{\sigma,g}$, nous déduisons que $\sigma(g) = h_{\sigma,g}[g] =$ `p1 :D(n1 :A,n1,n2 :B(n1))`. □

Proposition 2.5.9 Soient g un graphe et σ une substitution tels que g soit compatible avec σ. Alors $\sigma(g)$ est un graphe défini de manière unique. ◇

Preuve : Nous montrons que g est compatible avec $h_{\sigma,g} : l_{\sigma,g} \to r_{\sigma,g}$. Premièrement, $l_{\sigma,g}$ est un sous-graphe de g, donc $l_{\sigma,g}$ est compatible avec g. De plus, $r_{\sigma,g}$ est une somme de $\sigma(x)$, et g (donc $l_{\sigma,g}$) est compatible avec $\sigma(x)$ pour tout $x \in \mathcal{D}\sigma$ par compatibilité de g avec σ. Par conséquent, g, $l_{\sigma,g}$ et $r_{\sigma,g}$ sont compatibles deux à deux. Deuxièmement, $\mathcal{B}(g, l_{\sigma,g})$ est un ensemble de nœuds étiquetés par des variables dans g et ces variables sont dans $\mathcal{D}\sigma$. De plus, $r_{\sigma,g}$ est une somme de $\sigma(x)$. Comme g, $l_{\sigma,g}$ et $r_{\sigma,g}$ sont compatibles deux à deux et $\mathcal{D}\sigma \cap \mathcal{I}\sigma = \emptyset$, on en conclut que $\mathcal{B}(g, l_{\sigma,g}) \cap \mathcal{N}_{(r_{\sigma,g})} = \emptyset$. Donc g est compatible avec $h_{\sigma,g}$ ce qui implique que $\sigma(g) = h_{\sigma,g}[g]$ est un graphe défini de manière unique d'après la Proposition 2.4.11. □

Nous définissons maintenant la composition de deux substitutions :

Définition 2.5.10 (Composition de substitutions)
Soient σ_1 et σ_2 deux substitutions telles que $\sigma_1(x)$ soit compatible avec σ_2 pour tout $x \in \mathcal{D}\sigma_1$. La *composition de* σ_1 *avec* σ_2, notée $\sigma_2 \circ \sigma_1$, est la substitution σ telle que $\sigma(x) = \sigma_2(\sigma_1(x))$ pour tout $x \in \mathcal{D}\sigma_1$ et $\sigma(x) = \sigma_2(x)$ pour tout $x \in (\mathcal{D}\sigma_2 - \mathcal{D}\sigma_1)$. ◇

Exemple 2.5.11 Soient $\sigma_1 = \{$`X` $\mapsto$ `n1 :B(n2 :U)`, `Y` $\mapsto$ `n3 :B(n4 :V)`$\}$ et $\sigma_2 = \{$`U` $\mapsto$ `n5 :A`$\}$. Le lecteur peut vérifier que $\sigma_2 \circ \sigma_1 = \{$`X` $\mapsto$ `n1 :B(n5 :A)`, `Y` $\mapsto$ `n3 :B(n4 :V)`, `U` $\mapsto$ `n5 :A`$\}$. □

Nous avons abordé la notion d'homomorphisme engendré par une substitution et un graphe dans la Définition 2.5.7. Nous nous intéressons ci-dessous à la notion duale de *substitution engendrée par un homomorphisme* : étant donné un homomorphisme $h : g_1 \to g_2$, on définit la substitution σ_h qui lie les variables de g_1 avec leurs valeurs dans g_2 par h.

Définition 2.5.12 (Substitution engendrée par un homomorphisme)
Soient g_1 et g_2 deux graphes compatibles et $h : g_1 \to g_2$ un homomorphisme. Soit N l'ensemble des nœuds étiquetés par des variables de g_1 qui sont affectées avec des termes-graphes de g_2 par h et soit V l'ensemble des étiquettes des nœuds de N :

$$N = \{n \in \mathcal{N}_{g_1} \mid \mathcal{L}_{g_1}(n) \in \mathcal{X} \text{ et } \mathcal{L}_{g_2}(h(n)) \neq \mathcal{L}_{g_1}(n)\} \text{ et } V = \{\mathcal{L}_{g_1}(n) \mid n \in N\}$$

Nous définissons la *substitution engendrée par l'homomorphisme* h, notée σ_h, comme la substitution de domaine V telle que $\sigma_h(x) = g_{2|h(n)}$ pour tout $n \in N$ et pour tout $x \in V$ tels que $\mathcal{L}_{g_1}(n) = x$. ◇

Exemple 2.5.13 Considérons l'homomorphisme $\mu : L \to G_{|\texttt{n2}}$ de l'Exemple 2.3.1. σ_μ est définie par $\sigma_\mu(\texttt{u}) = \sigma_\mu(\texttt{v}) = \texttt{n4 :a}$. Le lecteur peut vérifier que $\sigma_\mu(L) = \texttt{l1 :g(l2 :s(n4 :a),l4 :s(n4))}$ et $\mu[L] = G_{|\texttt{n2}} = \texttt{n2 :g(n3 :s(n4 :a),n3)}$. Il est clair que $\sigma_\mu(L)$ et $\mu[L]$ ne sont pas égaux. En fait, on peut montrer qu'il existe un homomorphisme $\phi : \sigma_\mu(L) \to \mu[L]$. □

2.6 Préordres et égalités sur les graphes et les substitutions

Nous finissons ce chapitre de préliminaires en faisant la liste des préordres et des égalités définis sur les graphes et les substitutions que nous utiliserons dans la suite de cette thèse. Concernant les relations d'égalités sur les graphes, nous avons déjà utilisé l'*égalité syntaxique* $=$ de deux graphes. Nous donnons ci-dessous la définition de trois autres égalités : égalité au renommage (des nœuds et des variables) près $\approx$, égalité au renommage des nœuds près $\sim$ et bisimilarité $\doteq$.

On dit que deux graphes sont égaux au renommage près si on peut obtenir l'un à partir de l'autre en renommant ses nœuds et ses variables :

Définition 2.6.1 (Egalité au renommage près)
Deux graphes g_1 et g_2 sont *égaux au renommage (des nœuds et des variables) près* ssi il existe un isomorphisme $\phi : g_1 \to g_2$. Nous écrirons alors $g_1 \approx_\phi g_2$ ou simplement $g_1 \approx g_2$. ◇

On dit que deux graphes sont égaux au renommage des nœuds près si on peut obtenir l'un à partir de l'autre en renommant ses nœuds uniquement (et pas ses variables). Pour définir formellement cette notion d'égalité, on introduit la notion (importante) de $\mathcal{V}$-homomorphisme :

Définition 2.6.2 ($\mathcal{V}$-homomorphisme)
Soient g_1 et g_2 deux graphes et $h : g_1 \to g_2$ un homomorphisme. h est un *$\mathcal{V}$-homomorphisme* ssi pour tout nœud $n \in \mathcal{N}_{g_1}$ et toute variable $x \in \mathcal{V}_{g_1}$, si $\mathcal{L}_{g_1}(n) = x$, alors $\mathcal{L}_{g_2}(h(n)) = x$. ◇

Exemple 2.6.3 Dans la Figure 2.8, il existe un homomorphisme $h : g_1 \to g_2$ mais h n'est pas un $\mathcal{V}$-homomorphisme parce qu'il affecte la variable `X` de g_3 avec `n4 :A`. Les homomorphismes $h' : g_1 \to g_3$ et $h'' : g_2 \to g_4$ sont, eux, des $\mathcal{V}$-homomorphismes. □

Définition 2.6.4 (Egalité au renommage des nœuds près)
Deux graphes g_1 et g_2 sont *égaux au renommage des nœuds près* ssi il existe un $\mathcal{V}$-isomorphisme $\phi : g_1 \to g_2$. Nous écrirons alors $g_1 \sim_\phi g_2$ ou simplement $g_1 \sim g_2$. ◇

Avec l'égalité $\sim$, on ne peut changer que les nœuds d'un graphe mais pas ses variables (puisque l'isomorphisme utilisé est un $\mathcal{V}$-isomorphisme). L'égalité au renommage des nœuds près ressemble à l'α-conversion dans le λ-calcul [Bar84].

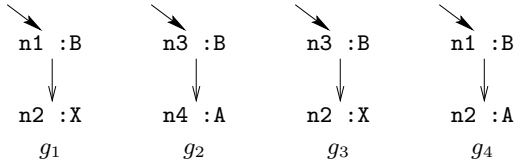


FIG. 2.8:

Nous définissons maintenant la *bisimilarité*. Informellement, deux termes-graphes sont bisimilaires s'ils représentent le même arbre infini quand on les déplie. La bisimilarité est appelée *tree-equivalence* dans [BvEG+87]. La bisimilarité entre termes-graphes sans variable fait l'objet d'une longue étude dans [AK96]. Notre définition est une conséquence de leur Théorème 3.9.

Définition 2.6.5 (Bisimilarité)
Deux termes-graphes compatibles g_1 et g_2 sont *bisimilaires*, noté $g_1 \doteq g_2$, ssi il existe un graphe g avec deux racines et un $\mathcal{V}$-homomorphisme $h : (g_1 \oplus g_2) \to g$ tels que $h(g_1) = h(g_2)$, ou de manière équivalente, ssi il existe un terme-graphe g et deux $\mathcal{V}$-homomorphismes $h_1 : g_1 \to g$ et $h_2 : g_2 \to g$. ◇

Exemple 2.6.6 Dans la Figure 2.9, les termes-graphes g_1 et g_2 sont bisimilaires puisqu'il existe un terme-graphe g et deux $\mathcal{V}$-homomorphismes $h_1 : g_1 \to g$ et $h_2 : g_2 \to g$. On remarque qu'on obtient le terme infini t lorsqu'on déplie les graphes g_1, g_2 et g. □

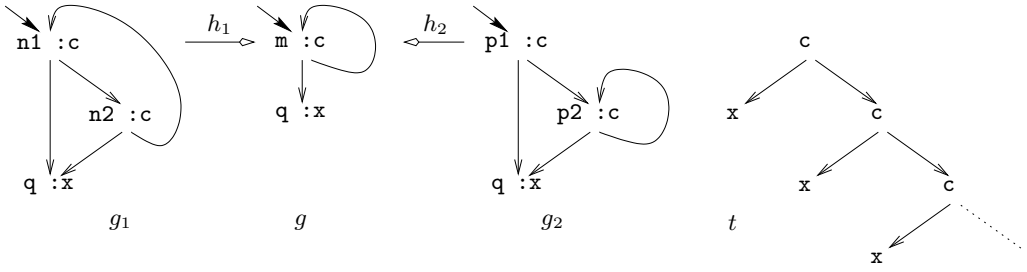


FIG. 2.9:

En fait, la notion de bisimilarité est la même que celle d'unification, sauf pour les variables qui sont traitées comme des constantes dans le cas de la bisimilarité. Deux termes-graphes avec des ensembles de variables différents ne peuvent pas être bisimilaires.

Définition 2.6.7 (Préordre sur les graphes)
Soient g_1 et g_2 deux graphes. On pose $g_1 \dot{\leq} g_2$ ssi il existe une substitution θ telle que $\theta(g_1) \doteq g_2$. ◇

Exemple 2.6.8 Soient $g_1 =$ `n1 :B(n2 :X,n2)` et $g_2 =$ `m1 :B(m2 :A,m3 :A)`. On vérifie que $g_1 \dot{\leq} g_2$. En effet, si $\theta = \{\mathtt{X} \mapsto \mathtt{p\ :A}\}$, alors $\theta(g_1) =$ `n1 :B(p :A,p)` $\doteq g_2$. □

Nous étendons maintenant les relations $\dot{\leq}$ et $\doteq$ définies sur les graphes aux substitutions.

Définition 2.6.9 (Préordre sur les substitutions)
Soient σ_1 et σ_2 deux substitutions et V un ensemble de variables. On dit que σ_1 *est plus générale que* σ_2 *sur* V, noté $\sigma_1 \dot{\leq} \sigma_2$ $[V]$, ssi il existe une substitution θ telle que $\theta \circ \sigma_1(x) \doteq \sigma_2(x)$ pour tout $x \in V$. On omettra d'écrire l'ensemble V lorsque $V = \mathcal{D}\sigma_1$. ◇

Exemple 2.6.10 Soient $\sigma_1 = \{\mathtt{U} \mapsto \mathtt{n1\ :B(n2\ :X,n2)}, \mathtt{V} \mapsto \mathtt{n3\ :A}\}$ et $\sigma_2 = \{\mathtt{U} \mapsto \mathtt{m1\ :B(m2\ :A,m3\ :A)}, \mathtt{W} \mapsto \mathtt{m4\ :A}\}$. Il est facile de vérifier que $\sigma_1 \dot{\leq} \sigma_2\ [\{\mathtt{U}\}]$. En effet, soit $\theta = \{\mathtt{X} \mapsto \mathtt{p\ :A}\}$. Alors, $\theta \circ \sigma_1(\mathtt{U}) = \mathtt{n1\ :B(p\ :A,p)} \doteq \sigma_2(\mathtt{U})$. □

Définition 2.6.11 (Relation d'équivalence sur les substitutions)
Soient σ_1 et σ_2 deux substitutions et V un ensemble de variables. On dit que σ_1 *est bisimilaire à* σ_2 *sur* V, noté $\sigma_1 \doteq \sigma_2\ [V]$, ssi $\sigma_1 \dot{\leq} \sigma_2\ [V]$ et $\sigma_2 \dot{\leq} \sigma_1\ [V]$, c'est-à-dire, $\sigma_1(x) \doteq \sigma_2(x)$ pour tout $x \in V$. ◇

Chapitre 3

Graphes admissibles, réécriture et confluence

Dans les langages logico-fonctionnels que nous considérons, les données sont modélisées à l'aide de graphes cycliques et les programmes sont vus comme des systèmes de réécriture de graphes. L'étude de ces systèmes fait l'objet d'une importante littérature (voir [SPvE93, ET96, Roz97, EEKR99], entre autres). Ils sont utilisés dans l'implantation des langages fonctionnels [Pey87] comme par exemple CLEAN [PvE93] ou HOPS [Kah94].

Une première façon d'utiliser un langage logico-fonctionnel à base de graphes consiste en l'évaluation par réécriture d'expressions modélisées sous la forme de graphes cycliques. Cette sémantique opérationnelle a plusieurs propriétés remarquables. Outre les gains en terme d'expressivité, d'espace et de temps de calcul que nous avons vus dans le chapitre d'introduction, nous nous intéressons aux "bonnes" propriétés du calcul, en particulier, la confluence de la relation de réécriture. Elle exprime le déterminisme de l'évaluation d'une expression, l'unicité de la valeur d'une expression lorsque cette valeur existe.

Les systèmes de réécriture de graphes que nous considérons (c'est-à-dire, les programmes) peuvent être vus comme une extension naturelle des systèmes de réécriture faiblement orthogonaux de termes avec constructeurs. Ceux-ci sont confluents dans le cadre des termes [Hue80], mais cette propriété n'est pas préservée dans le cadre des graphes cycliques. En effet, le système orthogonal formé par les règles `(R1) A(x)` $\rightarrow$ `x` et `(R2) B(x)` $\rightarrow$ `x` induit une relation de réécriture qui est confluente sur les termes mais pas sur les graphes, comme le montre l'exemple suivant [KKSV94] : le graphe `n :A(B(n))` peut se réécrire en deux graphes différents `u :A(u)` et `v :B(v)`.

Ce système de réécriture n'est pas confluent parce qu'il utilise des règles effondrantes[1], c'est-à-dire des règles dont les membres droits sont réduits à de simples variables. L'usage de telles règles ne peut pas être interdit dans un langage de programmation. En effet, la plupart des fonctions d'accès (comme `car` et `cdr` sur les listes) sont définies à l'aide de règles effondrantes.

Dans ce chapitre, nous mettons en évidence une classe de graphes dits *admissibles* qui est stable par réécriture et sur laquelle la relation de réécriture que nous considérons est confluente. Un graphe est admissible si ses cycles ne contiennent pas d'opération définie. Le graphe `cdr(n :cons(0,n))` est un graphe admissible. Ce n'est pas le cas du graphe `n :cdr(cons(0,n))` puisque l'opération définie `cdr` appartient à un cycle.

[1] *collapsing rules* en anglais.

Nous montrons de plus que la relation de réécriture est confluente modulo la bisimilarité par rapport aux graphes admissibles. Autrement dit, une expression modélisée par deux graphes admissibles "équivalents" (bisimilaires) a une unique valeur (forme normale) lorsque cette valeur existe. Nous aurons besoin de ce résultat dans les Chapitres 5 et 6 traitant de la surréduction de graphes.

L'ensemble de ces résultats a fait l'objet de plusieurs publications [EJ98a, EJ98b, EJ97c]. Quant aux détails techniques, ils se trouvent dans deux rapports [EJ97b, EJ98c].

Dans la section suivante, nous définissons les graphes admissibles, les règles de réécriture de graphes et des systèmes de réécriture de graphes que nous étudions dans cette thèse. Nous donnons la définition du pas de réécriture dans la Section 3.2 et nous montrons la stabilité de la classe des graphes admissibles par réécriture. Nous faisons ensuite une liste de nos résultats concernant la confluence de la relation de réécriture dans la Section 3.3. Les preuves de ces résultats sont données dans les Sections 3.4 et 3.5. Nous terminons ce chapitre avec une étude succinte de la confluence dans un cas particulier de systèmes de réécriture de graphes dits faiblement admissibles avec bisimilarité.

3.1 Graphes admissibles et systèmes de réécriture considérés

Pour des raisons aussi bien pratiques [O'D77] que théoriques [HH82], les langages déclaratifs utilisent des signatures avec constructeurs, c'est-à-dire des signatures où des fonctions qu'on appelle les *constructeurs*, qui sont utilisées pour construire les données, sont distinguées des fonctions qu'on appelle les *opérations définies*, qui sont utilisées pour calculer des informations sur les données et qui sont décrites à l'aide de règles de réécriture.

Définition 3.1.1 (Signature avec constructeurs)
Une *signature avec constructeurs* Σ est un triplet $\langle S, \mathcal{C}, \mathcal{D}\rangle$ tel que S soit un ensemble de sortes, $\mathcal{C}$ soit une famille S-indicée d'ensembles de *symboles de constructeurs* et $\mathcal{D}$ soit une famille S-indicée d'ensembles d'*opérations définies* telles que $\mathcal{C} \cap \mathcal{D} = \emptyset$ et $\langle S, \mathcal{C} \cup \mathcal{D}\rangle$ soit une signature. ◇

Exemple 3.1.2 Soit $\Sigma = \langle S, \mathcal{C}, \mathcal{D}\rangle$ la signature avec constructeurs telle que $S = \{\zeta_1, \zeta_2\}$, $\mathcal{C} = \{\mathtt{c} : \zeta_1\zeta_2 \to \zeta_2, \mathtt{s} : \zeta_1 \to \zeta_1, \mathtt{a} : \varepsilon \to \zeta_1\}$ et $\mathcal{D} = \{\mathtt{f} : \zeta_1\zeta_1 \to \zeta_2, \mathtt{g} : \zeta_1\zeta_1 \to \zeta_1, \mathtt{h} : \zeta_1\zeta_1 \to \zeta_1\}$. □

Comme nous faisons la distinction entre les opérations définies et les symboles constructeurs dans une signature, on peut définir différentes sortes de nœuds dans un graphe :

Définition 3.1.3 Soit g un graphe défini sur une signature avec constructeurs. Un nœud $n \in \mathcal{N}_g$ est un *nœud fonctionnel* (resp. *constructeur*, *variable*) si n est étiqueté avec une opération définie (resp. un symbole constructeur, une variable) dans g. On dit que g est un *terme-graphe de racine fonctionnelle* (resp. *constructeur*) si $\mathcal{R}\mathrm{oot}_g$ est un nœud fonctionnel (resp. constructeur). ◇

Exemple 3.1.4 Supposons que les graphes de la Figure 3.1 soient définis sur la signature Σ de l'Exemple 3.1.2 où `g` est une opération définie, `c`, `s` et `a` sont des symboles constructeurs et `x` et `y` sont des variables. Le nœud `n1` est un nœud constructeur, le nœud `l1` est un nœud fonctionnel et les nœuds `l3` et `l4` sont des nœuds variables. G est un terme-graphe de racine constructeur et L est un terme-graphe de racine fonctionnelle. □

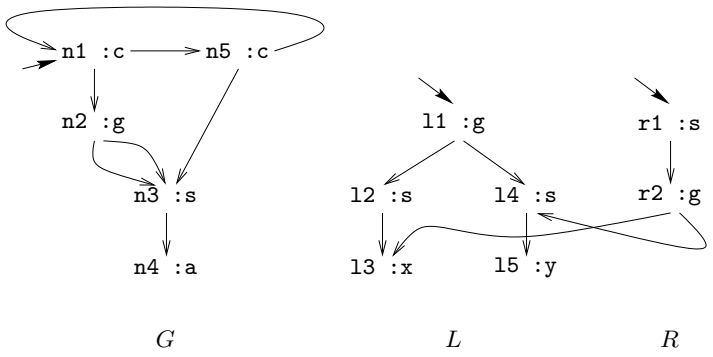


FIG. 3.1:

Dans cette thèse, nous considérons la réécriture et la surréduction de certains graphes qu'on appelle des graphes *admissibles*. Les termes-graphes admissibles correspondent, sur le plan de la programmation impérative, à des appels imbriqués de fonctions dont les paramètres sont des graphes cycliques constructeurs modélisant les structures de données classiques.

Définition 3.1.5 (Graphes admissibles et graphes constructeurs)
Un graphe g est un *graphe admissible* si il n'existe pas de chemin allant d'un nœud fonctionnel de g vers lui-même : Pour tout $n \in \mathcal{N}_g$, si $\mathcal{L}_g(n) \in \mathcal{D}$, alors $n \| n$ (i.e., $\mathcal{P}\text{aths}_g(n,n) = \emptyset$).
g est un *graphe constructeur* si g n'a pas de nœud fonctionnel : Pour tout $n \in \mathcal{N}_g$, $\mathcal{L}_g(n) \in \mathcal{C}$ ou $\mathcal{L}_g(n) \in \mathcal{X}$. Un graphe constructeur est nécessairement admissible. ◇

Exemple 3.1.6 Le graphe G de la Figure 3.1 est un terme-graphe admissible mais `z :g(z,z)` et `z :g(n :s(z),n)` n'en sont pas (puisque `g` est une opération définie appartenant à un cycle). □

Nous distinguons ci-dessous certains termes-graphes admissibles qu'on appelle des *motifs*[2] et qui sont nécessaires pour définir les règles de réécriture :

Définition 3.1.7 (Motif)
Un terme-graphe admissible g est un *motif* ssi :

1. Tous les nœuds de g sont soit des nœuds constructeurs soit des nœuds variables, sauf la racine qui est un nœud fonctionnel : Pour tout $n \in \mathcal{N}_g$, $\mathcal{L}_g(n) \in \mathcal{D} \Longleftrightarrow n = \mathcal{R}\text{oot}_g$.
2. g a une structure d'arbre, i.e., il existe au plus un chemin de $\mathcal{R}\text{oot}_g$ vers n'importe quel nœud de g : Pour tout $n \in \mathcal{N}_g$, si $n = \mathcal{R}\text{oot}_g$ alors $\mathcal{P}\text{aths}_g(\mathcal{R}\text{oot}_g, n) = \emptyset$, sinon si $n \neq \mathcal{R}\text{oot}_g$ alors $cardinal(\mathcal{P}\text{aths}_g(\mathcal{R}\text{oot}_g, n)) = 1$.

◇

Exemple 3.1.8 Dans la Figure 3.1, le graphe L est un motif. Le terme-graphe `p :B(q :X,q)` n'est pas un motif puisqu'il existe deux chemins allant de `p` vers `q`. En fait, un motif est un terme-graphe correspondant à un terme du premier ordre linéaire ne contenant qu'une seule occurence d'opération définie située à la racine. □

La définition qui suit introduit la notion de *règle de réécriture admissible*. Ces règles sont définies de sorte que l'ensemble des graphes admissibles soit stable par rapport à la relation de réécriture qu'elles induisent (Théorème 3.2.8).

[2] *pattern* en anglais.

Définition 3.1.9 (Règle de réécriture admissible)
Une *règle de réécriture* est un graphe avec deux racines noté $l \to r$ [BvEG+87]. l (resp. r) est un terme-graphe qu'on appelle le *membre gauche* (resp. *membre droit*) de la règle. Une règle e' est une *variante* d'une autre règle e ssi e' est obtenue à partir de e en renommant toutes les variables et tous les nœuds (i.e., $e \approx e'$ et tous les nœuds et toutes les variables de e' sont nouveaux).
Une règle de réécriture $l \to r$ est dite *admissible* ssi :

1. l est un *motif*.
2. r est un terme-graphe *admissible*.
3. l n'est pas un sous-graphe de r.
4. $\mathcal{V}_r \subseteq \mathcal{V}_l$.

On dit que deux règles admissibles sont *superposables* si leurs membres gauches s'unifient. ◇

Exemple 3.1.10 Le graphe $L \to R$ de la Figure 3.1 montre un exemple de règle admissible. Des règles non admissibles sont utilisées dans les Exemples 3.2.9 et 3.3.6. On remarque que les variables apparaissant dans une règle de réécriture sont toujours partagées entre le membre gauche et le membre droit (par définition des graphes). □

Nous sommes prêts pour définir les systèmes de réécriture de graphes qui sont étudiés dans la suite :

Définition 3.1.11 (Systèmes de réécriture considérés)
Un *système de réécriture de graphes avec constructeurs* (cGRS[3]) est un couple $SP = \langle \Sigma, \mathcal{R} \rangle$ tel que Σ soit une signature avec constructeurs et $\mathcal{R}$ soit un ensemble de règles de réécriture admissibles.
On dit que SP est un *système de réécriture admissible* (AGRS[4]) ssi pour toute règle distincte R_1 et R_2 de $\mathcal{R}$, R_1 et R_2 ne sont pas superposables.
On dit que SP est un *système de réécriture faiblement admissible* (WAGRS[5]) ssi $\mathcal{R}$ est un ensemble de règles telles que si $l_1 \to r_1$ et $l_2 \to r_2$ sont deux règles superposables, alors leurs membres droits instanciés sont égaux au renommage des nœuds près, i.e., si il existe un graphe admissible g et un homomorphisme $h : (l_1 \oplus l_2) \to g$ tel que $h(l_1) = h(l_2)$, alors $h[r_1] \sim h[r_2]$. ◇

Exemple 3.1.12 Soit Σ la signature avec constructeurs de l'Exemple 3.1.2. Considérons les règles de réécriture suivantes (cf. Figure 3.2) :

```
(R1) l1:f(l2:a,l3:x) -> r1:c(l3:x,r1)
(R2) l1:f(l2:x,l3:s(l4:a)) -> r1:c(r2:s(r3:a),r1)
(R3) l1:g(l2:a,l3:x) -> r1:s(l2:a)
(R4) l1:g(l2:x,l3:a) -> r1:s(l2:x)
(R5) l1:g(l2:s(l3:x),l4:s(l5:y)) -> r1:s(r2:g(l3:x,l4:s(l5:y)))
(R6) l1:h(l2:x,l3:y) -> r1:h(l3:y,l2:x)
```

Soient $\mathcal{R}_1 = \{\texttt{R1}, \texttt{R3}, \texttt{R5}, \texttt{R6}\}$ et $SP_1 = \langle \Sigma, \mathcal{R}_1 \rangle$. Comme les règles de $\mathcal{R}_1$ ne sont pas superposables deux à deux, SP_1 est un AGRS.

Posons maintenant $\mathcal{R}_2 = \{\texttt{R1}, \texttt{R2}, \texttt{R3}, \texttt{R4}, \texttt{R5}, \texttt{R6}\}$ et $SP_2 = \langle \Sigma, \mathcal{R}_2 \rangle$. Les règles `R1` et `R2` (resp. `R3` et `R4`) sont superposables et leurs membres droits instanciés sont égaux au renommage des nœuds près. Donc SP_2 est un WAGRS. □

[3] *Constructor-based Graph Rewriting System* en anglais.
[4] *Admissible Graph Rewriting System* en anglais.
[5] *Weakly Admissible Graph Rewriting System* en anglais.

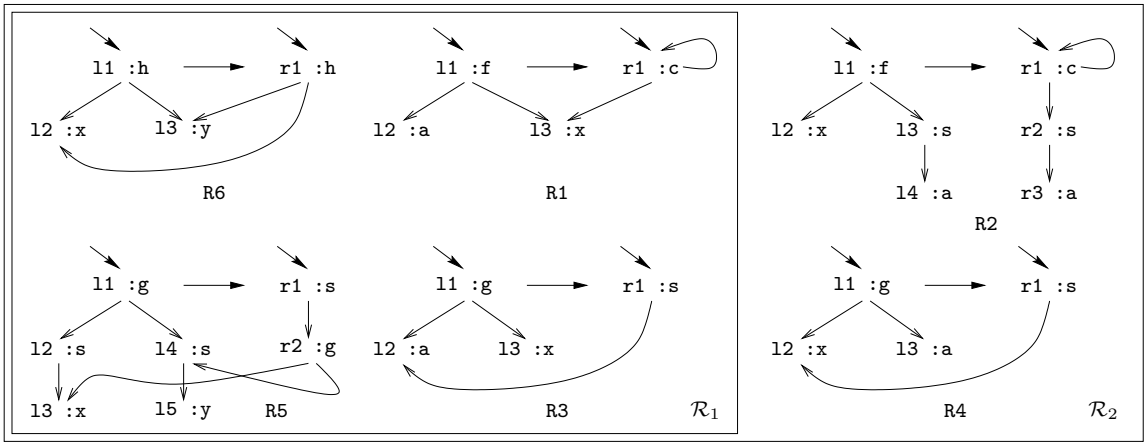


FIG. 3.2:

3.2 Pas et relation de réécriture

Réécrire un graphe se fait en deux étapes [BvEG+87]. Tout d'abord, on calcule le filtre entre le membre gauche d'une règle et un sous-graphe du graphe à réécrire ; ensuite, on remplace ce sous-graphe par le membre droit instancié de la règle.

Définition 3.2.1 (Pas de réécriture)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, g_1 et g_2 deux graphes, R une règle de réécriture de $\mathcal{R}$ et p un nœud de g_1. On dit que g_1 *se réécrit en* g_2 *au nœud* p *avec la règle* R, noté $g_1 \rightarrow_{[p,R]} g_2$, ssi :

1. Il existe une variante $l \rightarrow r$ de R et un homomorphisme $h : l \rightarrow g_{1|p}$ (tels que l filtre g_1 au nœud p avec l'homomorphisme h).
2. $g_2 = g_1[p \leftarrow h[r]]$.

Dans ce cas, on dit que $g_{1|p}$ est un *rédex* de g_1 de racine p. ◇

Exemple 3.2.2 Reprenons les Figures 2.6 page 29 et 2.2 page 2.2 et montrons le pas de réécriture du graphe G avec la règle $L \rightarrow R$ au nœud `n2` :

1. Filtrage (Figure 2.6 (a)) : On "projette" le membre gauche de la règle, L, sur le sous-graphe du graphe à réécrire, $G_{|\texttt{n2}}$. Le filtre est un homomorphisme $\mu : L \rightarrow G_{|\texttt{n2}}$.
2. On instancie le membre droit de la règle, R (Figure 2.6 (b)) : Ce calcul revient à remplacer tous les sous-graphes communs entre R et L, c'est-à-dire `l3 :x` et `l4 :s(y)`, par leurs images par μ, c'est-à-dire `n4 :a` et `n3 :s(n4 :a)`, respectivement. On note $\mu[R]$ le graphe qu'on obtient. Remarquons que la Figure 2.6 (b) correspond à la Figure 2.2 (b) (i.e., $\mu[R] = D$).
3. On remplace $G_{|\texttt{n2}}$ par $\mu[R]$ dans G (Figure 2.2 (c) et (d)) : Pour calculer ce remplacement, on redirige toutes les flèches pointant sur `n2` (ici, la première flèche partant de `n1`) vers la racine de $\mu[R]$, `r1`. On obtient le graphe H_2 (Figure 2.2 (c)) à partir de H_1 (Figure 2.2 (b)). Pour finir, il ne reste plus qu'à faire l'étape de *garbage collection*. Le graphe resultat, $G' = G[\texttt{n2} \leftarrow D]$, est celui de la Figure 2.2 (d).

Nous obtenons donc $G \rightarrow_{[\texttt{n2},\texttt{R5}]} G'$ (voir Figure 3.3). □

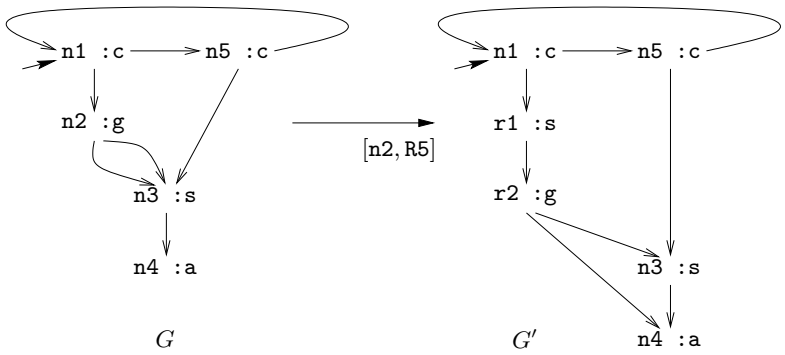


FIG. 3.3:

Définition 3.2.3 (Dérivation de réécriture)
Soient SP un cGRS. On note $\xrightarrow{*}$ la fermeture réflexive et transitive de $\rightarrow$ et on parle de *dérivation (de réécriture)* d'un graphe g_1 vers un graphe g_2 lorsque $g_1 \xrightarrow{*} g_2$. On note $\xrightarrow{+}$ la fermeture transitive de $\rightarrow$. Enfin, $\xrightarrow{\varepsilon}$[6] dénote la fermeture réflexive de $\rightarrow$ et on dit qu'un graphe g_1 *se réécrit en au plus un pas* en un graphe g_2 lorsque $g_1 \xrightarrow{\varepsilon} g_2$. ◇

Exemple 3.2.4 Dans l'Exemple 3.2.2, nous avons vu que $G \rightarrow_{[\texttt{n2},\texttt{R5}]} G'$ avec $G' =$ `n1 :c(r1 :s(r2 :g(n4 :a,n3 :s(n4))),n5 :c(n3,n1))`. Le membre gauche de la règle `R3` (cf. Exemple 3.1.12 et Figure 3.2) filtre G' au nœud `r2`. Donc G' se réécrit au nœud `r2` avec la règle `R3` en un graphe G'' avec $G'' =$ `n1 :c(r1 :s(r3 :s(n4 :a)),n5 :c(n3 :s(n4),n1))` (voir Figure 3.4). Par définition d'une dérivation de réécriture, nous concluons que $G \xrightarrow{*} G''$.

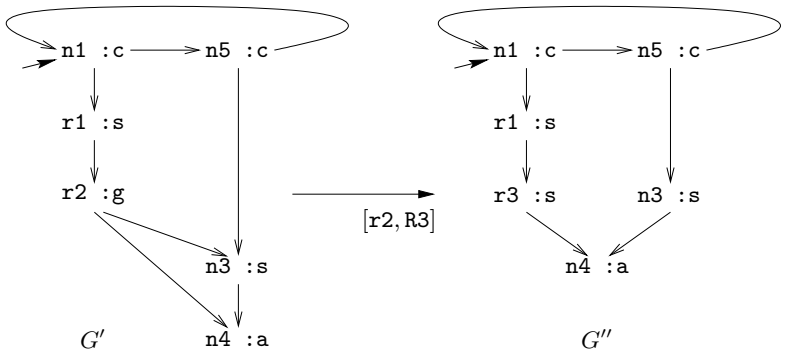


FIG. 3.4:

□

Dans les langages que nous considérons, on évalue une expression modélisée sous forme d'un graphe cyclique par réécriture. On considère qu'on a calculé la valeur d'une expression lorsque les deux conditions suivantes sont réunies :

1. Le graphe qui représente cette expression n'a plus de rédex. On dit qu'il est *sous forme normale*.
2. Le graphe obtenu est un graphe constructeur.

[6] Cette notation est introduite dans [Hue80]. Dans [Klo92], elle se note $\rightarrow^{\equiv}$.

Cette dernière condition est vérifiée dans la plupart des langages qui suivent une discipline constructeur. En effet, une forme normale qui contient des fonctions définies (par exemple `car(nil)`) provoque une erreur d'exécution. On dit qu'un graphe est *$\mathcal{C}$-normalisable*[7] s'il admet une forme normale constructeur.

Définition 3.2.5 (Forme normale, $\mathcal{C}$-normalisation)
Soient SP un cGRS. Un terme-graphe admissible est *sous forme normale* s'il n'a pas de rédex. Un terme-graphe admissible g est *$\mathcal{C}$-normalisable* s'il existe un graphe constructeur c et une dérivation de réécriture $g \xrightarrow{*} c$. ◇

Exemple 3.2.6 Le graphe G'' de l'Exemple 3.2.4 est un graphe constructeur. Donc G'' est sous forme normale. De plus, G'' est un graphe constructeur et $G \xrightarrow{*} G''$, donc G est $\mathcal{C}$-normalisable. □

Nous montrons ci-dessous la bonne définition d'un pas de réécriture, c'est-à-dire que la réécriture d'un graphe permet bien de calculer un graphe.

Proposition 3.2.7 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, g_1 un graphe, $l \to r$ une variante d'une règle de réécriture R de $\mathcal{R}$ et p un nœud de g_1. Si l filtre g_1 au nœud p, alors il existe un graphe g_2 tel que $g_1 \to_{[p,R]} g_2$. ◇

Preuve : Soit $h : l \to g_{1|p}$ le filtre de l sur g_1 au nœud p. Nous devons montrer que l'expression $g_1[p \leftarrow h[r]]$ désigne un graphe. Montrons tout d'abord qu'on peut calculer $h[r]$. Par hypothèse, l, r et g_1 sont compatibles deux à deux. De plus $\mathcal{B}(l,r) \subseteq \mathcal{N}_{(l \oplus r)}$ donc $\mathcal{B}(l,r) \cap \mathcal{N}_{(g_{1|p})} = \emptyset$. On conclut donc que h est compatible avec r, et par conséquent, $h[r]$ est bien défini. Montrons maintenant que g_1 et $h[r]$ sont compatibles. Supposons que $\mathcal{B}(l,r) = \{p_1, \ldots, p_k\}$. Par définition de l'application d'un homomorphisme à un graphe, $h[r] = r[p_1 \leftarrow (g_{1|p})_{|h(p_1)}] \ldots [p_k \leftarrow (g_{1|p})_{|h(p_k)}]$. Le graphe précédent est composé de nœuds de r et de sous-graphes de g_1. Ces sous-graphes ne sont pas modifiés par les redirections de pointeurs car $p_i \in \mathcal{N}_{(l \oplus r)}$, donc $p_i \notin \mathcal{N}_{g_1}$, pour tout $i \in 1..k$. Donc $h[r]$ est compatible avec g_1, ce qui nous permet de conclure que $g_1[p \leftarrow h[r]]$ peut être calculé, i.e., que g_2 peut être calculé. □

Pour finir cette section, nous montrons que l'ensemble des termes-graphes admissibles est stable par réécriture avec des règles admissibles.

Theorème 3.2.8 Soient SP un cGRS et $g_1 \to_{[p,R]} g_2$ un pas de réécriture. Si g_1 est un graphe *admissible*, alors g_2 est un graphe *admissible*. ◇

L'exemple suivant montre que cette proposition est fausse si les règles de réécriture utilisées ne sont pas admissibles.

Exemple 3.2.9 La règle `l1 :B(l2 :A,l3 :X)` → `r1 :B(l1,r2 :A)` n'est pas admissible puisque la racine de son membre gauche est un nœud de son membre droit, i.e., son membre gauche est un sous-graphe de son membre droit, et donc la troisième condition de la Définition 3.1.9 n'est pas remplie. En utilisant cette règle, le lecteur peut vérifier que le terme-graphe admissible `n1 :B(n2 :A,n3 :A)` se réécrit en `r1 :B(r1,r2 :A)`. Ce dernier graphe n'est pas admissible puisque l'opération définie `B` appartient à un cycle. □

Pour prouver le Théorème 3.2.8, nous établissons la proposition suivante qui donne une condition suffisante pour qu'un graphe obtenu par remplacement d'un sous-graphe dans un graphe admissible soit admissible.

[7] [SR93] parle de *terme légal* (*legal term* en anglais) dans le cadre des termes du premier ordre $\mathcal{C}$-normalisable.

Proposition 3.2.10 Soient g un graphe admissible, p un nœud de g et u un terme-graphe admissible tels que g et u soient compatibles. Si $p \notin \mathcal{N}_u$, alors $g[p \leftarrow u]$ est un graphe admissible. ◇

Preuve : Puisque p n'est pas un nœud de u, il n'existe pas de chemin allant de $\mathcal{R}oot_u$ vers p dans le graphe $g \oplus u$. Donc la redirection de pointeurs de p vers $\mathcal{R}oot_u$ utilisée pour calculer $g[p \leftarrow u]$ ne modifie pas u pour créer un cycle passant par un nœud fonctionnel dans $g \oplus u$. Donc $g[p \leftarrow u]$ est un graphe admissible. □

Preuve du Théorème 3.2.8 : Soit $h : l \to g_{1|p}$. Pour montrer que $g_1[p \leftarrow h[r]]$ est un graphe admissible, il suffit de montrer que $p \notin \mathcal{N}_{h[r]}$, d'après la Proposition 3.2.10. Supposons que $\mathcal{B}(l, r) = \{p_1, \ldots, p_k\}$. Alors $h[r] = r[p_1 \leftarrow (g_{1|p})_{|h(p_1)}] \ldots [p_k \leftarrow (g_{1|p})_{|h(p_k)}]$. Le graphe précédent est composé de nœuds de r et de sous-graphes de g_1 qui ne sont pas modifiés par les redirections de pointeurs (puisque $p_i \notin \mathcal{N}_{g_1}$ pour tout $i \in 1..k$). p n'est pas un nœud de r puisque $p \in \mathcal{N}_{g_1}$ et $\mathcal{N}_{g_1} \cap \mathcal{N}_r = \emptyset$. Donc si p apparaît dans $h[r]$, alors p est un nœud de $(g_{1|p})_{|h(p_i)}$ pour un i donné. Autrement dit, $h(p_i) \preceq p$ dans g_1. De plus, nous affirmons que $p \preceq h(p_i)$ dans g_1. En effet, comme $p_i \in \mathcal{B}(l, r)$, p_i est un nœud de l, donc $\mathcal{R}oot_l \preceq p_i$ (dans l). Comme h est un homomorphisme, $h(\mathcal{R}oot_l) \preceq h(p_i)$ dans g_1, et comme $h(\mathcal{R}oot_l) = p$, nous en déduisons que $p \preceq h(p_i)$ dans g_1. Nous avons donc $h(p_i) \preceq p$ et $p \preceq h(p_i)$ dans g_1. Or p est un nœud fonctionnel de g_1 et g_1 est un graphe admissible. Donc il n'y a pas de cycle sur p dans g_1. Par conséquent, $h(p_i) = p$. $h : l \to g_{1|p}$ est un homomorphisme, donc $h(\mathcal{R}oot_l) = p$ et pour tout $n \in \mathcal{N}_l$ tel que $n \neq \mathcal{R}oot_l$, $h(n) \neq p$ (sinon il existerait un cycle sur p dans g_1). Donc si $h(p_i) = p = h(\mathcal{R}oot_l)$, alors $p_i = \mathcal{R}oot_l$. Pourtant, $p_i \in \mathcal{B}(l, r)$ et $\mathcal{R}oot_l \notin \mathcal{N}_r$ par définition d'une règle admissible, donc $p_i \neq \mathcal{R}oot_l$. On en conclut que p ne peut pas être un nœud de $h[r]$. Par conséquent, $g_1[p \leftarrow h(r)]$ est un graphe admissible, d'après la Proposition 3.2.10. □

3.3 Confluence et confluence modulo la bisimilarité

La confluence est une des principales propriétés des relations de réécriture : elle exprime le déterminisme des calculs effectués, l'unicité du résultat d'un calcul lorsqu'il termine. Ce problème, abordé pour la première fois dans [CR36], a fait l'objet de nombreuses études (voir [Klo92]) dont deux étapes essentielles furent [New42] et [KB70]. Malheureusement, ces études ne s'étendent pas immédiatement aux graphes cycliques, comme nous l'avons remarqué en introduction de ce chapitre. Dans cette section, nous établissons que la réécriture des graphes admissibles est confluente.

Définition 3.3.1 (Confluence)
Soit SP un cGRS. On dit que *la relation de réécriture $\to$ est confluente (au renommage des nœuds près) par rapport aux termes-graphes admissibles* si pour tout terme-graphe *admissible* g_1, g_2, g_1' et g_2' tels que g_1 et g_2 soient égaux au renommage des nœuds près ($g_1 \sim g_2$), $g_1 \xrightarrow{*} g_1'$ et $g_2 \xrightarrow{*} g_2'$, il existe deux termes-graphes admissibles g_1'' et g_2'' tels que $g_1' \xrightarrow{*} g_1''$, $g_2' \xrightarrow{*} g_2''$ et $g_1'' \sim g_2''$ (cf. Figure 3.5). ◇

Remarque 3.3.2 Dans les figures, une ligne continue représente une hypothèse alors qu'une ligne en pointillés représente une conclusion. ◇

Dans [KKSV94], il est prouvé que la relation de réécriture engendrée par les systèmes de réécriture orthogonaux de graphes (donc en particulier les systèmes de réécriture admissibles) est confluente modulo l'équivalence de certains graphes dits hyper-effondrants[8] ; un graphe est hyper-effondrant s'il se réécrit en un pas en lui-même (i.e., $g \to g$).

[8] *hypercollapsing graphs* en anglais.

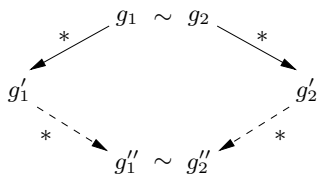


FIG. 3.5:

Exemple 3.3.3 Soient $g =$ `n1 :A(n2 :B(n1))` un terme-graphe et `S1` et `S2` les deux règles suivantes :

```
(S1) l1:A(l2:X) -> l2:X
(S2) l3:B(l4:Y) -> l4:Y
```

g se réécrit en deux termes-graphes distincts $g_1 =$ `n2 :B(n2)` (avec la règle `S1`) et $g_2 =$ `n1 :A(n1)` (avec la règle `S2`). g_1 (resp. g_2) se réécrit uniquement en lui-même avec la règle `S2` (resp. `S1`). Donc g_1 et g_2 sont des graphes hyper-effondrants. Il est clair que la relation de réécriture n'est pas confluente, mais elle est confluente modulo l'équivalence des graphes hyper-effondrants. On note que le graphe initial g n'est pas admissible. □

Nous ne nous intéressons pas à la confluence modulo l'équivalence des graphes hyper-effondrants dans cette thèse. Nous obtenons le résultat suivant :

Theorème 3.3.4 Soit SP un WAGRS. Alors la relation de réécriture $\rightarrow$ est confluente (modulo $\sim$) par rapport aux termes-graphes admissibles. ◇

Le Théorème 3.3.4 est prouvé dans la Section 3.4. Dans [AK96], tout système de réécriture de graphes sans règle superposable mais dont les membres gauches sont éventuellement non linéaires est confluent. Toutefois, ce résultat (surprenant) est établi pour une relation de réécriture particulière qui est différente de celle que nous considérons ici. Citons aussi [Plu94] qui propose une définition des pairs critiques dans le cadre des systèmes de réécriture de graphes acycliques, puis qui donne un critère permettant de décider la confluence de la relation de réécriture lorsqu'elle est nœthérienne (i.e., terminante).

Nous posons maintenant le problème de la confluence modulo la bisimilarité. Nous aurons besoin de ce résultat dans le Chapitre 6.

Définition 3.3.5 (Confluence modulo la bisimilarité)
Soit SP un cGRS. On dit que *la relation de réécriture $\rightarrow$ est confluente modulo la bisimilarité $\doteq$ par rapport aux termes-graphes admissibles* si pour tout terme-graphe admissible g_1, g_2, g'_1 et g'_2 tels que $g_1 \doteq g_2$, $g_1 \xrightarrow{*} g'_1$ et $g_2 \xrightarrow{*} g'_2$, il existe deux termes-graphes admissibles g''_1 et g''_2 tels que $g'_1 \xrightarrow{*} g''_1$, $g'_2 \xrightarrow{*} g''_2$ et $g''_1 \doteq g''_2$ (cf. Figure 3.6). ◇

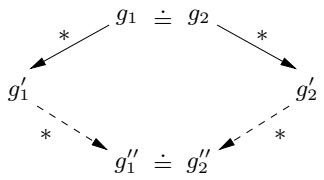


FIG. 3.6:

Notons que si les membres gauches des règles ne sont pas des arbres, alors la confluence de la réécriture modulo la bisimilarité n'est pas assurée :

Exemple 3.3.6 Considérons les termes-graphes admissibles $g_1 =$ `n1 :B(n2 :A,n2)` et $g_2 =$ `m1 :B(m2 :A,m3 :A)` et la règle `l1 :B(l2 :X,l2)` $\rightarrow$ `l2 :X`. g_1 se réécrit à la racine en $g'_1 =$ `n2 :A` alors que le graphe g_2 ne peut pas se réécrire. Comme g'_1 et g_2 ne sont pas bisimilaires, $\rightarrow$ n'est pas confluente modulo $\doteq$ par rapport aux termes-graphes admissibles. □

Nous obtenons le résultat suivant :

Theorème 3.3.7 Soit SP un WAGRS. La relation de réécriture $\rightarrow$ est confluente modulo la bisimilarité par rapport aux termes-graphes admissibles. ◇

Le Théorème 3.3.7 est prouvé dans la Section 3.5. Le lecteur trouvera dans [AKP97] un catalogue des propriétés de confluence et de confluence modulo la bisimilarité sur les termes-graphes acycliques.

Une autre question intéressante concerne la confluence de la relation de réécriture engendrée par des cGRS tels que les membres droits instanciés de deux règles superposables soient bisimilaires (et donc pas nécessairement égaux au renommage des nœuds près). Nous aborderons ce problème dans la Section 3.6.

3.4 Preuve de confluence des WAGRS

Dans cette section, nous prouvons le Théorème 3.3.4 :

Théorème 3.3.4 Soit SP un WAGRS. La relation de réécriture $\rightarrow$ est confluente au renommage des nœuds près par rapport aux termes-graphes admissibles, i.e., pour tout terme-graphe *admissible* g_1, g_2, g'_1 et g'_2 tels que $g_1 \sim g_2$, $g_1 \xrightarrow{*} g'_1$ et $g_2 \xrightarrow{*} g'_2$, il existe deux termes-graphes admissibles g''_1 et g''_2 tels que $g'_1 \xrightarrow{*} g''_1$, $g'_2 \xrightarrow{*} g''_2$ et $g''_1 \sim g''_2$ (cf. Figure 3.5). □

Nous commençons par faire une liste de propriétés techniques concernant les sous-graphes et le remplacement. Ces propriétés s'inspirent de celles utilisées dans [Hue80] pour la preuve de confluence de la réécriture de termes. Toutefois, leurs utilisations dans notre cadre d'étude requiert des précautions : nous avons vu que le résultat d'un remplacement pouvait parfois être surprenant (cf. Exemple 2.2.9).

Propriétés de persistence

Nous commençons par établir deux propositions qui simplifient le calcul d'un sous-graphe dans un graphe obtenu par remplacement.

Proposition 3.4.1 (Persistence globale)
Soient g un graphe, u un terme-graphe compatible avec g et n_1 et n_2 deux nœuds de g. Si il n'existe pas de chemin de n_2 vers n_1 dans g et qu'il existe un chemin d'une racine de g vers n_2 qui ne passe pas par n_1, alors $(g[n_1 \leftarrow u])_{|n_2} = g_{|n_2}$. Cette proposition s'applique dans le cas particulier où $n_1 \| n_2$. ◇

Preuve : Par hypothèse, il existe un chemin d'une racine r de g vers n_2 qui ne passe pas par n_1. Donc n_2 reste un nœud de $g[n_1 \leftarrow u]$ (i.e., le chemin allant de r vers n_2 n'est pas modifié par la redirection de pointeurs de n_1 vers $\mathcal{R}\mathrm{oot}_u$). De plus, il n'existe pas de chemin de n_2 vers n_1 dans g, i.e., $n_1 \notin \mathcal{N}_{(g_{|n_2})}$.

Donc $g_{|n_2}$ est un sous-graphe de $g[n_1 \leftarrow u]$ (i.e., $g_{|n_2}$ n'est pas modifié par la redirection de pointeurs de n_1 vers $\mathcal{R}\text{oot}_u$). On en conclut que $(g[n_1 \leftarrow u])_{|n_2} = g_{|n_2}$. □

Proposition 3.4.2 (Persistence locale [9])
Soient g un graphe, u un terme-graphe compatible avec g et n_1 et n_2 deux nœuds de g. Si $n_1 \notin \mathcal{N}_u$ et $n_2 \in \mathcal{N}_u$, alors $(g[n_1 \leftarrow u])_{|n_2} = u_{|n_2}$. ◇

Preuve : Par hypothèse, $n_1 \notin \mathcal{N}_u$, donc u n'est pas modifié par la redirection de pointeurs de n_1 vers $\mathcal{R}\text{oot}_u$. De plus, n_2 reste un nœud de u dans $g[n_1 \leftarrow u]$ et par conséquent, $(g[n_1 \leftarrow u])_{|n_2} = u_{|n_2}$. □

Propriétés concernant les doubles remplacements

Nous travaillons maintenant sur des propositions qui mettent en jeu des doubles remplacements. Il apparaît qu'une expression telle que $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$ n'est pas toujours bien définie. En effet, supposons que g soit un graphe compatible avec deux termes-graphes u_1 et u_2. Comme g est compatible avec u_1, on peut calculer $g[n_1 \leftarrow u_1]$. Mais ce graphe peut ne plus être compatible avec u_2.

Exemple 3.4.3 Soient g, u_1 et u_2 trois termes-graphes compatibles tels que $g =$ `n1 :A(n2 :B(n3 :C))`, $u_1 =$ `n4 :D` et $u_2 =$ `n5 :E(n2 :B(n3 :C))`. Alors $g[$`n3` $\leftarrow u_1] =$ `n1 :A(n2 :B(n4 :D))`. Ce graphe n'est plus compatible avec u_2 puisque le nœud `n2` n'a pas les mêmes successeurs dans $g[$`n3` $\leftarrow u_1]$ et dans u_2. Le problème vient du fait que `n3` apparaît à la fois dans g et dans u_2. □

Dans la proposition suivante, nous donnons une condition suffisante pour éviter ce problème.

Proposition 3.4.4 Soient g un graphe, n_1 et n_2 deux nœuds de g et u_1 et u_2 deux termes-graphes tels que g, u_1 et u_2 soient compatibles. Si $n_1 \notin \mathcal{N}_{u_2}$, alors $g[n_1 \leftarrow u_1]$ et u_2 sont compatibles, et donc $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$ est bien défini. ◇

Preuve : Supposons que $g[n_1 \leftarrow u_1]$ et u_2 ne soient pas compatibles alors que g, u_1 et u_2 le sont. Par définition de la compatibilité, il existe forcément un nœud p commun à $g[n_1 \leftarrow u_1]$ et à u_2 tel que $\mathcal{S}_{g \oplus u_1}(p) = \mathcal{S}_{u_2}(p)$ mais $\mathcal{S}_{g[n_1 \leftarrow u_1]}(p) \neq \mathcal{S}_{u_2}(p)$. Supposons que $\mathcal{S}_{g \oplus u_1}(p) = p_1 \dots p_k = \mathcal{S}_{u_2}(p)$. Soit ρ la redirection de pointeurs telle que $\rho(n_1) = \mathcal{R}\text{oot}_{u_1}$ et $\rho(q) = q$ pour tout $q \neq n_1$. Par définition d'un remplacement, $\mathcal{S}_{g[n_1 \leftarrow u_1]}(p) = \rho(p_1) \dots \rho(p_k)$. Puisque $\mathcal{S}_{u_2}(p) = p_1 \dots p_k$ et $\mathcal{S}_{g[n_1 \leftarrow u_1]}(p) \neq \mathcal{S}_{u_2}(p)$, il existe $i \in 1..k$ tel que $\rho(p_i) \neq p_i$. Aussi il existe $i \in 1..k$ tel que $p_i = n_1$. Puisque $\mathcal{S}_{u_2}(p) = p_1 \dots p_k$, on en conclut que $n_1 \in \mathcal{N}_{u_2}$. Donc si n_1 n'est pas un nœud de u_2, alors $g[n_1 \leftarrow u_1]$ et u_2 sont compatibles. □

Proposition 3.4.5 (Associativité)
Soient g un graphe, n_1 et n_2 deux nœuds de g et u_1 et u_2 deux termes-graphes tels que g, u_1 et u_2 soient compatibles. Si $n_1 \notin \mathcal{N}_{u_2}$, alors $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])]$. ◇

Preuve : Soit $E_1 = (g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$. Comme $n_1 \notin \mathcal{N}_{u_2}$, l'expression E_1 est bien définie (d'après la Proposition 3.4.4). Soient $\rho_1 = \{n_1 \mapsto \mathcal{R}\text{oot}_{u_1}\}$ et $\rho_2 = \{n_2 \mapsto \mathcal{R}\text{oot}_{u_2}\}$ deux redirections de pointeurs. On a : $E_1 = (\rho_2(\rho_1(g \oplus u_1 \oplus u_2)))_{|\rho_2(\rho_1(\mathcal{R}\text{oots}_g))}$.

[9] Le mot *Embedding* est utilisé dans [Hue80].

Soit $E_2 = (g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])]$. L'expression E_2 est, elle aussi, bien définie car si g et u_1 sont compatibles, alors $g[n_2 \leftarrow u_2]$ et $u_1[n_2 \leftarrow u_2]$ restent compatibles. Soit $\rho_3 = \{n_2 \mapsto \rho_2(\mathcal{R}\mathrm{oot}_{u_1})\}$ une nouvelle redirection de pointeurs. On a : $E_2 = (\rho_3(\rho_2(g \oplus u_1 \oplus u_2)))_{|\rho_3(\rho_2(\mathcal{R}\mathrm{oots}_g))}$.
Pour montrer que $E_1 = E_2$, il suffit donc de montrer que $\rho_2(\rho_1(p)) = \rho_3(\rho_2(p))$ pour tout $p \in \mathcal{N}_{g \oplus u_1 \oplus u_2}$, ce qui est trivial avec une analyse des différents cas pour p. □

Proposition 3.4.6 (Commutativité)
Soient g un graphe, n_1 et n_2 deux nœuds de g et u_1 et u_2 deux termes-graphes tels que g, u_1 et u_2 soient compatibles. Si $n_1 \notin \mathcal{N}_{u_2}$ et $n_2 \notin \mathcal{N}_{u_1}$, alors $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow u_1]$. ◇

Preuve : D'après la Proposition 3.4.5 (Associativité), $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])]$, puisque $n_1 \notin \mathcal{N}_{u_2}$. Or $n_2 \notin \mathcal{N}_{u_1}$, donc $u_1[n_2 \leftarrow u_2] = u_1$. Aussi, nous concluons que $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow u_1]$. □

Proposition 3.4.7 (Dominance faible)
Soient g un graphe, n_1 et n_2 deux nœuds de g et u_1 et u_2 deux termes-graphes tels que g, u_1 et u_2 soient compatibles. Si $n_1 \notin \mathcal{N}_{u_2}$ et tous les chemins allant d'une racine de g vers n_1 passent par n_2, i.e., $\forall r \in \mathcal{R}\mathrm{oots}_g, \forall C \in \mathcal{P}\mathrm{aths}_g(r, n_1), n_2 \in C$, alors $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = g[n_2 \leftarrow u_2]$. ◇

Preuve : D'après la Proposition 3.4.5 (Associativité), $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])]$, puisque $n_1 \notin \mathcal{N}_{u_2}$. Or n_1 n'est pas un nœud de $g[n_2 \leftarrow u_2]$. En effet, n_1 n'est pas un nœud de u_2 et tous les chemins allant d'une racine de g vers n_1 passent par n_2, donc n_1 a été éliminé par le remplacement dans $g[n_2 \leftarrow u_2]$. En conséquence, $(g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])] = g[n_2 \leftarrow u_2]$, et donc $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = g[n_2 \leftarrow u_2]$. □

Preuve de confluence proprement dite

La proposition suivante établit que la réécriture de deux graphes égaux au renommage des nœuds près conduit à deux graphes égaux au renommage des nœuds près. La preuve est évidente.

Proposition 3.4.8 Soient SP un cGRS et g_1, g_2 et g_1' trois termes-graphes admissibles tels que $g_1 \sim g_2$ et $g_1 \to g_1'$ (resp. $g_1 \xrightarrow{*} g_1'$). Alors il existe un terme-graphe admissible g_2' tel que $g_2 \to g_2'$ (resp. $g_2 \xrightarrow{*} g_2'$) et $g_1' \sim g_2'$ (cf. Figure 3.7). ◇

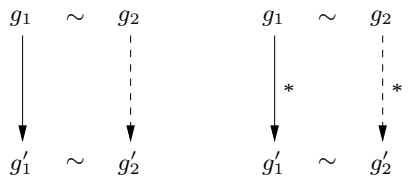


FIG. 3.7:

Le lemme ci-dessous est un résultat clé pour montrer le Théorème 3.3.4. Il établit que si un terme-graphe admissible se réécrit de deux manières différentes, alors les deux graphes qu'on obtient peuvent se réécrire en un même troisième (au renommage des nœuds près). Cette propriété est presque la même que celle de sous-commutativité [Klo92].

Lemme 3.4.9 Soient SP un WAGRS et g, g_1 et g_2 trois termes-graphes admissibles tels que $g \to g_1$ et $g \to g_2$. Alors il existe deux termes-graphes admissibles g'_1 et g'_2 tels que $g_1 \xrightarrow{\varepsilon} g'_1$, $g_2 \xrightarrow{\varepsilon} g'_2$ et $g'_1 \sim g'_2$ (cf. Figure 3.8). ◇

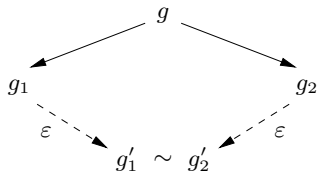


FIG. 3.8:

Preuve : Soient g, g_1 et g_2 trois graphes admissibles tels que $g \to_{[n_1, l_1 \to r_1]} g_1$ et $g \to_{[n_2, l_2 \to r_2]} g_2$. Par définition d'un pas de réécriture, il existe $h_1 : l_1 \to g_{|n_1}$ et $h_2 : l_2 \to g_{|n_2}$ tels que $g_1 = g[n_1 \leftarrow h_1[r_1]]$ et $g_2 = g[n_2 \leftarrow h_2[r_2]]$. Si $n_1 = n_2$, alors nécessairement $h_1[r_1] \sim h_2[r_2]$ par définition d'un WAGRS et donc le lemme est vrai (i.e., $g'_1 = g_1$ et $g'_2 = g_2$). A partir de maintenant, nous supposons que $n_1 \neq n_2$. Nous considérons deux cas dépendants des positions de n_1 et n_2 dans g : soit n_1 et n_2 sont disjoints (i.e., $n_1 \| n_2$), soit il existe un chemin allant de n_1 vers n_2 (i.e., $n_1 \prec n_2$). Le cas $n_2 \prec n_1$ est symétrique.

Cas 1 : n_1 et n_2 sont disjoints ($n_1 \| n_2$) :
D'après la définition de g_1, $g_{1|n_2} = (g[n_1 \leftarrow h_1[r_1]])_{|n_2}$. Puisque $n_1 \| n_2$, nous déduisons de la Proposition 3.4.1 (Persistence globale) que $(g[n_1 \leftarrow h_1[r_1]])_{|n_2} = g_{|n_2}$, i.e., $g_{1|n_2} = g_{|n_2}$. Par hypothèse, h_2 est un homomorphisme de l_2 vers $g_{|n_2}$, et comme $g_{|n_2} = g_{1|n_2}$, h_2 est aussi un homomorphisme de l_2 vers $g_{1|n_2}$, i.e., l_2 filtre g_1 au nœud n_2 avec h_2. Donc g_1 se réécrit avec la règle $l_2 \to r_2$ en $g'_1 = g_1[n_2 \leftarrow h_2[r_2]]$, c'est-à-dire $g'_1 = (g[n_1 \leftarrow h_1[r_1]])[n_2 \leftarrow h_2[r_2]]$. Pour les mêmes raisons, g_2 se réécrit en $g'_2 = (g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow h_1[r_1]]$ avec la règle $l_1 \to r_1$. Nous montrons maintenant que $g'_1 = g'_2$. Pour cela, nous vérifions que n_1 n'est pas un nœud de $h_2[r_2]$ et que n_2 n'est pas un nœud de $h_1[r_1]$, ce qui nous permet d'utiliser la Proposition 3.4.6 (Commutativité). Le graphe $h_2[r_2]$ est composé de nœuds de r_2 et de nœuds qui sont introduits par h_2, c'est-à-dire des nœuds de $g_{|n_2}$ (puisque h_2 est un homomorphisme de l_2 vers $g_{|n_2}$). Par hypothèse, $\mathcal{N}_g \cap \mathcal{N}_{r_2} = \emptyset$, donc n_1 n'est pas un nœud de r_2. De plus, n_1 et n_2 sont disjoints, donc $n_2 \not\prec n_1$ dans g, i.e., n_1 n'est pas un nœud de $g_{|n_2}$. Nous en déduisons que n_1 n'est pas un nœud de $h_2[r_2]$. Avec un raisonnement similaire, on montre que n_2 n'est pas un nœud de $h_1[r_1]$. Et donc, d'après la Proposition 3.4.6, $g'_1 = g'_2$.

Cas 2 : Il existe un chemin de n_1 vers n_2 ($n_1 \prec n_2$) (cf. Figure 3.9) :
Comme $n_1 \prec n_2$, n_2 est un nœud de $g_{|n_1}$. Par hypothèse, $h_1(l_1) = g_{|n_1}$, donc n_2 est un nœud de $h_1(l_1)$. l_1 est un motif donc tous ses nœuds sont contructeurs ou variables sauf sa racine qui est fonctionnelle. Comme n_2 est étiqueté par une fonction définie et que $h_1(\mathcal{R}\text{oot}_{l_1}) = n_1 \neq n_2$, n_2 est introduit dans le graphe $h_1(l_1)$ par h_1 grâce à une variable de l_1. Autrement dit, il existe au moins un nœud variable p_1 de l_1 tel que n_2 soit un nœud de $g_{|h_1(p_1)}$. Il peut exister d'autres nœuds variables comme p_1 dans l_1. Aussi, nous définissons l'ensemble $P = \{p_1, p_2, \ldots, p_k\}$ des nœuds variables p_i de l_1 tels que n_2 soit un nœud de $g_{|h_1(p_i)}$. Il y a deux sous-cas à étudier, selon que $P \cap \mathcal{N}_{r_1} = \emptyset$ ou que $P \cap \mathcal{N}_{r_1} \neq \emptyset$, c'est-à-dire selon qu'il existe au moins un nœud variable de P apparaissant dans le membre droit de la règle $l_1 \to r_1$ ou pas.

Cas 2.1 : Il existe un nœud $p_k \in P$ tel que $p_k \in \mathcal{N}_{r_1}$ ($P \cap \mathcal{N}_{r_1} \neq \emptyset$) :
Ce sous-cas est représenté dans la Figure 3.9. Par définition de P, n_2 est un nœud de $g_{|h_1(p_k)}$. Par hypothèse, p_k est un nœud variable de r_1, donc n_2 est un nœud du graphe $h_1[r_1]$. Par conséquent, n_2 est un nœud de $g_1 = g[n_1 \leftarrow h_1[r_1]]$.

Nous prétendons que $g_{1|n_2} = g_{|n_2}$. En effet, $n_1 \notin \mathcal{N}_{h_1[r_1]}$ et $n_2 \in \mathcal{N}_{h_1[r_1]}$, donc $g_{1|n_2} = (h_1[r_1])_{|n_2}$ d'après la Proposition 3.4.2 (Persistence locale). Le graphe $h_1[r_1]$ est composé de nœuds de r_1 et de nœuds qui sont introduits par h_1, c'est-à-dire des nœuds de $g_{|n_1}$ (puisque h_1 est un homomorphisme de l_1 vers $g_{|n_1}$). Par hypothèse, $\mathcal{N}_g \cap \mathcal{N}_{r_1} = \emptyset$, donc n_2 n'est pas un nœud de r_1. Par conséquent, $(h_1[r_1])_{|n_2}$ est un sous-graphe de $g_{|n_1}$, i.e., $(h_1[r_1])_{|n_2} = g_{|n_2}$. Nous en concluons que $g_{1|n_2} = g_{|n_2}$.

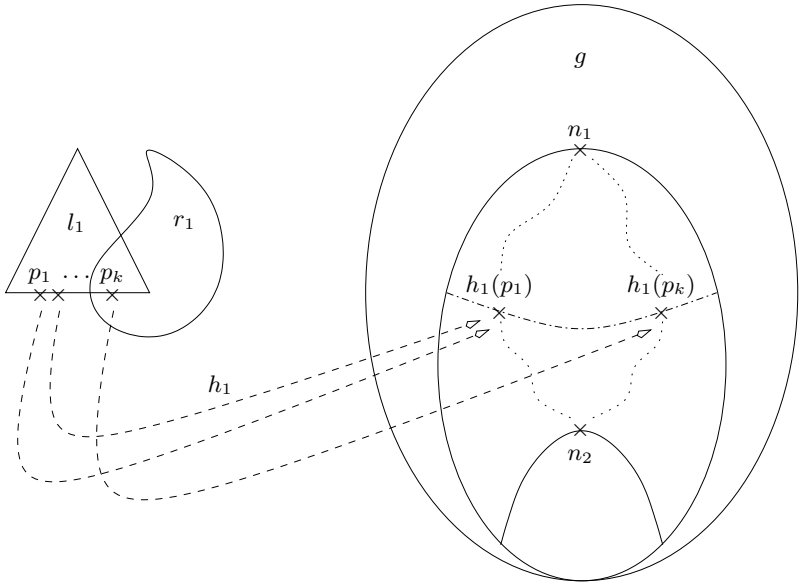


FIG. 3.9:

Par hypothèse, h_2 est un homomorphisme de l_2 vers $g_{|n_2}$ et nous avons montré que $g_{|n_2} = g_{1|n_2}$, donc h_2 est aussi un homomorphisme de l_2 vers $g_{1|n_2}$, i.e., l_2 filtre g_1 au nœud n_2 avec h_2. Aussi, g_1 se réécrit avec la règle $l_2 \to r_2$ en $g'_1 = g_1[n_2 \leftarrow h_2[r_2]]$, c'est-à-dire $g'_1 = (g[n_1 \leftarrow h_1[r_1]])[n_2 \leftarrow h_2[r_2]]$.

Nous venons de définir g'_1 et nous définissons maintenant g'_2. Par hypothèse, $n_1 \prec n_2$ et $n_2 \not\prec n_1$ (car g est admissible). Donc n_1 reste un nœud de $g_2 = g[n_2 \leftarrow h_2[r_2]]$ et $g_{2|n_1} = (g_{|n_1})[n_2 \leftarrow h_2[r_2]]$. Nous affirmons qu'il existe un filtre $h'_1 : l_1 \to g_{2|n_1}$. En effet, il existe un filtre $h_1 : l_1 \to g_{|n_1}$. De plus, l_1 est un motif et n_2 est un nœud fonctionnel de $g_{|n_1}$, donc l_1 continue de filtrer $(g_{|n_1})[n_2 \leftarrow h_2[r_2]]$. Comme $(g_{|n_1})[n_2 \leftarrow h_2[r_2]] = g_{2|n_1}$, nous déduisons l'existence du filtre $h'_1 : l_1 \to g_{2|n_1}$. Donc g_2 se réécrit avec la règle $l_1 \to r_1$ en $g'_2 = g_2[n_1 \leftarrow h'_1[r_1]]$, c'est-à-dire $g'_2 = (g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow h'_1[r_1]]$. Il est clair que $h'_1[r_1] = h_1[r_1][n_2 \leftarrow h_2[r_2]]$. Avec l'égalité précédente, on conclut que $g'_2 = (g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow (h_1[r_1][n_2 \leftarrow h_2[r_2]])]$.

Pour finir, g_1 se réécrit en $g'_1 = (g[n_1 \leftarrow h_1[r_1]])[n_2 \leftarrow h_2[r_2]]$ et g_2 se réécrit en $g'_2 = (g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow (h_1[r_1][n_2 \leftarrow h_2[r_2]])]$. Comme n_1 n'est pas un nœud de $h_2[r_2]$, on obtient $g'_1 = g'_2$, avec la Proposition 3.4.5 (Associativité).

Cas 2.2 : Il n'existe pas de nœud variable de P qui apparaisse dans r_1 ($P \cap \mathcal{N}_{r_1} = \emptyset$) :
Dans ce cas, n_2 n'est pas un nœud de $h_1[r_1]$. Mais n_2 peut quand même être un nœud de $g[n_1 \leftarrow h_1[r_1]]$ si il existe un chemin allant d'une racine de g vers n_2 qui ne passe pas par n_1. Il y a donc deux nouveaux sous-cas.

Cas 2.2.1 : Tous les chemins allant d'une racine de g vers n_2 passe par n_1 :
Alors n_2 n'est plus un nœud de $g_1 = g[n_1 \leftarrow h_1[r_1]]$: il est effacé par le remplacement. Concernant g_2, par hypothèse $n_1 \prec n_2$ et $n_2 \not\prec n_1$, donc n_1 est un nœud de $g_2 = g[n_2 \leftarrow h_2[r_2]]$. De plus, il existe un homomorphisme $h'_1 : l_1 \to g_{2|n_1}$ (pour la même raison que dans le Cas 2.1). Donc l_1 filtre g_2 au nœud n_1 et g_2 se réécrit en $g'_2 = g_2[n_1 \leftarrow h'_1[r_1]]$. Par hypothèse, n_2 n'est pas un nœud de $h_1[r_1]$, donc $h'_1[r_1] = h_1[r_1]$. Ainsi g_2 peut se réécrire en $g'_2 = (g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow h_1[r_1]]$. Comme n_2 n'est pas un nœud de $h_1[r_1]$ et comme tous les chemins allant d'une racine de g vers n_2 passe par n_1, d'après la Proposition 3.4.7 (Dominance faible), nous obtenons $(g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow h_1[r_1]] = g[n_1 \leftarrow h_1[r_1]]$, c'est-à-dire $g_2 \to_{[n_1, l_1 \to r_1]} g_1$. Pour finir, soient $g'_1 = g'_2 = g_1$ et le lemme est vérifié.

Cas 2.2.2 : Il existe un chemin C allant d'une racine de g vers n_2 tel que $n_1 \notin C$:
Dans ce cas, le nœud n_2 n'est pas un nœud de $h_1[r_1]$ mais n_2 reste un nœud de $g_1 = g[n_1 \leftarrow h_1[r_1]]$ car le chemin C n'est pas modifié par le remplacement au nœud n_1. Il n'existe pas de chemin

de n_2 vers n_1 et il existe un chemin de la racine de g qui ne passe pas par n_1, donc d'après la Proposition 3.4.1 (Persistence globale), $(g[n_1 \leftarrow h_1[r_1]])_{|n_2} = g_{|n_2}$, i.e., $g_{1|n_2} = g_{|n_2}$. Par hypothèse, l_2 filtre $g_{|n_2}$, donc l_2 filtre $g_{1|n_2}$. Par conséquent, g_1 peut se réécrire en $g'_1 = g_1[n_2 \leftarrow h_2[r_2]]$, c'est-à-dire $g'_1 = (g[n_1 \leftarrow h_1[r_1]])[n_2 \leftarrow h_2[r_2]]$. Concernant g_2, par hypothèse, $n_1 \prec n_2$ et $n_2 \not\prec n_1$, donc n_1 est un nœud de $g_2 = g[n_2 \leftarrow h_2[r_2]]$. De plus, il existe un homomorphisme $h'_1 : l_1 \to g_{2|n_1}$ (pour la même raison que dans le Cas 2.1). Aussi, l_1 filtre g_2 au nœud n_1 et g_2 peut se réécrire en $g'_2 = g_2[n_1 \leftarrow h'_1[r_1]]$. Par hypothèse, n_2 n'est pas un nœud de $h_1[r_1]$, donc $h'_1[r_1] = h_1[r_1]$. Par conséquent, g_2 se réécrit en $g'_2 = (g[n_2 \leftarrow h_2[r_2]])[n_1 \leftarrow h_1[r_1]]$ avec la règle $l_1 \to r_1$. Pour finir, n_1 n'est pas un nœud de $h_2[r_2]$ (puisque $n_1 \prec n_2$) et n_2 n'est pas un nœud de $h_1[r_1]$ (par hypothèse), donc d'après la Proposition 3.4.6 (Commutativité), $g'_1 = g'_2$. □

Du Lemme 3.4.9 et de la Proposition 3.4.8, on déduit la proposition suivante. La preuve se fait facilement par induction.

Proposition 3.4.10 Soient SP un WAGRS et g, g_1 et g_2 trois termes-graphes admissibles tels que $g \xrightarrow{*} g_1$ et $g \xrightarrow{\varepsilon} g_2$ (resp. $g \xrightarrow{*} g_2$). Alors il existe deux termes-graphes admissibles g'_1 et g'_2 tels que $g_1 \xrightarrow{\varepsilon} g'_1$ (resp. $g_1 \xrightarrow{*} g'_1$), $g_2 \xrightarrow{*} g'_2$ et $g'_1 \sim g'_2$ (cf. Figure 3.10). ◇

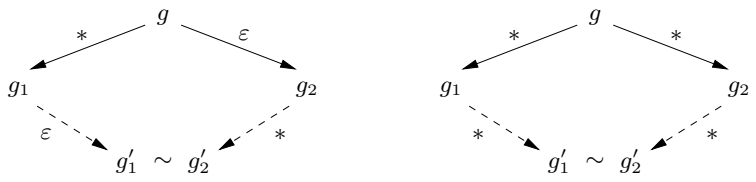


FIG. 3.10:

Nous sommes prêts pour démontrer la confluence des WAGRS par rapport à l'ensemble des termes-graphes admissibles :

Preuve du Théorème 3.3.4 : Soient SP un WAGRS et g_1, g_2, g'_1 et g'_2 quatre termes-graphes admissibles tels que $g_1 \sim g_2$, $g_1 \xrightarrow{*} g'_1$ et $g_2 \xrightarrow{*} g'_2$. Comme $g_1 \xrightarrow{*} g'_1$ et $g_1 \sim g_2$, nous déduisons de la Proposition 3.4.8 qu'il existe un terme-graphe admissible d tel que $g_2 \xrightarrow{*} d$ et $g'_1 \sim d$. Comme $g_2 \xrightarrow{*} d$ et $g_2 \xrightarrow{*} g'_2$, il existe deux termes-graphes admissibles d' et g''_2 tels que $d \xrightarrow{*} d'$, $g'_2 \xrightarrow{*} g''_2$ et $d' \sim g''_2$, d'après la Proposition 3.4.10. Puisque $g'_1 \sim d$ et $d \xrightarrow{*} d'$, nous inférons avec la Proposition 3.4.8 qu'il existe un terme-graphe admissible g''_1 tel que $g'_1 \xrightarrow{*} g''_1$ et $g''_1 \sim d'$. Pour finir, comme $g''_1 \sim d'$ et $d' \sim g''_2$, nous concluons que $g''_1 \sim g''_2$. □

3.5 Preuve de confluence modulo la bisimilarité des WAGRS

Dans cette section, nous prouvons le Théorème 3.3.7 :

Théorème 3.3.7 Soit SP un WAGRS. La relation de réécriture $\to$ est confluente modulo la bisimilarité $\doteq$ par rapport aux termes-graphes admissibles, i.e., pour tout terme-graphe *admissible* g_1, g_2, g'_1 et g'_2 tels que g_1 et g_2 soient bisimilaires ($g_1 \doteq g_2$), $g_1 \xrightarrow{*} g'_1$ et $g_2 \xrightarrow{*} g'_2$, il existe deux termes-graphes admissibles g''_1 et g''_2 tels que $g'_1 \xrightarrow{*} g''_1$, $g'_2 \xrightarrow{*} g''_2$ et $g''_1 \doteq g''_2$ (cf. Figure 3.6). □

La proposition suivante établi que si deux graphes sont bisimilaires et le membre gauche d'une règle admissible filtre l'un d'entre eux, alors il filtre aussi l'autre.

Proposition 3.5.1 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, g_1 et g_2 deux termes-graphes admissibles et $l \to r$ une règle de $\mathcal{R}$.

1. Si il existe un homomorphisme $h : g_1 \to g_2$ et que l filtre g_1 à la racine, alors l filtre g_2 à la racine.
2. Si il existe un $\mathcal{V}$-*homomorphisme* $h : g_1 \to g_2$ et que l filtre g_2 à la racine, alors l filtre g_1 à la racine.
3. Si g_1 et g_2 sont bisimilaires et que l filtre g_1 à la racine, alors l filtre g_2 à la racine.

◇

La proposition ci-dessus est valide parce que les membres gauches des règles d'un cGRS sont des motifs, donc des arbres finis et linéaires. Cette proposition est fausse en général, comme le montre l'exemple suivant :

Exemple 3.5.2 Considérons la règle `l1 :A(l2 :B,l2)` $\to$ `l2`. Son membre gauche, l, ne filtre pas le terme-graphe admissible $g_1 =$ `n1 :A(n2 :B,n3 :B)` puisqu'il n'existe pas d'homomorphisme de l vers g_1. Aussi g_1 ne peut pas être réécrit. Pourtant l et g_1 sont bisimilaires et l se réécrit en `l2 :B`. Considérons maintenant la règle `l1 :A(l2 :X,l3 :B(l3))` $\to$ `l3`. Son membre gauche filtre le graphe $g_3 =$ `n1 :A(n2 :C,n3 :B(n3))` mais il ne filtre pas le graphe $g_4 =$ `n1 :A(n2 :C,n3 :B(n4 :B(n3)))` alors que g_3 et g_4 sont bisimilaires. □

Preuve de la Proposition 3.5.1 : La première partie de cette proposition est évidente : s'il existe un homomorphisme $h_1 : l \to g_1$ et un homomorphisme $h : g_1 \to g_2$, alors $h \circ h_1$ est un homomorphisme de l vers g_2, i.e., l filtre g_2 à la racine.

Nous montrons maintenant la seconde partie de la proposition. Par définition d'un cGRS, l est un motif, donc un arbre fini et linéaire. Dans la suite, nous utilisons cette propriété en faisant une preuve par induction sur la hauteur de l, c'est-à-dire la longueur maximale des chemins allant de la racine de l vers les feuilles de l.

Cas de base : Soient l un arbre linéaire de hauteur $\eta = 0$, g_1 et g_2 deux termes-graphes admissibles et $h : g_1 \to g_2$ un $\mathcal{V}$-homomorphisme tel que l filtre g_2 à la racine. Comme $\eta = 0$, l est soit une variable, soit une constante. Si l est une variable, il est clair que l filtre g_2. Si l est une constante, alors g_2 est exactement la même constante puisque $h_2 : l \to g_2$ est un homomorphisme. Et comme $h : g_1 \to g_2$ est un $\mathcal{V}$-homomorphisme, g_1 est aussi la même constante. Donc l filtre g_1 à la racine.

Cas d'induction : Soit $\eta > 0$. On suppose que pour tout arbre linéaire de hauteur inférieur ou égale à η, pour tout terme-graphe admissible g_1 et g_2 et pour tout $\mathcal{V}$-homomorphisme $h : g_1 \to g_2$, si l filtre g_2 à la racine, alors l filtre g_1 à la racine. Soient l un arbre linéaire de hauteur $\eta + 1$, g_1 et g_2 deux termes-graphes admissibles et $h : g_1 \to g_2$ un $\mathcal{V}$-homomorphisme tels qu'il existe un homomorphisme $h_2 : l \to g_2$. Nous construisons un homomorphisme $h_1 : l \to g_1$.

l est ni une variable ni une constante, puisque c'est un arbre de hauteur $\eta + 1$. Donc sa racine est étiquetée par une fonction d'arité non vide f (un constructeur ou une opération définie). Comme $h_2 : l \to g_2$ est un homomorphisme, la racine de g_2 est aussi étiquetée par f, et comme $h : g_1 \to g_2$ est un $\mathcal{V}$-homomorphisme, la racine de g_1 est elle aussi étiquetée par f. Supposons maintenant que $\mathcal{S}_l(\mathcal{R}\text{oot}_l) = n_1 \dots n_k$, $\mathcal{S}_{g_1}(\mathcal{R}\text{oot}_{g_1}) = u_1 \dots u_k$ et $\mathcal{S}_{g_2}(\mathcal{R}\text{oot}_{g_2}) = v_1 \dots v_k$. Pour tout $i \in 1..k$, $h_{|u_i}$ est un $\mathcal{V}$-homomorphime allant de $g_{1|u_i}$ vers $g_{2|v_i}$. Pour tout $i \in 1..k$, $l_{|n_i}$ est un arbre linéaire de hauteur inférieure ou égale à η et $h_{2|n_i}$ est un $\mathcal{V}$-homomorphisme allant de $l_{|n_i}$ vers $g_{2|v_i}$. Donc par hypothèse d'induction, il existe un filtre $\phi_i : l_{|n_i} \to g_{1|u_i}$ pour tout $i \in 1..k$. Soit $h_1 : \mathcal{N}_l \to \mathcal{N}_{g_1}$ une fonction telle que $h_1(\mathcal{R}\text{oot}_l) = \mathcal{R}\text{oot}_{g_1}$ et pour tout $i \in 1..k$, pour tout $n \in \mathcal{N}_{(l_{|n_i})}$, $h_1(n) = \phi_i(n)$. Nous allons prouver que h_1 est un homomorphisme de l vers g_1. Premièrement, l est un arbre linéaire, donc pour tout $i, j \in 1..k$, si $i \neq j$, alors $\mathcal{N}_{(l_{|n_i})} \cap \mathcal{N}_{(l_{|n_j})} = \emptyset$ et $\mathcal{V}_{(l_{|n_i})} \cap \mathcal{V}_{(l_{|n_j})} = \emptyset$. Par conséquent, $h_1(n)$ est défini une et une seule fois, pour tout nœud $n \in \mathcal{N}_l$, i.e., h_1 est une fonction. Deuxièmement, h_1 préserve la fonction d'étiquetage de l puisque $\mathcal{L}_{g_1}(h_1(\mathcal{R}\text{oot}_l)) = \mathcal{L}_{g_1}(\mathcal{R}\text{oot}_{g_1}) = \mathcal{L}_l(\mathcal{R}\text{oot}_l) = f$ et tous les ϕ_i préservent la fonction d'étiquetage de $l_{|n_i}$. Troisièmement, h_1 préserve la fonction successeur

de l puisque $\mathcal{S}_{g_1}(\mathcal{R}\text{oot}_{g_1}) = u_1 \dots u_k = \phi_1(n_1) \dots \phi_k(n_k) = h_1(\mathcal{S}_l(\mathcal{R}\text{oot}_l))$ et tous les ϕ_i préservent la fonction successeur de $l_{|n_i}$. Quatrièmement, h_1 préserve la racine de l. Nous en concluons que $h_1 : l \to g_1$ est un homomorphisme et donc que l filtre g_1 à la racine.

Pour finir, nous montrons la dernière partie de la proposition. Soient g_1 et g_2 deux termes-graphes bisimilaires tel que l filtre g_1 à la racine. Par définition de la relation de bisimilarité, il existe un graphe g et deux $\mathcal{V}$-homomorphimes $\mu_1 : g_1 \to g$ et $\mu_2 : g_2 \to g$. l filtre g_1 à la racine, donc l filtre g à la racine (d'après la première partie de la proposition). l filtre g à la racine et $\mu_2 : g_2 \to g$ est un $\mathcal{V}$-homomorphime, donc l filtre g_2 à la racine (d'après la seconde partie de la proposition). □

Nous établissons maintenant un résultat clé pour la preuve de confluence modulo la bisimilarité de la relation de réécriture.

Proposition 3.5.3 Soient SP un cGRS, g_1 et g_2 deux termes-graphes admissibles et $h : g_1 \to g_2$ un *$\mathcal{V}$-homomorphisme*. Si il existe un graphe g_2' tel que $g_2 \to g_2'$ (resp. $g_2 \xrightarrow{*} g_2'$), alors il existe un terme-graphe admissible g_1' et un $\mathcal{V}$-homomorphisme $h' : g_1' \to g_2'$ tel que $g_1 \xrightarrow{+} g_1'$ (resp. $g_1 \xrightarrow{*} g_1'$) (cf. Figure 3.11). ◇

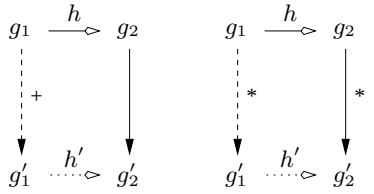


FIG. 3.11:

Une proposition similaire est donnée dans [AK96, Proposition 5.13] (pour une relation de réécriture différente de la notre, comme nous l'avons déjà dit).

Preuve : Nous donnons la preuve dans le cas d'un seul pas de réécriture de g_2 vers g_2'. Le cas de plusieurs pas se déduit immédiatement (par induction). Supposons que $g_2 \to_{[q,\, l \to r]} g_2'$. Par définition du pas de réécriture, il existe un homomorphisme $\psi : l \to g_{2|q}$ et $g_2' = g_2[q \leftarrow \psi[r]]$.

Soit $P = h^{-1}(q) = \{p_1, p_2, \dots, p_n\}$. Tous les nœuds de P sont disjoints deux à deux : sinon, si $p_i \prec p_j$ et $i \neq j$, alors il existe un chemin $C = [p_i, n_1, u_1, \dots, u_{k-1}, n_k, p_j]$ dans g_1 ; h étant un homomorphisme, $h(C) = [h(p_i), n_1, h(u_1), \dots, h(u_{k-1}), n_k, h(p_j)]$ est un chemin de g_2 ; or $h(p_i) = h(p_j) = q$, donc il existe un cycle sur le nœud q dans g_2, ce qui est impossible puisque q est un nœud fonctionnel et que g_2 est admissible. Par conséquent tous les nœuds de P sont disjoints deux à deux.

Soient $\{l_i \to r_i \mid i \in 1..n\}$ un ensemble de n variantes de la règle $l \to r$ et α_i l'isomorphisme allant du graphe $l_i \to r_i$ vers le graphe $l \to r$ pour tout $i \in 1..n$ (i.e., α_i est l'isomorphisme allant du graphe $l_i \oplus r_i$ vers le graphe $l \oplus r$ qui renomme tous les nœuds et toutes les variables de $l_i \oplus r_i$). ψ est un homomorphisme allant de l vers $g_{2|q}$ et $h_{|p_i}$ est un homomorphisme allant de $g_{1|p_i}$ vers $g_{2|q}$ pour tout $i \in 1..n$. Donc d'après la Proposition 3.5.1, il existe un homomorphisme ϕ_i allant de l vers $g_{1|p_i}$ pour tout $i \in 1..n$. Par conséquent, g_1 se réécrit au nœud p_i avec la règle $l_i \to r_i$ en $g_1[p_i \leftarrow \phi_i[r_i]]$ pour tout $i \in 1..n$. Toutes les positions dans P sont disjointes deux à deux, donc d'après la Proposition 3.4.6 (Commutativité), nous pouvons faire tous ces pas de réécriture dans n'importe quel ordre et nous obtenons le graphe $g_1' = g_1[p_1 \leftarrow \phi_1[r_1]] \dots [p_n \leftarrow \phi_n[r_n]]$. Il est clair que $g_1 \xrightarrow{+} g_1'$.

Montrons maintenant qu'il existe un $\mathcal{V}$-homomorphisme $h' : g_1' \to g_2'$. Pour cela, nous décomposons la construction de g_1' et de g_2' en trois étapes successives (en suivant les trois étapes de la définition d'un pas de remplacement) et nous prouvons à chaque étape qu'il existe un $\mathcal{V}$-homomorphisme.

Etape 1 : Soient $G_1 = g_1 \oplus \phi_1[r_1] \oplus \ldots \oplus \phi_n[r_n]$ et $G_2 = g_2 \oplus \psi[r]$. Il existe un $\mathcal{V}$-homomorphisme $\mu : G_1 \to G_2$ tel que pour tout $n \in \mathcal{N}_{g_1}$, $\mu(n) = h(n)$ et pour tout $i \in 1..k$, pour tout $n \in (\mathcal{N}_{r_i} - \mathcal{N}_{l_i})$, $\mu(n) = \alpha_i(n)$.
Etape 2 : Nous appliquons la redirection multiple de pointeurs ρ_1 sur G_1 et la redirection de pointeurs ρ_2 sur g_2 avec $\rho_1 = \{p_1 \mapsto \mathcal{R}\text{oot}_{\phi_1[r_1]}, \ldots, p_n \mapsto \mathcal{R}\text{oot}_{\phi_n[r_n]}\}$ et $\rho_2 = \{q \mapsto \mathcal{R}\text{oot}_{\psi[r]}\}$. μ est un homomorphisme allant de G_1 vers G_2 ; nous allons prouver que μ est aussi un homomorphisme allant de $\rho_1(G_1)$ vers $\rho_2(G_2)$. Premièrement, μ est une fonction allant de $\mathcal{N}_{G_1}$ vers $\mathcal{N}_{G_2}$, donc μ est aussi une fonction allant de $\mathcal{N}_{\rho_1(G_1)}$ vers $\mathcal{N}_{\rho_2(G_2)}$. Deuxièmement, μ préserve la fonction d'étiquetage de G_1, donc μ préserve aussi la fonction d'étiquetage de $\rho_1(G_1)$. Troisièmement, μ préserve la fonction successeur de $\rho_1(G_1)$. Pour montrer cela, nous devons vérifier que $\mathcal{S}_{\rho_2(G_2)}(\mu(n)) = \mu(\mathcal{S}_{\rho_1(G_1)}(n))$ pour tout $n \in \mathcal{N}_{\rho_1(G_1)}$. Comme $\mathcal{N}_{\rho_1(G_1)} = \mathcal{N}_{G_1}$, cela revient à montrer que pour tout $n \in \mathcal{N}_{G_1}$, $\rho_2(\mathcal{S}_{G_2}(\mu(n))) = \mu(\rho_1(\mathcal{S}_{G_1}(n)))$. Puisque μ est un homomorphisme allant de G_1 vers G_2, nous savons que $\mathcal{S}_{G_2}(\mu(n)) = \mu(\mathcal{S}_{G_1}(n))$, donc il faut établir que $\rho_2(\mu(\mathcal{S}_{G_1}(n))) = \mu(\rho_1(\mathcal{S}_{G_1}(n)))$. Ceci est vrai si $\rho_2(\mu(n)) = \mu(\rho_1(n))$ pour tout $n \in \mathcal{N}_{G_1}$. Il y a deux cas à étudier selon n. Premier cas, si $n \notin P$, alors $\rho_1(n) = n$ et donc $\mu(\rho_1(n)) = \mu(n)$. Comme $n \notin P$, $\mu(n) \neq q$, donc $\rho_2(\mu(n)) = \mu(n)$. Par conséquent, $\mu(\rho_1(n)) = \rho_2(\mu(n))$. Second cas, si $n \in P$, alors il existe un $i \in 1..k$ tel que $n = p_i$. Donc $\rho_1(n) = \rho_1(p_i) = \mathcal{R}\text{oot}_{\phi_i[r_i]} = \phi_i(\mathcal{R}\text{oot}_{r_i})$, et donc $\mu(\rho_1(n)) = \mu(\phi_i(\mathcal{R}\text{oot}_{r_i}))$. D'autre part, $\rho_2(\mu(n)) = \rho_2(\mu(p_i)) = \rho_2(q) = \mathcal{R}\text{oot}_{\psi[r]} = \psi(\mathcal{R}\text{oot}_r)$. Or par définition de μ, $\mu(\phi_i(\mathcal{R}\text{oot}_{r_i})) = \psi(\mathcal{R}\text{oot}_r)$. Donc dans le cas où $n \in P$, nous déduisons encore que $\mu(\rho_1(n)) = \rho_2(\mu(n))$, ce qui implique que μ préserve la fonction successeur de $\rho_1(G_1)$. Quatrièmement, $\mu(\mathcal{R}\text{oots}_{\rho_1(G_1)}) = \mathcal{R}\text{oots}_{\rho_2(G_2)}$. Donc nous en concluons que μ est un $\mathcal{V}$-homomorphisme allant de $\rho_1(G_1)$ vers $\rho_2(G_2)$.
Etape 3 : Soient $g'_1 = \rho_1(G_1)_{|\rho_1(\mathcal{R}\text{oots}_{g_1})}$, $g'_2 = \rho_2(G_2)_{|\rho_2(\mathcal{R}\text{oots}_{g_2})}$ et $h' = \mu_{|\mathcal{R}\text{oot}_{g'_1}}$. Il est clair que h' est un $\mathcal{V}$-homomorphisme allant de g'_1 vers g'_2. □

Nous établissons maintenant un second résultat fondamental pour faire la preuve du Théorème 3.3.7.

Proposition 3.5.4 Soient SP un WAGRS, g_1 et g_2 deux termes-graphes admissibles et $h : g_1 \to g_2$ un $\mathcal{V}$-homomorphisme. Si il existe un graphe g'_1 tel que $g_1 \to g'_1$, alors il existe deux termes-graphes admissibles g''_1 et g'_2 et un homomorphisme $h' : g''_1 \to g'_2$ tels que $g'_1 \xrightarrow{*} g''_1$ et $g_2 \to g'_2$. (cf. Figure 3.12). ◇

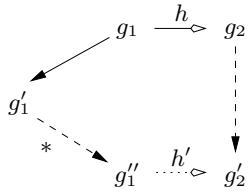


FIG. 3.12:

Preuve : Un schéma de cette preuve est donné dans la Figure 3.13. Supposons $g_1 \to_{[p_1, l \to r]} g'_1$. Soient $q = h(p_1)$ et $P = h^{-1}(q) = \{p_1, \ldots p_n\}$ (i.e., P est l'ensemble des antécédents de q par h). Comme il existe un homomorphisme allant de l vers $g_{1|p_1}$ (i.e., le filtre) et que $h_{|p_1}$ est un homomorphisme allant de $g_{1|p_1}$ vers $g_{2|q}$, il existe un homomorphisme allant de l vers $g_{2|q}$ (par composition). Donc le graphe g_2 se réécrit au nœud q avec la règle $l \to r$: il existe un graphe g'_2 tel que $g_2 \to_{[q, l \to r]} g'_2$. Puisque $g_2 \to_{[q, l \to r]} g'_2$ et qu'il existe un $\mathcal{V}$-homomorphisme $h : g_1 \to g_2$, on déduit avec la Proposition 3.5.3 qu'il existe un graphe d et un homomorphisme $j : d \to g'_2$ tels que $g_1 \xrightarrow{+} d$ (cf. A dans la Figure 3.13). En reprenant la preuve de la Proposition 3.5.3, on infère que tous les pas de la dérivation de réécriture $g_1 \xrightarrow{+} d$ sont exécutés avec les nœuds de P. De plus, on sait que les nœuds de P sont deux à deux disjoints

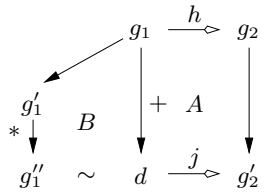


FIG. 3.13:

dans g_1. Comme p_1 est un nœud de P et que $g_1 \rightarrow_{[p_1, l \rightarrow r]} g'_1$, il existe un graphe g''_1 tel que $g'_1 \stackrel{*}{\rightarrow} g''_1$ et $g''_1 \sim d$ (cf. B dans la Figure 3.13). Les nœuds utilisés dans la dérivation de réécriture $g'_1 \stackrel{*}{\rightarrow} g''_1$ sont ceux de $P - \{p_1\}$, c'est-à-dire $\{p_2, \ldots, p_n\}$. Pour finir, g''_1 et d sont égaux au renommage des nœuds près, donc il existe un $\mathcal{V}$-isomorphisme $\alpha : g''_1 \rightarrow d$. De plus, $j : d \rightarrow g'_2$ est un $\mathcal{V}$-homomorphisme. Donc par composition, $\alpha \circ j$ est un $\mathcal{V}$-homomorphisme allant de g''_1 vers g'_2. □

Nous généralisons maintenant la Proposition 3.5.4.

Proposition 3.5.5 Soient SP un WAGRS, g_1 et g_2 deux termes-graphes admissibles et $h : g_1 \rightarrow g_2$ un $\mathcal{V}$-homomorphisme. Si il existe un graphe g'_1 tel que $g_1 \stackrel{*}{\rightarrow} g'_1$, alors il existe deux termes-graphes admissibles g''_1 et g'_2 et un $\mathcal{V}$-homomorphisme $h' : g''_1 \rightarrow g'_2$ tels que $g'_1 \stackrel{*}{\rightarrow} g''_1$ et $g_2 \stackrel{*}{\rightarrow} g'_2$ (cf. Figure 3.12). ◇

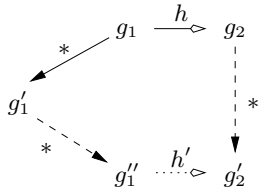


FIG. 3.14:

Preuve : Nous faisons cette preuve par induction sur la longueur de la dérivation $g_1 \stackrel{*}{\rightarrow} g'_1$, comme indiqué dans la Figure 3.15. Pour une dérivation de longueur 0, la proposition est évidente (i.e.,

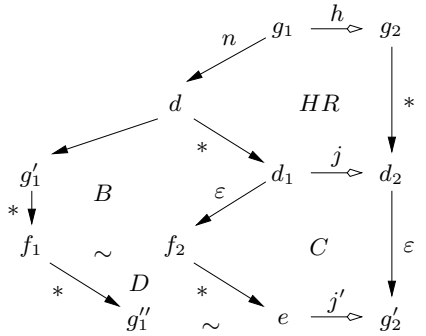


FIG. 3.15:

$g''_1 = g_1$, $g'_2 = g_2$ et $h' = h$). Supposons que la proposition soit vraie pour une longueur n donnée. Soient g_1 et g_2 deux termes-graphes admissibles et $h : g_1 \rightarrow g_2$ un $\mathcal{V}$-homomorphisme tels qu'il existe

un graphe g'_1 avec $g_1 \overset{n+1}{\to} g'_1$. Alors il existe un graphe d tel que $g_1 \overset{n}{\to} d \to g'_1$. Comme $h : g_1 \to g_2$ est un $\mathcal{V}$-homomorphisme et que $g_1 \overset{n}{\to} d$, on en déduit qu'il existe deux graphes d_1 et d_2 et un $\mathcal{V}$-homomorphisme $j : d_1 \to d_2$ tels que $d \overset{*}{\to} d_1$ et $g_2 \overset{*}{\to} d_2$ par hypothèse d'induction (cf. HR dans la Figure 3.15). Comme $d \to g'_1$ et $d \overset{*}{\to} d_1$, d'après le Théorème 3.3.4, il existe deux graphes f_1 et f_2 tels que $g'_1 \overset{*}{\to} f_1$, $d_1 \overset{\varepsilon}{\to} f_2$ et $f_1 \sim f_2$ (cf. B dans la Figure 3.15). Puisqu'il existe un $\mathcal{V}$-homomorphisme $j : d_1 \to d_2$ et que $d_1 \overset{\varepsilon}{\to} f_2$, d'après la Proposition 3.5.4, il existe deux graphes e et g'_2 et un $\mathcal{V}$-homomorphisme $j' : e \to g'_2$ tels que $f_2 \overset{*}{\to} e$ et $d_2 \overset{\varepsilon}{\to} g'_2$ (cf. C dans la Figure 3.15). Puisque $f_2 \overset{*}{\to} e$ et $f_1 \sim f_2$, d'après la Proposition 3.4.8, il existe un graphe g''_1 tel que $f_1 \overset{*}{\to} g''_1$ et $g''_1 \sim e$ (cf. D dans la Figure 3.15). Pour finir, comme $g''_1 \sim e$, il existe un $\mathcal{V}$-isomorphisme $\alpha : g''_1 \to e$. De plus, $j' : e \to g'_2$ est un $\mathcal{V}$-homomorphisme. Donc $j' \circ \alpha : g''_1 \to g'_2$ est un $\mathcal{V}$-homomorphisme par composition. □

Nous sommes prêts pour démontrer le Théorème 3.3.7. Cette preuve est directe grâce aux propositions précédentes.

Preuve du Théorème 3.3.7 : Soient SP un WAGRS et g_1 et g_2 deux termes-graphes admissibles et bisimilaires ($g_1 \doteq g_2$) tels qu'il existe deux graphes g'_1 et g'_2 avec $g_1 \overset{*}{\to} g'_1$ et $g_2 \overset{*}{\to} g'_2$. Nous prouvons qu'il existe deux termes-graphes admissibles g''_1 et g''_2 tels que $g'_1 \overset{*}{\to} g''_1$, $g'_2 \overset{*}{\to} g''_2$ et $g''_1 \doteq g''_2$. Nous donnons le schéma de cette preuve dans la Figure 3.16. D'après la Définition 2.6.5, il existe un

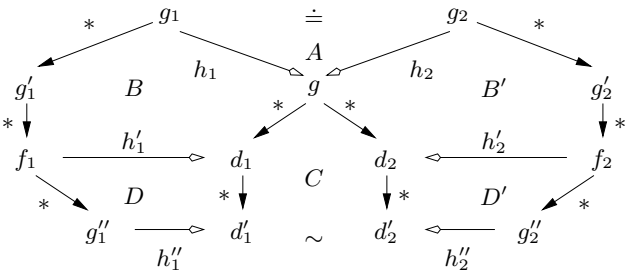


FIG. 3.16:

terme-graphe admissible g, un $\mathcal{V}$-homomorphisme $h_1 : g_1 \to g$ et un $\mathcal{V}$-homomorphisme $h_2 : g_2 \to g$ (cf. A dans la Figure 3.16). Puisque $g_1 \overset{*}{\to} g'_1$ et qu'il existe un $\mathcal{V}$-homomorphisme $h_1 : g_1 \to g$, d'après la Proposition 3.5.5, il existe deux termes-graphes admissibles f_1 et d_1 et un $\mathcal{V}$-homomorphisme $h'_1 : f_1 \to d_1$ tel que $g'_1 \overset{*}{\to} f_1$ et $g \overset{*}{\to} d_1$ (cf. B dans la Figure 3.16). Pour les mêmes raisons, il existe deux termes-graphes admissibles f_2 et d_2 et un $\mathcal{V}$-homomorphisme $h'_2 : f_2 \to d_2$ tels que $g'_2 \overset{*}{\to} f_2$ et $g \overset{*}{\to} d_2$ (cf. B' dans la Figure 3.16). Comme $g \overset{*}{\to} d_1$ et $g \overset{*}{\to} d_2$, d'après le Théorème 3.3.4 (Confluence), il existe deux termes-graphes admissibles d'_1 et d'_2 tels que $d_1 \overset{*}{\to} d'_1$, $d_2 \overset{*}{\to} d'_2$ et $d'_1 \sim d'_2$ (cf. C dans la Figure 3.16). Comme $d_1 \overset{*}{\to} d'_1$ et qu'il existe un $\mathcal{V}$-homomorphisme $h'_1 : f_1 \to d_1$, d'après la Proposition 3.5.3, il existe un graphe g''_1 et un $\mathcal{V}$-homomorphisme $h''_1 : g''_1 \to d'_1$ tels que $f_1 \overset{*}{\to} g''_1$ (cf. D dans la Figure 3.16). Pour les mêmes raisons, il existe un graphe g''_2 et un $\mathcal{V}$-homomorphisme $h''_2 : g''_2 \to d'_2$ tels que $f_2 \overset{*}{\to} g''_2$ (cf. D' dans la Figure 3.16). Pour finir, h''_1 est un $\mathcal{V}$-homomorphisme allant de g''_1 vers d'_1, d'_1 et d'_2 sont égaux au renommage des nœuds près et h''_2 est un $\mathcal{V}$-homomorphisme allant de g''_2 vers d'_2, donc d'après la Définition 2.6.5, nous concluons que $g''_1 \doteq g''_2$. □

3.6 Systèmes bWAGRS

Nous avons vu dans la section précédente que les WAGRS étaient confluents modulo la bisimilarité. On peut aussi se demander si la classe des cGRS telle que les membres droits instanciés de deux règles superposables soient bisimilaires (et donc pas nécessairement égaux au

renommage des nœuds près) induisent, eux aussi, une relation de réécriture qui est confluente modulo la bisimilarité.

Définition 3.6.1 (bWAGRS)
Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS. On dit que SP est un *système de réécriture de graphes faiblement admissible avec bisimilarité (bWAGRS)* ssi $\mathcal{R}$ est un ensemble de règles admissibles tel que si deux règles $l_1 \to r_1$ et $l_2 \to r_2$ sont superposables, alors leurs membres droits instanciés sont bisimilaires, i.e., si il existe un graphe admissible g et un homomorphisme $h : (l_1 \oplus l_2) \to g$ tel que $h(l_1) = h(l_2) = g$, alors $h[r_1] \doteq h[r_2]$. ◇

Exemple 3.6.2 Considérons les règles suivantes :

```
(T1) l1:A(l2:B,l3:X) -> r1:C(r2:C(r1,l3:X),l3)
(T2) l1:A(l2:Y,l3:D(l4:B)) -> r1:C(r1,r2:D(r3:B))
```

Comme `T1` et `T2` sont superposables et que leurs membres droits instancés sont bisimilaires, ce cGRS est un bWAGRS. □

Il est clair qu'un bWAGRS ne peut pas être confluent au renommage de nœuds près, en général. Mais un bWAGRS peut quand même être confluent modulo la bisimilarité. Nous montrons cette propriété dans le cas où la relation de réécriture considérée est *nœthérienne* (ou *terminante*) *par rapport aux termes-graphes admissibles*.

Définition 3.6.3 (Relation de réécriture nœthérienne)
Soit SP un cGRS. On dit que la relation de réécriture est *nœthérienne par rapport aux termes-graphes admissibles* ssi pour tout terme-graphe admissible g, il n'existe pas de dérivation de réécriture infinie $g \to g_1 \to g_2 \to \ldots$. ◇

Dauchet a montré que la terminaison des systèmes de réécriture de termes était indécidable, même si ceux-ci ne contiennent qu'une seule règle de réécriture [Dau89]. Toutefois, de nombreuses techniques de preuve de terminaison existent (voir [HO80, DJ90, Klo92]). Concernant la terminaison les systèmes de réécriture de graphes, [Plu98b] a récemment montré qu'elle était aussi indécidable en général (pour le cas d'une définition de la réécriture avec des *double pushouts* [EKL90]). Une technique de preuve de terminaison est développée dans [Plu97] dans le cas des systèmes de réécriture de graphes acycliques. Cette technique peut être étendue pour traiter les bWAGRS dans le cas des graphes cycliques admissibles.

Theorème 3.6.4 Soit SP un bWAGRS. Si la relation de réécriture est nœthérienne par rapport aux termes-graphes admissibles, alors $\to$ est confluente modulo la bisimilarité par rapport aux termes-graphes admissibles (cf. Figure 3.6). ◇

Notre preuve du Théorème 3.6.4 repose sur la notion de *confluence locale modulo la bisimilarité* qui est définie ci-dessous :

Définition 3.6.5 (Confluence locale modulo la bisimilarité)
Soit SP un cGRS. On dit que *la relation de réécriture est localement confluente modulo la bisimilarité par rapport aux termes-graphes admissibles* ssi les conditions suivantes sont vérifiées :

1. Pour tout terme-graphe admissible g, g_1 et g_2 tels que $g \to g_1$ et $g \to g_2$, il existe deux termes-graphes admissibles g'_1 et g'_2 tels que $g_1 \stackrel{*}{\to} g'_1$, $g_2 \stackrel{*}{\to} g'_2$ et $g'_1 \doteq g'_2$ (cf. Figure 3.17 (a)).

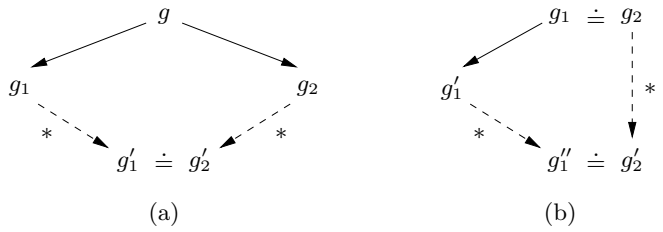


FIG. 3.17:

2. Pour tout terme-graphe admissible g_1, g_2 et g'_1 tels que $g_1 \doteq g_2$ et $g_1 \rightarrow g'_1$, il existe deux termes-graphes admissibles g''_1 et g'_2 tels que $g'_1 \stackrel{*}{\rightarrow} g''_1$, $g_2 \stackrel{*}{\rightarrow} g'_2$ et $g''_1 \doteq g'_2$ (cf. Figure 3.17 (b)).

◇

Quand une relation de réécriture est nœthérienne et localement confluente modulo une relation d'équivalence, on peut montrer qu'elle est confluente modulo cette relation d'équivalence :

Lemme 3.6.6 Soit SP un cGRS. Si $\rightarrow$ est nœthérienne par rapport aux termes-graphes admissibles, alors $\rightarrow$ est confluente modulo $\doteq$ par rapport aux termes-graphes admissibles ssi $\rightarrow$ est localement confluente modulo $\doteq$ par rapport aux termes-graphes admissibles. ◇

Preuve : Conséquence évidente de [Hue80, Lemma 2.7]. □

Nous établissons ci-dessous deux propositions impliquant la confluence locale modulo $\doteq$ par rapport aux termes-graphes admissibles de la relation de réécriture engendrée par un bWAGRS.

Proposition 3.6.7 Soit SP un bWAGRS. Pour tout terme-graphe admissible g, g_1 et g_2 tels que $g \rightarrow g_1$ et $g \rightarrow g_2$, il existe deux termes-graphes admissibles g'_1 et g'_2 tels que $g_1 \stackrel{\varepsilon}{\rightarrow} g'_1$, $g_2 \stackrel{\varepsilon}{\rightarrow} g'_2$ et $g'_1 \doteq g'_2$ (cf. Figure 3.18). ◇

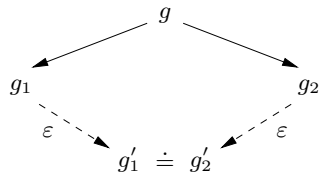


FIG. 3.18:

Preuve : Cette preuve nécessite une analyse par cas. Soient g, g_1 et g_2 trois termes-graphes admissibles tels que $g \rightarrow_{[n_1, l_1 \rightarrow r_1]} g_1$ et $g \rightarrow_{[n_2, l_2 \rightarrow r_2]} g_2$. Par définition d'un pas de réécriture, il existe un filtre $h_1 : l_1 \rightarrow g_{|n_1}$ tel que $g_1 = g[n_1 \leftarrow h_1[r_1]]$. De même, il existe un filtre $h_2 : l_2 \rightarrow g_{|n_2}$ tel que $g_2 = g[n_2 \leftarrow h_2[r_2]]$. Si $n_1 = n_2$, alors $h_1[r_1]$ et $h_2[r_2]$ sont bisimilaires, par définition d'un bWAGRS. Aussi, $g[n_1 \leftarrow h_1[r_1]]$ et $g[n_2 \leftarrow h_2[r_2]]$ sont bisimilaires, i.e., $g_1 \doteq g_2$. Donc le lemme est vérifié (i.e., $g'_1 = g_1$ et $g'_2 = g_2$). Si $n_1 \neq n_2$, alors la preuve est exactement la même que celle du Lemme 3.4.9, i.e., il existe deux termes-graphes admissibles g'_1 et g'_2 tels que $g_1 \stackrel{\varepsilon}{\rightarrow} g'_1$, $g_2 \stackrel{\varepsilon}{\rightarrow} g'_2$ et $g'_1 \sim g'_2$ (cf. Figure 3.8). Comme g'_1 et g'_2 sont égaux au renommage des nœuds près, g'_1 et g'_2 sont bisimilaires. Donc le lemme est vérifié. □

Proposition 3.6.8 Soit SP un bWAGRS. Pour tout terme-graphe admissible g_1, g_2 et g_1' tels que $g_1 \to g_1'$ et $g_1 \doteq g_2$, il existe deux termes-graphes admissibles g_1'' et g_2' tels que $g_1' \overset{*}{\to} g_1''$, $g_2 \overset{*}{\to} g_2'$ et $g_1'' \doteq g_2'$ (cf. Figure 3.19). ◇

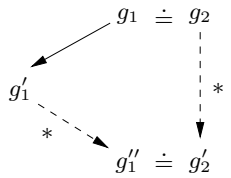


FIG. 3.19:

Preuve : Soient g_1, g_2 et g_1' trois termes-graphes admissibles tels que $g_1 \to g_1'$ et $g_1 \doteq g_2$. Comme $g_1 \doteq g_2$, il existe un terme-graphe admissible g et deux $\mathcal{V}$-homomorphismes $h_1 : g_1 \to g$ et $h_2 : g_2 \to g$, d'après la Definition 2.6.5. Puisqu'il existe un $\mathcal{V}$-homomorphisme $h_1 : g_1 \to g$ et que $g_1 \to g_1'$, il existe deux termes-graphes admissibles g_1'' et g' et un $\mathcal{V}$-homomorphisme $h_1' : g_1'' \to g'$ tels que $g_1' \overset{*}{\to} g_1''$ et $g \to g'$ (cf. Figure 3.12). Comme il existe un $\mathcal{V}$-homomorphisme $h_2 : g_2 \to g$ et que $g \to g'$, il existe un terme-graphe admissible g_2' et un $\mathcal{V}$-homomorphisme $h_2' : g_2' \to g'$ tels que $g_2 \overset{+}{\to} g_2'$ (cf. Figure 3.11). Pour finir, il existe deux $\mathcal{V}$-homomorphismes $h_1' : g_1'' \to g'$ et $h_2' : g_2' \to g'$, donc $g_1'' \doteq g_2'$. □

Preuve du Théorème 3.6.4 : Les Propositions 3.6.7 et 3.6.8 impliquent que la relation de réécriture engendrée par le bWAGRS est localement confluente modulo $\doteq$ par rapport aux termes-graphes admissibles. De plus nous supposons qu'elle est nœthérienne par rapport aux termes-graphes admissibles. Donc le Théorème 3.6.4 est une conséquence immédiate du Lemme 3.6.6. □

Chapitre 4

Stratégies de réécriture de graphes admissibles

Les programmes des langages logico-fonctionnels que nous considérons sont modélisés à l'aide de systèmes de réécriture de graphes avec constructeurs. Une manière d'utiliser ces programmes consiste en l'évaluation par réécriture d'expressions représentées sous la forme de graphes cycliques particuliers, les graphes admissibles, qui ne contiennent pas d'opération définie dans leurs cycles. Nous avons montré dans le chapitre précédent que la relation de réécriture engendrée par les systèmes de réécriture de graphes faiblement admissibles avec constructeurs (WAGRS) était confluente par rapport aux graphes admissibles.

Lorsqu'un système de réécriture est confluent, on peut calculer une expression de manière déterministe et efficace en utilisant des *stratégies de réécriture*. Appliquée à un graphe, une stratégie de réécriture désigne les rédex de ce graphe qu'il est utile de réécrire pour calculer sa forme normale constructeur quand elle existe (c'est-à-dire la valeur de l'expression que ce graphe représente).

De nombreuses stratégies ont été proposées dans le cadre des systèmes de réécriture *orthogonaux* de termes du premier ordre (OTRS) (par exemple [HL91]) et de termes infinis (OT$^\infty$RS) (par exemple [KKSV95]). Une stratégie adaptée aux systèmes de réécriture de termes avec constructeurs est développée dans [Ant92]. Nous montrons dans ce chapitre que cette stratégie, notée Φ, peut être étendue aux graphes admissibles tout en préservant ses "bonnes" propriétés.

De même, plusieurs stratégies ont été proposées dans le cadre des systèmes de réécriture *faiblement orthogonaux* de termes (WOTRS). O'Donnell a par exemple montré que la stratégie qui consiste en la réécriture en parallèle de tous les plus hauts rédex d'un terme[1] permet de calculer la forme normale de ce terme [O'D77]. Sekar and Ramakrishnan [SR93] ainsi qu'Antoy [Ant92] ont amélioré cette stratégie. Nous montrons dans ce chapitre que la seconde stratégie d'Antoy, notée $\bar{\Phi}$, peut être étendue aux graphes admissibles, tout comme Φ. Pour faire cette extension, nous définissons la relation de *réécriture parallèle* de graphes admissibles.

Nos résultats concernant Φ sont publiés dans [EJ97c, EJ98a, EJ98b]. Ceux concernant $\bar{\Phi}$ se trouvent dans [EJ99b]. Ils sont développés en détails dans [EJ98c, EJ97b].

Nous commençons ce chapitre avec des préliminaires précisant le vocabulaire et les notions utilisés dans la suite. Dans les Sections 4.2 et 4.3, nous étudions successivement les extensions des stratégies Φ et $\bar{\Phi}$ aux graphes admissibles tout en soulignant leurs propriétés d'optimalité.

[1] *parallel-outermost strategy* en anglais.

4.1 Préliminaires

Pour calculer la forme normale constructeur d'un graphe, il faut exhiber un nœud de ce graphe ainsi qu'une règle permettant de réécrire ce nœud :

Définition 4.1.1 (Stratégie de réécriture de graphes)
Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS. Une *stratégie (séquentielle) de réécriture de graphes* est une fonction partielle $\mathcal{S}$ qui prend un terme-graphe admissible g et qui retourne un couple (p, R) tel que p soit un nœud de g, R soit une règle de réécriture de $\mathcal{R}$ et g se réécrive au nœud p avec la règle R. On note $g \rightarrow_{\mathcal{S}} g'$ le *$\mathcal{S}$-pas de réécriture* de g vers g' tel que $\mathcal{S}(g) = (p, R)$ et $g \rightarrow_{[p,R]} g'$. On note $\overset{*}{\rightarrow}_{\mathcal{S}}$ la fermeture réflexive et transitive de $\rightarrow_{\mathcal{S}}$ et on parle de *$\mathcal{S}$-dérivation de réécriture* de g en g' lorsque $g \overset{*}{\rightarrow}_{\mathcal{S}} g'$. ◇

Plusieurs propriétés des stratégies sont intéressantes. On exigera toujours d'une stratégie qu'elle permette de calculer la forme normale constructeur d'un graphe lorsqu'elle existe. C'est la propriété dite de *$\mathcal{C}$-normalisation*.

Définition 4.1.2 ($\mathcal{C}$-normalisation)
Soit SP un cGRS. Une stratégie $\mathcal{S}$ est *$\mathcal{C}$-normalisante* ssi pour tout terme-graphe admissible g et pour tout terme-graphe constructeur c tels que $g \overset{*}{\rightarrow} c$, il existe un terme-graphe constructeur c' et une $\mathcal{S}$-dérivation $g \overset{*}{\rightarrow}_{\mathcal{S}} c'$ tels que $c' \doteq c$. ◇

Une autre propriété plus forte que la $\mathcal{C}$-normalisation stipule que l'usage arbitraire mais régulier d'une stratégie engendre des dérivations de réécriture qui aboutissent à la forme normale constructeur d'un graphe lorsqu'elle existe. Cette propriété permet de combiner une stratégie de réécriture avec d'autres stratégies de réécriture tout en conservant la propriété de $\mathcal{C}$-normalisation. C'est la propriété dite de *$\mathcal{C}$-hyper-normalisation*.

Définition 4.1.3 ($\mathcal{C}$-hyper-normalisation)
Soit SP un cGRS. Une stratégie $\mathcal{S}$ est *$\mathcal{C}$-hyper-normalisante* ssi pour tout terme-graphe admissible g et pour tout terme-graphe constructeur c tels que $g \overset{*}{\rightarrow} c$, toute dérivation de réécriture D qui commence avec g et qui alterne des $\mathcal{S}$-pas de réécriture avec un ou plusieurs pas de réécriture quelconques termine par une forme normale constructeur c' telle que $c' \doteq c$. Une stratégie $\mathcal{C}$-hyper-normalisante est forcément $\mathcal{C}$-normalisante. ◇

De nombreuses stratégies ont été définies dans le cadre de la réécriture de termes du premier ordre (voir le catalogue établi dans [Klo92]). Dans ce chapitre, nous nous intéressons à l'une d'entre elle, Φ, définie dans [Ant92] pour certains systèmes de réécriture de termes avec constructeurs qui sont dits *inductivement séquentiels*. Très efficace, cette stratégie calcule des *rédex nécessaires*[2] [HL91]. Cette propriété permet de montrer que Φ est $\mathcal{C}$-hyper-normalisante. Dans la définition suivante, nous généralisons la notion de rédex nécessaires aux termes-graphes admissibles.

Définition 4.1.4 (Rédex nécessaire)
Soient SP un cGRS, g et g' deux termes-graphes admissibles, $D = g \overset{*}{\rightarrow} g'$ une dérivation de réécriture et q un nœud fonctionnel de g.
On dit que q est *un nœud résiduel par D* si q reste un nœud de g' ; dans ce cas, on appelle *descendant de $g_{|q}$ par D* le sous-graphe $(g')_{|q}$.
On dit qu'un rédex de racine q est *nécessaire* dans g ssi dans toute dérivation de g vers une forme normale constructeur, un descendant de g se réécrit au nœud q. ◇

[2] *needed redexes* en anglais.

Exemple 4.1.5 Soit g = `n1 :A(n2 :A(n3 :B,n3),n4 :A(n2,n3))` où `A` est définie avec la règle `(T) l1 :A(l2 :B,l3 :X)` → `l2 :B`. Pour calculer la forme normale constructeur de g, il faut réécrire g à la racine en un terme-graphe constructeur. Ceci n'est possible que si $g_{|\texttt{n2}}$ se réécrit en un terme-graphe constructeur. Comme $g_{|\texttt{n2}}$ est un rédex, nous en déduisons que $g_{|\texttt{n2}}$ est un rédex nécessaire de g. Nous obtenons $g \rightarrow_{[\texttt{n2},\texttt{T}]} g'$ avec g' = `n1 :A(n3 :B,n4 :A(n3,n3))`. Pour calculer la forme normale constructeur de g', il suffit de réécrire g' à la racine en g'' = `n3 :B`. On constate que $g'_{|\texttt{n4}}$ est aussi un rédex de g'. Mais il n'est pas nécessaire de le réécrire pour calculer la forme normale de g'. $g'_{|\texttt{n4}}$ n'est donc pas un rédex nécessaire de g'. □

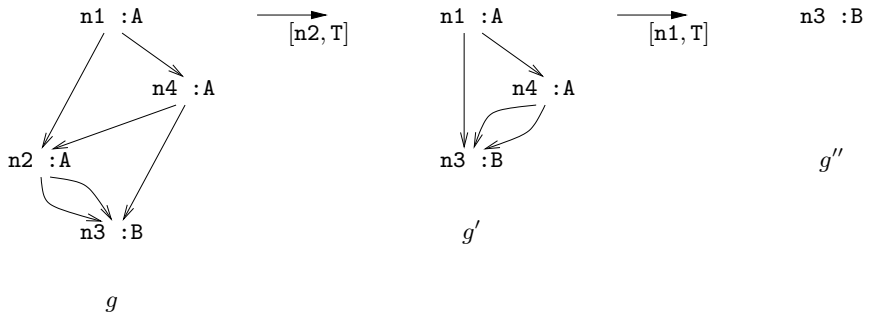


FIG. 4.1:

L'exemple ci-dessus illustre la notion de rédex nécessaire mais cette notion n'est pas toujours pertinente dans le cas général d'un WAGRS.

Exemple 4.1.6 On considère de nouveau le graphe g de l'Exemple 4.1.5 mais on suppose maintenant que `A` est définie avec les règles `(T1) l1 :A(l2 :B,l3 :X)` → `l2 :B` et `(T2) l1 :A(l2 :X,l3 :B)` → `l3 :B`. Dans ce cas, les sous-graphes $g_{|\texttt{n2}}$ et $g_{|\texttt{n4}}$ sont des rédex de g, mais aucun d'entre eux n'est nécessaire au sens de la Définition 4.1.4. Par contre, il faut forcément réduire l'un d'entre eux pour calculer la forme normale constructeur de g. □

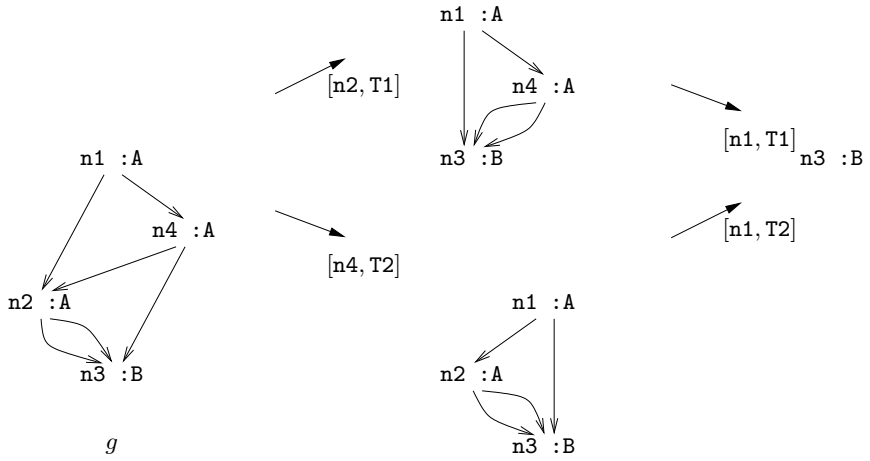


FIG. 4.2:

Dans l'exemple précédent, nous avons besoin d'une notion qui généralise celle de rédex nécessaire. Nous introduisons ci-dessous la notion d'*ensemble suffisant de rédex*[3] [SR93] :

Définition 4.1.7 (Ensemble suffisant de rédex)
Soient SP un cGRS et g un terme-graphe admissible. On dit qu'un ensemble de rédex $S = \{u_1, \ldots, u_n\}$ est un *ensemble suffisant de rédex* dans g ssi dans toute dérivation de réécriture allant de g vers une forme normale constructeur, un descendant d'*au moins un* rédex $u \in S$ est réécrit à la racine. ◇

Exemple 4.1.8 Dans l'Exemple 4.1.6, l'ensemble $S = \{g_{|\mathtt{n2}}, g_{|\mathtt{n4}}\}$ est un ensemble suffisant de rédex dans g. □

Nous avons vu dans l'Exemple 4.1.6 un cas de WAGRS où la notion de rédex nécessaire n'est pas pertinente. En fait, la stratégie Φ de [Ant92] n'est pas définie pour ce système de réécriture car il n'est pas inductivement séquentiel.

Dans le cas général des systèmes de réécriture de termes avec constructeurs qui sont faiblement orthogonaux mais pas inductivement séquentiels, [Ant92] propose une seconde "stratégie", $\bar{\Phi}$, qui calcule des ensembles suffisants de rédex. Le mot "stratégie" ne correspond pas à celui de la Définition 4.1.1 : $\bar{\Phi}$ est une stratégie *parallèle* de réécriture.

Définition 4.1.9 (Stratégie parallèle de réécriture)
Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS. Une *stratégie parallèle de réécriture de graphes* est une fonction partielle $\bar{\mathcal{S}}$ qui prend un terme-graphe admissible g et qui retourne *un ensemble de couples* (p, R) tels que p soit un nœud de g, R soit une règle de réécriture de $\mathcal{R}$ et g se réécrive au nœud p avec la règle R. ◇

Tous les rédex calculés par $\bar{\Phi}$ doivent être réécrits si on veut être sûr de calculer la forme normale constructeur d'un graphe lorsqu'elle existe. Ces pas de réécriture peuvent être vus de manière atomique en utilisant un *pas de réécriture parallèle* de graphes. C'est la raison pour laquelle on qualifie de *parallèle* ce type de stratégies.

Dans la Section 4.2, nous définissons les systèmes de réécriture de graphes inductivement séquentiels puis nous étendons la stratégie Φ aux termes-graphes admissibles. Nous montrons ensuite que Φ calcule des rédex nécessaires puis qu'elle est $\mathcal{C}$-hyper-normalisante. Enfin, nous verrons que Φ engendre la dérivation de réécriture la plus courte à partir d'un graphe.

Dans la Section 4.3, nous commençons par définir le pas de réécriture parallèle de graphes, puis nous étendons la stratégie $\bar{\Phi}$ aux WAGRS. Nous verrons que $\bar{\Phi}$ calcule des ensembles suffisants de rédex puis qu'elle est $\mathcal{C}$-hyper-normalisante.

4.2 Stratégie Φ

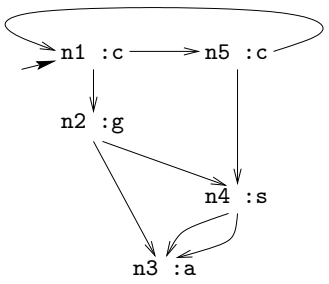
Dans cette section, nous étendons la stratégie Φ définie dans [Ant92] aux systèmes de réécriture de graphes inductivement séquentiels. Cette stratégie calcule des *plus hauts rédex nécessaires*. Nous avons déjà vu la notion de rédex nécessaire (cf. Définition 4.1.4). Nous explicitons ci-dessous les notions de *plus haut nœud*, de *plus haut rédex*, de *plus haut nœud à gauche* et de *plus haut rédex à gauche* dans un graphe admissible.

Définition 4.2.1 Soient SP un cGRS et g un terme-graphe admissible.

[3] *necessary set of redexes* en anglais.

On dit qu'un nœud fonctionnel q est un *plus haut nœud fonctionnel* de g ssi $q = \mathcal{R}\text{oot}_g$ lorsque $\mathcal{R}\text{oot}_g$ est un nœud fonctionnel ou sinon $\mathcal{S}_g(\mathcal{R}\text{oot}_g) = r_1 \dots r_k$ et il existe $i \in 1..k$ tel que q soit un plus haut nœud fonctionnel de $g_{|r_i}$.
Un rédex u de racine q dans g est un *plus haut rédex* de g ssi $q = \mathcal{R}\text{oot}_g$ lorsque g est un rédex ou sinon $\mathcal{S}_g(\mathcal{R}\text{oot}_g) = r_1 \dots r_k$ et il existe $i \in 1..k$ tel que u soit un plus haut rédex de $g_{|r_i}$.
Un nœud fonctionnel q est *le plus haut nœud fonctionnel à gauche* de g ssi $q = \mathcal{R}\text{oot}_g$ lorsque $\mathcal{R}\text{oot}_g$ est un nœud fonctionnel ou sinon $\mathcal{S}_g(\mathcal{R}\text{oot}_g) = r_1 \dots r_k$ et il existe $i \in 1..k$ tel que q soit un plus haut nœud fonctionnel à gauche dans $g_{|r_i}$ et les sous-graphes $g_{|r_j}$ sont constructeurs pour tout $j < i$.
Un rédex u de racine q dans g est *le plus haut rédex à gauche* de g ssi $q = \mathcal{R}\text{oot}_g$ lorsque g est un rédex ou sinon $\mathcal{S}_g(\mathcal{R}\text{oot}_g) = r_1 \dots r_k$ et il existe $i \in 1..k$ tel que u soit un plus haut rédex à gauche dans $g_{|r_i}$ et les sous-graphes $g_{|r_j}$ sont constructeurs pour tout $j < i$. ◇

Exemple 4.2.2 Soit g = `n1 :c(n2 :g(n3 :a,n4 :h(n3,n3)),n5 :c(n4,n1))` (cf. Figure 4.3) et $SP_2 = \langle \Sigma, \mathcal{R}_2 \rangle$ le WAGRS de l'Exemple 3.1.12. Les plus hauts rédex de g sont $g_{|\mathtt{n2}}$ et $g_{|\mathtt{n4}}$. $g_{|\mathtt{n2}}$ est le plus haut rédex à gauche de g. Le lecteur remarque qu'il existe un chemin allant de `n2` vers `n4`, mais $g_{|\mathtt{n4}}$ ne disparait pas si on réécrit $g_{|\mathtt{n2}}$ (car le chemin $[\mathtt{n1}, 2, \mathtt{n5}, 1, \mathtt{n4}]$ n'est pas modifié par un pas de réécriture au nœud `n2`). Notons encore que les



G

FIG. 4.3:

notions de plus haut nœud fonctionnel (à gauche) et de plus haut rédex (à gauche) sont bien définies dans le cadre des graphes admissibles ; en effet, si p et q sont deux nœuds fonctionnels tels qu'il existe un chemin de p vers q, c'est-à-dire, tels que p soit plus haut que q, alors il n'existe pas de chemin de q vers p, par admissibilité. □

La stratégie Φ que nous développons ici utilise une extension des arbres de définition [Ant92]. Un arbre de définition est une structure hiérarchique dont les feuilles sont des règles d'un cGRS utilisées pour définir une opération. Dans la définition suivante, *branch* et *rule* sont des symboles non interprétés utilisés pour construire les nœuds d'un arbre de définition.

Définition 4.2.3 (Arbre de définition)
Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS. Un arbre $\mathcal{T}$ est un *arbre de définition partiel (pdt) de motif* π dans les deux cas suivants :

- $\mathcal{T} = rule(\pi \to r)$, où $\pi \to r$ est une variante d'une règle de $\mathcal{R}$.
- $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_k)$, où o est un nœud variable de π, o a pour sorte ζ, $c_1, \dots, c_k$ ($k > 0$) sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt

de motif $\pi[o \leftarrow p : c_j(o_1 : x_1, \ldots, o_n : x_n)]$, tel que n soit le nombre d'arguments de c_j, $x_1, \ldots, x_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds.

$pattern(\mathcal{T})$ désigne le motif du pdt $\mathcal{T}$.
Un *arbre de définition* $\mathcal{T}$ d'une opération définie f est un pdt *fini* avec un motif de la forme $p : f(o_1 : x_1, \ldots, o_n : x_n)$ où n est le nombre d'arguments de f, $x_1, \ldots, x_n$ sont des variables fraîches et $p, o_1, \ldots, o_n$ sont des nouveaux nœuds. ◇

Exemple 4.2.4 Considérons le cGRS $SP_2 = \langle \Sigma, \mathcal{R}_2 \rangle$ de l'Exemple 3.1.12, en particulier les trois règles de réécriture définissant l'opération `g` :

```
(R3) l1:g(l2:a,l3:x) -> r1:s(l2:a)
(R4) l1:g(l2:x,l3:a) -> r1:s(l2:x)
(R5) l1:g(l2:s(l3:x),l4:s(l5:y)) -> r1:s(r2:g(l3:x,l4:s(l5:y)))
```

Un arbre de définition $\mathcal{T}^1_{\mathtt{g}}$ de l'opération `g` est donné par :

$\mathcal{T}^1_{\mathtt{g}} = branch($`k19 :g(k20 :x8,k21 :x9)`,
`k20`,
$rule($`k19 :g(k22 :a,k21 :x9)` $\rightarrow$ `k23 :s(k22 :a)`$)$,
$branch($`k19 :g(k24 :s(k25 :x10),k21 :x9)`,
`k21`,
$rule($`k19 :g(k24 :s(k25 :x10),k26 :s(k27 :x11))` $\rightarrow$
`k28 :s(k29 :g(k25 :x10,k26 :s(k27 :x11)))`$)))$

$\mathcal{T}^1_{\mathtt{g}}$ est représenté dans la Figure 4.4. On remarque que la règle `R4` n'est pas représentée par une feuille de $\mathcal{T}^1_{\mathtt{g}}$. En fait, il est impossible de construire un seul arbre de définition contenant toutes les règles définissant `g`. □

Nous définissons ci-dessous les systèmes de réécriture inductivement séquentiels comme étant ceux tels que toutes les règles définissant une opération puissent être rangées dans un seul arbre de définition.

Définition 4.2.5 (ISGRS)
Un cGRS $SP = \langle \Sigma, \mathcal{R} \rangle$ est un *système de réécriture de graphes inductivement séquentiel (ISGRS)* ssi pour toute opération définie f, il existe un arbre de définition $\mathcal{T}$ de f tel que pour toute règle $l \rightarrow r$ de $\mathcal{R}$, où l est de la forme $f(g_1, \ldots, g_n)$, il existe une feuille $rule(R)$ de $\mathcal{T}$ telle que R soit une variante de $l \rightarrow r$. ◇

Exemple 4.2.6 Nous remarquons dans la Figure 4.4 que l'AGRS $SP_1 = \langle \Sigma, \mathcal{R}_1 \rangle$ de l'Exemple 3.1.12 est un ISGRS. Par contre, le WAGRS $SP_2 = \langle \Sigma, \mathcal{R}_2 \rangle$ n'en est pas un. □

Dans l'exemple précédent, nous constatons qu'un WAGRS n'est pas inductivement séquentiel en général. En fait, seuls les AGRS peuvent être inductivement séquentiels : un ISGRS est forcément un AGRS. Mais tous les AGRS ne sont pas inductivement séquentiels :

Exemple 4.2.7 Considérons le cGRS suivant :

```
(U1) l1:A(l2:B,l3:C,l4:X) -> r1:O
(U2) l1:A(l2:X,l3:B,l4:C) -> r1:O
(U2) l1:A(l2:C,l3:X,l4:B) -> r1:O
```

Comme aucune de ces règles n'est superposable à une autre, ce cGRS est un AGRS. Pourtant, il n'est pas possible de construire un arbre de définition contenant toutes les règles de réécriture définissant l'opération `A`. Autrement dit, cet AGRS n'est pas un ISGRS. □

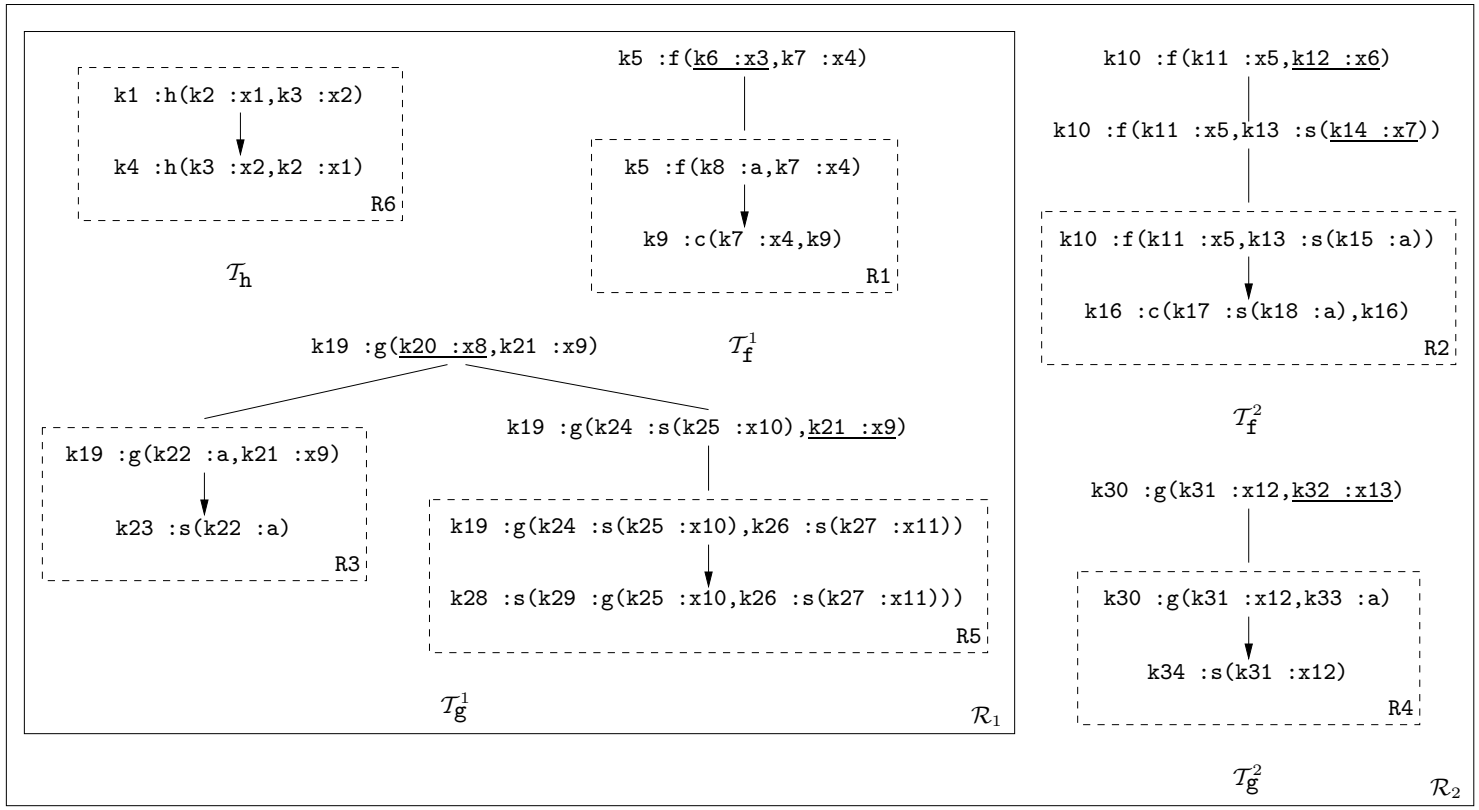

k1 :h(k2 :x1,k3 :x2)
k4 :h(k3 :x2,k2 :x1)
R6
$\mathcal{T}_h$
k5 :f(k6 :x3,k7 :x4)
k5 :f(k8 :a,k7 :x4)
k9 :c(k7 :x4,k9)
R1
$\mathcal{T}_f^1$
k19 :g(k20 :x8,k21 :x9)
k19 :g(k22 :a,k21 :x9)
k23 :s(k22 :a)
R3
k19 :g(k24 :s(k25 :x10),k21 :x9)
k19 :g(k24 :s(k25 :x10),k26 :s(k27 :x11))
k28 :s(k29 :g(k25 :x10,k26 :s(k27 :x11)))
R5
$\mathcal{T}_g^1$
$\mathcal{R}_1$
k10 :f(k11 :x5,k12 :x6)
k10 :f(k11 :x5,k13 :s(k14 :x7))
k10 :f(k11 :x5,k13 :s(k15 :a))
k16 :c(k17 :s(k18 :a),k16)
R2
$\mathcal{T}_f^2$
k30 :g(k31 :x12,k32 :x13)
k30 :g(k31 :x12,k33 :a)
k34 :s(k31 :x12)
R4
$\mathcal{T}_g^2$
$\mathcal{R}_2$

FIG. 4.4:

Nous sommes prêts pour définir la stratégie de réécriture Φ. Cette stratégie est une fonction partielle qui opère sur les termes-graphes admissibles dans le cadre d'un ISGRS. $\Phi(g)$ retourne (quand c'est possible) un couple (p, R) tel que p soit un nœud de g, R soit une règle de réécriture et g se réécrive au nœud p en utilisant la règle R. Φ utilise une fonction auxilliaire φ qui prend deux arguments : un terme-graphe de racine fonctionnelle et un pdt de cette fonction.

Définition 4.2.8 (Stratégie Φ)
Soit SP un ISGRS et g un terme-graphe admissible. Φ est la fonction partielle telle que $\Phi(g) = \varphi(g_{|p}, \mathcal{T})$, où p désigne le plus haut nœud fonctionnel à gauche de g, p est étiqueté par une opération définie f et $\mathcal{T}$ est un arbre de définition de f.
Soit g un terme-graphe admissible de racine fonctionnelle et $\mathcal{T}$ un pdt tel que $pattern(\mathcal{T})$ filtre g à la racine. On définit ci-dessous la fonction partielle φ :

$$\varphi(g,\mathcal{T}) = \begin{cases} (p,R) & \text{si } \mathcal{T} = rule(\pi \to r), p = \mathcal{R}\text{oot}_g \text{ et} \\ & R \text{ est une variante de } \pi \to r\,; \\ \varphi(g,\mathcal{T}_i) & \text{si } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ et} \\ & pattern(\mathcal{T}_i) \text{ filtre } g \text{ pour un } i \in 1..k\,; \\ (p,R) & \text{si } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \pi \text{ filtre } g \text{ par l'homomorphisme } h : \pi \to g, \\ & h(o) \text{ est étiqueté par l'opération } f \text{ (dans } g), \\ & \mathcal{T}' \text{ est un arbre de définition de } f \text{ et} \\ & \varphi(g_{|h(o)}, \mathcal{T}') = (p,R). \end{cases}$$

◇

Exemple 4.2.9 Considérons le terme-graphe g de la Figure 4.5 :
g = `n1 :c(n2 :g(n3 :s(n4 :g(n5 :a,n6 :h(n5,n5))),n4),n7 :c(n4,n1))`. On calcule $\Phi(g)$ dans le cas de l'ISGRS $SP_1 = \langle \Sigma, \mathcal{R}_1 \rangle$ de l'Exemple 3.1.12 (cf. Figures 3.2 et 4.4). Le

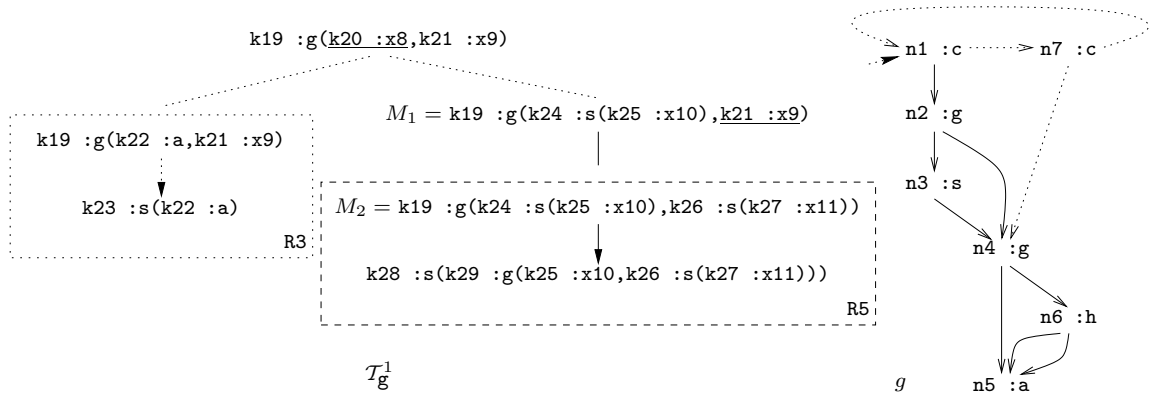


FIG. 4.5:

plus haut nœud fonctionnel à gauche de g est `n2`. Aussi, $\Phi(g) = \varphi(g_{|\mathtt{n2}}, \mathcal{T}_g^1)$.
Dans $\mathcal{T}_g^1$, le motif M_1 = `k19 :g(k24 :s(k25 :x10),k21 :x9)` filtre le terme-graphe $g_{|\mathtt{n2}}$, mais le motif M_2 = `k19 :g(k24 :s(k25 :x10),k26 :s(k27 :x11))` ne filtre pas $g_{|\mathtt{n2}}$. En effet, le nœud `k26` est étiqueté par `s` alors que le nœud `n4` est étiqueté par `g`. Aussi, $\varphi(g_{|\mathtt{n2}}, \mathcal{T}_g^1)$ fait un appel récursif à $\varphi(g_{|\mathtt{n4}}, \mathcal{T}_g^1)$ (cf. Figure 4.6). On constate que le motif M_3 =

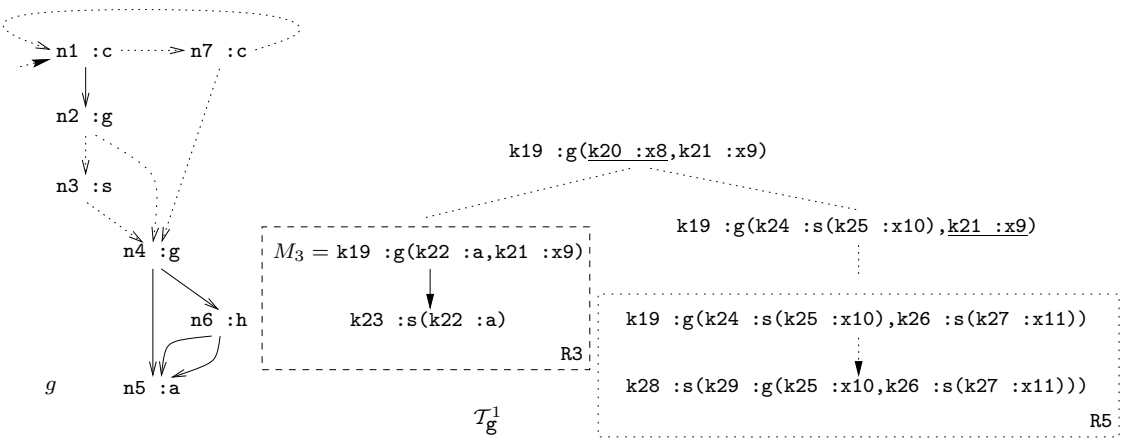


FIG. 4.6:

`k19 :g(k22 :a,k21 :x9)` filtre le terme-graphe $g_{|\mathtt{n4}}$. Comme ce motif est le membre gauche de la règle de réécriture `R3`, on conclut que $\varphi(g_{|\mathtt{n4}}, \mathcal{T}_g^1) = (\mathtt{n4}, \mathtt{R3})$, et donc, $\Phi(g) = (\mathtt{n4}, \mathtt{R3})$. □

Remarque 4.2.10 $\Phi(g)$ utilise le plus haut nœud fonctionnel *à gauche* de g. En fait, n'importe quel plus haut nœud fonctionnel de g pourrait être utilisé. Mais comme notre stratégie est déterministe, nous choisissons celui de gauche entre tous. ◇

Nous établissons maintenant quelques propriétés de la stratégie Φ.

Proposition 4.2.11 Soient SP un ISGRS et g un terme-graphe admissible.

- Si $\Phi(g) = (p, R)$, alors $g_{|p}$ est un plus haut rédex nécessaire de g et g se réécrit au nœud p avec la règle R.
- Si $\Phi(g)$ n'est pas définie, alors g n'admet pas de forme normale constructeur.

◇

La Proposition 4.2.11 est une conséquence immédiate du lemme suivant, qui s'inspire de [AEH94b, Theorem 3] :

Lemme 4.2.12 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un ISGRS, g un terme-graphe admissible de racine fonctionnelle f et $\mathcal{T}$ un arbre de définition de f.

- Si $\varphi(g, \mathcal{T}) = (p, R)$, alors
  1. dans toute dérivation de réécriture de g vers un terme-graphe de racine constructeur, un descendant de $g_{|p}$ est réécrit à la racine, en un ou plusieurs pas, en un terme-graphe de racine constructeur ;
  2. $g_{|p}$ est un rédex de g filtré par le membre gauche de la règle R ;
  3. $g_{|p}$ est un plus haut rédex de g.
- Si $\varphi(g, \mathcal{T})$ n'est pas définie, alors g ne peut pas se réécrire en un terme-graphe de racine constructeur.

◇

Preuve : Dans la suite, $\sqsubset$ dénote l'ordre nœthérien défini par $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ ssi le graphe g_1 a moins d'occurrence d'opération définie que le graphe g_2, ou alors $g_1 = g_2$ et $\mathcal{T}_1$ est un sous-arbre propre de $\mathcal{T}_2$. Cette preuve se fait par induction nœthérienne sur $\sqsubset$. Nous étudions les différents cas de la définition de φ :

Cas de base : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle, $\mathcal{T} = rule(\pi \to r)$ et $\pi \to r$ est une variante d'une règle de $\mathcal{R}$ tels que π filtre g à la racine. Par définition de φ, $\varphi(g, \mathcal{T})$ est défini et $\varphi(g, \mathcal{T}) = (\mathcal{R}\text{oot}_g, R)$ où R est une variante de $\pi \to r$. (1) Comme g est de racine fonctionnelle, toute dérivation de réécriture de g vers un terme-graphe de racine constructeur contient un pas de réécriture au nœud $\mathcal{R}\text{oot}_g$. Donc dans toute dérivation de réécriture de g vers un terme-graphe de racine constructeur, un descendant de $g_{|\mathcal{R}\text{oot}_g}$ doit être réécrit à la racine, en un ou plusieurs pas, en un terme-graphe de racine constructeur. (2) Par hypothèse, π filtre g à la racine, et comme R est une variante de $\pi \to r$, $g_{|\mathcal{R}\text{oot}_g}$ est un rédex (de g) filtré par le membre gauche de la règle R ; (3) Il est clair que $g_{|\mathcal{R}\text{oot}_g} = g$ est un plus haut rédex de g.

Cas d'induction : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle, $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, o est un nœud variable de π, o est de sorte ζ, $c_1, \ldots, c_k$ ($k > 0$) sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt de motif $\pi[o \leftarrow p : c_j(o_1 : x_1, \ldots, o_n : x_n)]$ tel que n soit le nombre d'arguments de c_j, $x_1, \ldots, x_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds. Nous supposons que $pattern(\mathcal{T})$ filtre g à la racine donc il existe un homomorphisme $h : \pi \to g$. Trois cas sont à étudier, selon l'étiquette du nœud $h(o)$ dans g.

Cas 1 : $h(o)$ est un nœud constructeur de g :
Deux cas sont possibles. Si il existe $i \in 1..k$ tel que $pattern(\mathcal{T}_i)$ filtre g à la racine, alors par définition de φ, $\varphi(g, \mathcal{T}) = \varphi(g, \mathcal{T}_i)$. Toutes les clauses du lemme sont vérifiées par $\varphi(g, \mathcal{T}_i)$, d'après l'hypothèse d'induction. Donc elles le sont aussi par $\varphi(g, \mathcal{T})$. Si il n'existe pas de $i \in 1..k$ tel que $pattern(\mathcal{T}_i)$ filtre g à la racine, alors l'opération $\mathcal{L}_g(\mathcal{R}\text{oot}_g)$ est incomplètement définie par le système de réécriture $\mathcal{R}$. Donc g ne peut pas être réécrit en un terme-graphe de racine constructeur et $\varphi(g, \mathcal{T})$ n'est pas défini.

Cas 2 : $h(o)$ est un nœud fonctionnel de g :
Posons $\mathcal{L}_g(h(o)) = f$. Soit $\mathcal{T}'$ l'arbre de définition de f. De même que dans le Cas 1, il existe deux possibilités selon que $\varphi(g_{|h(o)}, \mathcal{T}')$ est défini ou pas.

Cas 2.1 : Supposons que $\varphi(g_{|h(o)}, \mathcal{T}') = (p, R)$, où p est un nœud de $g_{|h(o)}$ et R est une variante d'une règle de $\mathcal{R}$. Alors $\varphi(g, \mathcal{T}) = (p, R)$.
(1) Par hypothèse, $pattern(\mathcal{T})$ filtre g à la racine. Par définition d'un ISGRS, toutes les règles $l \to r$ de $\mathcal{R}$ dont le membre gauche filtre g à la racine sont représentées par une feuille $rule(l' \to r')$ de $\mathcal{T}$, où $l' \to r'$ est une variante de $l \to r$. Par construction d'un arbre de définition, si $l \to r$ est représentée par une feuille de $\mathcal{T}$, alors $pattern(\mathcal{T})$ filtre l à la racine, i.e., il existe un homomorphisme $h' : \pi \to l$. Par hypothèse, $\mathcal{T}$ est un pdt dont la racine est étiquetée par le symbole *branch* et pas par le symbole *rule*. Donc $pattern(\mathcal{T})$ filtre l sans être un renommage de l. Ceci implique que $h'(o)$ est un nœud constructeur de l. Pourtant, dans le cas que nous considérons, $h(o)$ est un nœud fonctionnel de g. Donc il n'existe pas de règle de $\mathcal{R}$ dont le membre gauche filtre g à la racine, i.e., g n'est pas un rédex. De plus, dans toute dérivation de réécriture de g qui contient un pas de réécriture à la racine, un descendant de $g_{|h(o)}$ doit être réécrit en un terme-graphe de racine constructeur. Comme g est de racine fonctionnelle, dans toute dérivation de g vers un terme-graphe de racine constructeur, un descendant de g est réécrit à la racine, et donc, un descendant de $g_{|h(o)}$ est réécrit en un terme-graphe de racine constructeur. Par hypothèse d'induction, dans toute dérivation de réécriture de $g_{|h(o)}$ vers un terme-graphe de racine constructeur, un descendant de $(g_{|h(o)})_{|p}$ est réécrit en un terme-graphe de racine constructeur. Comme $(g_{|h(o)})_{|p} = g_{|p}$, on conclut que dans toute dérivation de réécriture de g vers un terme-graphe de racine constructeur, un descendant de $g_{|p}$ est réécrit en un terme-graphe de racine constructeur.
(2) Par hypothèse d'induction, $g_{|p}$ est un rédex de $g_{|h(o)}$ filtré par le membre gauche de R ; donc $g_{|p}$ est un rédex de g filtré par le membre gauche de R.
(3) Par hypothèse, π filtre g à la racine, i.e., il existe un homomorphisme $h : \pi \to g$. Par définition d'un arbre de définition, π est un motif et o est un nœud de π. Ces conditions impliquent qu'il existe

un chemin allant de $\mathcal{R}oot_g$ vers $h(o)$ dans g et que tous les nœuds de ce chemin sont constructeurs, sauf $\mathcal{R}oot_g$ et $h(o)$. Dans les cGRS, les rédex sont toujours de racine fonctionnelle. Nous venons de montrer que g n'était pas un rédex. Donc il n'existe pas de rédex sur le chemin allant de $\mathcal{R}oot_g$ vers $h(o)$. Comme $g_{|p}$ est un plus haut rédex de $g_{|h(o)}$ par hypothèse d'induction, $g_{|p}$ est aussi un plus haut rédex de g.

Cas 2.2 : Supposons que $\varphi(g_{|h(o)}, \mathcal{T}')$ ne soit pas défini. Alors $\varphi(g, \mathcal{T})$ n'est pas défini, lui non plus. De plus, par hypothèse d'induction, $g_{|h(o)}$ ne peut pas être réécrit en un terme-graphe de racine constructeur. Pourtant, comme dans le cas 2.1, g n'est pas un rédex et dans toute dérivation de g vers un terme-graphe de racine constructeur, un descendant de $g_{|h(o)}$ doit être réécrit en un terme-graphe de racine constructeur. Donc g ne peut pas être réécrit en un terme-graphe de racine constructeur.

Cas 3 : $h(o)$ est un nœud variable de g :

Dans ce cas, $\varphi(g, \mathcal{T})$ n'est pas défini. Nous affirmons que g n'a pas de forme normale constructeur. En effet, π filtre g à la racine par hypothèse. Toute règle dont le membre gauche filtre g à la racine est représentée par une feuille de $\mathcal{T}$. Si $l \to r$ est une règle représentée par une feuille de $\mathcal{T}$, alors il existe un homomorphisme $h' : \pi \to l$. De plus, π n'est pas égal à l à un renommage près. Donc par construction d'un arbre de définition, $h'(o)$ est un nœud constructeur de l. Pourtant, dans le cas que nous considérons, $h(o)$ est un nœud variable de g. Par conséquent, g n'est pas un rédex. De plus, dans toute dérivation de réécriture de g vers un terme-graphe de racine constructeur, un descendant de $g_{|h(o)}$ doit être réécrit en un terme-graphe de racine constructeur, ce qui est impossible puisque $g_{|h(o)}$ est une variable. Donc g n'admet pas de forme normale constructeur. □

Dans le cadre des systèmes de réécriture orthogonaux de termes, la réécriture des rédex nécessaires d'un terme du premier ordre permet de calculer sa forme normale. Nous obtenons un résultat similaire dans le cadre des ISGRS.

Theorème 4.2.13 Soit SP un ISGRS. La stratégie Φ est $\mathcal{C}$-hyper-normalisante, donc $\mathcal{C}$-normalisante. ◇

Pour prouver le Théorème 4.2.13, nous avons besoin du lemme suivant. Avec la Proposition 4.2.11, nous avons vu que toute dérivation de réécriture d'un graphe g vers sa forme normale constructeur devait exécuter un pas en utilisant le nœud p et la règle R calculés par $\Phi(g)$. De plus, nous savons que g se réécrit au nœud p avec la règle R. Le lemme suivant stipule qu'étant donnée une dérivation $D = g \xrightarrow{*} c$, si on commence par exécuter le pas calculé par $\Phi(g)$, on obtient une nouvelle dérivation de longueur inférieure (ou égale) à celle de D.

Lemme 4.2.14 Soient SP un ISGRS, g un terme-graphe admissible $\mathcal{C}$-normalisable non constructeur et c un terme-graphe constructeur tels qu'il existe une dérivation de réécriture $g \xrightarrow{*} c$ de longueur n. Alors il existe un terme-graphe admissible g' et un terme-graphe constructeur c' tels que $g \to_\Phi g'$, $g' \xrightarrow{*} c'$, $c' \sim c$ et si n' est la longueur de $g' \xrightarrow{*} c'$, alors $n' < n$. ◇

Preuve : On donne un schéma de cette preuve à la Figure 4.7. Par hypothèse, il existe une dérivation

$$
\begin{array}{ccccccccccccc}
g & \to_{[u_1,R_1]} & a_1 & \to_{[u_2,R_2]} & \cdots & \to_{[u_{k-1},R_{k-1}]} & a_{k-1} & \to_{[p,R]} & a_k & \to_{[u_{k+1},R_{k+1}]} & \cdots & \to_{[u_n,R_n]} & c \\
 & & \downarrow [p,R] & & & & \downarrow [p,R] & & & & & & \\
\downarrow [p,R] & A & b_1 & & B & & b_{k-1} & C & \sim & & D & & \sim \\
 & & \sim & & & & \sim & & & & & & \\
g' & \xrightarrow{\varepsilon}_{[u_1,R_1]} & a'_1 & \xrightarrow{\varepsilon}_{[u_2,R_2]} & \cdots & \xrightarrow{\varepsilon}_{[u_{k-1},R_{k-1}]} & a'_{k-1} & = & a'_{k-1} & \to_{[u_{k+1},R_{k+1}]} & \cdots & \to_{[u_n,R_n]} & c'
\end{array}
$$

FIG. 4.7:

de réécriture D allant de g vers le graphe constructeur c. Donc d'après la Proposition 4.2.11, $\Phi(g)$ est

défini et si $\Phi(g) = (p, R)$, alors (1) g se réécrit au nœud p avec la règle R et (2) dans toute dérivation de g vers c, un descendant de $g_{|p}$ est réécrit en un terme-graphe de racine constructeur. D'après le point (1), il existe un graphe g' tel que $g \to_{[p,R]} g'$ et d'après le point (2), la dérivation D est de la forme $D = g \to_{[u_1,R_1]} a_1 \to_{[u_2,R_2]} \cdots \to_{[u_{k-1},R_{k-1}]} \underline{a_{k-1} \to_{[p,R]} a_k} \to_{[u_{k+1},R_{k+1}]} \cdots \to_{[u_n,R_n]} c$, avec $u_i \neq p$ pour tout $i \in 1..(k-1)$ et pour tout $i \in (k+1)..n$.

Premièrement, d'après le Lemme 3.4.9, puisque $g \to_{[u_1,R_1]} a_1$ et $g \to_{[p,R]} g'$, il existe deux graphes b_1 et a'_1 tels que $a_1 \overset{\varepsilon}{\to}_{[p,R]} b_1$, $g' \overset{\varepsilon}{\to}_{[u_1,R_1]} a'_1$ et $b_1 \sim a'_1$. De plus, $g_{|p}$ est un rédex *nécessaire* de g et $u_1 \neq p$, donc p est un nœud de a_1. Par conséquent, $a_1 \neq b_1$ et $a_1 \to_{[p,R]} b_1$ (cf. A dans la Figure 4.7).

Plus généralement, comme $g \to_{[u_1,R_1]} a_1 \ldots \to_{[u_{k-1},R_{k-1}]} a_{k-1}$ et $g \to_{[p,R]} g'$, on déduit qu'il existe deux graphes b_{k-1} et a'_{k-1} tels que $a_{k-1} \overset{\varepsilon}{\to}_{[p,R]} b_{k-1}$, $g' \overset{\varepsilon}{\to}_{[u_1,R_1]} \cdots \overset{\varepsilon}{\to}_{[u_{k-1},R_{k-1}]} a'_{k-1}$ et $b_{k-1} \sim a'_{k-1}$, d'après la Proposition 3.4.10. De plus, $g_{|p}$ est un rédex *nécessaire* de g et $p \notin \{u_1, \ldots, u_{k-1}\}$, donc p est un nœud de a_{k-1}. Par conséquent, $a_{k-1} \neq b_{k-1}$ et $a_{k-1} \to_{[p,R]} b_{k-1}$ (cf. B dans la Figure 4.7). La longueur de la dérivation $g \to_{[u_1,R_1]} \cdots \to_{[u_{k-1},R_{k-1}]} a_{k-1}$ est exactement $k-1$ alors que la longueur de la dérivation $g' \overset{\varepsilon}{\to}_{[u_1,R_1]} \cdots \overset{\varepsilon}{\to}_{[u_{k-1},R_{k-1}]} a'_{k-1}$ est inférieure ou égale à $k-1$.

Deuxièmement, on constate à l'aide de la partie C dans la Figure 4.7 que $a_{k-1} \to_{[p,R]} a_k$ et $a_{k-1} \to_{[p,R]} b_{k-1}$, donc $a_k \sim b_{k-1}$. Comme $a'_{k-1} \sim b_{k-1}$ et $b_{k-1} \sim a_k$, on infère que $a'_{k-1} \sim a_k$. La longueur de la dérivation $g \to_{[u_1,R_1]} \cdots \to_{[u_{k-1},R_{k-1}]} a_{k-1} \to_{[p,R]} b_{k-1}$ est exactement k alors que la longueur de la dérivation $g' \overset{\varepsilon}{\to}_{[u_1,R_1]} \cdots \overset{\varepsilon}{\to}_{[u_{k-1},R_{k-1}]} a'_{k-1}$ est inférieure ou égale à $k-1$.

Troisièmement, nous avons $a_k \to_{[u_{k+1},R_{k+1}]} \cdots \to_{[u_n,R_n]} c$ par hypothèse. Comme $a'_{k-1} \sim a_k$, il est clair qu'il existe un graphe constructeur c' tel que $c' \sim c$ et $a'_{k-1} \to_{[u_{k+1},R_{k+1}]} \cdots \to_{[u_n,R_n]} c'$ (cf. D dans la Figure 4.7). La longueur de la dérivation
$D = g \to_{[u_1,R_1]} a_1 \to_{[u_2,R_2]} \cdots \to_{[u_{k-1},R_{k-1}]} a_{k-1} \to_{[p,R]} a_k \to_{[u_{k+1},R_{k+1}]} \cdots \to_{[u_n,R_n]} c$
est exactement n alors que la longueur de la dérivation
$g' \overset{\varepsilon}{\to}_{[u_1,R_1]} \cdots \overset{\varepsilon}{\to}_{[u_{k-1},R_{k-1}]} a'_{k-1} \to_{[u_{k+1},R_{k+1}]} \cdots \to_{[u_n,R_n]} c'$
est inférieure ou égale à $n-1$. □

Preuve du Théorème 4.2.13 : On donne un schéma de cette preuve à la Figure 4.8. Soient g_0 un

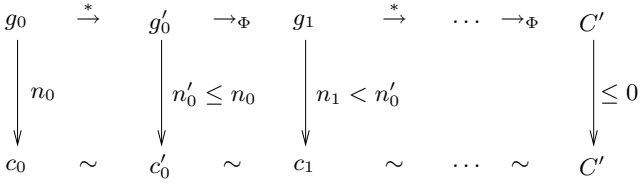


FIG. 4.8:

terme-graphe admissible et c_0 un terme-graphe constructeur tel qu'il existe une dérivation $g_0 \overset{*}{\to} c_0$ de longueur n_0. Nous montrons ce théorème par induction sur n_0. Le cas $n_0 = 0$ est évident. Soit $n_0 > 0$ et supposons le théorème vrai pour toutes les longueurs $p < n_0$. Soit D une dérivation commençant avec g contenant des Φ-pas infiniment souvent, i.e., D est de la forme $g_0 \overset{*}{\to} g'_0 \to_\Phi g_1 \overset{*}{\to} g'_1 \to_\Phi g_2 \overset{*}{\to} \ldots$ Par confluence (Théorème 3.3.4), comme (1) $g_0 \overset{*}{\to} g'_0$, (2) $g_0 \overset{*}{\to} c_0$ est de longueur n_0 et (3) c_0 est un graphe constructeur, il existe un graphe constructeur $c'_0 \sim c_0$ et une dérivation $g'_0 \overset{*}{\to} c'_0$ de longueur $n'_0 \leq n_0$. Comme (1) $g'_0 \overset{*}{\to} c'_0$ est de longueur n'_0, (2) $g'_0 \to_\Phi g_1$, et (3) c'_0 est un graphe constructeur, nous déduisons du Lemme 4.2.14 l'existence d'un graphe constructeur $c_1 \sim c'_0$ et d'une dérivation $g_1 \overset{*}{\to} c_1$ de longueur $n_1 < n'_0$. Soit D' la sous-dérivation de D commençant avec g_1. La dérivation $g_1 \overset{*}{\to} c_1$ est de longueur $n_1 < n_0$ et D' contient des Φ-pas infiniment souvent (puisque D contient des Φ-pas infiniment souvent). Donc par hypothèse d'induction, D' termine, avec un graphe constructeur $C' \sim c_1$. Par conséquent, D termine, avec un graphe constructeur $C' \sim c_0$. □

La réécriture de graphes ne duplique pas les données. Aussi, le nombre de pas nécessaires pour calculer une forme normale constructeur peut être optimisé. Nous obtenons le théorème d'optimalité suivant :

Theorème 4.2.15 Soient SP un ISGRS, g un terme-graphe admissible $\mathcal{C}$-normalisable et c un terme-graphe constructeur tel qu'il existe une dérivation de réécriture $g \xrightarrow{*} c$. Alors il existe un terme-graphe constructeur c' tel que $c' \sim c$ et la longueur de la Φ-dérivation $g \xrightarrow{*}_{\Phi} c'$ soit inférieure (ou égale) à la longueur de la dérivation $g \xrightarrow{*} c$. ◇

Preuve : Par induction sur la longueur de la dérivation $g \xrightarrow{*} c$. Le cas $n = 0$ est immédiat. Soit $n > 0$ et supposons le théorème vrai pour toute longueur $p < n$. D'après le Lemme 4.2.14, il existe un terme-graphe admissible g' et un terme-graphe constructeur $c' \sim c$ tels que $g \rightarrow_{\Phi} g'$, $g' \xrightarrow{*} c'$ et si n' est la longueur ce $g' \xrightarrow{*} c'$ alors $n' < n$. Par hypothèse d'induction, il existe un terme-graphe constructeur $c'' \sim c'$ et une Φ-dérivation $g' \xrightarrow{*}_{\Phi} c''$ de longueur $n'' \leq n'$. La longueur de la dérivation $g \rightarrow_{\Phi} g' \xrightarrow{*}_{\Phi} c''$ est $n'' + 1 \leq n' + 1 \leq n$ et $c'' \sim c$. □

4.3 Stratégie parallèle $\bar{\Phi}$

Dans la section précédente, nous avons étendu la stratégie Φ définie dans [Ant92] aux systèmes de réécriture de graphes inductivement séquentiels. Cette stratégie ne peut plus être utilisée dans le cadre général des WAGRS. En effet, Φ calcule des rédex nécessaires et cette notion n'est pas pertinente dans le cadre des WAGRS (cf. Exemple 4.1.6). De plus, Φ utilise des arbres de définition contenant toutes les règles de réécriture définissant une opération et de tels arbres ne peuvent pas être construits en général dans le cadre d'un WAGRS (cf. Exemple 4.2.7).

Dans cette section, nous étendons la stratégie parallèle $\bar{\Phi}$ définie dans [Ant92] aux WAGRS. $\bar{\Phi}$ ne calcule pas des rédex nécessaires, comme Φ, mais des ensembles suffisants de rédex. Tous les rédex calculés par $\bar{\Phi}$ doivent être réécrits si on veut être sûr de calculer la forme normale constructeur d'un graphe. Ces pas de réécriture peuvent être vus de manière atomique en utilisant un pas de réécriture parallèle de graphes. Nous donnons une définition de la relation de réécriture parallèle $\Vdash\!\!\rightarrow$ dans le paragraphe suivant. Nous étudierons la stratégie $\bar{\Phi}$ proprement dite dans le Paragraphe 4.3.2.

4.3.1 Réécriture parallèle

La réécriture parallèle est courante dans le cadre des systèmes de réécriture de termes du premier ordre. Dans [PvE93, Chap. 14], une réécriture parallèle de graphes a été étudiée du point de vue de l'implantation : des annotations peuvent être ajoutées aux membres droits des règles ce qui permet d'indiquer quels rédex doivent être évalués en parallèle. Dans cette section, nous nous intéressons davantage à une notion générale de réécriture parallèle permettant de réécrire plusieurs rédex arbitraires d'un terme-graphe admissible en un seul pas.

La réécriture parallèle d'un graphe nécessite plus de précautions que la réécriture parallèle de termes [Hue80, O'D77] du fait du partage des sous-graphes. En effet, on ne peut réécrire en parallèle les rédex d'un terme que si ceux-ci sont disjoints. Mais on ne peut plus faire cette hypothèse dans le cadre des graphes :

Exemple 4.3.1 Soient t = `A(B(C),D(B(C)))` un terme et
g = `p1 :A(p2 :B(p3 :C),p4 :D(p2))` un terme-graphe admissible
"équivalent" (bisimilaire) (cf. Figure 4.9). Considérons les deux règles suivantes :

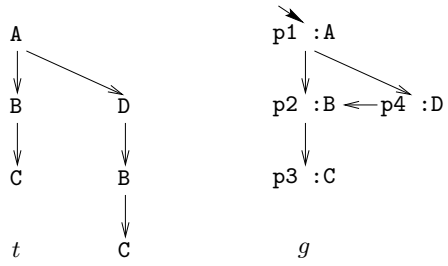


FIG. 4.9:

```
(V1) l1:B(l2:X) -> r1:E(l2:X)
(V2) l1:D(l2:X) -> l2:X
```

Les sous-termes `B(C)` et `D(B(C))` sont des rédex disjoints du terme t. Mais les sous-graphes correspondants `p2 :B(p3 :C)` et `p4 :D(p2 :B(p3 :C))` ne sont pas disjoints dans g. □

Nous discutons ci-dessous deux définitions possibles de la réécriture parallèle de graphes. La seconde sera utilisée dans la suite.

Définition 4.3.2 (Réécriture parallèle "naïve")
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, g_1 un terme-graphe admissible, g_2 un graphe, $R_1, \ldots, R_n$ n règles de réécriture de $\mathcal{R}$ et $p_1, \ldots, p_n$ n nœuds distincts de g_1. On dit que g_1 *se réécrit "naïvement" en parallèle en* g_2 *aux nœuds* $p_1, \ldots, p_n$ *avec les règles* $R_1, \ldots, R_n$, noté $g_1 \Vdash\!\!\rightarrow_{[p_1,R_1]\ldots[p_n,R_n]} g_2$ ssi :

1. Soient $l_i \to r_i$ une variante de R_i et $h_i : l_i \to g_{1|p_i}$ un filtre de l_i sur g_1 au nœud p_i pour tout $i \in 1..n$.
2. Soit $H = g \oplus h_1[r_1] \oplus \ldots \oplus h_n[r_n]$.
3. Soit ρ la redirection de pointeurs (cf. Définition 2.2.5) telle que $\rho(p_i) = \mathcal{R}\text{oot}_{h_i[r_i]}$ pour tout $i \in 1..n$ et $\rho(p) = p$ pour tout p tel que $p \neq p_1, \ldots, p \neq p_n$.
4. Soient $H' = \rho(H)$ et $r = \rho(\mathcal{R}\text{oot}_g)$.
5. $g_2 = H'_{|r}$.

◇

La définition précédente utilise des redirections de pointeurs en parallèle sans tenir compte des positions relatives des rédex.

Exemple 4.3.3 Considérons le graphe g et les règles `V1` et `V2` de l'Exemple 4.3.1.
Le membre gauche de `V1` $= l_1 \to r_1$ filtre g au nœud `p2` avec l'homomorphisme $h_1 : l_1 \to g_{|\texttt{p2}}$ et on a $h_1[r_1]$ = `r1 :E(p3 :C)`. D'autre part, le membre gauche de `V2` $= l_2 \to r_2$ filtre g au nœud `p4` avec l'homomorphisme $h_2 : l_2 \to g_{|\texttt{p4}}$ et on a $h_2[r_2]$ = `p2 :B(p3 :C)`. Soit $H = g \oplus h_1[r_1] \oplus h_2[r_2]$ = `p1 :A(p2 :B(p3 :C),p4 :D(p2)) + r1 :E(p3) + p2` (cf. Figure 4.10). Soit $\rho = \{\texttt{p2} \mapsto \texttt{r1}, \texttt{p4} \mapsto \texttt{p2}\}$. Le lecteur peut vérifier que $\rho(H) = H_1$ = `p1 :A(r1 :E(p3 :C),p2 :B(p3)) + r1 + r1 + p2 + p4 :D(r1)` (cf. Figure 4.10) et $\rho(\mathcal{R}\text{oot}_g)$ = `p1`. Donc d'après la Définition 4.3.2, $g \Vdash\!\!\rightarrow_{[\texttt{p2},\texttt{V1}][\texttt{p4},\texttt{V2}]} g_1$ où g_1 = `p1 :A(r1 :E(p3 :C),p2 :B(p3))`. □

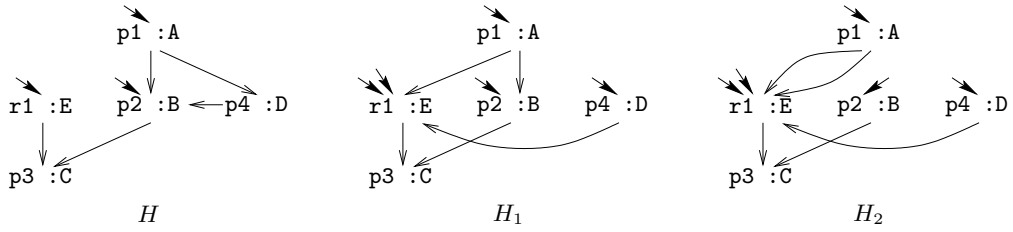


FIG. 4.10:

En poursuivant l'Exemple 4.3.1 (Figure 4.9) présenté au début de ce paragraphe, le lecteur peut vérifier que la réécriture parallèle du terme t et la réécriture parallèle du graphe g conduisent à deux graphes qui sont bisimilaires. En ce sens, notre pas de réécriture parallèle de graphes est cohérent avec le pas de la réécriture parallèle de termes. Pourtant, la relation $\Vdash\!\!\rightarrow$ n'est pas satisfaisante :

1. D'une part, un rédex d'un graphe qui est réécrit avec un $\Vdash\!\!\rightarrow$-pas de réécriture peut continuer d'être un rédex dans le graphe résultant. C'est le cas, par exemple, du rédex $g_{|\texttt{p2}} = \texttt{p2 :B(p3 :C)}$ qui reste un sous-graphe de g_1 dans l'Exemple 4.3.3.
2. D'autre part, un $\Vdash\!\!\rightarrow$-pas de réécriture ne peut pas toujours être simulé avec une dérivation de réécriture séquentielle. En effet, dans l'Exemple 4.3.3, on a $g \Vdash\!\!\rightarrow g_1$ mais $g \overset{*}{\not\rightarrow} g_1$.

Ce genre de comportement se produit parce que nous utilisons des redirections de pointeurs de manière aveugle dans la définition de $\Vdash\!\!\rightarrow$.

Nous définissons ci-dessous un second pas de réécriture parallèle, noté $\Vdash\!\!\!\!\rightarrow$, qui utilise une nouvelle redirection de pointeurs calculée de manière à imposer un ordre dans l'application des différentes redirections de pointeurs.

Définition 4.3.4 (Réécriture parallèle)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, g_1 un terme-graphe admissible, g_2 un graphe, $R_1, \ldots, R_n$ n règles de réécriture de $\mathcal{R}$ et $p_1, \ldots, p_n$ n nœuds distincts de g_1. On dit que g_1 *se réécrit en parallèle en* g_2 *aux nœuds* $p_1, \ldots, p_n$ *avec les règles* $R_1, \ldots, R_n$, noté $g_1 \Vdash\!\!\!\!\rightarrow_{[p_1, R_1] \ldots [p_n, R_n]} g_2$ ssi :

1. Soient $l_i \rightarrow r_i$ une variante de R_i et $h_i : l_i \rightarrow g_{1|p_i}$ un filtre de l_i sur g_1 au nœud p_i pour tout $i \in 1..n$.
2. Soit $H = g \oplus h_1[r_1] \oplus \ldots \oplus h_n[r_n]$.
3. Soient $\rho_1, \ldots, \rho_n$ des redirections de pointeurs telles que pour tout $i \in 1..k$, $\rho_i(p_i) = \mathcal{R}\text{oot}_{h_i[r_i]}$ et $\rho_i(p) = p$ pour tout nœud p tel que $p \neq p_i$.
4. Soit $\rho = \rho_{\mu(1)} \circ \ldots \circ \rho_{\mu(n)}$ où $\mu : 1..n \rightarrow 1..n$ est une permutation telle que si $i < j$, alors il n'existe pas de chemin de $p_{\mu(i)}$ vers $p_{\mu(j)}$.
5. Soient $H' = \rho(H)$ et $r = \rho(\mathcal{R}\text{oot}_g)$.
6. $g_2 = H'_{|r}$.

On note $\Vdash\!\!\!\!\rightarrow^*$ la fermeture réflexive et transitive de $\Vdash\!\!\!\!\rightarrow$ et on parle de *dérivation de réécriture parallèle* de g en g' lorsque $g \Vdash\!\!\!\!\rightarrow^* g'$. ◇

On remarque que dans le cadre des termes-graphes admissibles, il existe toujours au moins une permutation satisfaisant les conditions du point 4.

Exemple 4.3.5 Considérons de nouveau l'Exemple 4.3.3. Soient $\rho_1 = \{\texttt{p2} \mapsto \texttt{r1}\}$ et $\rho_2 = \{\texttt{p4} \mapsto \texttt{p2}\}$. Il existe un chemin allant de `p4` vers `p2` dans g mais aucun chemin allant de `p2` vers `p4`. Donc on pose $\rho = \rho_1 \circ \rho_2 = \{\texttt{p2} \mapsto \texttt{r1}, \texttt{p4} \mapsto \texttt{r1}\}$. Le lecteur peut vérifier que $\rho(H) = H_2 = \texttt{p1 :A(r1 :E(p3 :C),r1) + r1 + r1 + p2 :B(p3) + p4 :D(r1)}$ (cf. Figure 4.10) et que $\rho(\mathcal{R}\text{oot}_g) = \texttt{p1}$. Donc d'après la Définition 4.3.4, $g \Vdash_{[\texttt{p2},\texttt{V1}][\texttt{p4},\texttt{V2}]} g_2$ où $g_2 = \texttt{p1 :A(r1 :E(p3 :C),r1)}$. □

Proposition 4.3.6 Soit SP un cGRS.

- Si $g_1 \Vdash g_2$, alors $g_1 \xrightarrow{*} g_2$.
- Si $g_1 \rightarrow g_2$, alors $g_1 \Vdash g_2$.

◇

Preuve : Du fait de la redirection de pointeurs utilisée, la réécriture en parallèle des rédex d'un graphe consiste en la réécriture de ces rédex, les uns après le autres, de bas en haut dans ce graphe. En conséquence, on infère que si $g_1 \Vdash g_2$, alors $g_1 \xrightarrow{*} g_2$. Le second point de la proposition est évident. □

Theorème 4.3.7 Soit SP un WAGRS. $\Vdash$ est confluente modulo $\sim$ et $\doteq$ par rapport aux termes-graphes admissibles. ◇

Preuve : Conséquence évidente du Théorème 3.3.4 et de la Proposition 4.3.6. □

4.3.2 Stratégie $\bar{\Phi}$

Dans ce paragraphe, nous étendons la stratégie $\bar{\Phi}$ définie dans [Ant92] aux WAGRS. $\bar{\Phi}$ est une stratégie parallèle de réécriture de graphes, c'est-à-dire une fonction partielle qui prend un terme-graphe admissible g et qui retourne *un ensemble de couples* (p, R) tels que p soit un nœud de g, R soit une règle de réécriture de $\mathcal{R}$ et g se réécrive au nœud p avec la règle R.

Comme tous les rédex calculés par une stratégie $\bar{\mathcal{S}}$ peuvent être utiles pour calculer la forme normale constructeur d'un terme-graphe g, on les considère tous en effectuant des pas de réécriture parallèle. On note $g \Vdash_{\bar{\mathcal{S}}} g'$ le *$\bar{\mathcal{S}}$-pas de réécriture parallèle* de g vers g' tel que $g \Vdash_{\bar{\mathcal{S}}(g)} g'$. On note $\Vdash^*_{\bar{\mathcal{S}}}$ la fermeture réflexive et transitive de $\Vdash_{\bar{\mathcal{S}}}$ et on parle de *$\bar{\mathcal{S}}$-dérivation* de g en g' lorsque $g \Vdash^*_{\bar{\mathcal{S}}} g'$.

Nous étendons les notions de $\mathcal{C}$-normalisation et de $\mathcal{C}$-hyper-normalisation définies pour les stratégies séquentielles aux stratégies parallèles :

Définition 4.3.8 ($\mathcal{C}$-normalisation)
Soit SP un cGRS. Une stratégie parallèle $\bar{\mathcal{S}}$ est *$\mathcal{C}$-normalisante* ssi pour tout terme-graphe admissible $\mathcal{C}$-normalisable g et pour tout terme-graphe constructeur c tels que $g \xrightarrow{*} c$, il existe un terme-graphe constructeur c' et une $\bar{\mathcal{S}}$-dérivation $g \Vdash^*_{\bar{\mathcal{S}}} c'$ tels que $c' \doteq c$.
Une stratégie $\bar{\mathcal{S}}$ est *$\mathcal{C}$-hyper-normalisante* ssi pour tout terme-graphe admissible $\mathcal{C}$-normalisable g et pour tout terme-graphe constructeur c tels que $g \xrightarrow{*} c$, toute dérivation de réécriture D qui commence avec g et qui alterne des $\bar{\mathcal{S}}$-pas de réécriture parallèle avec des pas de réécriture quelconques termine par une forme normale constructeur c' telle que $c' \doteq c$.
◇

Comme Φ, la stratégie $\bar{\Phi}$ est basée sur des arbres de définition (cf. Définition 4.2.3). Nous avons vu dans les Exemples 4.2.4, 4.2.6 et 4.2.7 qu'il est impossible en général de construire un arbre de définition contenant toutes les règles de réécriture définissant une opération dans le cas des WAGRS. En d'autres termes, les WAGRS ne sont généralement pas des

ISGRS. Toutefois, un WAGRS peut être décomposé en plusieurs sous-systèmes inductivement séquentiels. Aussi, une opération f peut être complètement décrite avec un *ensemble* d'arbres de définition $\{\mathcal{T}_f^1, \mathcal{T}_f^2, \ldots, \mathcal{T}_f^k\}$ qu'on appelle une *forêt* d'arbres de définition et qu'on note $\mathcal{F}_f$.

Proposition 4.3.9 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS. L'ensemble des règles $\mathcal{R}$ peut être partitionné en une union disjointe d'ensembles de règles $\mathcal{R} = \uplus_{i=1}^{n} \mathcal{R}_i$ tel que $\langle \Sigma, \mathcal{R}_i \rangle$ soit un ISGRS pour tout $i \in 1..n$. ◇

Preuve : Immédiat puisque si $\mathcal{R} = \{l_1 \to r_1, \ldots, l_k \to r_k\}$, alors $\langle \Sigma, \{l_i \to r_i\} \rangle$ est un ISGRS pour tout $i \in 1..k$. □

Définition 4.3.10 (Forêt d'arbres de définition)
Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS. Soit $\mathcal{R} = \uplus_{i=1}^{n} \mathcal{R}_i$ une partition de l'ensemble de règles de $\mathcal{R}$ telle que $SP_i = \langle \Sigma, \mathcal{R}_i \rangle$ soit un ISGRS pour tout $i \in 1..n$. Soient f une opération définie de Σ et $\mathcal{T}_i$ l'arbre de définition de f dans SP_i pour tout $i \in 1..n$. On appelle *forêt d'arbres de définition* de f l'ensemble $\mathcal{F}_f$ des arbres $\mathcal{T}_i$, c'est-à-dire $\mathcal{F}_f = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$. ◇

Exemple 4.3.11 Dans la Figure 4.4, nous représentons un exemple de forêts d'arbres de définition des opérations `f`, `g` et `h` du WAGRS $SP_2 = \langle \Sigma, \mathcal{R}_2 \rangle$ de l'Exemple 3.1.12. On pose $\mathcal{F}_{\mathtt{f}} = \{\mathcal{T}_{\mathtt{f}}^1, \mathcal{T}_{\mathtt{f}}^2\}$, $\mathcal{F}_{\mathtt{g}} = \{\mathcal{T}_{\mathtt{g}}^1, \mathcal{T}_{\mathtt{g}}^2\}$ et $\mathcal{F}_{\mathtt{h}} = \{\mathcal{T}_{\mathtt{h}}\}$. □

Notre stratégie parallèle $\bar{\Phi}$ est une fonction partielle qui opère sur les termes-graphes admissibles en présence d'un WAGRS et de forêts d'arbres de définition. $\bar{\Phi}(g)$ retourne (lorsque c'est possible) un ensemble de couples (p, R) tels que p soit un nœud de g, R soit une règle de réécriture de $\mathcal{R}$ et g se réécrive au nœud p avec la règle R. $\bar{\Phi}$ utilise deux fonctions auxilliaires $\bar{\varphi}$ et $Outer$. $\bar{\varphi}$ prend deux arguments : un terme-graphe admissible de racine fonctionnelle et un pdt de cette opération.

Définition 4.3.12 ($\bar{\varphi}$)
Soient SP un WAGRS, g un terme-graphe admissible de racine fonctionnelle et $\mathcal{T}$ un pdt tel que $pattern(\mathcal{T})$ filtre g à la racine. On définit la fonction partielle $\bar{\varphi}$ de la manière suivante :

$$\bar{\varphi}(g, \mathcal{T}) = \begin{cases} \{(p, R)\} & \text{si} \quad \mathcal{T} = rule(\pi \to r),\ p = \mathcal{R}\mathrm{oot}_g \text{ et} \\ & \quad R \text{ est une variante de } \pi \to r\,; \\ \bar{\varphi}(g, \mathcal{T}_i) & \text{si} \quad \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ et} \\ & \quad pattern(\mathcal{T}_i) \text{ filtre } g \text{ pour un } i \in 1..k\,; \\ S & \text{si} \quad \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \pi \text{ filtre } g \text{ par l'homomorphisme } h : \pi \to g, \\ & \quad h(o) \text{ est étiqueté par l'opération } f \text{ (dans } g), \\ & \quad \mathcal{F}_f = \{\mathcal{T}_1', \ldots, \mathcal{T}_k'\} \text{ est une forêt} \\ & \quad\quad \text{d'arbres de définition de } f \text{ et} \\ & \quad S = \bar{\varphi}(g_{|h(o)}, \mathcal{T}_1') \cup \ldots \cup \bar{\varphi}(g_{|h(o)}, \mathcal{T}_k'). \end{cases}$$

◇

Dans la définition ci-dessus, $\bar{\varphi}(g, \mathcal{T})$ calcule un ensemble S de couples (p, R) tels que R soit une règle dont le membre gauche filtre g au nœud p. Certains des couples (p, R) sont inutiles dans le calcul de la stratégie $\bar{\Phi}$. Aussi, on définit la fonction $Outer(g, S)$ qui choisit un sous-ensemble maximal (pour l'inclusion) de couples (p, R) de S tels que p soit un plus haut nœud fonctionnel de g. Si un plus haut nœud fonctionnel p de g apparaît plusieurs fois dans S, un seul couple (p, R) interviendra dans $Outer(g, S)$. $Outer(g, S)$ peut être défini de manière déterministe en utilisant un ordre sur les règles de réécriture.

Définition 4.3.13 (*Outer*)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS, g un terme-graphe admissible et
$S = \{(p_1, R_1), \ldots, (p_n, R_n)\}$ un ensemble de couples tels que p_i soit un nœud de g et R_i soit une règle de réécriture admissible. On définit $Outer(g, S)$ comme un sous-ensemble maximal $\{(q_1, S_1), \ldots, (q_k, S_k)\}$ de S tel que :

1. Pour tout $i, j \in 1..k$, $i \neq j \Longrightarrow q_i \neq q_j$.
2. Pour tout $i \in 1..k$, il existe un chemin $[\mathcal{R}\text{oot}_g, i_0, u_1, i_1, \ldots, i_{k-1}, q_i]$ tel que pour tout $j \in 0..k-1$ et pour toute variante R d'une règle de $\mathcal{R}$, $(u_j, R) \notin S$.

◇

Définition 4.3.14 (Stratégie $\bar{\Phi}$)
Soient SP un WAGRS, g un terme-graphe admissible, p le plus haut nœud fonctionnel à gauche de g, f l'étiquette de p dans g et $\mathcal{F}_f = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ une forêt d'arbres de définition de f. $\bar{\Phi}$ est la fonction partielle définie par $\bar{\Phi}(g) = Outer(g, S)$ où $S = \bar{\varphi}(g_{|p}, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(g_{|p}, \mathcal{T}_k)$.
◇

Exemple 4.3.15 Considérons les forêts de l'Exemple 4.3.11 et de la Figure 4.4. Soit $g =$ `n1 :c(n2 :f(n3 :h(n4 :g(n5 :a,n5),n6 :g(n4,n5)),n6),n7 :c(n6,n1))` le graphe de la Figure 4.11. D'après la Définition 4.3.14, $\bar{\Phi}(g) = Outer(g, S)$ où

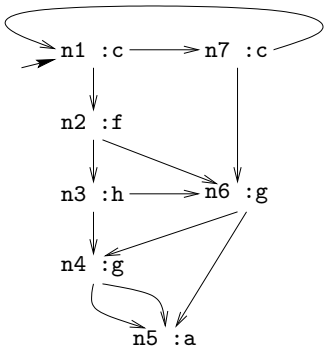


FIG. 4.11:

$$\begin{aligned} S &= \bar{\varphi}(g_{|\mathtt{n2}}, \mathcal{T}^1_{\mathtt{f}}) \cup \bar{\varphi}(g_{|\mathtt{n2}}, \mathcal{T}^2_{\mathtt{f}}) \\ &= \bar{\varphi}(g_{|\mathtt{n3}}, \mathcal{T}_{\mathtt{h}}) \cup \bar{\varphi}(g_{|\mathtt{n6}}, \mathcal{T}^1_{\mathtt{g}}) \cup \bar{\varphi}(g_{|\mathtt{n6}}, \mathcal{T}^2_{\mathtt{g}}) \\ &= \{(\mathtt{n3}, \mathtt{R6})\} \cup \bar{\varphi}(g_{|\mathtt{n4}}, \mathcal{T}^1_{\mathtt{g}}) \cup \bar{\varphi}(g_{|\mathtt{n4}}, \mathcal{T}^2_{\mathtt{g}}) \cup \{(\mathtt{n6}, \mathtt{R4})\} \\ &= \{(\mathtt{n3}, \mathtt{R6}), (\mathtt{n4}, \mathtt{R3}), (\mathtt{n4}, \mathtt{R4}), (\mathtt{n6}, \mathtt{R4})\} \end{aligned}$$

D'après la seconde condition de la Définition 4.3.13, $Outer(g, S)$ sélectionne les plus hauts rédex de S dans g, donc $\bar{\Phi}(g) = \{(\mathtt{n3}, \mathtt{R6}), (\mathtt{n6}, \mathtt{R4})\}$.
On remarque sur cet exemple que S contient deux couples utilisant le nœud `n4`, à savoir $(\mathtt{n4}, \mathtt{R3})$ et $(\mathtt{n4}, \mathtt{R4})$. Si $g_{|\mathtt{n4}}$ avait été un plus haut rédex de S dans g, alors la première condition de la Définition 4.3.13 aurait nécessité de choisir un couple parmi $(\mathtt{n4}, \mathtt{R3})$ et $(\mathtt{n4}, \mathtt{R4})$ pour calculer $\bar{\Phi}(g)$. Ce choix n'a pas d'importance puisque réécrire g avec les règles `R3` ou `R4` conduit au même graphe (par définition d'un WAGRS). □

Nous faisons maintenant la liste des propriétés de $\bar{\Phi}$.

Proposition 4.3.16 Soit SP un WAGRS et g un terme-graphe admissible. L'ensemble des rédex calculés par $\bar{\Phi}(g)$ constitue un ensemble suffisant de rédex de g. ◇

Comme cas particulier du théorème ci-dessus, on retrouve le fait que si le système de réécriture considéré est un $ISGRS$, alors $\bar{\Phi}(g)$ calcule un singleton $\{(p, R)\}$ tel que $g_{|p}$ soit un rédex nécessaire de g (cf. Proposition 4.2.11).

La Proposition 4.3.16 est une conséquence du lemme suivant, qui s'inspire de [AEH97a, Lemma 2] :

Lemme 4.3.17 Soient SP un WAGRS, g un terme-graphe admissible de racine fonctionnelle, f l'étiquette de la racine de g, $\mathcal{F}_f = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ une forêt d'arbres de définition de f et $S = \bar{\varphi}(g, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(g, \mathcal{T}_n)$. Aucun descendant de g ne peut être réécrit en un terme-graphe de racine constructeur à moins qu'un descendant de $g_{|p}$ ne soit réécrit en un terme-graphe de racine constructeur pour un couple $(p, R) \in S$. En d'autres termes, S est un ensemble suffisant de rédex de g. ◇

Preuve : Il y a deux cas à étudier dépendant du fait que g soit un rédex ou pas.
Cas 1 : Si g est un rédex :
Dans ce cas, il existe une ou plusieurs règles R dans $\mathcal{R}$ dont le membre gauche filtre g à la racine. Par construction de $\mathcal{F}_f$, pour toutes ces règles R, il existe un arbre de définition $\mathcal{T} \in \mathcal{F}_f$ tel que R soit représentée par une feuille de $\mathcal{T}$. D'après la définition de $\bar{\varphi}$, $\bar{\varphi}(g, \mathcal{T})$ est définie et $\bar{\varphi}(g, \mathcal{T}) = \{(\mathcal{R}\text{oot}_g, R)\}$. Comme nous travaillons avec des WAGRS, le membre gauche de R filtre tous les descendants de g qui ne sont pas réécrits à la racine. Par conséquent, aucun descendant de g ne peut être réécrit en un terme-graphe de racine constructeur à moins qu'un descendant de $g_{|\mathcal{R}\text{oot}_g}$ ne soit réécrit en un terme-graphe de racine constructeur (avec la règle R ou avec une autre règle).
Cas 2 : Si g n'est pas un rédex :
Nous prouvons le Lemme 4.3.17 par induction sur le nombre d'opérations définies de g. Si g n'a qu'une opération définie, alors elle est située à la racine et comme g n'est pas un rédex, g ne peut pas être réécrit en un terme-graphe de racine constructeur. Donc le cas de base est évident. Supposons que g ait plus d'une opération définie. Soient f l'étiquette de la racine de g, $\mathcal{F}_f = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ une forêt d'arbres de définition de f et $S = \bar{\varphi}(g, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(g, \mathcal{T}_n)$. g n'est pas un rédex, donc pour tout $i \in 1..n$, soit $\bar{\varphi}(g, \mathcal{T}_i) = \emptyset$ soit il existe un sous-arbre $\mathcal{T}_i'$ de $\mathcal{T}_i$ tel que $\mathcal{T}_i' = branch(\pi_i, o_i, \ldots)$, π_i filtre g à la racine avec l'homomorphisme $h_i : \pi_i \to g$, $h_i(o_i)$ soit étiqueté avec l'opération définie f_i dans g, $\mathcal{F}_{f_i} = \{\mathcal{T}_1'', \ldots, \mathcal{T}_{m_i}''\}$ et $\bar{\varphi}(g, \mathcal{T}_i) \neq \emptyset = \bar{\varphi}(g_{|h_i(o_i)}, \mathcal{T}_1'') \cup \ldots \cup \bar{\varphi}(g_{|h_i(o_i)}, \mathcal{T}_{m_i}'')$. Soit $R = l \to r$ une des règles permettant de réécrire un descendant de g à la racine en un terme-graphe de racine constructeur. Par construction d'une forêt d'arbres de définition, il existe $i \in 1..n$ tel que la règle R soit représentée par une feuille du sous-arbre $\mathcal{T}_i'$ de l'arbre $\mathcal{T}_i$ (tel que $\bar{\varphi}(g, \mathcal{T}_i) \neq \emptyset$). Par construction d'un arbre de définition, il existe un homomorphisme $\mu : \pi_i \to l$ et $\mu(o_i)$ est étiqueté par un constructeur (puisque l est un motif). Or dans le cas que nous considérons, $h_i(o_i)$ est étiqueté par une opération définie dans g. Donc il est nécessaire de réécrire un descendant de $g_{|h_i(o_i)}$ en un terme-graphe de racine constructeur pour pouvoir réécrire un descendant de g avec la règle R en un terme-graphe de racine constructeur. Autrement dit, aucun descendant de g ne peut être réécrit à la racine en un terme-graphe de racine constructeur à moins qu'un descendant de $g_{|h_i(o_i)}$ ne soit réécrit à la racine en un terme-graphe de racine constructeur pour au moins un $i \in 1..n$. $g_{|h_i(o_i)}$ est un sous-graphe propre de g. Donc par hypothèse d'induction, aucun descendant de $g_{|h_i(o_i)}$ ne peut être réécrit à la racine en un terme-graphe de racine constructeur à moins qu'un descendant de $(g_{|h_i(o_i)})_{|p}$ ne soit réécrit à la racine en un terme-graphe de racine constructeur pour un $(p, R) \in \bar{\varphi}(g_{|h_i(o_i)}, \mathcal{T}_j'')$. Par transitivité, aucun descendant de g ne peut être réécrit à la racine en un terme-graphe de racine constructeur à moins qu'un descendant de $g_{|p}$ ne soit réécrit à la racine en un terme-graphe de racine constructeur pour un $(p, R) \in \bar{\varphi}(g, \mathcal{T}_i)$ (donc pour un $(p, R) \in S$). □

Preuve de la Proposition 4.3.16 : On traite le cas où g est un terme-graphe admissible de racine fonctionnelle tel que f soit l'étiquette de la racine de g, $\mathcal{F}_f = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ soit une forêt d'arbres

de définition de f et $S = \bar{\varphi}(g, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(g, \mathcal{T}_n)$. Le cas où g est un terme-graphe admissible de racine constructeur est une conséquence immédiate. D'après le Lemme 4.3.17, l'ensemble S est un ensemble suffisant de rédex de g, i.e., dans toute dérivation de réécriture allant de g vers une forme normale constructeur, un descendant d'au moins un rédex $(p, R) \in S$ est réécrit à la racine. Soit $O = Outer(g, S)$. Nous montrons que O reste un ensemble suffisant de rédex de g. Il suffit de montrer que pour toute dérivation de réécriture D allant de g vers une forme normale constructeur telle qu'un descendant du rédex $g_{|p}$ soit réécrit à la racine avec la règle R pour $(p, R) \in (S - O)$, il existe un autre rédex $g_{|q}$ dont un descendant dans D est réécrit à la racine avec une règle R' telle que $(q, R') \in O$.

Il y a deux cas à étudier selon les raisons pour lesquelles le couple (p, R) n'appartient pas à O.
Cas 1 : $(p, R) \notin O$ car il existe R' tel que $(p, R') \in O$:
Par définition d'un WAGRS, on peut aussi réécrire le descendant de $g_{|p}$ aussi bien avec la règle R qu'avec la règle R' et on obtient le même résultat. Donc dans D, un descendant de $g_{|p}$ est réécrit à la racine avec une règle R'.
Cas 2 : $(p, R) \notin O$ car pour tous les chemins C allant de $\mathcal{R}oot_g$ vers p, il existe un nœud q et une règle R' tel que $q \in C$ et $(q, R') \in O$:
Si g est un rédex, il est clair que $q = \mathcal{R}oot_g$ et qu'un descendant de $g_{|\mathcal{R}oot_g}$ est réécrit dans D. Si g n'est pas un rédex, tous les chemins allant de $\mathcal{R}oot_g$ vers p passent par des nœuds fonctionnels q tel que $(q, R') \in O$ et $g_{|q}$ soit un rédex de g. Donc au moins un descendant des rédex $g_{|q}$ est réécrit dans D pour calculer la forme normale constructeur de g.
On en conclut que $O = Outer(g, S) = \bar{\Phi}(g)$ est un ensemble suffisant de rédex de g. □

De la proposition précédente, on déduit que si un terme-graphe admissible a une forme normale constructeur c, alors toute dérivation allant de g vers c doit contenir un pas de réécriture au nœud p avec la règle R pour un couple $(p, R) \in \bar{\Phi}(g)$. Le lemme suivant établit qu'étant donnée une dérivation $D = g \xrightarrow{*} c$, la réécriture parallèle de tous les rédex calculés par $\bar{\Phi}(g)$ conduit à une nouvelle dérivation de longueur inférieure (ou égale) à celle de D.

Lemme 4.3.18 Soient SP un WAGRS, g un terme-graphe admissible non constructeur $\mathcal{C}$-normalisable et c un graphe constructeur tel qu'il existe une dérivation de réécriture $g \xrightarrow{*} c$ de longueur n. Alors il existe un terme-graphe admissible g' tel que $g \amalg\!\!\rightarrow_{\bar{\Phi}} g'$ et une dérivation de réécriture $g' \xrightarrow{*} c'$ de longueur n' tels que $n' < n$ et que c' soit un graphe constructeur avec $c' \sim c$. ◇

Preuve : Par hypothèse, il existe une dérivation $A = g \xrightarrow{*} c$ de longueur n. Supposons que $A = g \rightarrow_{[p_1, R_1]} u_1 \ldots u_{n-1} \rightarrow_{[p_n, R_n]} c$. Soit g' un terme-graphe admissible tel que $g \amalg\!\!\rightarrow_{\bar{\Phi}} g'$. Supposons que $\bar{\Phi}(g) = \{(q_1, S_1), \ldots, (q_m, S_m)\}$. D'après la Proposition 4.3.6, il existe une dérivation de réécriture B allant de g vers g' telle que $B = g \rightarrow_{[q_1, S_1]} v_1 \ldots v_{m-1} \rightarrow_{[q_m, S_m]} g'$. Comme c est un terme-graphe constructeur, nous pouvons montrer à partir de la preuve de la Proposition 3.4.10 qu'il existe un terme-graphe constructeur c' et une dérivation $A' = g' \xrightarrow{\varepsilon}_{[p_1, R_1]} a_1 \ldots a_{n-1} \xrightarrow{\varepsilon}_{[p_n, R_n]} c'$ tels que $c' \sim c''$. D'après la Proposition 4.3.16, $\{(q_1, S_1), \ldots, (q_m, S_m)\}$ est un ensemble suffisant de rédex, donc il existe au moins un couple (q_i, S_i) dans la dérivation B et un couple (p_j, R_j) dans la dérivation A tels que $(p_j, R_j) = (q_i, S_i)$. Comme $(q_i, S_i) = (p_j, R_j)$, p_j n'est pas un nœud de g' : il est effacé par le pas de réécriture $v_{i-1} \rightarrow_{[q_i, S_i]} v_i$ de la dérivation B. Le couple (p_j, R_j) apparaît également dans la dérivation A' puisque $A' = g' \xrightarrow{\varepsilon}_{[p_1, R_1]} a_1 \ldots a_{j-1} \xrightarrow{\varepsilon}_{[p_j, R_j]} a_j \ldots a_{n-1} \xrightarrow{\varepsilon}_{[p_n, R_n]} c'$. Comme p_j n'est pas un nœud de g', p_j ne peut pas être un nœud de a_{j-1}, donc de $a_j = a_{j-1}$. Nous en concluons que la longueur de la dérivation A' est inférieure ou égale à $n - 1$. □

Du lemme précédent, nous déduisons le théorème suivant :

Theorème 4.3.19 Soit SP un WAGRS. $\bar{\Phi}$ est un stratégie $\mathcal{C}$-hyper-normalisante (donc $\mathcal{C}$-normalisante). ◇

Preuve : Soient g_0 un terme-graphe admissible et c_0 un graphe constructeur tel qu'il existe une dérivation de réécriture $g_0 \xrightarrow{*} c_0$ de longueur n_0. On fait cette preuve par induction sur n_0. Le cas de base $n_0 = 0$ est évident. Soit $n_0 > 0$ et D une dérivation de la forme
$g_0 \xrightarrow{*} g'_0 \amalg\!\!\rightarrow_{\bar{\Phi}} g_1 \xrightarrow{*} g'_1 \amalg\!\!\rightarrow_{\bar{\Phi}} g_2 \xrightarrow{*} \ldots$ Comme g_0 se réécrit en g'_0 et qu'il existe une dérivation de longueur n_0 allant de g_0 vers le graphe constructeur c_0, on peut montrer (preuve de la Proposition 3.4.10) qu'il existe un graphe constructeur $c'_0 \sim c_0$ et une dérivation $g'_0 \xrightarrow{*} c'_0$ de longueur $n'_0 \leq n_0$. Comme la longueur de $g'_0 \xrightarrow{*} c'_0$ est n'_0 et que $g'_0 \amalg\!\!\rightarrow_{\bar{\Phi}} g_1$, il existe un graphe constructeur $c_1 \sim c'_0$ et une dérivation $g_1 \xrightarrow{*} c_1$ de longueur $n_1 < n'_0$, d'après le Lemme 4.3.18. Soit D' la sous-dérivation de D commençant avec g_1. Comme la longueur de la dérivation $g_1 \xrightarrow{*} c_1$ est $n_1 < n_0$, nous utilisons l'hypothèse d'induction : la dérivation D', donc la dérivation D, termine avec un graphe constructeur $C' \sim c_1 \sim c_0$. Par conséquent, nous concluons que $\bar{\Phi}$ est $\mathcal{C}$-hyper-normalisante. □

Il est clair que tout sur-ensemble d'un ensemble suffisant de rédex reste un ensemble suffisant de rédex. Comme l'ensemble de tous les plus hauts rédex d'un graphe g est un sur-ensemble de $\bar{\Phi}(g)$, on déduit le corollaire suivant du Théorème 4.3.19. Ce corollaire est une extension d'un résultat montré dans le cadre des terme du premier ordre par [O'D77] :

Corollaire 4.3.20 Soit SP un WAGRS. La stratégie parallèle qui consiste en la réécriture de tous les plus hauts rédex d'un graphe[4] est une stratégie $\mathcal{C}$-hyper-normalisante. ◇

Enfin, la Proposition 4.3.16 montre que la stratégie $\bar{\Phi}$ calcule un ensemble suffisant de rédex et l'Exemple 4.1.6 montre qu'on peut construire des dérivations de réécriture calculant la forme normale constructeur d'un graphe g sans forcément considérer tous les rédex calculés par $\bar{\Phi}(g)$. On pourrait donc imaginer des stratégies calculant des ensembles plus petits de rédex que ceux de $\bar{\Phi}(g)$ ...

Néanmoins, une stratégie de réécriture $\bar{\mathcal{S}}$ *qui ne prend pas en compte les membres droits des règles de réécriture* ne peut pas "faire mieux" que $\bar{\Phi}$. En ce sens, $\bar{\Phi}$ est une stratégie parallèle *optimale*. Pour développer cette notion d'optimalité de $\bar{\Phi}$, il faut définir une notion de réécriture arbitraire de graphes et une notion de recouvrement[5] d'un graphe par un ensemble de rédex. Le théorème d'optimalité de $\bar{\Phi}$ se montrerait alors en utilisant les techniques de preuve de [SR93].

[4] *parallel-outermost graph rewriting strategy* en anglais.
[5] *cover* en anglais.

Chapitre 5

Surréduction de graphes admissibles

Dans les précédents chapitres, nous avons utilisé des programmes logico-fonctionnels pour évaluer par réécriture des expressions représentées sous la forme de graphes cycliques admissibles. Dans ce chapitre, nous utilisons les programmes pour évaluer par *surréduction* des expressions *partiellement définies* représentées sous la forme de graphes admissibles *contenant des variables*. On utilise ce mode d'évaluation dans les langages logico-fonctionnels pour résoudre des buts. Comme notre étude vise à introduire les graphes dans les langages logico-fonctionnels, il est nécessaire d'étudier la relation de surréduction des graphes admissibles.

Surréduire un graphe g en un graphe g' consiste à trouver une substitution σ instanciant suffisamment les variables de g pour que $\sigma(g)$ puisse se réécrire en g'. Considérons par exemple le graphe H et la règle $L \rightarrow R$ de la Figure 5.1. Il est clair que H ne peut pas être réécrit et

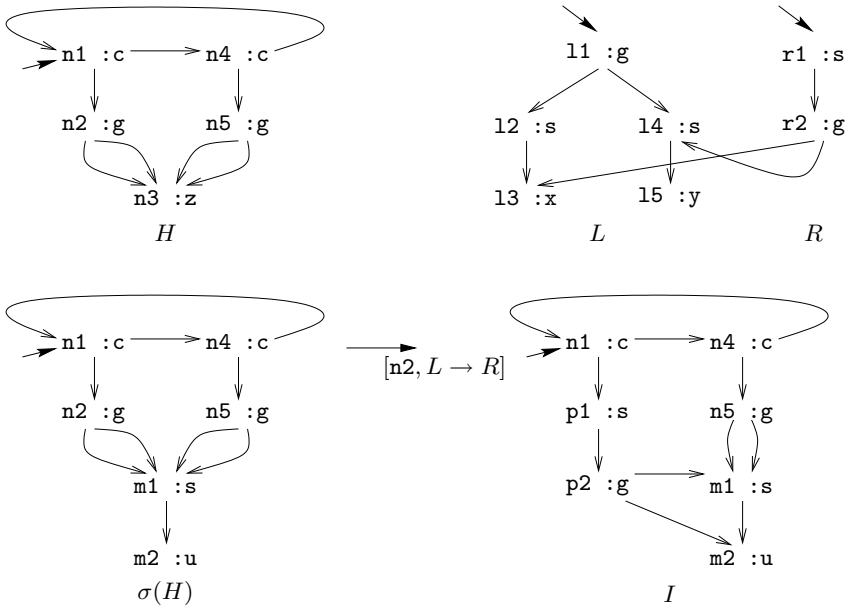


FIG. 5.1:

qu'il n'est pas non plus sous forme normale constructeur puisque le nœud `n2` est étiqueté par

l'opération définie `g`. Néanmoins, en instanciant la variable `z` avec le graphe `m1 :s(m2 :u)` à l'aide d'une substitution σ, on peut réécrire $\sigma(H)$ en I et se rapprocher ainsi d'une forme normale constructeur.

La relation de surréduction a été introduite dans [Sla74]. Son utilisation en démonstration automatique a été illustrée pour la première fois dans [Fay79] et [Hul80a]. Une première étude de la surréduction de graphes a été faite dans [Yam93, Kri96, HP96] dans le cadre des graphes acycliques et dans [EJ98a, EJ98b] dans le cadre des graphes cycliques admissibles. Toutefois, nous ne profitons pas pleinement de la structure de graphe dans le calcul que nous avons brièvement décrit ci-dessus. En effet, dans la Figure 5.1, on constate que $\sigma(H)$ contient deux sous-graphes qui ne sont pas partagés : celui de racine `n2` et celui de racine `n5`. En conséquence, le sous-graphe de racine `n5` continue d'apparaître dans I. Il faudra donc le réécrire pour progresser de nouveau vers une forme normale constructeur.

Pour pallier ce "défaut", nous combinons la relation de surréduction avec une relation dite de *compression* ou de *compaction de graphes*[1] [HP99]. On dit qu'un graphe g_1 se compacte en un graphe g_2 si g_1 et g_2 représentent la même expression (i.e., g_1 et g_2 sont bisimilaires) mais que la taille de g_2, c'est-à-dire le nombre de nœuds de g_2, est plus petit que celui de g_1. On parle de compression maximale de g_1 en g_2 si g_2 ne peut plus être compacté. Nous montrons dans la Figure 5.2 le graphe H' obtenu par compression maximale de $\sigma(H)$. Il est clair qu'en réécrivant H' au nœud `p2`, on progresse plus rapidement vers une forme normale constructeur qu'en réécrivant $\sigma(H)$ au nœud `n2`.

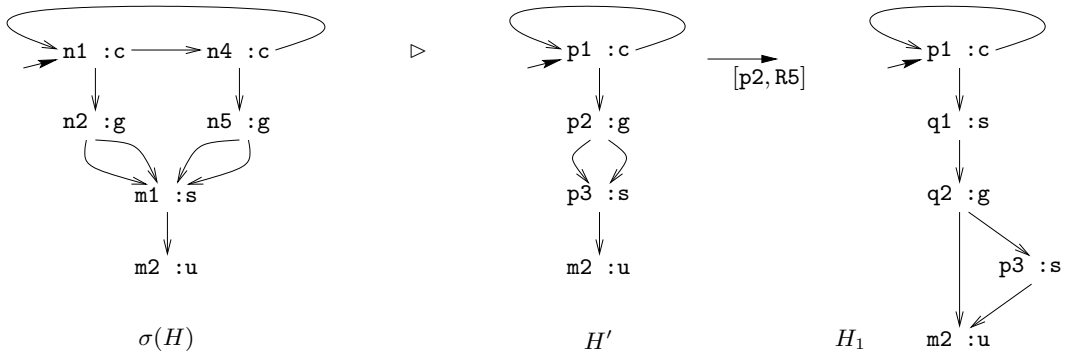


FIG. 5.2:

Dans ce chapitre, nous étudions donc la relation de *surréduction compressante* des graphes cycliques admissibles. Cette relation est "meilleure" que toutes les autres relations de surréduction. Considérons par exemple la règle `A(O)` $\rightarrow$ `O` et le terme `C(A(X),A(O),A(O))` où `A` est une opération définie, `C` et `O` sont des constructeurs et `X` est une variable. Dans la Figure 5.3, nous représentons trois dérivations de surréduction. La première correspond à une dérivation de surréduction de termes ; sa longueur est 3. La seconde correspond à une dérivation de surréduction "classique" de graphes ; sa longueur est 2. La troisième correspond à une dérivation de surréduction compressante maximale de graphes ; sa longueur est 1.

Nous avons mené une première étude de la surréduction de graphes admissibles dans [EJ98a, EJ98b, EJ97a]. La surréduction *compressante* des graphes admissibles fait l'objet d'un rapport [EJ99a].

Dans la section suivante, nous développons l'ensemble des notions qui sont nécessaires

[1] *graph collapsing* en anglais.

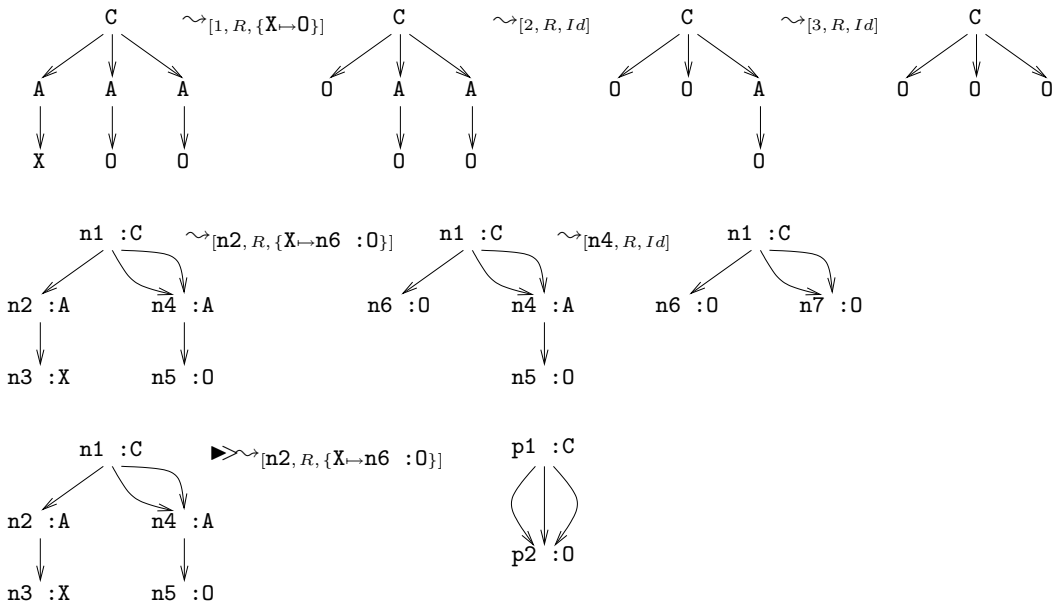


FIG. 5.3:

pour comprendre la définition du pas de surréduction compressante. Nous donnons cette définition dans la Section 5.2. Nous énonçons les théorèmes de cohérence et de complétude dans la Section 5.3 puis nous les démontrons dans les Sections 5.4 et 5.5.

5.1 Préliminaires

Un pas de surréduction compressante d'un graphe g_1 en un graphe g_2 se fait en trois temps :

1. Tout d'abord, il faut calculer une substitution σ qui instancie suffisamment les variables de g_1 pour qu'on puisse réécrire $\sigma(g_1)$. Cette substitution est un unificateur d'un sous-graphe de g_1 *par rapport* au membre gauche d'une règle de réécriture.
2. Ensuite, il faut compresser l'instance $\sigma(g_1)$ en un graphe g_1'.
3. Enfin, il faut réécrire le graphe g_1'.

Dans cette section de préliminaire, nous commençons par définir la notion d'unificateur d'un graphe par rapport à un motif, puis nous posons les bases de la compression de graphes.

5.1.1 Unificateur d'un graphe *par rapport à* un motif

Dans la Section 2.3, nous avons défini l'unification des termes-graphes avec des homomorphismes (voir Définition 2.3.7). Comme les homomorphismes sont moins pratiques que les substitutions pour définir les pas de surréduction, nous introduisons la notion suivante :

Définition 5.1.1 (Unificateur d'un graphe *par rapport à* un motif)
Soient g un terme-graphe admissible, l un motif et σ une substitution tels que g soit compatible avec σ. On dit que σ *est un unificateur de* g *par rapport à* l ssi il existe un unificateur $h : L \to M$ de g et de l tel que $\sigma = (\sigma_h)_{|\mathcal{V}_g}$. On dit que σ *est un unificateur le plus général de* g *par rapport à* l ssi h est un unificateur le plus général de g et de l. ◇

Exemple 5.1.2 Soient H = `n1 :c(n2 :g(n3 :z,n3),n4 :c(n5 :g(n3,n3),n1))` un terme-graphe admissible et L = `l1 :g(l2 :s(l3 :x),l4 :s(l5 :y))` un motif (cf. Figure 5.4). Soit $\sigma = \{\texttt{z} \mapsto \texttt{m1 :s(m2 :u)}\}$. Le lecteur peut vérifier que $\sigma(H) =$

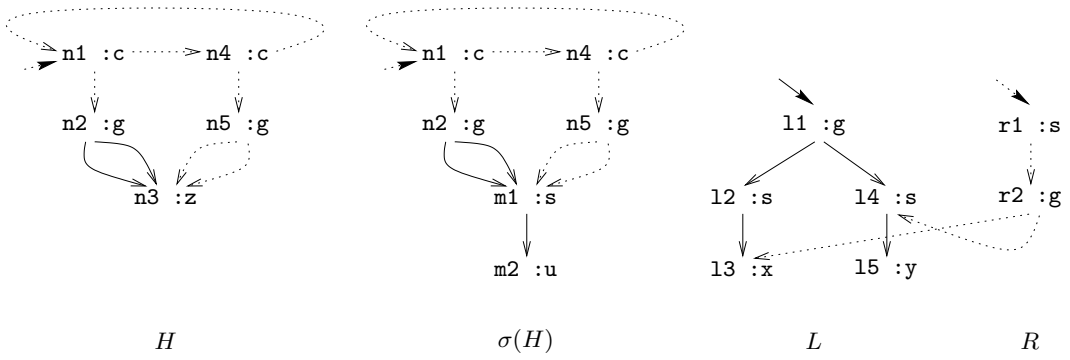


FIG. 5.4:

`n1 :c(n2 :g(m1 :s(m2 :u),m1),n4 :c(n5 :g(m1,m1),n1))`. Nous affirmons que σ est un unificateur de $H_{|\texttt{n2}}$ par rapport à L. En effet, il existe un homomorphisme

$$\upsilon : (H_{|\texttt{n2}} \oplus L) \to (\sigma(H_{|\texttt{n2}}) \oplus \sigma(H_{|\texttt{n2}}))$$

tel que $\upsilon(\texttt{n2}) = \upsilon(\texttt{l1}) = \texttt{n2}$, $\upsilon(\texttt{n3}) = \upsilon(\texttt{l2}) = \upsilon(\texttt{l4}) = \texttt{m1}$ et $\upsilon(\texttt{l3}) = \upsilon(\texttt{l5}) = \texttt{m2}$. De plus, on constate que υ est un unificateur le plus général de $H_{|\texttt{n2}}$ et de L. Donc σ est un unificateur *le plus général* de $H_{|\texttt{n2}}$ par rapport à L. □

5.1.2 Compression d'un graphe

On dit qu'un graphe g_1 se compacte (ou se compresse) en un graphe g_2 si g_1 et g_2 sont bisimilaires mais que le nombre de nœuds de g_2 est plus petit que celui de g_1 :

Définition 5.1.3 (Compression d'un graphe)
Soient g_1 et g_2 deux graphes compatibles. On dit que g_1 *se compacte en* g_2, noté $g_1 \rhd g_2$, ssi il existe un $\mathcal{V}$-homomorphisme $h : g_1 \to g_2$. ◇

Exemple 5.1.4 Considérons les graphes $\sigma(H)$ et H' de la Figure 5.5. Comme il existe un

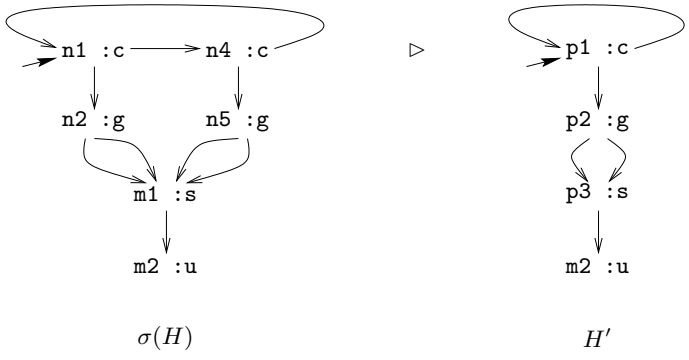


FIG. 5.5:

$\mathcal{V}$-homomorphisme $h : \sigma(H) \to H'$ tel que $h(\texttt{n1}) = h(\texttt{n4}) = \texttt{p1}$, $h(\texttt{n2}) = h(\texttt{n5}) = \texttt{p2}$, $h(\texttt{m1}) = \texttt{p3}$ et $h(\texttt{m2}) = \texttt{m2}$, on conclut que $\sigma(H) \rhd H'$. □

Pour définir la relation de surréduction compressante de graphe, nous avons besoin d'utiliser un algorithme de compression de graphes :

Définition 5.1.5 (Stratégie de compression)
On appelle *stratégie de compression* une fonction totale $\blacktriangleright$ définie sur les graphes telle que si $\blacktriangleright(g) = g'$, alors g se compacte en g' (i.e., $g \rhd g'$). On suppose que $\blacktriangleright$ est une fonction déterministe au renommage des nœuds près (i.e., $\blacktriangleright(g)$ est unique au renommage des nœuds près). ◇

Dans la suite, nous utiliserons deux stratégies de compression remarquables :

- La première stratégie de compression est l'identité. Cette stratégie particulière n'a aucun effet de compression sur les graphes.
- La seconde stratégie de compression, notée $\blacktriangleright\!\!\rangle$, est celle qui compresse les graphes au maximum. Formellement, $\blacktriangleright\!\!\rangle(g_1) = g_2$ ssi (1) $g_1 \rhd g_2$ et (2) pour tout terme-graphe admissible g, si $g_1 \rhd g$, alors $g \rhd g_2$.

Exemple 5.1.6 Dans l'Exemple 5.1.4, nous avons compressé le graphe $\sigma(H)$ au maximum pour obtenir H'. On a donc $\blacktriangleright\!\!\rangle(\sigma(H)) = H'$. □

5.2 Pas de surréduction compressante

Plusieurs définitions du pas de surréduction compressante sont possibles, comme nous le verrons plus loin. Nous choisissons la suivante :

Définition 5.2.1 (Pas de surréduction compressante)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, $\blacktriangleright$ une stratégie de compression, g_1 et g_2 deux graphes, p un nœud fonctionnel de g_1, $l \to r$ une règle de réécriture de $\mathcal{R}$ et σ une substitution.
Un *pas de surréduction compressante de g_1 en g_2 au nœud p avec la règle $l \to r$, la substitution σ et la stratégie de compression* $\blacktriangleright$ se note $g_1 \blacktriangleright\!\!\rightsquigarrow_{[p, l \to r, \sigma]} g_2$ et satisfait les trois conditions suivantes :

1. σ est un unificateur de $g_{1|p}$ par rapport à l.
2. Il existe un graphe $g'_1 = \blacktriangleright(\sigma(g_1))$ et un $\mathcal{V}$-homomorphisme $h : \sigma(g_1) \to g'_1$.
3. $g'_1 \to_{[h(p), R]} g_2$.

On dit que $g_1 \blacktriangleright\!\!\rightsquigarrow_{[p, l \to r, \sigma]} g_2$ est un *pas de surréduction compressante la plus générale* ssi σ est un unificateur *le plus général* de $g_{1|p}$ par rapport à l. ◇

Nous avons défini deux stratégies particulières de compression dans le paragraphe précédent. Nous disposons donc de deux relations de surréduction compressante :

- la relation de surréduction "classique" de graphes $\rightsquigarrow$ qui utilise l'identité comme stratégie de compression et
- la relation de *surréduction compressante maximale* de graphes $\blacktriangleright\!\!\rangle\!\!\rightsquigarrow$ qui utilise la stratégie de compression maximale $\blacktriangleright\!\!\rangle$.

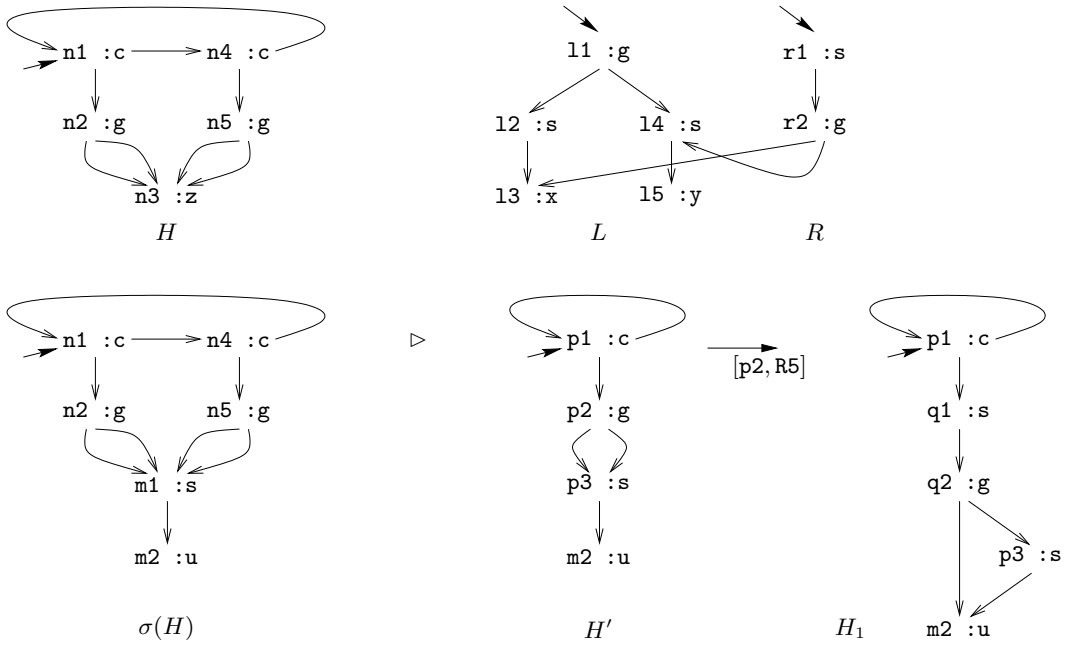


FIG. 5.6:

Exemple 5.2.2 Soient H le terme-graphe admissible et $L \to R$ la règle de la Figure 5.6. La règle $L \to R$ est exactement la règle `R5` de la Figure 3.2 et de l'Exemple 3.1.12. Soit $\sigma = \{\mathtt{z} \mapsto \mathtt{m1 :s(m2 :u)}\}$. Nous avons vu dans l'Exemple 5.1.2 que σ était un unificateur le plus général de $H_{|\mathtt{n2}}$ par rapport à L. Soit $H' = \mathtt{p1 :c(p2 :g(p3 :s(m2 :u),p3),p1)}$. Nous avons vu dans l'Exemple 5.1.6 que $\blacktriangleright\!\!\gg(\sigma(H)) = H'$ avec le $\mathcal{V}$-homomorphisme $h : \sigma(H) \to H'$ défini dans l'Exemple 5.1.4. H' se réécrit au nœud $h(\mathtt{n2}) = \mathtt{p2}$ avec la règle `R5` en $H_1 =$ `p1 :c(q1 :s(q2 :g(m2 :u,p3 :s(m2))),p1)`. Donc nous concluons que $H \blacktriangleright\!\!\gg\!\!\rightsquigarrow_{[\mathtt{n2},\mathtt{R5},\sigma]} H_1$. Ce pas de surréduction est un pas de surréduction compressante maximale la plus générale. □

La proposition ci-dessous établit d'une part qu'un pas de surréduction compressante est bien défini (au sens où la surréduction d'un graphe permet bien de calculer un graphe) et d'autre part que l'ensemble des termes-graphes admissibles est stable par surréduction.

Proposition 5.2.3 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, $\blacktriangleright$ une stratégie de compression, g_1 un terme-graphe admissible, $R = l \to r$ une règle de réécriture de $\mathcal{R}$, p un nœud fonctionnel de g_1 et σ un unificateur de $g_{1|p}$ par rapport à l. Alors il existe un terme-graphe admissible g_2 tel que $g_1 \blacktriangleright\!\!\rightsquigarrow_{[p,R,\sigma]} g_2$. ◇

Preuve : Puisque $g_1 \blacktriangleright\!\!\rightsquigarrow_{[p,R,\sigma]} g_2$, il existe un graphe g_1' et un $\mathcal{V}$-homomorphisme $h : \sigma(g_1) \to g_1'$ tels que $g_1' \to_{[h(p),R]} g_2$. Comme σ est un unificateur de $g_{1|p}$ par rapport à l, σ est une substitution telle que g_1 soit compatible avec σ. Donc on infère que $\sigma(g_1)$ est un terme-graphe admissible, à partir de la preuve de la Proposition 2.5.9. $\sigma(g_1)$ est un terme-graphe admissible et $\sigma(g_1) \rhd g_1'$, donc g_1' est aussi un terme-graphe admissible. D'après la Proposition 3.2.7 et le Théorème 3.2.8, la réécriture du terme-graphe admissible g_1' permet bien de calculer un terme-graphe admissible. Nous concluons donc que g_2 est un terme-graphe admissible. □

Notons qu'il existe d'autres définitions du pas de surréduction, en particulier celles qui utilisent des homomorphismes [Yam93, EJ98a, EJ98b]. Toutefois, nous préférons une définition

utilisant des substitutions car elle se rapproche de la définition de la surréduction des termes du premier ordre. Elle facilite donc la compréhension des exemples, des résultats et des preuves. Notre définition est équivalente aux définitions à base d'homomorphismes dans le cadre des cGRS.

Après avoir posé les bases du calcul de surréduction compressante, nous donnons ci-dessous les raisons qui nous ont poussées à choisir *cette* définition de la relation de surréduction compressante. En effet, il existe plusieurs manières de combiner la compression de graphes et la surréduction de graphes :

1. On peut commencer par faire le pas de compression puis celui de surréduction. La relation $\blacktriangleright\odot\rightsquigarrow$ qu'on obtient est définie de la manière suivante :

$$g_1 \blacktriangleright\odot\rightsquigarrow_{[p,\,R,\,\sigma]} g_2 \iff \begin{cases} \text{il existe un graphe } g_1' \text{ tel que } \blacktriangleright(g_1) = g_1', \\ \text{il existe un } \mathcal{V}\text{-homomorphisme } h : g_1 \to g_1' \text{ et} \\ \sigma(g_1') \to_{[h(p),\,R]} g_2 \end{cases}$$

2. On peut aussi commencer par faire le pas de surréduction puis celui de compression. La relation $\rightsquigarrow\odot\blacktriangleright$ qu'on obtient est définie de la manière suivante :

$$g_1 \rightsquigarrow\odot\blacktriangleright_{[p,\,R,\,\sigma]} g_2 \iff \sigma(g_1) \to_{[p,\,R]} g_1' \text{ et } \blacktriangleright(g_1') = g_2$$

3. Rappelons enfin la définition de la relation de surréduction compressante $\blacktriangleright\rightsquigarrow$ que nous avons choisie :

$$g_1 \blacktriangleright\rightsquigarrow_{[p,\,R,\,\sigma]} g_2 \iff \begin{cases} \text{il existe un graphe } g_1' \text{ tel que } \blacktriangleright(\sigma(g_1)) = g_1', \\ \text{il existe un } \mathcal{V}\text{-homomorphisme } h : \sigma(g_1) \to g_1' \text{ et} \\ \sigma(g_1') \to_{[h(p),\,R]} g_2 \end{cases}$$

Mais des trois relations ci-dessus, la relation de surréduction compressante $\blacktriangleright\rightsquigarrow$ semble "meilleure" que les autres, au sens où elle permet de calculer plus rapidement des formes normales constructeurs que $\blacktriangleright\odot\rightsquigarrow$ et $\rightsquigarrow\odot\blacktriangleright$. Cette propriété semble vérifiée lorsque la stratégie de compression utilisée est $\blacktriangleright\!\!>$. C'est en tout cas ce que suggère l'exemple ci-dessous. Par contre, nous ignorons si cette propriété est vraie pour une stratégie de compression quelconque.

Exemple 5.2.4 Soient $g =$ `n1 :A(n2 :B(n3 :X,n3),n4 :B(n5 :O,n5))` un terme-graphe admissible (cf. Figure 5.7) où `A` et `O` sont des symboles constructeurs, `B` est une opération définie et `X` est une variable. Soient $R =$ `l1 :B(l2 :O,l3 :O)` $\to$ `l2 :O` une règle de réécriture, $p =$ `n2` un nœud fonctionnel de g et $\sigma = \{$`X` $\mapsto$ `n6 :O`$\}$ une substitution. Exécutons

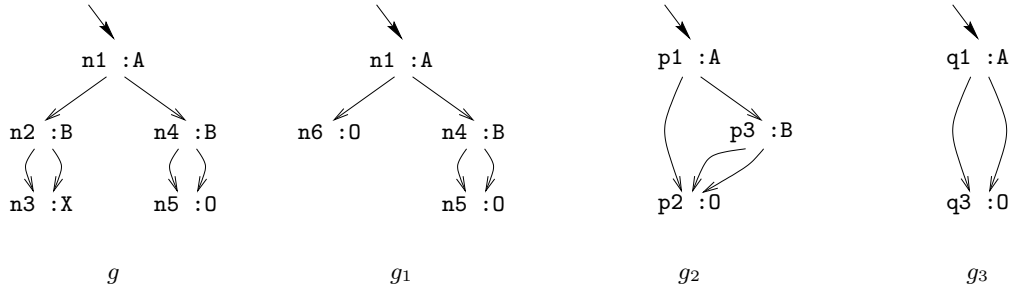


FIG. 5.7:

successivement les trois pas de surréduction définis précédemment :

1. Soit g_1 le graphe tel que $g \blacktriangleright\!\!\gg\!\odot\!\rightsquigarrow_{[p,R,\sigma]} g_1$. On ne peut pas compacter g plus qu'il ne l'est déjà, donc $g \blacktriangleright\!\!\gg\!\odot\!\rightsquigarrow_{[p,R,\sigma]} g_1$ ssi $\sigma(g) \rightarrow_{[p,R]} g_1$. Comme $\sigma(g) =$ `n1 :A(n2 :B(n6 :O,n6),n4 :B(n5 :O,n5))`, le lecteur peut vérifier que $g_1 =$ `n1 :A(n6 :O,n4 :B(n5 :O,n5))` (cf. Figure 5.7).
2. Soit g_2 le graphe tel que $g \rightsquigarrow\!\odot\!\blacktriangleright\!\!\gg_{[p,R,\sigma]} g_2$. Nous venons juste de montrer que $\sigma(g) \rightarrow_{[p,R]} g_1$ où $g_1 =$ `n1 :A(n6 :O,n4 :B(n5 :O,n5))`. En compactant g_1 au maximum, on obtient le graphe $g_2 =$ `p1 :A(p2 :O,p3 :B(p2,p2))` (cf. Figure 5.7).
3. Nous effectuons maintenant le pas de surréduction compressante $g \blacktriangleright\!\!\gg\!\rightsquigarrow_{[p,R,\sigma]} g_3$. En compactant $\sigma(g)$ au maximum, on obtient $g'_1 =$ `q1 :A(q2 :B(q3 :O,q3),q2)`. Le lecteur peut vérifier que $\sigma(g'_1)$ se réécrit au nœud `q2` en $g_3 =$ `q1 :A(q3 :O,q3)` (cf. Figure 5.7).

g_3 est une forme normale constructeur alors qu'il faut encore surréduire g_1 et g_2 pour calculer leurs formes normales constructeurs. On constate donc sur cet exemple que $\blacktriangleright\!\!\gg\!\rightsquigarrow$ est meilleure que $\blacktriangleright\!\!\gg\!\odot\!\rightsquigarrow$ et $\rightsquigarrow\!\odot\!\blacktriangleright\!\!\gg$. □

5.3 Cohérence et complétude de $\blacktriangleright\!\rightsquigarrow$

Dans cette partie, nous définissons les notions de cohérence et de complétude de la surréduction compressante. Ces définitions ne considèrent pas la surréduction comme une règle d'inférence permettant de résoudre des buts particuliers mais plutôt comme un modèle général de calcul d'expressions arbitraires représentées sous la forme de graphes cycliques admissibles. Les buts traditionnels comme les équations peuvent être représentés comme des expressions booléennes. La solution d'un but est une substitution.

Définition 5.3.1 (Substitution calculée par surréduction compressante)
Soient SP un cGRS, $\blacktriangleright$ une stratégie de compression et g_1 et g_2 deux termes-graphes admissibles. On dit qu'*une substitution σ est calculée par surréduction compressante de g_1 en g_2*, noté $g_1 \overset{*}{\blacktriangleright\!\rightsquigarrow}_\sigma g_2$, ssi il existe une dérivation $g_1 \blacktriangleright\!\rightsquigarrow_{[p_1,R_1,\sigma_1]} \ldots \blacktriangleright\!\rightsquigarrow_{[p_k,R_k,\sigma_k]} g_2$ et $\sigma = (\sigma_k \circ \ldots \circ \sigma_1)_{|\mathcal{V}_{g_1}}$. ◇

Exemple 5.3.2 Soit $H \blacktriangleright\!\!\gg\!\rightsquigarrow_{[\texttt{n2},\texttt{R5},\sigma]} H_1$ le pas de surréduction compressante de la Figure 5.6 et de l'Exemple 5.2.2 où $H_1 =$ `p1 :c(q1 :s(q2 :g(m2 :u,p3 :s(m2))),p1)` et $\sigma = \{$`z` $\mapsto$ `m1 :s(m2 :u)`$\}$. Considérons la règle $L' \rightarrow R'$ de la Figure 5.8. Cette règle est exactement la règle `R3` de Figure 3.2 et de l'Exemple 3.1.12. Soit $\sigma' = \{$`u` $\mapsto$ `m3 :a`$\}$ une substitution. Le lecteur peut vérifier de σ' est un unificateur le plus général de $H_{1|\texttt{q2}}$ par rapport à L'. De plus, comme $\sigma'(H_1) =$ `p1 :c(q1 :s(q2 :g(m3 :a,p3 :s(m3))),p1)` (cf. Figure 5.8), $\sigma'(H_1)$ ne peut pas être plus compressé qu'il ne l'est déjà. Donc $\blacktriangleright\!\!\gg(\sigma'(H_1)) = \sigma'(H_1)$. Enfin, $\sigma'(H_1)$ se réécrit au nœud `q2` avec la règle `R3` en $H_2 =$ `p1 :c(q1 :s(q3 :s(m3 :a)),p1)`. Par conséquent, nous concluons qu'il existe une dérivation de surréduction compressante $H \overset{*}{\blacktriangleright\!\!\gg\!\rightsquigarrow}_\theta H_2$ où $\theta = (\sigma' \circ \sigma)_{|\mathcal{V}_H} =$ {`z` $\mapsto$ `m1 :s(m3 :a)`. θ est calculée par surréduction compressante de H en H_2. □

La cohérence et la complétude de la surréduction compressante sont définies par rapport à la réécriture de graphes.

Définition 5.3.3 (Cohérence)
Soit SP un cGRS et $\blacktriangleright$ une stratégie de compression. On dit que la relation de surréduction compressante $\blacktriangleright\!\rightsquigarrow$ est *cohérente* ssi pour tout terme-graphe admissible g, pour tout graphe

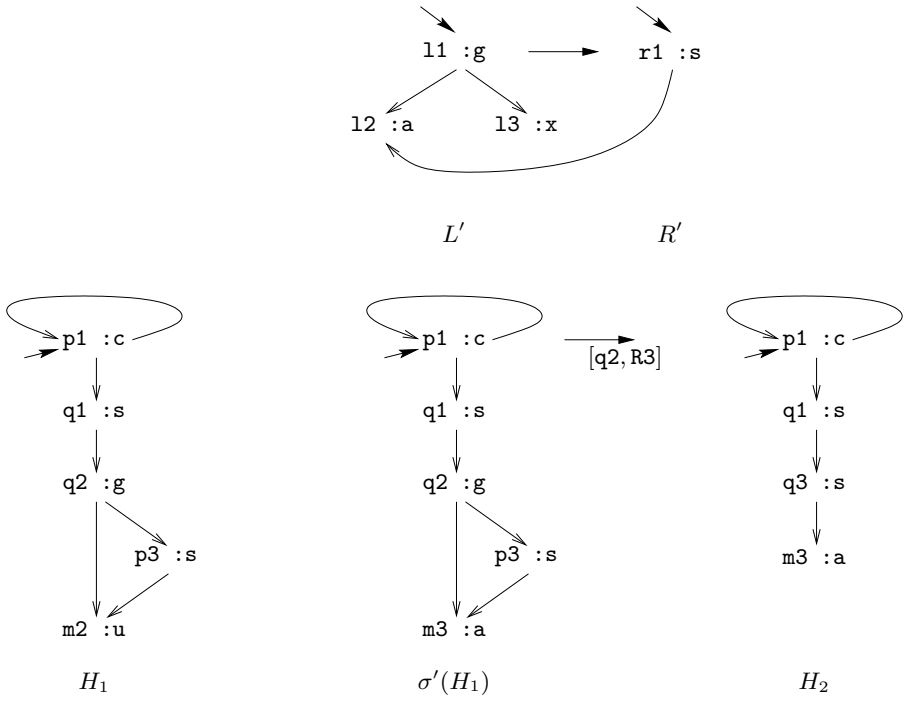


FIG. 5.8:

constructeur c et pour toute substitution θ tels que $g \blacktriangleright\!\!\overset{*}{\rightsquigarrow}_\theta c$, il existe un graphe constructeur s tel que $\theta(g) \overset{*}{\rightarrow} s$ et $s \doteq c$. ◇

Theorème 5.3.4 Soit SP un cGRS et $\blacktriangleright$ une stratégie de compression. La relation de surréduction compressante $\blacktriangleright\!\rightsquigarrow$ est cohérente. ◇

Nous montrons le Théorème 5.3.4 dans la Section 5.4.

Définition 5.3.5 (Complétude)
Soit SP un cGRS et $\blacktriangleright$ une stratégie de compression. On dit que la relation de surréduction compressante $\blacktriangleright\!\rightsquigarrow$ est *complète* ssi pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution constructeur[2] θ tels que $\theta(g) \overset{*}{\rightarrow} c$, il existe un terme-graphe constructeur s et une substitution σ tels que $g \blacktriangleright\!\!\overset{*}{\rightsquigarrow}_\sigma s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. ◇

Theorème 5.3.6 Soit SP un WAGRS et $\blacktriangleright$ une stratégie de compression. La relation de surréduction compressante la plus générale $\blacktriangleright\!\rightsquigarrow$ est complète. ◇

Nous prouvons le Théorème 5.3.6 dans la Section 5.5. Comme conséquence des deux théorèmes précédents, nous obtenons le résultat suivant :

Corollaire 5.3.7 Soit SP un WAGRS. Alors $\blacktriangleright\!\!\gg\!\!\rightsquigarrow$ et $\rightsquigarrow$ sont cohérentes et complètes. ◇

On remarque que nous restreignons le Théorème 5.3.6 et le Corollaire 5.3.7 à des WAGRS. En effet, les preuves de ces résultats nécessitent la confluence de la relation de réécriture (Théorème 3.3.4). En fait, on peut déduire de la preuve du Théorème 5.3.6 que $\rightsquigarrow$ est complète dans le cas général d'un cGRS. Mais ce n'est pas vrai pour $\blacktriangleright\!\!\gg\!\!\rightsquigarrow$ (et donc pour $\blacktriangleright\!\rightsquigarrow$), comme le montre l'exemple suivant :

[2]Une substitution constructeur θ est une substitution telle que $\theta(x)$ soit un terme-graphe constructeur pour tout $x \in \mathcal{D}\theta$.

Exemple 5.3.8 Les trois règles suivantes constituent un cGRS non confluent :

```
(R1) l1:A(l2:B(l3:X)) -> r1:C(r2:D,r3:D)
(R2) l1:D -> r1:E
(R3) l1:D -> r1:F
```

On suppose que `A` et `D` sont des opérations définies et `B`, `C`, `E`, `F` et `O` sont des symboles constructeurs. Soient $s =$ `n1 :A(n2 :Y)` et $t = \theta(s) =$ `n1 :A(n3 :B(n4 :O))` où $\theta = \{$`Y` $\mapsto$ `n3 :B(n4 :O)`$\}$ (cf. Figure 5.9). Le lecteur peut vérifier que $t \xrightarrow{*} t'$ avec $t' =$ `m1 :C(m4 :E,m5 :F)`. Considérons les dérivations de surréduction compressante maximale

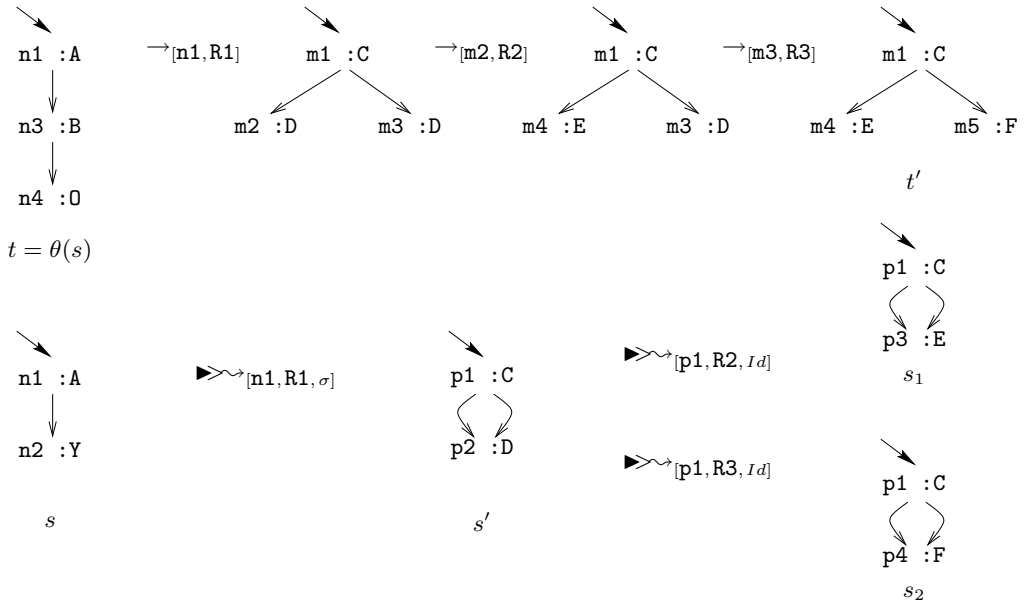


FIG. 5.9:

partant de s (cf. Figure 5.9). Tout d'abord, il existe une seule manière de commencer une dérivation de surréduction compressante maximale à partir de s : $s \blacktriangleright\!\!\rightsquigarrow_{[\mathtt{n1},\mathtt{R1},\sigma]} s'$ avec $\sigma = \{$`Y` $\mapsto$ `q1 :B(q2 :Z)`$\}$ et $s' =$ `p1 :C(p2 :D,p2)`. Ensuite, s' peut être surréduit de deux manières différentes en $s_1 =$ `p1 :C(p3 :E,p3)` ou en $s_2 =$ `p1 :C(p4 :F,p4)` qui sont des graphes constructeurs. Il est clair que $s_1 \not\dot{\leq} t'$ et que $s_2 \not\dot{\leq} t'$. Nous concluons donc que $\blacktriangleright\!\!\rightsquigarrow$ et, plus généralement, $\blacktriangleright\!\rightsquigarrow$ ne sont pas complètes dans le cas général d'un cGRS. □

Pour finir cette section, nous abordons une autre question intéressante concernant la complétude de la relation de surréduction *décompressante* de graphes. On dit qu'un graphe g_1 se décompresse en un graphe g_2, noté $g_1 \lhd g_2$, ssi g_2 se compacte en g_1 (i.e., $g_2 \rhd g_1$). On définit un pas de surréduction décompressante de graphes en posant $g_1 \blacktriangleleft\!\rightsquigarrow_{[p,R,\sigma]} g_2$ ssi (1) il existe un graphe g' tel que $\blacktriangleright(g') = \sigma(g_1)$ avec le $\mathcal{V}$-homomorphisme $h : g' \to \sigma(g_1)$ (c'est-à-dire $\sigma(g_1) \lhd g'$), (2) il existe un nœud $q \in \mathcal{N}_{g'}$ tel que $h(q) = p$ et (3) $g' \to_{[q,R]} g_2$. L'exemple suivant montre que la surréduction décompressante n'est pas complète dans le cadre des WAGRS :

Exemple 5.3.9 Considérons la simple règle `(R) l1 :A(l2 :O)` $\to$ `l2 :O` où `A` est une opération définie et `O` et `C` sont des symboles constructeurs. Soit $s =$ `n :C(`$\underline{\mathtt{A(O)}}$`,n)`. A partir

de s, on peut construire une dérivation de surréduction décompressante infinie :

$$
\begin{array}{rcl}
s & \lhd & \texttt{m :C(}\underline{\texttt{A(O)}}\texttt{,C(A(O),m))} \\
 & \to & \texttt{m :C(O,C(}\underline{\texttt{A(O)}}\texttt{,m))} \\
 & \lhd & \texttt{p :C(O,C(}\underline{\texttt{A(O)}}\texttt{,C(O,C(A(O),p))))} \\
 & \to & \texttt{p :C(O,C(O,C(O,C(}\underline{\texttt{A(O)}}\texttt{,p))))} \\
 & \lhd & \texttt{q :C(O,C(O,C(O,C(}\underline{\texttt{A(O)}}\texttt{,C(O,C(O,C(O,C(A(O),q))))))))} \\
 & \to & \texttt{q :C(O,C(O,C(O,C(O,C(O,C(O,C(O,C(}\underline{\texttt{A(O)}}\texttt{,q))))))))} \\
 & \lhd & \ldots
\end{array}
$$

Pourtant, $s \overset{*}{\leadsto}_{Id} \texttt{n :C(O,C(O,n))}$. □

5.4 Preuve de cohérence de ▶⇝

Dans cette section, nous prouvons le théorème suivant :

Théorème 5.3.4 Soit SP un cGRS et $\blacktriangleright$ une stratégie de compression. La relation de surréduction compressante $\blacktriangleright\!\!\leadsto$ est cohérente, i.e., pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution θ tels que $g \overset{*}{\blacktriangleright\!\!\leadsto}_\theta c$, il existe un graphe constructeur s tel que $\theta(g) \overset{*}{\to} s$ et $s \doteq c$. □

La preuve du Théorème 5.3.4 repose sur la proposition suivante :

Proposition 5.4.1 Soient SP un cGRS et g_1 et g_2 deux termes-graphes admissibles tels que $g_1 \rhd g_2$. Si il existe un graphe g_2' tel que $g_2 \to g_2'$ (resp. $g_2 \overset{*}{\to} g_2'$), alors il existe un terme-graphe admissible g_1' tel que $g_1 \overset{+}{\to} g_1'$ (resp. $g_1 \overset{*}{\to} g_1'$) et $g_1' \rhd g_2'$ (cf. Figure 5.10). ◇

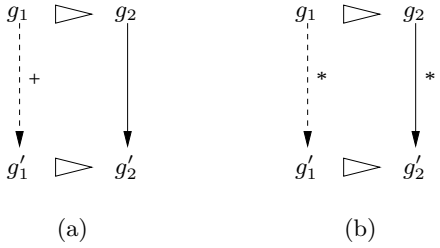


FIG. 5.10:

Preuve : Immédiat avec la définition de $\rhd$ et la Proposition 3.5.3. □

Preuve du Théorème 5.3.4 : Par induction sur la longueur n de la dérivation de surréduction compressante. Le cas de base $n = 0$ est évident. Soit $n \geq 0$. Supposons que le théorème soit vrai pour toutes les dérivations de surréduction compressante de longueur n. Soit $g_1 \blacktriangleright\!\!\leadsto_\sigma g_2 \overset{n}{\blacktriangleright\!\!\leadsto}_\theta c$ une dérivation de surréduction compressante de longueur $n+1$. $g_2 \overset{n}{\blacktriangleright\!\!\leadsto}_\theta c$ est une dérivation de surréduction compressante de longueur n. Donc par hypothèse d'induction, il existe un terme-graphe constructeur s tel que $\theta(g_2) \overset{*}{\to} s$ et $s \doteq c$. Comme $g_1 \blacktriangleright\!\!\leadsto_\sigma g_2$, il existe un graphe g_1' tel que $\sigma(g_1) \rhd g_1'$ et $g_1' \to g_2$. Il est clair que $\theta(\sigma(g_1)) \rhd \theta(g_1')$ et $\theta(g_1') \to \theta(g_2)$. Puisque $\theta(\sigma(g_1)) \rhd \theta(g_1')$ et $\theta(g_1') \to \theta(g_2) \overset{*}{\to} s$, nous déduisons de la Proposition 5.4.1 qu'il existe un graphe s' tel que $\theta(\sigma(g_1)) \overset{*}{\to} s'$ et $s' \rhd s$. Comme $s' \rhd s$ et $s \doteq c$, on infère que $s' \doteq c$. Donc il existe un terme-graphe constructeur s' tel que $\theta(\sigma(g_1)) \overset{*}{\to} s'$ et $s' \doteq c$. □

5.5 Preuve de complétude de $\blacktriangleright\!\!\rightsquigarrow$

Nous consacrons cette section à la preuve du théorème suivant :

Théorème 5.3.6 Soient SP un WAGRS et $\blacktriangleright$ une stratégie de compression. Alors la relation de surréduction compressante la plus générale $\blacktriangleright\!\!\rightsquigarrow$ est complète, i.e., pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution constructeur θ tels que $\theta(g) \stackrel{*}{\rightarrow} c$, il existe un graphe constructeur s et une substitution σ tels que $g \stackrel{*}{\blacktriangleright\!\!\rightsquigarrow_\sigma} s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta$ $[\mathcal{V}_g]$. □

5.5.1 Lemme de Hullot

Comme pour la complétude de la surréduction la plus générale dans le cadre des termes du premier ordre, le Théorème 5.3.6 repose sur un lemme qui relie la réécriture de graphes et la surréduction de graphes. Nous l'appelons *lemme de Hullot*, en référence au premier chercheur qui a utilisé cette technique de preuve pour montrer la complétude de la relation de surréduction de termes [Hul80a, Hul80b]. Ce nom n'est pas standart. Nous l'utilisons à défaut d'une "bonne" traduction de *lifting lemma* en français).

Dans cette partie, nous établissons progressivement l'énoncé de ce lemme. Nous partons avec un énoncé "simpliste", puis nous donnons plusieurs contre-exemples nous permettant de raffiner cet énoncé. Nous répétons ce processus plusieurs fois, jusqu'à obtenir le véritable lemme du Hullot qui se trouve à la fin du paragraphe (Lemme 5.5.6 page 98). Cette manière de procéder nous permet d'introduire progressivement les notions qui sont nécessaires pour énoncer le lemme.

Pour simplifier cette analyse, nous supposons que la stratégie de compression est la fonction identité. Nous travaillons donc avec la relation de surréduction la plus générale classique de graphes $\rightsquigarrow$. Une première expression du lemme de Hullot qu'un lecteur pourrait imaginer serait sans doute la suivante :

Lemme Soient SP un WAGRS, s_1 et t_1 deux termes-graphes admissibles et θ_1 une substitution constructeur tels que $t_1 \sim \theta_1(s_1)$. Supposons qu'il existe un graphe t_2, un nœud fonctionnel $q \in \mathcal{N}_{t_1}$ et une règle $l \rightarrow r$ tels que $t_1 \rightarrow_{[q, l \rightarrow r]} t_2$.
Alors il existe un terme-graphe admissible s_2, un nœud fonctionnel $p \in \mathcal{N}_{s_1}$ et un unificateur le plus général σ de $s_{1|p}$ par rapport à l tels que $s_1 \rightsquigarrow_{[p, l \rightarrow r, \sigma]} s_2$. De plus, il existe une substitution constructeur θ_2 telle que $t_2 \sim \theta_2(s_2)$ et $\theta_2 \circ \sigma \doteq \theta_1$ $[\mathcal{V}_{s_1}]$ (cf. Figure 5.11). □

$$\begin{array}{ccc} t_1 & \rightarrow_{[q,\,R]} & t_2 \\ \sim & & \sim \\ \theta_1(s_1) & & \theta_2(s_2) \\ \\ s_1 & \rightsquigarrow_{[p,\,R,\,\sigma]} & s_2 \end{array}$$

FIG. 5.11:

Mais ce lemme est faux, comme le montre l'exemple suivant :

Exemple 5.5.1 Soient s_1 = `n1 :A(n2 :Z,n2)` un terme-graphe admissible, θ_1 une substitution constructeur telle que θ_1(`Z`) = `p :B(p)` et t_1 le terme-graphe tel que t_1 =

$\theta_1(s_1)$ = `n1 :B(p :A(p),p)` (cf. Figure 5.12). Considérons la règle $R = l \to r$ où l =

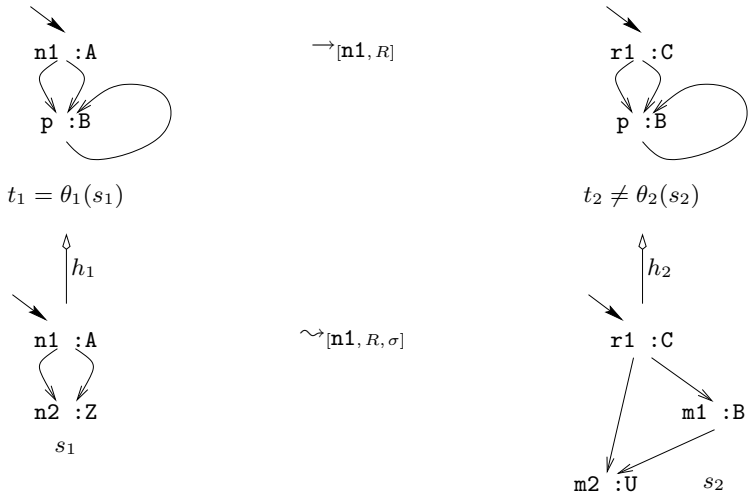


FIG. 5.12:

`l1 :A(l2 :B(l3 :X),l4 :Y)` et r = `r1 :C(l3 :X,l4 :Y)`. Le lecteur peut vérifier que $t_1 \to_{[\mathtt{n1}, R]} t_2$ avec t_2 = `r1 :C(p :B(p),p)`. D'un autre côté, la substitution $\sigma = \{\mathtt{Z} \mapsto$ `m1 :B(m2 :U)`$\}$ est un unificateur le plus général de s_1 par rapport à l. Donc s_1 se surréduit à la racine avec la règle R et la substitution σ en s_2 = `r1 :C(m2 :U,m1 :B(m2))`. Pourtant, il n'existe pas de substitution θ_2 telle que $\theta_2(s_2)$ soit égal à t_2 au renommage des nœuds près. En fait, il existe une substitution θ_2 telle que $\theta_2(s_2)$ soit *bisimilaire* à t_2. □

Nous constatons avec l'exemple ci-dessus qu'il n'existe pas de substitution constructeur θ_2 telle que $t_2 \sim \theta_2(s_2)$ mais qu'il existe une substitution constructeur θ_2 telle que $t_2 \doteq \theta_2(s_2)$. On est donc tenté de proposer le lemme de Hullot suivant :

Lemme Soient SP un WAGRS, s_1 et t_1 deux termes-graphes admissibles et θ_1 une substitution constructeur tels que $t_1 \doteq \theta_1(s_1)$. Supposons qu'il existe un graphe t_2, un nœud fonctionnel $q \in \mathcal{N}_{t_1}$ et une règle $l \to r$ tels que $t_1 \to_{[q, l \to r]} t_2$.
Alors il existe un terme-graphe admissible s_2, un nœud fonctionnel $p \in \mathcal{N}_{s_1}$ et un unificateur le plus général σ de $s_{1|p}$ par rapport à l tels que $s_1 \rightsquigarrow_{[p, l \to r, \sigma]} s_2$.
De plus, il existe une substitution constructeur θ_2 telle que $t_2 \doteq \theta_2(s_2)$ et $\theta_2 \circ \sigma \doteq \theta_1$ $[\mathcal{V}_{s_1}]$ (cf. Figure 5.13). □

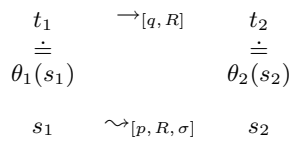


FIG. 5.13:

Mais ce lemme est faux, lui aussi :

Exemple 5.5.2 Soit s_1 = `n1 :A(n2 :B(n3 :Z),n4 :B(n3))` un terme-graphe admissible, θ_1 une substitution constructeur telle que θ_1(`Z`) = `n5 :O` et t_1 = `m1 :A(m2 :B(m3 :O),m2)`

un terme-graphe tel que $t_1 \doteq \theta_1(s_1)$ (cf. Figure 5.14). Considérons la règle $R =$ `l1 :B(l2 :O)` $\rightarrow$ `l2 :O`. Le lecteur peut vérifier que $t_1 \rightarrow_{[\texttt{m2}, R]} t_2$ avec $t_2 =$ `n1 :A(m3 :O,m3)`. D'un autre côté, s_1 est surréductible aux nœuds `m2` et `m4` mais il est *impossible* de construire un graphe s_2 tel que s_1 se surréduise *en un seul pas* en s_2 et $t_2 \doteq \theta_2(s_2)$. □

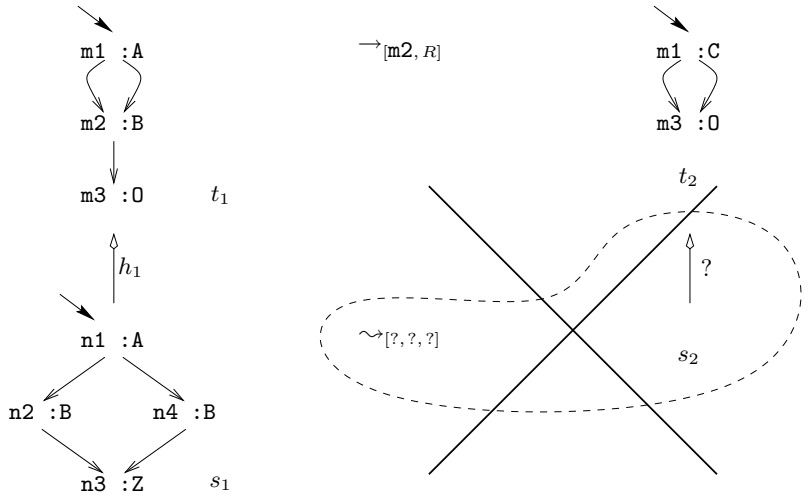


FIG. 5.14:

On constate avec l'Exemple 5.5.1 (resp. 5.5.2) que si on suppose $t_1 \sim \theta_1(s_1)$ (resp. $t_1 \doteq \theta_1(s_1)$), alors on ne peut pas montrer que $t_2 \sim \theta_2(s_2)$ (resp. $t_2 \doteq \theta_2(s_2)$). Néanmoins, on remarque dans l'Exemple 5.5.1 qu'il existe deux homomorphismes $h_1 : s_1 \rightarrow t_1$ et $h_2 : s_2 \rightarrow t_2$ qu'on pourrait essayer d'utiliser pour établir le lemme de Hullot.

Dans l'Exemple 5.5.2, il existe aussi un homomorphisme $h_1 : s_1 \rightarrow t_1$ mais pas d'homomorphisme $h_2 : s_2 \rightarrow t_2$. Notre idée n'est pas mauvaise pour autant : en fait, les homomorphismes qui nous intéressent ne doivent pas "coller" deux nœuds fonctionnels de s_1 dans t_1 (ou de s_2 dans t_2). C'est le cas des homomorphismes $h_1 : s_1 \rightarrow t_1$ et $h_2 : s_2 \rightarrow t_2$ de l'Exemple 5.5.1. Mais ce n'est pas le cas de l'homomorphisme $h_1 : s_1 \rightarrow t_1$ de l'Exemple 5.5.2 puisqu'il "colle" les nœuds fonctionnels `n2` et `n4` de s_1 en seul nœud `m2` dans t_1.

On appelle ces homomorphismes particuliers des $\mathcal{D}$-monomorphismes :

Définition 5.5.3 ($\mathcal{D}$-monomorphisme)
Soit SP un cGRS, g_1 et g_2 deux graphes admissibles et $h : g_1 \rightarrow g_2$ un homomorphisme.
On dit que h est un *$\mathcal{D}$-monomorphisme* ssi h est injectif sur tous les nœuds fonctionnels de g_1 : Pour tout nœud $n_1, n_2 \in \mathcal{N}_{g_1}$, si $\mathcal{L}_{g_1}(n_1) \in \mathcal{D}$, $\mathcal{L}_{g_1}(n_2) \in \mathcal{D}$ et $h(n_1) = h(n_2)$, alors $n_1 = n_2$. ◇

Avec de tels homomorphismes, nous proposons le lemme suivant :

Lemme Soient SP un WAGRS, s_1 et t_1 deux termes-graphes admissibles et $h_1 : s_1 \rightarrow t_1$ un $\mathcal{D}$-monomorphisme tel que σ_{h_1} soit une substitution constructeur. Supposons qu'il existe un graphe t_2, un nœud fonctionnel $q \in \mathcal{N}_{t_1}$ et une règle $l \rightarrow r$ tels que $t_1 \rightarrow_{[q, l \rightarrow r]} t_2$.
Alors il existe un terme-graphe admissible s_2, un nœud fonctionnel $p \in \mathcal{N}_{s_1}$ et un unificateur le plus général σ de $s_{1|p}$ par rapport à l tels que $h_1(p) = q$ et $s_1 \rightsquigarrow_{[p, l \rightarrow r, \sigma]} s_2$.
De plus, il existe un $\mathcal{D}$-monomorphisme $h_2 : s_2 \rightarrow t_2$ tel que σ_{h_2} soit une substitution constructeur et $\sigma_{h_2} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{s_1}]$ (cf. Figure 5.15). □

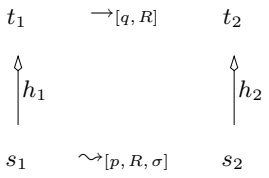


FIG. 5.15:

Mais ce lemme est faux, une fois de plus :

Exemple 5.5.4 Soit s_1 = `n1 :A(n2 :X,n3 :Y)` et t_1 = `m1 :A(m2 :O,m2)` deux termes-graphes admissibles (cf. Figure 5.16). Il est clair qu'il existe un $\mathcal{D}$-monomorphisme $h_1 : s_1 \rightarrow t_1$; la substitution σ_{h_1} est telle que σ_{h_1}(`X`) = σ_{h_1}(`Y`) = `m2 :O`. Considérons la règle R =

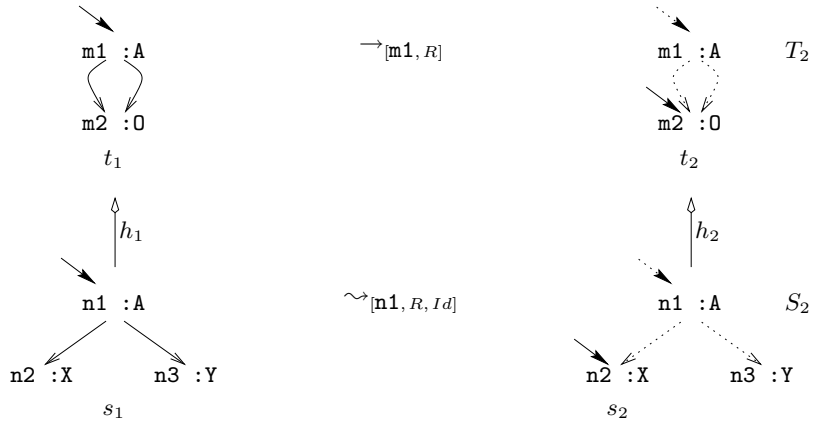


FIG. 5.16:

`l1 :A(l2 :U,l3 :V)` → `l2 :U` (où `U` et `V` sont des variables). Le lecteur peut vérifier que $t_1 \rightarrow_{[\mathtt{m1},\,R]} t_2$ avec t_2 = `m2 :O` et $s_1 \rightsquigarrow_{[\mathtt{n1},\,R,\,\sigma]} s_2$ avec $\sigma = Id$ et s_2 = `n2 :X`. On constate sur la Figure 5.16 qu'il existe bien un $\mathcal{D}$-monomorphisme $h_2 : s_2 \rightarrow t_2$; la substitution σ_{h_2} est telle que σ_{h_2}(`X`) = `m2 :O`. Nous obtenons $\sigma_{h_2} \circ \sigma = \sigma_{h_2}$ puisque $\sigma = Id$. Comme $\sigma_{h_1} = \{\mathtt{X} \mapsto \mathtt{m2\ :O}, \mathtt{Y} \mapsto \mathtt{m2\ :O}\}$ et $\sigma_{h_2} = \{\mathtt{X} \mapsto \mathtt{m2\ :O}\}$, il est clair que $\sigma_{h_2} \circ \sigma \not\doteq \sigma_{h_1}\ [\mathcal{V}_{s_1}]$. □

Dans l'exemple précédent, la relation $\sigma_{h_2} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{s_1}]$ est fausse car nous avons "perdu" la variable `Y` (donc la valeur de σ_{h_1}(`Y`)) pendant que nous calculions s_2. En fait, `Y` a été éliminée de s_2 lorsque nous avons éliminé les nœuds et les variables "inutiles" de s_2 (*garbage collection*).

Si nous ne faisons pas cette étape d'élimination pendant le calcul de s_2 et de t_2, nous obtenons les graphes S_2 et T_2 de la Figure 5.16. Il est clair qu'il existe un nouvel homomorphisme $h'_2 : S_2 \rightarrow T_2$. De plus, nous obtenons $\sigma_{h'_2} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{s_1}]$.

Nous proposons donc le lemme (correct !) suivant :

Lemme 5.5.5 Soient SP un WAGRS, s_1 et t_1 deux termes-graphes admissibles, S_1 et T_1 deux graphes admissibles et $h_1 : S_1 \rightarrow T_1$ un $\mathcal{D}$-monomorphisme tels que s_1 soit un sous-graphe de S_1, t_1 soit un sous-graphe de T_1, $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur. Supposons qu'il existe un graphe t_2, un nœud fonctionnel $q \in \mathcal{N}_{t_1}$ et une règle $l \rightarrow r$ tels que $t_1 \rightarrow_{[q,\,l \rightarrow r]} t_2$ (cf. Figure 5.17). Alors :

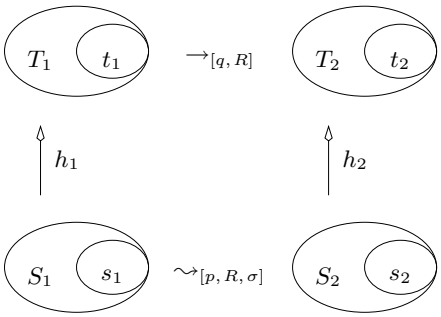


FIG. 5.17:

1. Il existe une terme-graphe admissible s_2, un nœud fonctionnel p de s_1 et un unificateur le plus général σ de $s_{1|p}$ par rapport à l tels que $s_1 \rightsquigarrow_{[p,l\rightarrow r,\sigma]} s_2$.
2. Il existe deux graphes admissibles S_2 et T_2 et un $\mathcal{D}$-monomorphisme $h_2 : S_2 \rightarrow T_2$ tels que s_2 soit un sous-graphe de S_2, t_2 soit un sous-graphe de T_2, $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ soit une substitution constructeur.
3. $\sigma_{h_2} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$.

◇

Le lemme ci-dessus est un cas particulier du lemme suivant qui, lui, prend en compte une stratégie de compression ▶ :

Lemme 5.5.6 (Lemme de Hullot)
Soient SP un cGRS, ▶ une stratégie de compression, s_1 et t_1 deux termes-graphes admissibles, S_1 et T_1 deux graphes admissibles et $h_1 : S_1 \rightarrow T_1$ un $\mathcal{D}$-monomorphisme tels que s_1 soit un sous-graphe de S_1, t_1 soit un sous-graphe de T_1, $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur. Supposons qu'il existe un graphe constructeur c et une dérivation de réécriture $t_1 \xrightarrow{+} c$ commençant par un pas de réécriture au nœud q avec la règle $R = l \rightarrow r$ (i.e., $t_1 \rightarrow_{[q,R]} u \xrightarrow{*} c$) (cf. Figure 5.18). Alors :

1. Il existe une terme-graphe admissible s_2, un nœud fonctionnel p de s_1 et un unificateur le plus général σ de $s_{1|p}$ par rapport à l tels que $s_1 \blacktriangleright\!\!\rightsquigarrow_{[p,l\rightarrow r,\sigma]} s_2$.
2. Il existe deux termes-graphes admissibles t'_1 et t_2 tels que $t_1 \rhd t'_1$ avec le $\mathcal{V}$-homomorphisme $\gamma : t_1 \rightarrow t'_1$ et $t'_1 \rightarrow_{[\gamma(q),l\rightarrow r]} t_2$.
3. Il existe deux graphes admissibles S_2 et T_2 et un $\mathcal{D}$-monomorphisme $h_2 : S_2 \rightarrow T_2$ tels que s_2 soit un sous-graphe de S_2, t_2 soit un sous-graphe de T_2, $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ soit une substitution constructeur.
4. Pour tout sous-graphe constructeur u de S_1, $\sigma(u)$ est bisimilaire à un sous-graphe constructeur de S_2.
5. $\sigma_{h_2} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$.

◇

5.5.2 Preuve du lemme de Hullot

Dans cette partie, nous nous attaquons à la preuve du Lemme 5.5.6. Nous utiliserons les deux propositions suivantes :

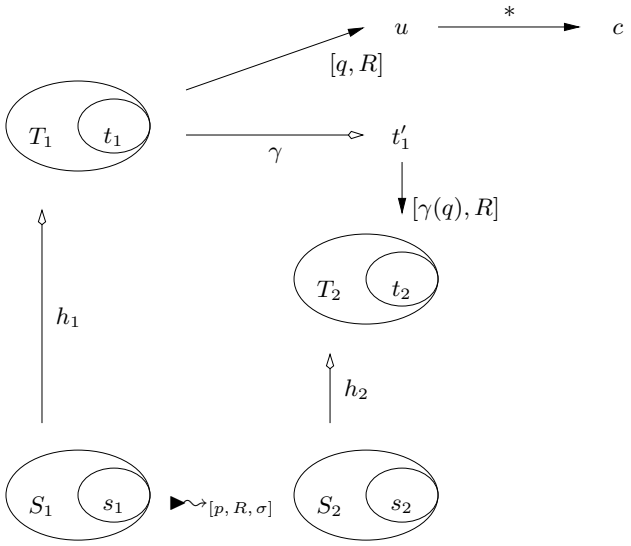


FIG. 5.18:

Proposition 5.5.7 Soit SP un cGRS, s et t deux termes-graphes admissibles et $h : s \to t$ un $\mathcal{D}$-monomorphisme tels que σ_h soit une substitution constructeur. Supposons qu'il existe un terme-graphe admissible s' tel que $s \rhd s'$ (cf. Figure 5.19). Alors, il existe un terme-graphe

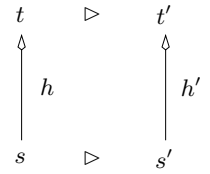


FIG. 5.19:

admissible t' tel que $t \rhd t'$. De plus, il existe un $\mathcal{D}$-monomorphisme $h' : s' \to t'$ tel que $\sigma_{h'}$ soit une substitution constructeur. ◇

Preuve : Comme $s \rhd s'$, il existe un $\mathcal{V}$-homomorphisme $\upsilon : s \to s'$. Par hypothèse, $h(s) = t$, donc $\sigma_h(s) \doteq t$. De plus, $\upsilon : s \to s'$ est un $\mathcal{V}$-homomorphisme, donc $s \doteq s'$. Comme $\sigma_h(s) \doteq t$ et $s \doteq s'$, nous inférons que $\sigma_h(s') \doteq t$. Par conséquent, s' et t sont unifiables : il existe un terme-graphe admissible t' et deux homomorphismes $\mu : t \to t'$ et $\delta : s' \to t'$ tels que pour tout terme-graphe admissible g et pour tout homomorphisme $\phi_1 : t \to g$ et $\phi_2 : s' \to g$, il existe un homomorphisme $\eta : t' \to g$.

Comme $\sigma_h(s') \doteq t$, aucune variable de t n'est affectée par unification. Donc nous déduisons que $\mu : t \to t'$ est un $\mathcal{V}$-homomorphisme, i.e., $t \rhd t'$. Nous avons choisi t' de sorte que $\mu \cup h' : (t \oplus s') \to (t' \oplus t')$ soit un unificateur le plus général de t et s'. Si h' n'était pas un $\mathcal{D}$-monomorphisme, alors h' "collerait" deux nœuds fonctionnels n_1 et n_2 de s'. Mais alors, nous pourrions construire un graphe g et deux homomorphismes $\phi_1 : t \to g$ et $\phi_2 : s' \to g$ qui ne "collent" pas n_1 et n_2. Par conséquent, il n'existerait pas d'homomorphisme $\eta : t' \to g$, ce qui est faux par construction. Donc h' est un $\mathcal{D}$-monomorphisme. □

Proposition 5.5.8 Soit SP un cGRS, g_1 et g_2 deux graphes admissibles et $h : g_1 \to g_2$ un $\mathcal{D}$-monomorphisme. Soient n_1 un nœud fonctionnel de g_1, p_1 un nœud de g_1 de même

sorte que n_1 et ρ_1 la redirection de pointeurs telle que $\rho_1(n_1) = p_1$ et $\rho_1(n) = n$ pour tout $n \neq n_1$. Soient n_2 un nœud fonctionnel de g_2, p_2 un nœud de g_2 de même sorte que n_2 et ρ_2 la redirection de pointeurs telle que $\rho_2(n_2) = p_2$ et $\rho_2(n) = n$ pour tout $n \neq n_2$. Supposons que $h(n_1) = n_2$ et $h(p_1) = p_2$. Alors :

1. $h(\rho_1(n)) = \rho_2(h(n))$ pour tout $n \in \mathcal{N}_{g_1}$.
2. h est un $\mathcal{D}$-monomorphisme allant de $\rho_1(g_1)$ vers $\rho_2(g_2)$.

◇

Preuve :
Point 1 : Nous prouvons que $\rho_2(h(q)) = h(\rho_1(q))$ pour tout nœud $q \in \mathcal{N}_{g_1}$. Il y a deux cas à étudier selon que $q \neq n_1$ ou que $q = n_1$. Premier cas, si $q \neq n_1$, alors par définition de ρ_1, $\rho_1(q) = q$ et donc $h(\rho_1(q)) = h(q)$. Comme $q \neq n_1$ et h est injectif sur n_1, $h(q) \neq h(n_1)$, donc $h(q) \neq n_2$. Aussi, par définition de ρ_2, $\rho_2(h(q)) = h(q)$. Par conséquent, $h(\rho_1(q)) = \rho_2(h(q))$. Deuxième cas, si $q = n_1$, alors par définition de ρ_1, $\rho_1(q) = \rho_1(n_1) = p_1$ et donc $h(\rho_1(q)) = h(p_1) = p_2$. Or $p_2 = \rho_2(n_2)$ et $n_2 = h(n_1) = h(q)$. Donc $h(\rho_1(q)) = p_2 = \rho_2(h(q))$.
Point 2 : h est un $\mathcal{D}$-monomorphisme allant de g_1 vers g_2. Nous prouvons que h est aussi un $\mathcal{D}$-monomorphisme allant de $\rho_1(g_1)$ vers $\rho_2(g_2)$. Tout d'abord, h est une fonction allant de $\mathcal{N}_{g_1}$ vers $\mathcal{N}_{g_2}$. Comme $\mathcal{N}_{\rho_1(g_1)} = \mathcal{N}_{g_1}$ et $\mathcal{N}_{\rho_2(g_2)} = \mathcal{N}_{g_2}$, h est aussi une fonction allant de $\mathcal{N}_{\rho_1(g_1)}$ vers $\mathcal{N}_{\rho_2(g_2)}$. De plus, h préserve la fonction d'étiquetage de $\rho_1(g_1)$. En effet, h préserve la fonction d'étiquetage de g_1, $\mathcal{L}_{\rho_1(g_1)} = \mathcal{L}_{g_1}$ et $\mathcal{L}_{\rho_2(g_2)} = \mathcal{L}_{g_2}$. Nous devons ensuite montrer que h préserve la fonction de successeur de $\rho_1(g_1)$, c'est-à-dire que $\mathcal{S}_{\rho_2(g_2)}(h(p)) = h(\mathcal{S}_{\rho_1(g_1)}(p))$ pour tout nœud $p \in \mathcal{N}_{\rho_1(g_1)}$. D'une part, $\mathcal{S}_{\rho(g)}(q) = \rho(\mathcal{S}_g(q))$ pour tout graphe g, pour toute redirection de pointeurs ρ et pour tout nœud $q \in \mathcal{N}_g$. D'autre part, $\mathcal{N}_{\rho_1(g_1)} = \mathcal{N}_{g_1}$. Donc nous cherchons à montrer que $\rho_2(\mathcal{S}_{g_2}(h(p))) = h(\rho_1(\mathcal{S}_{g_1}(p)))$ pour tout nœud $p \in \mathcal{N}_{g_1}$. h est un homomorphisme allant de g_1 vers g_2, donc h préserve la fonction de successeur de $g_1 : \mathcal{S}_{g_2}(h(p)) = h(\mathcal{S}_{g_1}(p))$. Aussi, nous devons vérifier que $\rho_2(h(\mathcal{S}_{g_1}(p))) = h(\rho_1(\mathcal{S}_{g_1}(p)))$ pour tout nœud $p \in \mathcal{N}_{g_1}$. Grâce au point 1 de la proposition, $\rho_2(h(q)) = h(\rho_1(q))$ pour tout nœud $q \in \mathcal{N}_{g_1}$. Donc nous déduisons la propriété qu'il faut vérifier, à savoir, $\rho_2(h(\mathcal{S}_{g_1}(p))) = h(\rho_1(\mathcal{S}_{g_1}(p)))$ pour tout nœud $p \in \mathcal{N}_{g_1}$. Nous concluons donc que h préserve la fonction de successeur de $\rho_1(g_1)$. Pour finir, nous devons vérifier que $h(\mathcal{R}\text{oots}_{\rho_1(g_1)}) = \mathcal{R}\text{oots}_{\rho_2(g_2)}$, c'est-à-dire $h(\rho_1(\mathcal{R}\text{oots}_{g_1}).n_1) = \rho_2(\mathcal{R}\text{oots}_{g_2}).n_2$, ce qui est évident puisque $h(n_1) = n_2$ et $h(\rho_1(\mathcal{R}\text{oots}_{g_1})) = \rho_2(h(\mathcal{R}\text{oots}_{g_1})) = \rho_2(\mathcal{R}\text{oots}_{g_2})$, d'après le point 1 de la proposition. □

Nous sommes prêts pour montrer le lemme de Hullot :

Preuve du Lemme 5.5.6 :
Construction de p : t_1 se réécrit au nœud q avec la règle $l \to r$. Donc il existe un filtre $\mu : l \to t_{1|q}$. l est un motif, donc $\mathcal{R}\text{oot}_l$ est un nœud fonctionnel de l. Comme $\mu : l \to t_{1|q}$ est un homomorphisme, nous inférons que q est aussi un nœud fonctionnel de t_1. $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ est une substitution constructeur, donc il existe au moins un nœud fonctionnel p de s_1 tel que $h_1(p) = q$. h_1 est un $\mathcal{D}$-monomorphisme, donc deux nœuds distincts de s_1 étiquetés par la même opération définie ont deux images distinctes par h_1. Aussi, nous concluons que p est le seul nœud de s_1 tel que $h_1(p) = q$.
Construction de σ : $\mu : l \to t_{1|q}$ et $h_{1|p} : s_{1|p} \to t_{1|q}$ sont tous les deux des homomorphismes, donc $h_{1|p} \cup \mu : (s_{1|p} \oplus l) \to (t_{1|q} \oplus t_{1|q})$ est un homomorphisme tel que $(h_{1|p} \cup \mu)(s_{1|p}) = (h_{1|p} \cup \mu)(l) = t_{1|q}$. Par conséquent, $s_{1|p}$ et l sont unifiables. Comme l est un motif, nous inférons qu'il existe un unificateur le plus général σ de $s_{1|p}$ par rapport à l et un homomorphisme $\upsilon : (s_{1|p} \oplus l) \to (\sigma(s_{1|p}) \oplus \sigma(s_{1|p}))$ tels que $\upsilon(s_{1|p}) = \upsilon(l) = \sigma(s_{1|p})$. De plus, comme $\upsilon : (s_{1|p} \oplus l) \to (\sigma(s_{1|p}) \oplus \sigma(s_{1|p}))$ est un unificateur le plus général de $s_{1|p}$ et l et $h_{1|p} \cup \mu : (s_{1|p} \oplus l) \to (t_{1|q} \oplus t_{1|q})$ est un simple unificateur de $s_{1|p}$ et l, il existe un homomorphisme $\delta : \sigma(s_{1|p}) \to t_{1|q}$ tel que $\delta \circ \upsilon_{|p} = h_{1|p}$ et $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_l} = \mu$ (cf. Figure 5.20).
Construction de s_2, t_1' et t_2 : Pour construire s_2 et t_2, on commence par étendre $\delta : \sigma(s_{1|p}) \to t_{1|q}$ au graphe $\sigma(s_1)$ tout entier. Soit δ_1 la fonction allant de $\mathcal{N}_{\sigma(s_1)}$ vers $\mathcal{N}_{t_1}$ telle que $\delta_1(n) = \delta(n)$ pour tout $n \in \mathcal{N}_{\sigma(s_{1|p})}$ et $\delta_1(n) = h_1(n)$ pour tout $n \in (\mathcal{N}_{\sigma(s_1)} - \mathcal{N}_{\sigma(s_{1|p})})$. Il est clair que δ_1 est un

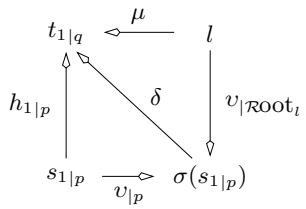


FIG. 5.20:

homomorphisme allant de $\sigma(s_1)$ vers t_1 et que $\delta = \delta_{1|p}$. Soit υ_1 l'homomorphisme allant de s_1 vers $\sigma(s_1)$. Nous avons $\delta_1 \circ \upsilon_1 = h_{1|\mathcal{R}\mathrm{oot}_{s_1}}$ (cf. A dans la Figure 5.21). Cette égalité nous permet de déduire

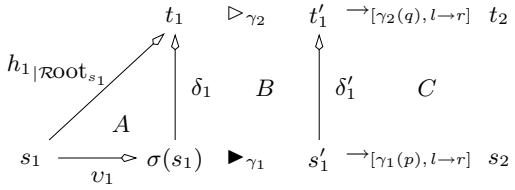


FIG. 5.21:

que δ_1 est un $\mathcal{D}$-monomorphisme allant de $\sigma(s_1)$ vers t_1 tel que σ_{δ_1} soit une substitution constructeur ; si ce n'était pas le cas, $h_{1|\mathcal{R}\mathrm{oot}_{s_1}} : s_1 \to t_1$ ne serait pas un $\mathcal{D}$-monomorphisme tel que $\sigma_{(h_{1|\mathcal{R}\mathrm{oot}_{s_1}})}$ soit une substitution constructeur (puisque $\delta_1 \circ \upsilon_1 = h_{1|\mathcal{R}\mathrm{oot}_{s_1}}$), ce qui est contraire aux hypothèses.

Nous effectuons maintenant le pas de compression de $\sigma(s_1)$ en s'_1, ce qui nous permet de définir le pas de compression de t_1 en t'_1. Soit s'_1 le terme-graphe admissible tel que $\blacktriangleright(\sigma(s_1)) = s'_1$. Par définition de $\blacktriangleright$, il existe un $\mathcal{V}$-homomorphisme $\gamma_1 : \sigma(s_1) \to s'_1$. De plus, δ_1 est un $\mathcal{D}$-monomorphisme allant de $\sigma(s_1)$ vers t_1 tel que σ_{δ_1} soit une substitution constructeur. Donc d'après la Proposition 5.5.7, il existe un terme-graphe admissible t'_1, un $\mathcal{V}$-homomorphisme $\gamma_2 : t_1 \to t'_1$ et un $\mathcal{D}$-monomorphisme $\delta'_1 : s'_1 \to t'_1$ tels que $t_1 \rhd t'_1$ et $\sigma_{\delta'_1}$ soit une substitution constructeur (cf. B dans la Figure 5.21).

Nous pouvons désormais construire t_2 par réécriture de t'_1 et s_2 par réécriture de s'_1 (cf. C dans la Figure 5.21). Par composition, $\gamma_{2|q} \circ \mu$ est un homomorphisme allant de l vers $t'_{1|\gamma_2(q)}$, donc t'_1 se réécrit au nœud $\gamma_2(q)$ avec la règle $l \to r$ en $t_2 = t'_1[\gamma_2(q) \leftarrow \gamma_2[\mu[r]]]$. Par composition, $\gamma_{1|p} \circ \upsilon_{|\mathcal{R}\mathrm{oot}_l}$ est un homomorphisme allant de l vers $s'_{1|\gamma_1(q)}$, donc s'_1 se réécrit au nœud $\gamma_1(p)$ avec la règle $l \to r$ en $s_2 = s'_1[\gamma_1(p) \leftarrow \gamma_1[\upsilon[r]]]$.

Construction de S_2 et T_2 : Par hypothèse, les graphes S_1 et T_1 contiennent les graphes s_1 et t_1. On calcule S_2 et T_2 de sorte qu'ils contiennent s_2 et t_2 ainsi que les nœuds et les variables de s_1 et t_1 qui ont été éliminés par réécriture de s_2 et t_2. Ce calcul est schématisé dans la Figure 5.22.

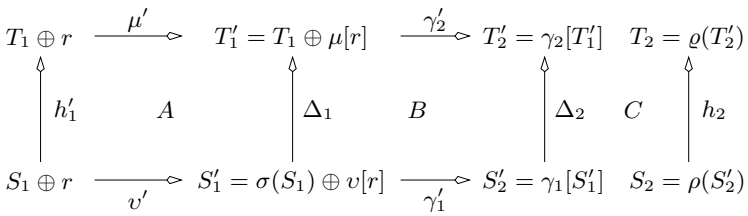


FIG. 5.22:

Considérons les graphes $T_1 \oplus r$ et $S_1 \oplus r$. Comme il existe un $\mathcal{D}$-monomorphisme $h_1 : S_1 \to T_1$, il

existe aussi un $\mathcal{D}$-monomorphisme $h'_1 : (S_1 \oplus r) \to (T_1 \oplus r)$. Soit $T'_1 = T_1 \oplus \mu[r]$. Il est clair qu'il existe un $\mathcal{D}$-monomorphisme $\mu' : (T_1 \oplus r) \to (T_1 \oplus \mu[r])$. Soit $S'_1 = \sigma(S_1) \oplus \upsilon[r]$. Il existe un homomorphisme $\upsilon' : (S_1 \oplus r) \to (\sigma(S_1) \oplus \upsilon[r])$. Nous prétendons qu'il existe un $\mathcal{D}$-monomorphisme $\Delta_1 : S'_1 \to T'_1$ (cf. A dans la Figure 5.22) ; Δ_1 est défini par $\Delta_1(n) = \delta(n)$ pour tout $n \in \mathcal{N}_{\sigma(s_{1|p})}$, $\Delta_1(n) = h_1(n)$ pour tout $n \in (\mathcal{N}_{S_1} - \mathcal{N}_{\sigma(s_{1|p})})$ et $\Delta_1(n) = n$ pour tout $n \in (\mathcal{N}_r - \mathcal{N}_l)$. De plus, nous obtenons $\Delta_1 \circ \upsilon' = \mu' \circ h'_1$. Comme μ' et h'_1 sont des $\mathcal{D}$-monomorphismes, $\mu' \circ h'_1$, donc $\Delta_1 \circ \upsilon'$, sont aussi des $\mathcal{D}$-monomorphismes. Si Δ_1 n'était pas un $\mathcal{D}$-monomorphisme, alors $\Delta_1 \circ \upsilon'$ ne serait pas non plus un $\mathcal{D}$-monomorphisme, ce qui n'est pas possible. Donc Δ_1 est un $\mathcal{D}$-monomorphisme.

Soit $T'_2 = \gamma_2[T'_1]$. Comme γ_2 est un $\mathcal{V}$-homomorphisme, nous déduisons qu'il existe un $\mathcal{V}$-homomorphisme $\gamma'_2 : T'_1 \to T'_2$. Soit $S'_2 = \gamma_1[S'_1]$. De même, il existe un $\mathcal{V}$-homomorphisme $\gamma'_1 : S'_1 \to S'_2$. De plus, il existe un $\mathcal{D}$-monomorphisme $\Delta_2 : S'_2 \to T'_2$ tel que $\Delta_2 \circ \gamma'_1 = \gamma'_2 \circ \Delta_1$ (cf. B dans la Figure 5.22).

Soit ϱ la redirection de pointeurs telle que $\varrho(\gamma_2(q)) = \mathcal{R}\text{oot}_{\gamma_2[\mu[r]]}$ et $\varrho(n) = n$ pour tout $n \neq \gamma_2(q)$. Soit $T_2 = \varrho(T'_2)$. t_1 est un sous-graphe de T_1, donc de T'_1. Comme $t'_1 = \gamma_2(t_1)$ et $T'_2 = \gamma_2[T'_1]$, t'_1 est un sous-graphe de T_2. t_2 est obtenu à partir de t'_1 en redirigeant toutes les flèches pointant sur $\gamma_2(q)$ pour qu'elles pointent sur $\mathcal{R}\text{oot}_{\gamma_2[\mu[r]]}$, i.e., t_2 est obtenu à partir de t'_1 avec ϱ. Donc il est clair que t_2 est un sous-graphe de T_2 : $t_2 = T_{2|\varrho(\mathcal{R}\text{oot}_{t'_1})}$.

Soit ρ la redirection de pointeurs telle que $\rho(\gamma_1(p)) = \mathcal{R}\text{oot}_{\gamma_1[\upsilon[r]]}$. Soit $S_2 = \rho(S'_2)$. s_1 est un sous-graphe de S_1. Donc $\sigma(s_1)$ est un sous-graphe de $\sigma(S_1)$, donc de S'_1. Comme $s'_1 = \gamma_1(\sigma(s_1))$ et $S_2 = \gamma_1[S'_1]$, s'_1 est un sous-graphe de S'_2. s_2 est obtenu à partir de s'_1 en redirigeant toutes les flèches pointant sur $\gamma_1(p)$ pour qu'elles pointent sur $\mathcal{R}\text{oot}_{\gamma_1[\upsilon[r]]}$, i.e., s_2 est obtenu à partir de s'_1 avec ρ. Donc il est clair que s_2 est un sous-graphe de S_2 : $s_2 = S_{2|\rho(\mathcal{R}\text{oot}_{s'_1})}$.

Construction de $h_2 : S_2 \to T_2$: Nous avons vu que $\Delta_2 : S'_2 \to T'_2$ était un $\mathcal{D}$-monomorphisme. Comme $\Delta_2(\gamma_1(p)) = \gamma_2(q)$ et $\Delta_2(\mathcal{R}\text{oot}_{\gamma_2[\mu[r]]}) = \mathcal{R}\text{oot}_{\gamma_1[\upsilon[r]]}$, nous déduisons avec la Proposition 5.5.8 que Δ_2 reste un $\mathcal{D}$-monomorphisme allant de S_2 vers T_2. Donc soit $h_2 = \Delta_2$ (cf. C dans la Figure 5.22). Il est clair que $h_2(s_2) = t_2$. De plus, $\sigma_{(h_2)|\mathcal{V}_{s_2}}$ est une substitution constructeur. En effet, δ'_1 est un $\mathcal{D}$-monomorphisme allant de s'_1 vers t'_1 tel que $\sigma_{h'_1}$ soit une substitution constructeur. Comme Δ_2 est un homomorphisme allant de S'_1 vers T'_1 et que s'_1 (resp. t'_1) est un sous-graphe de S'_1 (resp. T'_1), nous inférons que $\delta'_1 = \Delta_{2|\mathcal{R}\text{oot}_{s'_1}}$. Donc $\sigma_{(\Delta_2)|\mathcal{V}_{s'_1}} = \sigma_{(\delta'_1)|\mathcal{V}_{s'_1}}$. Comme $\sigma_{(\delta'_1)|\mathcal{V}_{s'_1}}$ est une substitution constructeur, nous déduisons que $\sigma_{(\Delta_2)|\mathcal{V}_{s'_1}}$ est une substitution constructeur. Pour finir, comme $h_2 = \Delta_2$ et $\mathcal{V}_{s_2} = \mathcal{V}_{s'_1}$, nous concluons que $\sigma_{(h_2)|\mathcal{V}_{s_2}}$ est aussi une substitution constructeur.

σ préserve les sous-graphes constructeurs de S_1 dans S_2 : Soit u un sous-graphe constructeur de S_1. Comme $S'_1 = \sigma(S_1) \oplus \upsilon[r]$, $\sigma(u)$ est un sous-graphe de S'_1. Comme σ est l'unificateur le plus général de $s_{1|p}$ par rapport à l, σ est une substitution constructeur. Donc $\sigma(u)$ est un sous-graphe *constructeur* de S'_1. Puisque $S'_2 = \gamma_1[S'_1]$, $\gamma_1[\sigma(u)]$ est un sous-graphe constructeur de S'_2. De plus, γ_1 est un $\mathcal{V}$-homomorphisme, donc $\gamma_1[\sigma(u)]$ est bisimilaire à $\sigma(u)$. Nous avons vu que $S_2 = \rho(S'_2)$ où ρ est la redirection de pointeurs allant du nœud *fonctionnel* $\gamma_1(p)$ vers $\mathcal{R}\text{oot}_{\gamma_1[\upsilon[r]]}$. Puisque $\gamma_1[\sigma(u)]$ est un graphe constructeur, $\gamma_1(p)$ n'est pas un nœud de $\gamma_1[\sigma(u)]$, donc $\gamma_1[\sigma(u)]$ n'est pas modifié par la redirection de pointeurs ρ. Par conséquent, $\gamma_1[\sigma(u)]$ est un sous-graphe constructeur de S_2. Comme $\sigma(u)$ est bisimilaire à $\gamma_1[\sigma(u)]$, nous concluons que $\sigma(u)$ est bisimilaire à un sous-graphe constructeur de S_2.

Vérification de l'équation $\sigma_{\mathbf{h_2}} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$: Soient $x \in \mathcal{V}_{S_1}$ et $n \in \mathcal{N}_{S_1}$ tels que $\mathcal{L}_{S_1}(n) = x$. Comme x est une variable de S_1, $\sigma_{h_1}(n : x) = h'_1(S_{1|n})$ (cf. Figure 5.22). h'_1 est une homomorphisme allant de $S_1 \oplus r$ vers $T_1 \oplus r$, donc $\sigma_{h_1}(n : x)$ est un sous-graphe de T_1. T_1 est invariant par μ', donc $\sigma_{h_1}(n : x)$ est un sous-graphe de T'_1 et $\sigma_{h_1}(n : x) = \mu' \circ h'_1(S_{1|n})$. Comme $\sigma_{h_1}(n : x)$ est un sous-graphe de T'_1, nous déduisons que $\gamma'_2(\sigma_{h_1}(n : x))$ est un sous-graphe de T'_2. Comme γ'_2 est un $\mathcal{V}$-homomorphisme, $\sigma_{h_1}(n : x) \doteq \gamma'_2(\sigma_{h_1}(n : x))$. Donc nous déduisons que $\sigma_{h_1}(n : x) \doteq \gamma'_2 \circ \mu' \circ h'_1(S_{1|n})$. Par composition, $\gamma'_2 \circ \mu' \circ h'_1 = \Delta_2 \circ \gamma'_1 \circ \upsilon'$, donc $\sigma_{h_1}(n : x) \doteq \Delta_2 \circ \gamma'_1 \circ \upsilon'(S_{1|n})$. υ' est un homomorphisme allant de $S_1 \oplus r$ vers $\sigma(S_1) \oplus \upsilon[r]$ et $n : x$ est un sous-graphe de S_1, donc $\upsilon'(n : x) = \sigma(n : x)$. Par conséquent,

$\sigma_{h_1}(n : x) \doteq \Delta_2(\gamma'_1(\sigma(n : x)))$. γ'_1 est un $\mathcal{V}$-homomorphisme, donc $\gamma'_1(\sigma(n : x)) \doteq \sigma(n : x)$ et par conséquent, $\sigma_{h_1}(n : x) \doteq \Delta_2(\sigma(n : x))$. $\sigma(n : x)$ est un sous-graphe constructeur de S'_2, donc nous inférons que $\Delta_2(\sigma(n : x)) = h_2(\sigma(n : x))$. Comme $h_2(\sigma(n : x)) \doteq \sigma_{h_2}(\sigma(n : x))$, nous déduisons que $\sigma_{h_1}(n : x) \doteq \sigma_{h_2}(\sigma(n : x))$. Nous concluons donc que $\sigma_{h_1} \doteq \sigma_{h_2} \circ \sigma\ [\mathcal{V}_{S_1}]$. □

5.5.3 Preuve du théorème de complétude

Nous prouvons ci-dessous le Théorème 5.3.6. Nous utiliserons le résultat clef suivant :

Proposition 5.5.9 Soient SP un WAGRS et g_1 et g_2 deux termes-graphes admissibles tels que $g_1 \rhd g_2$. Si il existe un graphe g'_1 tel que $g_1 \to g'_1$ (resp. $g_1 \xrightarrow{k} g'_1$), alors il existe deux termes-graphes admissibles g''_1 et g'_2 tels que $g'_1 \xrightarrow{*} g''_1$, $g_2 \to g'_2$ (resp. $g_2 \xrightarrow{k'} g'_2$ avec $k' \leq k$) et $g''_1 \rhd g'_2$ (cf. Figure 5.23). ◇

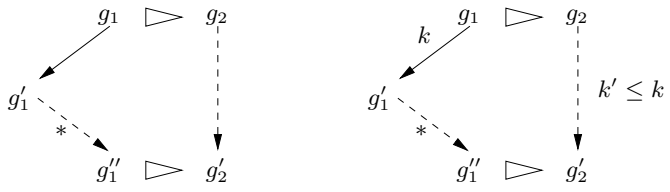


FIG. 5.23:

Preuve : Immédiat avec la définition de $\rhd$ et les Propositions 3.5.4 et 3.5.5. □

La proposition suivante généralise le lemme de Hullot :

Proposition 5.5.10 Soit SP un WAGRS, s_1 et t_1 deux termes-graphes admissibles, S_1 et T_1 deux graphes admissibles et $h_1 : S_1 \to T_1$ un $\mathcal{D}$-monomorphisme tels que s_1 soit un sous-graphe de S_1, t_1 soit un sous-graphe de T_1, $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur. Si il existe un graphe constructeur c_1 tel que $t_1 \xrightarrow{n} c_1$, alors il existe un graphe constructeur s et deux substitutions σ et η tels que $s_1 \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_\sigma s$, $\eta(s) \doteq c_1$ et $\eta \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. ◇

Preuve : Par induction sur la longueur n de la dérivation de réécriture $t_1 \xrightarrow{n} c_1$. Le cas $n = 0$ est évident. Soit $k_1 \geq 0$ et supposons que la propriété soit vraie pour tout $k < k_1$. Soit $t_1 \xrightarrow{k_1} c_1$ une dérivation de longueur k_1 (cf. Figure 5.24). Supposons que cette dérivation commence avec un pas de réécriture au nœud q avec la règle R. Alors, d'après le Lemme 5.5.6, (1) il existe un terme-graphe admissible s_2 et une substitution σ' tels que $s_1 \blacktriangleright\!\!\rightsquigarrow_{\sigma'} s_2$, (2) il existe deux termes-graphes admissibles t'_1 et t_2 tels que $t_1 \rhd t'_1$ avec l'homomorphisme $\gamma : t_1 \to t'_1$ et $t'_1 \to_{[\gamma(q), l \to r]} t_2$, (3) il existe deux graphes admissibles S_2 et T_2 et un $\mathcal{D}$-monomorphisme $h_2 : S_2 \to T_2$ tels que s_2 soit un sous-graphe de S_2, t_2 soit un sous-graphe de T_2, $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ soit une substitution constructeur, (4) pour tout sous-graphe constructeur u de S_1, $\sigma'(u)$ est bisimilaire à un sous-graphe constructeur de S_2 et (5) $\sigma_{h_2} \circ \sigma' \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$.

D'après le point (2), $t_1 \rhd t'_1$ avec l'homomorphisme $\gamma : t_1 \to t'_1$ et $t_1 \xrightarrow{k_1} c_1$, par hypothèse. Donc nous déduisons de la Proposition 5.5.9 qu'il existe un graphe constructeur c_2 et une dérivation de réécriture $t'_1 \xrightarrow{k'_1} c_2$ tels que $k'_1 \leq k_1$ et $c_2 \doteq c_1$. Le premier pas de la dérivation $t_1 \xrightarrow{k_1} c_1$ est au nœud q avec la règle R et t_1 se compacte en t'_1 avec l'homomorphisme γ. La preuve de la Proposition 5.5.9 montre que le premier pas de la dérivation $t'_1 \xrightarrow{k'_1} c_2$ est effectué au nœud $\gamma(q)$ avec la règle R, i.e.,

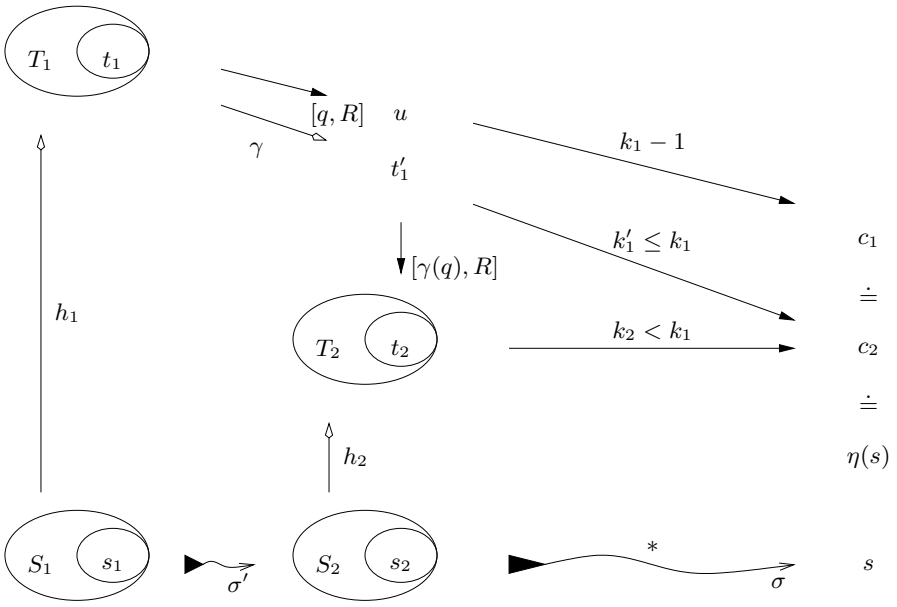


FIG. 5.24:

le premier pas de cette dérivation est $t'_1 \rightarrow_{[\gamma(q),\,R]} t_2$. Donc nous concluons qu'il existe une dérivation $t'_1 \rightarrow_{[\gamma(q),\,R]} t_2 \stackrel{k'_1-1}{\rightarrow} c_2$ et la longueur de la dérivation $t_2 \stackrel{*}{\rightarrow} c_2$ est $k_2 = k'_1 - 1 < k_1$.

s_2 est un sous-graphe de S_2, t_2 est un sous-graphe de T_2, $h_2 : S_2 \rightarrow T_2$ est un $\mathcal{D}$-monomorphisme tels que $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ est une substitution constructeur. De plus, $t_2 \stackrel{k_2}{\rightarrow} c_2$ et $k_2 < k_1$. Donc par hypothèse d'induction, il existe un graphe constructeur s et deux substitutions σ et η tels que $s_2 \blacktriangleright\!\!\stackrel{*}{\rightsquigarrow}_{\sigma} s$, $\eta(s) \doteq c_2$ et $\eta \circ \sigma \doteq \sigma_{h_2}\ [\mathcal{V}_{S_2}]$.

Nous devons prouver que $\eta(s) \doteq c_1$. C'est immédiat puisque $\eta(s) \doteq c_2$ et $c_2 \doteq c_1$. Nous prouvons maintenant que $\eta \circ (\sigma \circ \sigma') \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. Soient $x \in \mathcal{V}_{S_1}$ et $n \in \mathcal{N}_{S_1}$ tels que $\mathcal{L}_{s_1}(n) = x$. D'après le Lemme 5.5.6, $\sigma_{h_1}(n : x) \doteq \sigma_{h_2}(\sigma'(n : x))$. De plus, comme $n : x$ est un sous-graphe variable (donc constructeur) de S_1, $\sigma'(n : x)$ est bisimilaire à un sous-graphe constructeur de S_2. Donc $\mathcal{V}_{\sigma'(n:x)} \subseteq \mathcal{V}_{S_2}$. Par hypothèse d'induction, $\eta \circ \sigma \doteq \sigma_{h_2}\ [\mathcal{V}_{S_2}]$. Donc nous concluons que $\sigma_{h_1}(n : x) \doteq \sigma_{h_2}(\sigma'(n : x) \doteq \eta(\sigma(\sigma'(n : x)))$, i.e., $\eta \circ (\sigma \circ \sigma') \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. □

Le Théorème 5.3.6 est une conséquence immédiate de la Proposition 5.5.10 :

Preuve du Théorème 5.3.6 : Soient g un terme-graphe admissible, c un graphe constructeur et θ une substitution constructeur tels que $\theta(g) \stackrel{*}{\rightarrow} c$. Soient $S_1 = s_1 = g$ et $T_1 = t_1 = \theta(g)$. Il est clair qu'il existe un $\mathcal{D}$-monomorphisme $h_1 : S_1 \rightarrow T_1$ tel que $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur (puisque $(\sigma_{h_1})_{|\mathcal{V}_{s_1}} = \theta_{|\mathcal{V}_{s_1}}$). Donc d'après la Proposition 5.5.10, il existe un graphe constructeur s et deux substitutions σ et η tels que $g \blacktriangleright\!\!\stackrel{*}{\rightsquigarrow}_{\sigma} s$, $\eta(s) \doteq c$ et $\eta \circ \sigma \doteq \theta\ [\mathcal{V}_g]$. Comme $\eta(s) \doteq c$, nous déduisons que $s \dot{\leq} c$. Comme $\eta \circ \sigma \doteq \theta\ [\mathcal{V}_g]$, nous concluons que $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. □

Chapitre 6

Stratégies de surréduction de graphes admissibles

Dans le chapitre précédent, nous avons montré qu'on pouvait utiliser les programmes d'un langage logico-fonctionnel pour évaluer par surréduction des expressions partiellement définies modélisées sous la forme de graphes cycliques admissibles contenant des variables. Pour cela, nous avons défini la relation de surréduction compressante $\blacktriangleright\!\!\rightsquigarrow$ et nous avons montré qu'elle était cohérente et complète.

Toutefois, nous n'avons pas abordé le problème du choix des positions et des règles qu'il faut utiliser pour surréduire un graphe. C'est sur cette problématique que nous travaillons dans ce chapitre : nous définissons plusieurs *stratégies de surréduction*. De nombreuses stratégies de surréduction existent dans le cadre des termes du premier ordre, par exemple, la surréduction basique [Hul80a, MH92], *innermost* [Fri85], *outermost* [Ech88, Ech92], *outer* [You89], paresseuse [DG89, MKLR90, Red85] ou avec des tests de redondance [BKW92, KB91]. Nous nous intéressons plus particulièrement à la surréduction nécessaire[1] [AEH94a], à la surréduction faiblement nécessaire[2] [AEH97a] et à la surréduction parallèle[3] [AEH97a]. Ces stratégies récemment étudiées sont optimales selon de nombreux critères. Elles peuvent être facilement programmées et sont donc couramment utilisées dans l'implantation des langages logico-fonctionnels.

Dans ce chapitre, nous généralisons les trois stratégies précédentes à la surréduction compressante de graphes. Nous montrons que notre extension préserve les propriétés de cohérence, de complétude et d'optimalité qu'elles ont sur les termes du premier ordre. De plus, nous mettons en évidence plusieurs propriétés d'optimalité propres à la surréduction de graphes admissibles. Nos travaux ont partiellement été publiés dans [EJ99c, EJ98a, EJ98b]. Leurs fondements se trouvent dans deux rapports [EJ99a, EJ97a].

Une seule autre stratégie de surréduction de graphes existe dans la littérature, la surréduction basique de graphes acycliques[4] [Kri96, HP99]. La surréduction basique de termes est la première stratégie de surréduction à avoir été étudiée [Hul80a]. Elle ne possède pas les propriétés d'optimalité des trois stratégies qui nous intéressent.

Dans la Section 6.1, nous étendons les stratégies de surréduction nécessaire [AEH94a] et

[1] *needed narrowing* en anglais.
[2] *weakly needed narrowing* en anglais.
[3] *parallel narrowing* en anglais.
[4] *basic graph narrowing* en anglais.

de surréduction faiblement nécessaire [AEH97a] aux graphes admissibles. Ces stratégies dites séquentielles améliorent la surréduction compressante la plus générale de graphes dans le cadre des ISGRS et des WAGRS respectivement. Nous montrons que les relations de surréduction compressante qu'elles engendrent sont cohérentes et complètes. Nous étudions leurs propriétés d'optimalité dans la Section 6.2. Puis nous améliorons la relation de surréduction faiblement nécessaire de graphes dans la Section 6.3, en étendant la relation de surréduction parallèle de termes [AEH97a] aux graphes admissibles. Nous terminons ce chapitre en développant quelques propriétés d'optimalité de la surréduction parallèle compressante de graphes (Section 6.4).

6.1 Stratégies séquentielles de surréduction compressante

Nous avons vu dans le Chapitre 4 qu'une stratégie de réécriture était une fonction qui prenait un terme-graphe admissible g et qui retournait un ou plusieurs couple de la forme (p, R) où p est un nœud fonctionnel de g et R est une règle tels que g se réécrive au nœud p avec la règle R. La définition d'une stratégie de surréduction compressante est à peu près la même :

Définition 6.1.1 (Stratégie de surréduction compressante de graphes)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS et $\blacktriangleright$ une stratégie de compression. Une *stratégie (séquentielle) de surréduction compressante de graphes* est une fonction partielle $\mathcal{S}$ qui prend un terme-graphe admissible g et qui retourne un ensemble de triplets (p, R, σ) tels que p soit un nœud fonctionnel de g, R soit une règle de réécriture de $\mathcal{R}$, σ soit une substitution compatible avec g et $g \blacktriangleright\!\!\rightsquigarrow_{[p,\,R,\,\sigma]} g'$ pour un certain terme-graphe g'.
On note $g \blacktriangleright\!\!\rightsquigarrow_{\mathcal{S},[p,R,\sigma]}$ ou $g \blacktriangleright\!\!\rightsquigarrow_{\mathcal{S},\sigma} g'$ ou encore $g \blacktriangleright\!\!\rightsquigarrow_{\mathcal{S}} g'$ le $\mathcal{S}$-pas de surréduction compressante de g en g' tel que $\mathcal{S}(g) \ni (p, R, \sigma)$ et $g \blacktriangleright\!\!\rightsquigarrow_{[p,\,R,\,\sigma]} g'$. On note $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\mathcal{S},\sigma} g'$ ou simplement $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\mathcal{S}} g'$ toute $\mathcal{S}$-dérivation de surréduction compressante de g en g' calculant la substitution σ. ◇

Dans cette section, nous définissons deux stratégies de surréduction compressante. La première, Λ, est adaptée aux ISGRS. Nous l'introduisons dans le Paragraphe 6.1.1. La seconde, $\bar{\Lambda}$, étend Λ au cas général des WAGRS. Nous la définissons dans le Paragraphe 6.1.2. Puis nous montrons la cohérence et la complétude de $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ et de $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$ dans les Paragraphes 6.1.3 et 6.1.4.

6.1.1 Stratégie Λ (Cas des ISGRS)

La stratégie de surréduction Λ est une fonction partielle qui opère sur les termes-graphes admissibles en présence d'un ISGRS. $\Lambda(g)$ retourne, quand c'est possible, un ensemble de triplets de la forme $(p, l \to r, \sigma)$. σ est un unificateur particulier de $g_{|p}$ par rapport à l qui n'est généralement pas un unificateur le plus général de $g_{|p}$ par rapport à l. En fait, σ peut affecter des variables de g qui ne sont pas des variables de $g_{|p}$. Λ utilise une fonction auxilliaire λ qui prend deux arguments : un terme-graphe de racine fonctionnelle et un pdt de cette fonction.

Définition 6.1.2 (Stratégie Λ)
Soit SP un ISGRS et g un terme-graphe admissible. Λ est la fonction partielle telle que $\Lambda(g) = \lambda(g_{|p}, \mathcal{T})$ où p désigne le plus haut nœud fonctionnel à gauche de g, p est étiqueté par une opération définie f et $\mathcal{T}$ est un arbre de définition de f.

Soit g un terme-graphe admissible de racine fonctionnelle et $\mathcal{T}$ un pdt tel que $pattern(\mathcal{T})$ et g soient unifiables. $\lambda(g, \mathcal{T})$ est définie comme le plus petit ensemble tel que :

$$\lambda(g,\mathcal{T}) \supseteq \begin{cases} \{(p,R,\sigma)\} & \text{si } \mathcal{T} = rule(\pi \to r),\ p = \mathcal{R}\text{oot}_g,\ R = \pi \to r \text{ et} \\ & \sigma \text{ est un unificateur le plus général de } g \text{ par rapport à } \pi\,; \\ \lambda(g,\mathcal{T}_i) & \text{si } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ et} \\ & pattern(\mathcal{T}_i) \text{ et } g \text{ sont unifiables pour un certain } i \in 1..k\,; \\ \{(p,R,\sigma)\} & \text{si } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \tau \text{ est un unificateur le plus général de } g \text{ par rapport} \\ & \text{à } \pi \text{ et il existe un homomorphisme } h : \pi \to \tau(g), \\ & h(o) \text{ est étiqueté par une opération définie } f, \\ & \mathcal{T}' \text{ est un arbre de définition de } f, \\ & (p, R, \sigma') \in \lambda(\tau(g_{|h(o)}), \mathcal{T}') \text{ et } \sigma = \sigma' \circ \tau. \end{cases}$$

◇

Exemple 6.1.3 Considérons le terme-graphe g de la Figure 6.1 :
$g =$ `n1 :c(n2 :g(n3 :x,n4 :f(n5 :y,n5)),n6 :c(n7 :g(n3,n4),n1))`. On calcule $\Lambda(g)$ dans le cas de l'ISGRS $SP_1 = \langle \Sigma, \mathcal{R}_1 \rangle$ de l'Exemple 3.1.12 (cf. Figures 3.2 et 4.4). Le plus

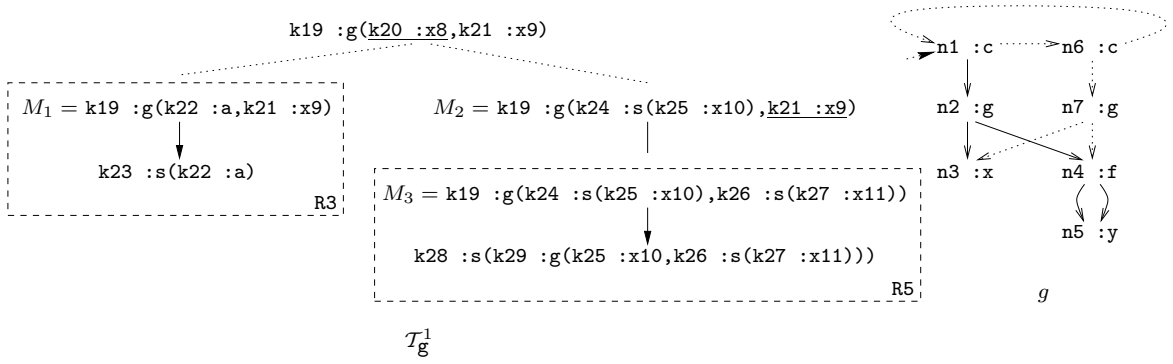
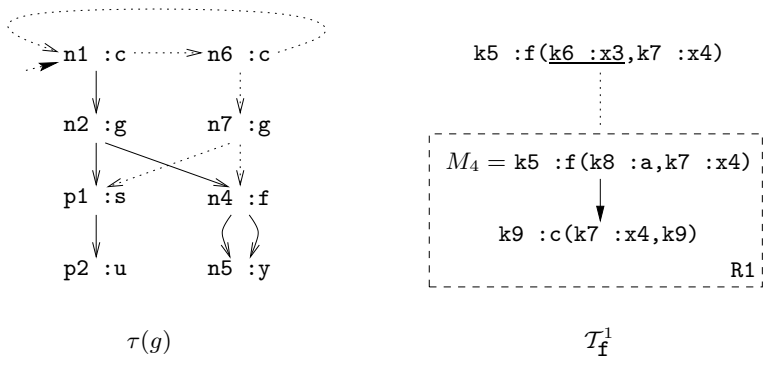


FIG. 6.1:

haut nœud fonctionnel à gauche de g est `n2`. Aussi, $\Lambda(g) = \lambda(g_{|\mathtt{n2}}, \mathcal{T}_g^1)$. Le terme-graphe $g_{|\mathtt{n2}}$ est unifiable avec le motif $M_1 =$ `k19 :g(k22 :a,k21 :x9)` de $\mathcal{T}_g^1$. La substitution $\sigma_1 = \{\mathtt{x} \mapsto$ `p1 :a`$\}$ est un unificateur le plus général de $g_{|\mathtt{n2}}$ par rapport à M_1. Comme le motif M_1 est le membre gauche de la règle de réécriture `R3`, on déduit que le triplet $(\mathtt{n2}, \mathtt{R3}, \sigma_1)$ est dans $\Lambda(g)$.

D'autre part, le terme-graphe admissible $g_{|\mathtt{n2}}$ est unifiable avec $M_2 =$ `k19 :g(k24 :s(k25 :x10),k21 :x9)` et $\tau = \{\mathtt{x} \mapsto$ `p2 :s(p3 :u)`$\}$ est un unificateur le plus général de $g_{|\mathtt{n2}}$ par rapport à M_2. On a $\tau(g_{|\mathtt{n2}}) =$ `n2 :g(p2 :s(p3 :u),n4 :f(n5 :y,n5))` (cf. Figure 6.2) et il existe un homomorphisme $h : M_2 \to \tau(g_{|\mathtt{n2}})$ tel que $h($`k21`$) =$ `n4`. Le motif $M_3 =$ `k19 :g(k24 :s(k25 :x10),k26 :s(k27 :x11))` ne s'unifie pas avec $g_{|\mathtt{n2}}$. En effet, le nœud `k26` est étiqueté par `s` dans M_3 alors que le nœud `n4` est étiqueté par `f` dans $\tau(g_{|\mathtt{n4}})$. Aussi, $\lambda(g_{|\mathtt{n2}}, \mathcal{T}_g^1)$ fait un appel récursif à $\lambda(\tau(g_{|\mathtt{n4}}), \mathcal{T}_f^1)$ (cf. Figure 6.2).

On constate que le terme-graphe $\tau(g_{|\mathtt{n4}})$ s'unifie avec le motif $M_4 =$ `k5 :f(k8 :a,k7 :x4)` et la substitution $\sigma' = \{\mathtt{y} \mapsto$ `p4 :a`$\}$ est un unificateur le plus général de $\tau(g_{|\mathtt{n4}})$ par rapport à M_4. Comme le motif M_4 est le membre gauche de la règle de réécriture `R1`, on déduit que $\Lambda(g)$ contient le triplet $(\mathtt{n4}, \mathtt{R1}, \sigma_2)$ où



$\tau(g)$ $\quad\quad$ $\mathcal{T}^1_{\mathtt{f}}$

FIG. 6.2:

$\sigma_2 = \sigma' \circ \tau = \{\mathtt{x} \mapsto \mathtt{p2\ :s(p3\ :u)}, \mathtt{y} \mapsto \mathtt{p4\ :a}\}$. On remarque que σ_2 n'est pas un unificateur le plus général de $\tau(g_{|\mathtt{n4}})$ par rapport au motif M_4.

Nous concluons donc que $\Lambda(g) = \{(\mathtt{n2}, \mathtt{R3}, \sigma_1), (\mathtt{n4}, \mathtt{R1}, \sigma_2)\}$ avec $\sigma_1 = \{\mathtt{x} \mapsto \mathtt{p1\ :a}\}$ et $\sigma_2 = \{\mathtt{x} \mapsto \mathtt{p2\ :s(p3\ :u)}, \mathtt{y} \mapsto \mathtt{p4\ :a}\}$. □

Nous montrerons plus loin que la relation de Λ-surréduction compressante, $\blacktriangleright\!\!\rightsquigarrow_\Lambda$, est cohérente et complète.

6.1.2 Stratégie $\bar{\Lambda}$ (Cas des WAGRS)

Nous venons de définir la stratégie Λ dans le cas des ISGRS. Cette stratégie ne peut plus être utilisée dans le cas d'un WAGRS puisque la définition de Λ repose sur l'existence d'un arbre de définition contenant *toutes* les règles de réécriture définissant une opération. Nous avons vu dans le Chapitre 4 qu'un tel arbre n'existait pas toujours dans le cas d'un WAGRS et qu'il fallait utiliser des forêts d'arbres de définition pour classer les règles d'un WAGRS. Dans cette partie, nous généralisons la stratégie Λ au cas des WAGRS de sorte qu'elle n'utilise pas de simples arbres de définition mais des forêts d'arbres de définition des opérations.

Cette nouvelle stratégie, $\bar{\Lambda}$, opère donc sur les termes-graphes admissibles en présence d'un WAGRS. $\bar{\Lambda}(g)$ retourne, quand c'est possible, un ensemble de triplets $(p, l \to r, \sigma)$ tels que g se surréduise au nœud p avec la règle $l \to r$ et la substitution σ. $\bar{\Lambda}$ utilise une fonction auxilliaire $\bar{\lambda}$ qui prend deux arguments : un terme-graphe admissible de racine fonctionnelle et un pdt de cette opération.

Définition 6.1.4 (Stratégie $\bar{\Lambda}$)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS et g un terme-graphe admissible. $\bar{\Lambda}$ est la fonction partielle telle que $\bar{\Lambda}(g) = \bar{\lambda}(g_{|p}, \mathcal{T}_1) \cup \ldots \cup \bar{\lambda}(g_{|p}, \mathcal{T}_n)$ où p désigne le plus haut nœud fonctionnel à gauche de g et $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ est une forêt d'arbres de définition de l'étiquette de p dans g.

Soient g un terme-graphe admissible de racine fonctionnelle et $\mathcal{T}$ un pdt tel que $pattern(\mathcal{T})$ et g s'unifient. $\bar{\lambda}(g, \mathcal{T})$ est un ensemble de triplets de la forme (p, R, σ), où p est un nœud fonctionnel de g, R est une règle de réécriture de $\mathcal{R}$ et σ est un unificateur de $g_{|p}$ par rapport

au membre gauche de R. $\bar{\lambda}(g, \mathcal{T})$ est défini comme le plus petit ensemble tel que :

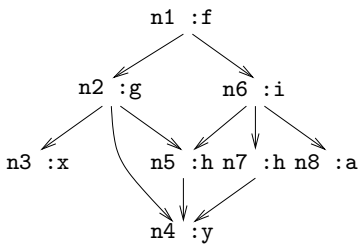
$$\bar{\lambda}(g,\mathcal{T}) \supseteq \begin{cases} \{(p,R,\sigma)\} & \text{si } \mathcal{T} = \mathit{rule}(\pi \to r),\ p = \mathcal{R}\mathrm{oot}_g,\ R = \pi \to r \text{ et} \\ & \sigma \text{ est un unificateur le plus général de } g \text{ par rapport à } \pi\,; \\ \bar{\lambda}(g,\mathcal{T}_i) & \text{si } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ et} \\ & g \text{ et } \mathit{pattern}(\mathcal{T}_i) \text{ s'unifient pour un certain } i \in 1..k\,; \\ \{(p,R,\sigma)\} & \text{si } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \tau \text{ est un unificateur le plus général de } g \text{ par rapport} \\ & \text{à } \pi \text{ et il existe un homomorphisme } h : \pi \to \tau(g), \\ & h(o) \text{ est étiqueté par une opération définie } f, \\ & \mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\} \text{ est une forêt d'arbres de définition de } f, \\ & S = \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_1) \cup \ldots \cup \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_k), \\ & (p, R, \sigma') \in S \text{ et } \sigma = \sigma' \circ \tau. \end{cases}$$

◇

Exemple 6.1.5 Pour traiter cet exemple, on considère un nouveau WAGRS défini par le système de réécriture suivant :

```
(R1) l1:f(l2:a,l3:x)        -> r1:c(l3:x,r1)
(R2) l1:f(l2:x,l3:s(l4:y)) -> r1:c(l3:s(l4:y),r1)
(R3) l1:g(l2:a,l3:a,l4:x)  -> l4:x
(R4) l1:h(l2:a)             -> l2:a
(R5) l1:i(l2:a,l3:x,l4:y)  -> l2:a
(R6) l1:i(l2:x,l3:a,l4:y)  -> l2:x
(R7) l1:i(l2:x,l3:y,l4:a)  -> l2:x
```

Des forêts d'arbres de définition des opérations `f`, `g`, `h` et `i` sont représentées dans la Figure 6.4. Soit g = `n1 :f(n2 :g(n3 :x,n4 :y,n5 :h(n4)),n6 :i(n5,n7 :h(n4),n8 :a)` le terme-graphe de la Figure 6.3.



g

FIG. 6.3:

Nous montrons ci-dessous le calcul de $\bar{\Lambda}(g)$. D'après la Définition 6.1.4,

$$\begin{aligned} \bar{\Lambda}(g) &= \bar{\lambda}(g_{|\mathtt{n1}}, \mathcal{T}^1_{\mathtt{f}}) \cup \bar{\lambda}(g_{|\mathtt{n1}}, \mathcal{T}^2_{\mathtt{f}}) \\ &= \bar{\lambda}(g_{|\mathtt{n2}}, \mathcal{T}_{\mathtt{g}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^1_{\mathtt{i}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^2_{\mathtt{i}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^3_{\mathtt{i}}) \\ &= \{(\mathtt{n2}, \mathtt{R3}, \sigma_1)\} \cup \bar{\lambda}(g_{|\mathtt{n5}}, \mathcal{T}_{\mathtt{h}}) \cup \bar{\lambda}(g_{|\mathtt{n7}}, \mathcal{T}_{\mathtt{h}}) \cup \{(\mathtt{n6}, \mathtt{R7}, \mathit{Id})\} \\ &= \{(\mathtt{n2}, \mathtt{R3}, \sigma_1), (\mathtt{n5}, \mathtt{R4}, \sigma_2), (\mathtt{n7}, \mathtt{R4}, \sigma_2), (\mathtt{n6}, \mathtt{R7}, \mathit{Id})\} \end{aligned}$$

où les substitutions σ_1 et σ_2 sont définies par
$\sigma_1 = \{\mathtt{x} \mapsto \mathtt{r1} : \mathtt{a}, \mathtt{y} \mapsto \mathtt{r2} : \mathtt{a}\}$ et $\sigma_2 = \{\mathtt{y} \mapsto \mathtt{r3} : \mathtt{a}\}$. □

k12 :g(k13 :x6,k14 :x7,k15 :x8)
k1 :f(k2 :x1,k3 :x2)
k6 :f(k7 :x3,k8 :x4)
k12 :g(k16 :a,k14 :x7,k15 :x8)
k18 :h(k19 :x9)
k1 :f(k4 :a,k3 :x2)
k5 :c(k3 :x2,k5)
R1
k6 :f(k7 :x3,k9 :s(k10 :x5))
k11 :c(k9 :s(k10 :x5),k11)
R2
k12 :g(k16 :a,k17 :a,k15 :x8)
k15 :x8
R3
k18 :h(k20 :a)
k20 :a
R4
$\mathcal{T}_{\mathtt{f}}^1$
$\mathcal{T}_{\mathtt{f}}^2$
$\mathcal{T}_{\mathtt{g}}$
$\mathcal{T}_{\mathtt{h}}$
k21 :i(k22 :x10,k23 :x11,k24 :x12)
k26 :i(k27 :x13,k28 :x14,k29 :x15)
k13 :i(k32 :x16,k33 :x17,k34 :x18)
k21 :i(k25 :a,k23 :x11,k24 :x12)
k25 :a
R5
k26 :i(k27 :x13,k30 :a,k29 :x15)
k27 :x13
R6
k31 :i(k32 :x16,k33 :x17,k35 :a)
k32 :x16
R7
$\mathcal{T}_{\mathtt{i}}^1$
$\mathcal{T}_{\mathtt{i}}^2$
$\mathcal{T}_{\mathtt{i}}^3$

FIG. 6.4:

De même que $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ est cohérente et complète, la relation de $\bar{\Lambda}$-surréduction compressante, $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$, est cohérente et complète. Nous montrons ces résultats dans les deux prochains paragraphes.

6.1.3 Cohérence de $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ et de $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$

Dans ce paragraphe, nous montrons le théorème suivant :

Theorème 6.1.6 Soient SP un WAGRS et $\blacktriangleright$ une stratégie de compression. $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$ est cohérente sur SP, i.e., pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution θ tels que $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\bar{\Lambda},\theta} c$, il existe un terme-graphe constructeur s tel que $\theta(g) \overset{*}{\rightarrow} s$ et $s \doteq c$. ◇

Nous constatons que la stratégie Λ est un cas particulier de $\bar{\Lambda}$ lorsque le système de réécriture de graphes considéré est un ISGRS plutôt qu'un WAGRS. Nous obtenons donc le corollaire suivant :

Corollaire 6.1.7 Soient SP un ISGRS et $\blacktriangleright$ une stratégie de compression. $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ est cohérente sur SP. ◇

Nous avons vu que la relation générale de surréduction compressante $\blacktriangleright\!\!\rightsquigarrow$ était cohérente (Théorème 5.3.4). Aussi, pour prouver le théorème précédent, il suffit de montrer que g peut se surréduire au nœud p avec la règle R et la substitution σ si $(p, R, \sigma) \in \bar{\Lambda}(g)$. Cette propriété est une conséquence immédiate de la proposition suivante :

Proposition 6.1.8 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS et s un terme-graphe admissible.
Si $(p, R, \sigma) \in \bar{\Lambda}(s)$, alors (1) p est un nœud fonctionnel de s, (2) R est une règle de $\mathcal{R}$, (3) σ est un unificateur de $s_{|p}$ par rapport au membre gauche de R et (4) $\sigma(x)$ est un arbre constructeur pour tout $x \in \mathcal{D}\sigma$. ◇

Preuve : Par définition de $\bar{\Lambda}$, si $(p, R, \sigma) \in \bar{\Lambda}(s)$, alors il existe un plus haut nœud fonctionnel q dans s et un arbre de définition $\mathcal{T}$ tels que $(p, R, \sigma) \in \bar{\lambda}(s_{|q}, \mathcal{T})$. Les points de la proposition sont vrais pour $(p, R, \sigma) \in \bar{\Lambda}(s)$ ssi ils sont vrais pour $(p, R, \sigma) \in \bar{\lambda}(s_{|q}, \mathcal{T})$. Donc nous supposons sans perte de généralité que s est un terme-graphe de racine fonctionnelle. On note $\sqsubset$ l'ordre nœthérien défini par $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ ssi le graphe g_1 a moins d'occurrences d'opérations définies que le graphe g_2, ou alors $g_1 = g_2$ et $\mathcal{T}_1$ est un sous-arbre propre de $\mathcal{T}_2$. Cette preuve se fait par induction nœthérienne sur $\sqsubset$. Nous étudions les différents cas de la définition de $\bar{\lambda}(s, \mathcal{T})$ où $\mathcal{T}$ est un pdt dont le motif s'unifie avec s :
Cas de base : Considérons $(s, \mathcal{T})$ où $\mathcal{T} = \mathit{rule}(\pi \rightarrow r)$ et $\pi \rightarrow r$ est une règle de $\mathcal{R}$. On a alors $\bar{\lambda}(s, \mathcal{T}) = \{(\mathcal{R}\mathrm{oot}_s, \pi \rightarrow r, \sigma)\}$ où σ est un unificateur le plus général de s par rapport à π. Donc tous les points de la proposition sont trivialement vrais.
Cas d'induction : Considérons $(s, \mathcal{T})$ où $\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, o est un nœud variable de π, o a pour sorte ζ, $c_1, \ldots, c_k$ $(k > 0)$ sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt de motif $\pi[o \leftarrow p : c_j(o_1 : X_1, \ldots, o_n : X_n)]$ tel que n soit le nombre d'arguments de c_j, $X_1, \ldots, X_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds. Soit $(p, l \rightarrow r, \sigma) \in \bar{\lambda}(s, \mathcal{T})$. Nous étudions les différents cas de la définition de $\bar{\lambda}(s, \mathcal{T})$ lorsque la racine du motif de $\mathcal{T}$ est étiquetée par *branch*.
Cas 1 : Supposons qu'il existe $i \in 1..k$ tel que s et $\mathit{pattern}(\mathcal{T}_i)$ soient unifiables. Alors $\bar{\lambda}(s, \mathcal{T}) \supseteq \bar{\lambda}(s, \mathcal{T}_i)$. De plus, le triplet $(p, l \rightarrow r, \sigma)$ que nous considérons est dans $\bar{\lambda}(s, \mathcal{T}_i)$ pour un certain $i \in 1..k$. Comme $(s, \mathcal{T}_i) \sqsubset (s, \mathcal{T})$, nous utilisons l'hypothèse d'induction : tous les points de la proposition sont vérifiés pour $(s, \mathcal{T}_i)$ donc ils le sont aussi pour $(s, \mathcal{T})$.
Cas 2 : Supposons qu'il n'existe pas $i \in 1..k$ tel que s et $\mathit{pattern}(\mathcal{T}_i)$ soient unifiables. Soient τ un unificateur le plus général de s par rapport à π et h l'homomorphisme allant de π vers $\tau(s)$. Dans le

cas que nous considérons ici, $h(o)$ est étiqueté par une opération définie f. Soit $\mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\}$ une forêt d'arbres de définition de f et soit $S = \bar{\lambda}(\tau(s_{|h(o)}), \mathcal{T}'_1) \cup \ldots \cup \bar{\lambda}(\tau(s_{|h(o)}), \mathcal{T}'_k)$.

Nous avons choisi $(p, l \to r, \sigma) \in \bar{\lambda}(s, \mathcal{T})$, donc par définition de $\bar{\lambda}$, il existe une substitution σ' et un entier $i \in 1..k$ tels que $(p, l \to r, \sigma') \in \bar{\lambda}(\tau(s_{|h(o)}), \mathcal{T}'_i)$ et $\sigma = \sigma' \circ \tau$. Comme τ est un unificateur le plus général de s par rapport à π, $\tau(x)$ est un arbre *constructeur* pour tout $x \in \mathcal{D}\tau$. Donc $\tau(s_{|h(o)})$ a moins d'opérations définies que s, i.e., $(\tau(s_{|h(o)}), \mathcal{T}'_i) \sqsubset (s, \mathcal{T})$. Aussi, par hypothèse d'induction, tous les points de la proposition sont vérifiés par $(p, l \to r, \sigma')$, à savoir (a) p est un nœud fonctionnel de $\tau(s_{|h(o)})$, (b) $l \to r$ est une règle de $\mathcal{R}$, (c) σ' est un unificateur de $\tau(s_{|h(o)})_{|p}$ par rapport à l et (d) $\sigma'(x)$ est un arbre constructeur pour tout $x \in \mathcal{D}\sigma'$.

(1) D'après le point (a), p est un nœud fonctionnel de $\tau(s_{|h(o)})$. Comme τ est un unificateur le plus général de s par rapport à π, τ est une substitution constructeur. Par conséquent p n'est pas introduit par τ dans $\tau(s_{|h(o)})$. Autrement dit, p est un nœud fonctionnel de $s_{|h(o)}$, donc de s.

(2) D'après le point (b), $l \to r$ est une règle de $\mathcal{R}$.

(3) D'après le point (c), σ' est un unificateur de $\tau(s_{|h(o)})_{|p}$ par rapport à l, donc il existe un homomorphisme $\mu : l \to \sigma'(\tau(s_{|h(o)})_{|p})$. De plus, $\sigma'(\tau(s_{|h(o)})_{|p}) = \sigma'(\tau(s_{|p})) = \sigma(s_{|p})$ (puisque $\sigma = \sigma' \circ \tau$). Donc μ est un homomorphisme allant de l vers $\sigma(s_{|p})$, i.e., σ est un unificateur de $s_{|p}$ par rapport à l.

(4) Soit $x \in \mathcal{D}\sigma$. Si $x \notin \mathcal{D}\tau$, alors $\sigma(x) = \sigma' \circ \tau(x) = \sigma'(x)$, donc $\sigma(x)$ est un arbre constructeur (d'après le point (d)). Si $x \in \mathcal{D}\tau$, alors $\tau(x)$ est un arbre constructeur puisque τ est un unificateur le plus général de s par rapport à π. Pour tout $u, v \in (\mathcal{D}\sigma' \cap \mathcal{V}_{\tau(x)})$ tels que $u \neq v$, $\sigma'(u)$ et $\sigma'(v)$ ne partage aucun nœud ni aucune variable et $\sigma'(u)$ et $\sigma'(v)$ sont des arbres constructeurs. Donc nous concluons que $\sigma(x) = \sigma'(\tau(x))$ est un arbre constructeur. □

6.1.4 Complétude de $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ et de $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$

Nous obtenons le résultat suivant :

Theorème 6.1.9 Soient SP un WAGRS et $\blacktriangleright$ une stratégie de compression. $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$ est complète sur SP, i.e., pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution constructeur θ tels que $\theta(g) \xrightarrow{*} c$, il existe un graphe constructeur s et une substitution σ tels que $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\bar{\Lambda},\sigma} s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta$ $[\mathcal{V}_g]$. ◇

Comme nous l'avons dit précédemment, la stratégie Λ est un cas particulier de $\bar{\Lambda}$ lorsque le WAGRS considéré est un ISGRS. Nous en déduisons le corollaire suivant :

Corollaire 6.1.10 Soient SP un ISGRS et $\blacktriangleright$ une stratégie de compression. Alors $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ est complète sur SP. ◇

Nous consacrons le reste de ce paragraphe à la preuve du théorème de complétude de $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$. Nous commençons par établir le lien entre le calcul de $\bar{\Phi}$ et le calcul de $\bar{\Lambda}$. Ce lemme s'apparente à [AEH97b, Lemma 4] :

Lemme 6.1.11 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS, s et t deux termes-graphes admissibles et $h : s \to t$ un $\mathcal{D}$-monomorphisme tels que σ_h soit une substitution constructeur. Si $(q, l \to r) \in \bar{\Phi}(t)$, alors il existe un nœud fonctionnel p de s et une substitution σ tels que :

1. $h(p) = q$.
2. $(p, l \to r, \sigma) \in \bar{\Lambda}(s)$.
3. $\sigma \dot{\leq} \sigma_h$ $[\mathcal{V}_s]$.

◇

Preuve : Soit m le plus haut nœud fonctionnel à gauche dans t et $\mathcal{F} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ une forêt d'arbres de définition de l'étiquette de m. Si $(q, l \to r) \in \bar{\Phi}(t)$, alors il existe $i \in 1..k$ tel que $(q, l \to r) \in \bar{\varphi}(t_{|m}, \mathcal{T}_i)$, par définition de $\bar{\Phi}$. Soit n le plus haut nœud fonctionnel à gauche dans s. Nous allons montrer qu'il existe un nœud fonctionnel p de s et une substitution σ tels que $(p, l \to r, \sigma) \in \bar{\lambda}(s_{|n}, \mathcal{T}_i)$, $h(p) = q$ et $\sigma \dot{\leq} \sigma_h\ [\mathcal{V}_s]$. Comme $\bar{\Lambda}(s) \supseteq \bar{\lambda}(s_{|n}, \mathcal{T}_i)$, nous aurons donc bien une preuve du Lemme 6.1.11.

n et m sont respectivement les plus hauts fonctionnels à gauche de s et t et σ_h est une substitution constructeur, donc $h(n) = m$. Par conséquent, $h_{|n}$ est un $\mathcal{D}$-monomorphisme allant de $s_{|n}$ vers $t_{|m}$ tel que $\sigma_{(h_{|n})}$ soit une substitution constructeur. Aussi, nous supposons sans perte de généralité que s et t sont des termes-graphes admissibles de racine fonctionnelle et qu'il existe un $\mathcal{D}$-monomorphisme $h : s \to t$ tel que σ_h soit une substitution constructeur.

Soit $\mathcal{T}$ un pdt tel que $pattern(\mathcal{T})$ filtre t à la racine. Nous montrons que si $(q, l \to r) \in \bar{\varphi}(t, \mathcal{T})$, alors il existe un nœud fonctionnel p de s et une substitution σ tels que $h(p) = q$, $(p, l \to r, \sigma) \in \bar{\lambda}(s, \mathcal{T})$ et $\sigma \dot{\leq} \sigma_h\ [\mathcal{V}_s]$. On note $\sqsubset$ l'ordre nœthérien défini par $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ ssi le graphe g_1 a moins d'occurrences d'opérations définies que le graphe g_2, ou alors $g_1 = g_2$ et $\mathcal{T}_1$ est un sous-arbre propre de $\mathcal{T}_2$. Cette preuve se fait par induction nœthérienne sur $\sqsubset$. Nous étudions les différents cas de la définition de $\bar{\varphi}(t, \mathcal{T})$ où $\mathcal{T}$ est un pdt dont le motif filtre t à la racine :

Cas de base : Considérons $(t, \mathcal{T})$ où $\mathcal{T} = rule(\pi \to r)$ et $\pi \to r$ est une règle de $\mathcal{R}$. On a alors $\bar{\varphi}(t, \mathcal{T}) = \{(\mathcal{R}\text{oot}_t, \pi \to r)\}$. π filtre t à la racine, donc il existe un homomorphisme $\mu : \pi \to t$. De plus, h est un homomorphisme allant de s vers t. Donc s et π sont unifiables et l'homomorphisme $(h \cup \mu) : (s \oplus \pi) \to (t \oplus t)$ tel que $(h \cup \mu)(s) = (h \cup \mu)(\pi) = t$ est un unificateur. Comme π est un motif, nous déduisons qu'il existe un unificateur le plus général τ de s par rapport à π et un homomorphisme $\upsilon : (s \oplus \pi) \to (\tau(s) \oplus \tau(s))$ tels que $\upsilon(s) = \upsilon(\pi) = \tau(s)$. De plus, il existe un homomorphisme $\delta : \tau(s) \to t$ tel que $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_s} = h$ et $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_\pi} = \mu$ (cf. Figure 6.5).

(2) Puisque s et π sont unifiables, $\bar{\lambda}(s, \mathcal{T})$ est défini et $\bar{\lambda}(s, \mathcal{T}) \ni (\mathcal{R}\text{oot}_s, \pi \to r, \tau)$. (1) h est un homomorphisme allant de s vers t, donc $h(\mathcal{R}\text{oot}_s) = \mathcal{R}\text{oot}_t$. (3) Nous avons vu que $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_s} = h$, donc $\sigma_\delta \circ \tau \doteq \sigma_h\ [\mathcal{V}_s]$ ce qui implique $\tau \dot{\leq} \sigma_h\ [\mathcal{V}_s]$.

Cas d'induction : Considérons $(t, \mathcal{T})$ où $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, o est un nœud variable de π, o est de sorte ζ, $c_1, \ldots, c_k$ $(k > 0)$ sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt de motif $\pi[o \leftarrow p : c_j(o_1 : X_1, \ldots, o_n : X_n)]$ tel que n soit le nombre d'arguments de c_j, $X_1, \ldots, X_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds.

π filtre t à la racine, donc il existe un homomorphisme $\mu : \pi \to t$. De plus, h est un homomorphisme allant de s vers t. Donc s et π sont unifiables et l'homomorphisme $(h \cup \mu) : (s \oplus \pi) \to (t \oplus t)$ tel que $(h \cup \mu)(s) = (h \cup \mu)(\pi) = t$ est un unificateur. Comme s et π sont unifiables, $\bar{\lambda}(s, \mathcal{T})$ est défini. Nous étudions les différents cas de la définition de $\bar{\varphi}(t, \mathcal{T})$ lorsque la racine du motif de $\mathcal{T}$ est étiquetée par *branch*.

Cas 1 : Supposons qu'il existe $i \in 1..k$ tel que $pattern(\mathcal{T}_i)$ filtre t à la racine. On a alors $\bar{\varphi}(t, \mathcal{T}) = \bar{\varphi}(t, \mathcal{T}_i) \ni (q, l \to r)$. Puisque $(t, \mathcal{T}_i) \sqsubset (t, \mathcal{T})$, nous utilisons l'hypothèse d'induction : il existe un nœud fonctionnel p de s et une substitution σ tels que (1) $h(p) = q$, (3) $\sigma \dot{\leq} \sigma_h\ [\mathcal{V}_s]$ et (2) $(p, l \to r, \sigma) \in \bar{\lambda}(s, \mathcal{T}_i)$. Comme $\bar{\lambda}(s, \mathcal{T}) \supseteq \bar{\lambda}(s, \mathcal{T}_i)$, nous concluons que $(p, l \to r, \sigma) \in \bar{\lambda}(s, \mathcal{T})$.

Cas 2 : Supposons qu'il n'existe pas $i \in 1..k$ tel que $pattern(\mathcal{T}_i)$ filtre t à la racine. Soit $b = \mu(o)$. Dans le cas que nous considérons actuellement, nous supposons que b est étiqueté par une opération définie f dans t. Soit $\mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\}$ une forêt d'arbres de définition de f. On a alors $\bar{\varphi}(t, \mathcal{T}) = \bar{\varphi}(t_{|b}, \mathcal{T}'_1) \cup \ldots \cup \bar{\varphi}(t_{|b}, \mathcal{T}'_k)$. Comme $(q, l \to r) \in \bar{\varphi}(t, \mathcal{T})$, il existe $i \in 1..k$ tel que $(q, l \to r) \in \bar{\varphi}(t_{|b}, \mathcal{T}'_i)$.

Nous avons vu que s et π étaient unifiables et que l'homomorphisme $(h \cup \mu) : (s \oplus \pi) \to (t \oplus t)$ tel que $(h \cup \mu)(s) = (h \cup \mu)(\pi) = t$ était un unificateur. Comme π est un motif, nous déduisons qu'il existe un unificateur le plus général τ de s par rapport à π et un homomorphisme $\upsilon : (s \oplus \pi) \to (\tau(s) \oplus \tau(s))$ tels que $\upsilon(s) = \upsilon(\pi) = \tau(s)$. De plus, il existe un homomorphisme $\delta : \tau(s) \to t$ tel que $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_s} = h$ et $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_\pi} = \mu$ (cf. Figure 6.5). Nous affirmons que $\delta : \tau(s) \to t$ est un $\mathcal{D}$-monomorphisme tel que σ_δ soit une substitution constructeur. En effet, $\delta \circ \upsilon_{|\mathcal{R}\text{oot}_s} = h$. Comme h est un $\mathcal{D}$-monomorphisme tel que σ_h soit une substitution constructeur, on déduit que δ est aussi un $\mathcal{D}$-monomorphisme tel que σ_δ soit une substitution constructeur (sinon h ne serait pas un $\mathcal{D}$-monomorphisme tel que σ_h soit une substitution constructeur).

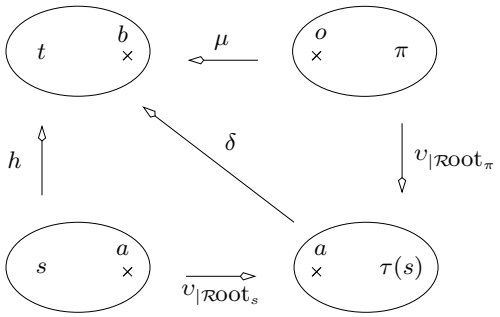


FIG. 6.5:

Soit $a = \upsilon(o)$. Puisque $\delta \circ \upsilon_{|\mathcal{R}\mathrm{oot}_\pi} = \mu$, $\delta(a) = \delta(\upsilon(o)) = \mu(o) = b$. Donc $\delta(a) = b$. Comme $b = \mu(o)$ est un nœud fonctionnel de t et comme $\delta : \tau(s) \to t$ est un $\mathcal{D}$-monomorphisme tel que σ_δ soit une substitution constructeur, nous déduisons que $a = \upsilon(o)$ est un nœud fonctionnel de $\tau(s)$. De plus, τ est un unificateur le plus général de s par rapport à π, donc τ est une substitution constructeur. Par conséquent, a est aussi un nœud fonctionnel de s. Aussi, $\bar{\lambda}(s, \mathcal{T}) \supseteq \bar{\lambda}(\tau(s_{|a}), \mathcal{T}'_1) \cup \ldots \cup \bar{\lambda}(\tau(s_{|a}), \mathcal{T}'_k)$, par définition de $\bar{\lambda}$.

Soit $h' = \delta_{|a}$. h' est un $\mathcal{D}$-monomorphisme allant de $\tau(s_{|a})$ vers $t_{|b}$ tel que σ'_h soit une substitution constructeur (puisque δ est un $\mathcal{D}$-monomorphisme tel que σ_δ soit une substitution constructeur). De plus, $\tau(s_{|a})$ et $t_{|b}$ sont des termes-graphes admissibles de racine fonctionnelle et $(q, l \to r) \in \bar{\varphi}(t_{|b}, \mathcal{T}'_i)$ pour un certain i. Pour finir, $t_{|b}$ a moins d'opérations définies que t, donc $(t_{|b}, \mathcal{T}'_i) \sqsubset (t, \mathcal{T})$. Par conséquent, nous pouvons utiliser l'hypothèse d'induction : il existe un nœud fonctionnel p dans $\tau(s_{|a})$ et une substitution σ' tels que $h'(p) = q$, $(p, l \to r, \sigma') \in \bar{\lambda}(\tau(s_{|a}), \mathcal{T}_i)$ et $\sigma' \dot{\leq} \sigma_{h'}\ [\mathcal{V}_{\tau(s_{|a})}]$.

(1) p est un nœud fonctionnel de $\tau(s_{|a})$ et τ est une substitution constructeur, donc p est un nœud fonctionnel de $s_{|a}$ et $h(p) = h_{|a}(p)$. Comme $\delta \circ \upsilon_{|\mathcal{R}\mathrm{oot}_s} = h$ et $h' = \delta_{|a}$, il est clair que $h' \circ \upsilon_{|a} = h_{|a}$. Donc $h_{|a}(p) = (h' \circ \upsilon_{|a})(p)$. $\upsilon_{|a}$ est un homomorphisme allant de $s_{|a}$ vers $\tau(s_{|a})$. Comme p est un nœud fonctionnel et τ est une substitution constructeur, $\upsilon_{|a}(p) = p$. Aussi $h(p) = h'(\upsilon_{|a}(p)) = h'(p) = q$.

(2) Soit $\sigma = \sigma' \circ \tau$. Par hypothèse d'induction, $(p, l \to r, \sigma') \in \bar{\lambda}(\tau(s_{|a}), \mathcal{T}_i)$, donc $(p, l \to r, \sigma) \in \bar{\lambda}(s, \mathcal{T})$.

(3) Nous avons vu que $\delta \circ \upsilon_{|\mathcal{R}\mathrm{oot}_s} = h$. Donc nous inférons que $\sigma_\delta \circ \tau \doteq \sigma_h\ [\mathcal{V}_s]$. Par hypothèse d'induction, $\sigma' \dot{\leq} \sigma_{h'}\ [\mathcal{V}_{\tau(s_{|a})}]$. De plus $h' = \delta_{|a}$, donc $\sigma' \dot{\leq} \sigma_\delta\ [\mathcal{V}_{\tau(s_{|a})}]$. σ' n'a pas d'effet sur les variables de $(\mathcal{V}_{\tau(s)} - \mathcal{V}_{\tau(s_{|a})})$, donc nous déduisons que $\sigma' \dot{\leq} \sigma_\delta\ [\mathcal{V}_{\tau(s)}]$, i.e., il existe une substitution θ telle que $\theta \circ \sigma' \doteq \sigma_\delta\ [\mathcal{V}_{\tau(s)}]$. Nous avons montré que $\sigma_\delta \circ \tau \doteq \sigma_h\ [\mathcal{V}_s]$ et $\theta \circ \sigma' \doteq \sigma_\delta\ [\mathcal{V}_{\tau(s)}]$, donc nous déduisons que $\theta \circ (\sigma' \circ \tau) \doteq \sigma_h\ [\mathcal{V}_s]$, i.e., $\theta \circ \sigma \doteq \sigma_h\ [\mathcal{V}_s]$. Par conséquent, $\sigma \dot{\leq} \sigma_h\ [\mathcal{V}_s]$. □

Nous établissons maintenant un lemme similaire au lemme de Hullot (Lemme 5.5.6). La seule différence consiste en l'utilisation d'une dérivation de réécriture calculée par $\bar{\Phi}$ plutôt qu'une dérivation de réécriture quelconque :

Lemme 6.1.12 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS, s_1 et t_1 deux termes-graphes admissibles, S_1 et T_1 deux graphes admissibles et $h_1 : S_1 \to T_1$ un $\mathcal{D}$-monomorphisme tels que s_1 soit un sous-graphe de S_1, t_1 soit un sous-graphe de T_1, $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur. Supposons qu'il existe un graphe constructeur c et une dérivation de réécriture $t_1 \xrightarrow{+}_{\bar{\Phi}} c$ commençant par un pas de réécriture au nœud q avec la règle $l \to r$, (i.e., $t_1 \to_{[q,R]} u \xrightarrow{*}_{\bar{\Phi}} c$ et $(q, l \to r) \in \bar{\Phi}(t_1)$). Alors :

1. Il existe un terme-graphe admissible s_2, un nœud fonctionnel p de s_1 et un unificateur σ de $s_{1|p}$ par rapport à l tels que $s_1 \blacktriangleright\!\!\rightsquigarrow_{[p, l \to r, \sigma]} s_2$ et $(p, l \to r, \sigma) \in \bar{\Lambda}(s_1)$.

2. Il existe deux termes-graphes admissibles t'_1 et t_2 tels que $t_1 \rhd t'_1$ avec le $\mathcal{V}$-homomorphisme $\gamma : t_1 \to t'_1$ et $t'_1 \to_{[\gamma(q), l \to r]} t_2$.
3. Il existe deux graphes admissibles S_2 et T_2 et un $\mathcal{D}$-monomorphisme $h_2 : S_2 \to T_2$ tels que s_2 soit un sous-graphe de S_2, t_2 soit un sous-graphe de T_2, $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ soit une substitution constructeur.
4. Pour tout sous-graphe constructeur u de S_1, $\sigma(u)$ est bisimilaire à un sous-graphe constructeur de S_2.
5. $\sigma_{h_2} \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$.

$\diamond$

Preuve : Par hypothèse, t_1 se réécrit au nœud q avec la règle $l \to r$ pour un certain $(q, l \to r) \in \bar{\Phi}(t_1)$. De plus $h_{1|\mathcal{R}\mathrm{oot}_{s_1}}$ est un $\mathcal{D}$-monomorphisme allant de s_1 vers t_1 tel que $\sigma_{(h_{1|\mathcal{R}\mathrm{oot}_{s_1}})}$ soit une substitution constructeur (puisque $\sigma_{(h_{1|\mathcal{R}\mathrm{oot}_{s_1}})} = (\sigma_{h_1})_{|\mathcal{V}_{s_1}}$). Donc d'après le Lemme 6.1.11, il existe un nœud fonctionnel p de s_1 et une substitution σ tels que (1) $h_1(p) = q$, (2) $(p, l \to r, \sigma) \in \bar{\Lambda}(s_1)$ et (3) $\sigma \dot{\leq} \sigma_{h_1}\ [\mathcal{V}_{s_1}]$. De plus, (4) σ est un unificateur de $s_{1|p}$ par rapport à l et (5) $\sigma(x)$ est un arbre constructeur pour tout $x \in \mathcal{D}\sigma$ d'après la Proposition 6.1.8. Nous déduisons du point (4) qu'il existe un terme-graphe admissible s_2 tel que $s_1 \blacktriangleright\!\!\rightsquigarrow_{[p, l \to r, \sigma]} s_2$. Soit υ_1 l'homomorphisme allant de s_1 vers $\sigma(s_1)$. Les points (3) et (5) impliquent qu'il existe un homomorphisme $\delta_1 : \sigma(s_1) \to t_1$ tel que $\delta_1 \circ \upsilon_1 = h_{1|\mathcal{R}\mathrm{oot}_{s_1}}$ (cf. Figure 6.6). Soit $\upsilon : (s_{1|p} \oplus l) \to (\sigma(s_{1|p}) \oplus \sigma(s_{1|p}))$ l'homomorphisme tel que

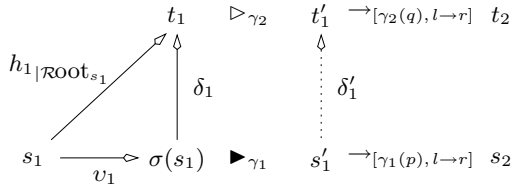


FIG. 6.6:

$\upsilon(l) = \upsilon(s_{1|p}) = \sigma(s_{1|p})$. Soit $\delta = \delta_{1|p}$ et $\mu = \delta \circ \upsilon_{|\mathcal{R}\mathrm{oot}_l}$ Il est clair que $\delta \circ \upsilon_{|p} = h_{1|p}$ (cf. Figure 6.7). Pour résumer, les relations suivantes sont satisfaites : (a) $\delta \circ \upsilon_{|\mathcal{R}\mathrm{oot}_l} = \mu$, (b) $\delta \circ \upsilon_{|p} = h_{1|p}$ et (c)

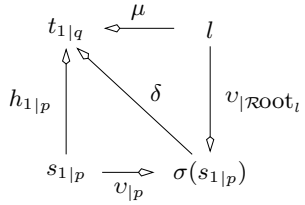


FIG. 6.7:

$\delta_1 \circ \upsilon_1 = h_{1|\mathcal{R}\mathrm{oot}_{s_1}}$. Ces relations sont exactement les mêmes que celles utilisées dans la preuve du lemme de Hullot 5.5.6, comme le montre les Figures 5.20 et 5.21. Aussi, la construction de t'_1, t_2, S'_1, T'_1, S'_2, T'_2, S_2, T_2, Δ_1, Δ_2 et h_2 sont exactement les mêmes que celles de la preuve du Lemme 5.5.6. Donc nous inférons que tous les points du Lemme 6.1.12 sont vérifiés. □

La proposition suivante stipule qu'on peut calculer la forme normale d'un graphe $\mathcal{C}$-normalisable en n'utilisant que des nœuds et des règles calculées par $\bar{\Phi}$:

Proposition 6.1.13 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS, t un terme-graphe admissible et c_1 un graphe constructeur tels qu'il existe une dérivation de réécriture $t \xrightarrow{*} c_1$ de longueur $n_1 \geq 0$. Alors il existe un graphe constructeur c_2 et une dérivation de réécriture

$$t = t_1 \rightarrow_{[p_1, R_1]} t_2 \rightarrow_{[p_2, R_2]} \cdots \rightarrow_{[p_{n_2}, R_{n_2}]} c_2$$

de longueur n_2 tels que $(p_i, R_i) \in \bar{\Phi}(t_i)$ pour tout $i \in 1..n_2$, $n_2 \leq n_1$ et $c_2 \sim c_1$. Par abus de langage, on note $t \xrightarrow{*}_{\bar{\Phi}} c_2$ la $\bar{\Phi}$-dérivation de réécriture précédente. $\diamond$

Preuve : Par induction sur n_1. La proposition est évidente lorsque $n_1 = 0$. Soit $n_1 \geq 0$ et $A = t \xrightarrow{*} c_1$ une dérivation de longueur n_1. Supposons que la proposition soit vraie pour tout $n'_1 < n_1$. La dérivation A est de la forme $A = t \rightarrow_{[q_1, S_1]} u_1 \ldots u_{n-1} \rightarrow_{[q_n, S_n]} c_1$. D'après la Proposition 4.3.16, $\bar{\Phi}(t)$ est un ensemble nécessaire de rédex. Donc il existe au moins un couple (q_i, S_i) de la dérivation A tel que $(q_i, S_i) \in \bar{\Phi}(t)$. Soit t' le terme-graphe admissible tel que $t \rightarrow_{[q_i, S_i]} t'$ (i.e., $t \rightarrow_{\bar{\Phi}} t'$). La preuve du théorème de confluence des WAGRS implique qu'il existe un graphe constructeur c'_1 et une dérivation de réécriture $t' \xrightarrow{*} c'_1$ de longueur n'_1 tels que $n'_1 < n_1$ et $c'_1 \sim c_1$. Par hypothèse d'induction, il existe un graphe constructeur c_2 et une dérivation de réécriture $t' \xrightarrow{*}_{\bar{\Phi}} c_2$ de longueur $n'_2 \leq n'_1$ tels que $c_2 \sim c'_1$. Donc nous concluons qu'il existe un graphe constructeur c_2 et une dérivation de réécriture $B = t \rightarrow_{\bar{\Phi}} t' \xrightarrow{*}_{\bar{\Phi}} c_2$. La longueur de B est $n_2 = n'_2 + 1$. Comme $n'_2 \leq n'_1 < n_1$, nous déduisons que $n_2 \leq n'_1 + 1 \leq n_1$. De plus, $c_2 \sim c'_1$ et $c'_1 \sim c_1$, donc $c_2 \sim c_1$. $\square$

Ci-dessous, nous généralisons le Lemme 6.1.12. Son énoncé rappelle celui de la Proposition 5.5.10 qu'on utilise dans la preuve de complétude de $\blacktriangleright\!\!\rightsquigarrow$:

Proposition 6.1.14 Soit SP un WAGRS, s_1 et t_1 deux termes-graphes admissibles, S_1 et T_1 deux graphes admissibles et $h_1 : S_1 \rightarrow T_1$ un $\mathcal{D}$-monomorphisme tels que s_1 soit un sous-graphe de S_1, t_1 soit un sous-graphe de T_1, $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur. Si il existe un graphe constructeur c_1 tel que $t_1 \xrightarrow{n}_{\bar{\Phi}} c_1$, alors il existe un graphe constructeur s et deux substitutions σ et η tels que $s_1 \overset{m}{\blacktriangleright\!\!\rightsquigarrow}_{\bar{\Lambda}, \sigma} s$, $m \leq n$, $\eta(s) \doteq c_1$ et $\eta \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. $\diamond$

Preuve : Par induction sur la longueur n de la dérivation de réécriture $t_1 \xrightarrow{n}_{\bar{\Phi}} c_1$. Le cas $n = 0$ est évident. Soit $k_1 \geq 0$ et supposons que la propriété soit vraie pour tout $k < k_1$. Soit $t_1 \xrightarrow{k_1}_{\bar{\Phi}} c_1$ une $\bar{\Phi}$-dérivation de longueur k_1. Supposons que cette dérivation commence par un pas de réécriture au nœud q avec la règle $l \rightarrow r$ tels que $(q, l \rightarrow r) \in \bar{\Phi}(t_1)$. Alors, d'après le Lemme 6.1.12, (1) il existe un terme-graphe admissible s_2 et une substitution σ' tels que $s_1 \blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}, \sigma'} s_2$, (2) il existe deux termes-graphes admissibles t'_1 et t_2 tels que $t_1 \rhd t'_1$ avec le $\mathcal{V}$-homomorphisme $\gamma : t_1 \rightarrow t'_1$ et $t'_1 \rightarrow_{[\gamma(q), l \rightarrow r]} t_2$, (3) il existe deux graphes admissibles S_2 et T_2 et un $\mathcal{D}$-monomorphisme $h_2 : S_2 \rightarrow T_2$ tels que s_2 soit un sous-graphe de S_2, t_2 soit un sous-graphe de T_2, $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ soit une substitution constructeur, (4) pour tout sous-graphe constructeur u de S_1, $\sigma'(u)$ est bisimilaire à un sous-graphe constructeur de S_2 et (5) $\sigma_{h_2} \circ \sigma' \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$.

D'après le point (2), $t_1 \rhd t'_1$ avec l'homomorphisme $\gamma : t_1 \rightarrow t'_1$ et $t_1 \xrightarrow{k_1} c_1$, par hypothèse. Donc nous déduisons de la Proposition 5.5.9 qu'il existe un graphe constructeur c'_1 et une dérivation de réécriture $t'_1 \xrightarrow{k'_1} c'_1$ tels que $k'_1 \leq k_1$ et $c'_1 \doteq c_1$. Le premier pas de la dérivation $t_1 \xrightarrow{k_1} c_1$ se fait au nœud q avec la règle $l \rightarrow r$ et t_1 se compacte en t'_1 avec l'homomorphisme γ. La preuve de la Proposition 5.5.9 montre que le premier pas de la dérivation $t'_1 \xrightarrow{k'_1} c'_1$ est au nœud $\gamma(q)$ avec la règle $l \rightarrow r$, i.e., le premier pas de cette dérivation est $t'_1 \rightarrow_{[\gamma(q), l \rightarrow r]} t_2$. Donc on conclut qu'il existe une dérivation $t'_1 \rightarrow_{[\gamma(q), l \rightarrow r]} t_2 \xrightarrow{k'_1 - 1} c'_1$ et la longueur de la dérivation $t_2 \xrightarrow{*} c'_1$ est $k'_2 = k'_1 - 1 < k_1$. t_2 est un terme-graphe admissible, c'_1 est un graphe constructeur et $t_2 \xrightarrow{*} c'_1$ est une dérivation de réécriture

de longueur $k_2' \geq 0$. Donc il existe un graphe constructeur c_2 et une dérivation de réécriture $t_2 \stackrel{*}{\rightarrow}_{\bar{\Phi}} c_2$ de longueur k_2 tels que $k_2 \leq k_2' < k_1$ et $c_2 \sim c_1'$, d'après la Proposition 6.1.13.

s_2 est un sous-graphe de S_2, t_2 est un sous-graphe de T_2, $h_2 : S_2 \rightarrow T_2$ est un $\mathcal{D}$-monomorphisme tel que $h_2(s_2) = t_2$ et $(\sigma_{h_2})_{|\mathcal{V}_{s_2}}$ soit une substitution constructeur. De plus, $t_2 \stackrel{k_2}{\rightarrow}_{\bar{\Phi}} c_2$ et $k_2 < k_1$. Donc par hypothèse d'induction, il existe un graphe constructeur s et deux substitutions σ et η tels que $s_2 \blacktriangleright\!\!\rightsquigarrow^{m'}_{\bar{\Lambda},\sigma} s$, $m' \leq k_2$, $\eta(s) \doteq c_2$ et $\eta \circ \sigma \doteq \sigma_{h_2}\ [\mathcal{V}_{S_2}]$.

Nous devons prouver que $\eta(s) \doteq c_1$. C'est immédiat puisque $\eta(s) \doteq c_2$, $c_2 \sim c_1'$ et $c_1' \doteq c_1$. Nous prouvons maintenant que $\eta \circ (\sigma \circ \sigma') \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. Soit $x \in \mathcal{V}_{S_1}$ et $n \in \mathcal{N}_{S_1}$ tels que $\mathcal{L}_{s_1}(n) = x$. D'après le Lemme 6.1.12, $\sigma_{h_1}(n : x) \doteq \sigma_{h_2}(\sigma'(n : x))$. De plus, comme $n : x$ est un sous-graphe variable (donc constructeur) de S_1, $\sigma'(n : x)$ est bisimilaire à un sous-graphe constructeur de S_2. Donc $\mathcal{V}_{\sigma'(n:x)} \subseteq \mathcal{V}_{S_2}$. Par hypothèse d'induction, $\eta \circ \sigma \doteq \sigma_{h_2}\ [\mathcal{V}_{S_2}]$. Donc nous concluons que $\sigma_{h_1}(n : x) \doteq \sigma_{h_2}(\sigma'(n : x) \doteq \eta(\sigma(\sigma'(n : x)))$, i.e., $\eta \circ (\sigma \circ \sigma') \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. Enfin, la dérivation $s_1 \blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda},\sigma'} s_2 \blacktriangleright\!\!\rightsquigarrow^{m'}_{\bar{\Lambda},\sigma} s$ est de longueur $m = m' + 1$. Comme $m' \leq k_2 < k_1$, on déduit que $m \leq k_1$. □

Nous sommes prêt pour montrer la complétude de $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$:

Preuve du Théorème 6.1.9 : Le Théorème 6.1.9 est une conséquence immédiate de la Proposition 6.1.14 : Soient g un terme-graphe admissible, c un graphe constructeur et θ une substitution constructeur tels que $\theta(g) \stackrel{*}{\rightarrow} c$. D'après la Proposition 6.1.13, il existe un graphe constructeur d et une dérivation de réécriture $\theta(g) \stackrel{*}{\rightarrow}_{\bar{\Phi}} d$ tels que $d \doteq c$. Soient $S_1 = s_1 = g$ et $T_1 = t_1 = \theta(g)$. Il est clair qu'il existe un $\mathcal{D}$-monomorphisme $h_1 : S_1 \rightarrow T_1$ tels que $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur (puisque $(\sigma_{h_1})_{|\mathcal{V}_{s_1}} = \theta_{|\mathcal{V}_{s_1}}$). Donc, d'après la Proposition 6.1.14, il existe un graphe constructeur s et deux substitutions σ et η tels que $g \blacktriangleright\!\!\rightsquigarrow^{*}_{\bar{\Lambda},\sigma} s$, $\eta(s) \doteq d$ et $\eta \circ \sigma \doteq \theta\ [\mathcal{V}_g]$. Comme $\eta(s) \doteq c$, nous déduisons que $s \dot{\leq} c$. Comme $\eta \circ \sigma \doteq \theta\ [\mathcal{V}_g]$, nous concluons que $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. □

6.2 Propriétés d'optimalité de $\blacktriangleright\!\!\rightsquigarrow_{\Lambda}$ et de $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$

Les relations de Λ-surréduction compressante et de $\bar{\Lambda}$-surréduction compressante sont plus intéressantes que la relation de surréduction compressante la plus générale étudiée dans le chapitre précédent. Tout d'abord, elle diminue drastiquement l'espace de recherche d'un algorithme fondé sur la surréduction de graphes. De plus, les substitutions calculées par Λ sont *disjointes*, ce qui implique que Λ ne fait pas de calcul redondant. Ensuite, les dérivations de Λ-surréduction compressante et de $\bar{\Lambda}$-surréduction compressante sont toujours plus courtes que celles engendrées par la relation de surréduction compressante la plus générale. Enfin, nous montrerons que Λ ne calcule des pas *nécessaires* de surréduction. Nous préciserons cette notion le moment venu.

6.2.1 Indépendance des substitutions calculées

Dans cette partie, nous nous intéressons aux substitutions calculées par Λ et par $\bar{\Lambda}$.

Définition 6.2.1 (Substitutions disjointes)
Etant données deux substitutions θ et θ' et un ensemble V de variables, on dit que θ et θ' sont *disjointes sur* V ssi il existe $x \in V$ tel que $\theta(x)$ et $\theta'(x)$ ne soient pas unifiables. ◇

On constate dans l'Exemple 6.1.5 que $\bar{\Lambda}(g)$ ne calcule pas des substitutions disjointes. En effet, nous avons trouvé que $\bar{\Lambda}(g) = \{(\mathtt{n2}, \mathtt{R3}, \sigma_1), (\mathtt{n5}, \mathtt{R4}, \sigma_2), (\mathtt{n7}, \mathtt{R4}, \sigma_2), (\mathtt{n6}, \mathtt{R7}, Id)\}$ où les substitutions σ_1 et σ_2 sont définies par $\sigma_1 = \{\mathtt{x} \mapsto \mathtt{r1} : \mathtt{a}, \mathtt{y} \mapsto \mathtt{r2} : \mathtt{a}\}$ et $\sigma_2 = \{\mathtt{y} \mapsto \mathtt{r3} : \mathtt{a}\}$. Il est clair que Id, σ_1 et σ_2 ne sont pas disjointes.

Par contre, nous obtenons le résultat suivant pour Λ :

Theorème 6.2.2 Soit SP un ISGRS, $\blacktriangleright$ une stratégie de compression, g un terme-graphe admissible, c et c' deux graphes constructeurs et $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma} c$ et $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'} c'$ deux Λ-dérivations de surréduction compressante distinctes. Alors σ et σ' sont disjointes sur $\mathcal{V}_g$. $\diamond$

Pour prouver ce théorème, on commence par montrer le lemme suivant qui s'inspire de [AEH94b, Proposition 3] :

Lemme 6.2.3 Soient SP un ISGRS et g un terme-graphe admissible. Si (p_1, R_1, σ_1) et (p_2, R_2, σ_2) sont deux triplets distincts de $\Lambda(g)$, alors σ_1 et σ_2 sont disjointes sur $\mathcal{V}_g$. $\diamond$

Preuve : Soit m le plus haut nœud fonctionnel à gauche dans g et $\mathcal{T}$ un arbre de définition de l'étiquette de m dans g. (p_1, R_1, σ_1) et (p_2, R_2, σ_2) sont deux triplets distincts de $\Lambda(g)$ si et seulement si (p_1, R_1, σ_1) et (p_2, R_2, σ_2) sont deux triplets distincts de $\lambda(g_{|m}, \mathcal{T})$, par définition de Λ. Aussi, nous supposons que g est un terme-graphe admissible de racine fonctionnelle et que $\mathcal{T}$ est un pdt dont le motif s'unifie avec g. On note $\sqsubset$ l'ordre nœthérien défini par $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ ssi le graphe g_1 a moins d'occurrences d'opérations définies que le graphe g_2, ou alors $g_1 = g_2$ et $\mathcal{T}_1$ est un sous-arbre propre de $\mathcal{T}_2$. On montre par induction nœthérienne sur $\sqsubset$ que si (p_1, R_1, σ_1) et (p_2, R_2, σ_2) sont deux triplets distincts de $\lambda(g, \mathcal{T})$, alors σ_1 et σ_2 sont disjointes sur $\mathcal{V}_g$. Nous étudions les différents cas de la définition de $\lambda(g, \mathcal{T})$:

Cas de base : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle et $\mathcal{T} = \mathit{rule}(\pi \to r)$ où $\pi \to r$ est une variante d'une règle de $\mathcal{R}$. Comme π et g sont unifiables, $\lambda(g, \mathcal{T})$ est défini et $\lambda(g, \mathcal{T}) = \{(\mathcal{R}\mathrm{oot}_g, \pi \to r, \sigma)\}$ où σ est un unificateur le plus général de g par rapport à π. Comme $\lambda(g, \mathcal{T})$ est un singleton, il n'existe pas deux triplets distincts dans $\lambda(g, \mathcal{T})$. Donc le lemme est trivialement vrai.

Cas d'induction : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle, $\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, o est un nœud variable de π, o est de sorte ζ, $c_1, \ldots, c_k$ $(k > 0)$ sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt de motif $\pi[o \leftarrow p : c_j(o_1 : x_1, \ldots, o_n : x_n)]$ tel que n soit le nombre d'arguments de c_j, $x_1, \ldots, x_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds. Comme π et g s'unifient et comme π est un motif, on déduit qu'il existe un unificateur le plus général τ de g par rapport à π ainsi qu'un homomorphisme $h : \pi \to \tau(g)$. Trois cas sont à étudier, selon l'étiquette du nœud $h(o)$ dans g.

Cas 1 : $h(o)$ est un nœud constructeur de g :
Puisque $\lambda(g, \mathcal{T})$ est défini, il existe $i \in 1..k$ tel que $\mathit{pattern}(\mathcal{T}_i)$ et g s'unifient et $\lambda(g, \mathcal{T}) \supseteq \lambda(g, \mathcal{T}_i)$. De plus, $h(o)$ est un nœud constructeur de g donc il existe un et un seul $i \in 1..k$ tel que $\mathit{pattern}(\mathcal{T}_i)$ et g s'unifient, par construction des nœuds étiquetés par branch dans les arbres de définition. En conséquence, $\lambda(g, \mathcal{T}) = \lambda(g, \mathcal{T}_i)$. Par hypothèse d'induction, le lemme est vrai pour $\lambda(g, \mathcal{T}_i)$ donc il est aussi vrai pour $\lambda(g, \mathcal{T})$.

Cas 2 : $h(o)$ est un nœud fonctionnel de g :
Posons $\mathcal{L}_g(h(o)) = f$. Soit $\mathcal{T}'$ l'arbre de définition de f. Puisque $\lambda(g, \mathcal{T})$ est défini, $\lambda(\tau(g_{|h(o)}), \mathcal{T}')$ est aussi défini. Soient (p_1, R_1, σ_1) et (p_2, R_2, σ_2) deux triplets distincts de $\lambda(g, \mathcal{T})$. Par définition de $\lambda(g, \mathcal{T})$, il existe deux triplets (p_1, R_1, σ'_1) et (p_2, R_2, σ'_2) dans $\lambda(\tau(g_{|h(o)}), \mathcal{T}')$ tels que $\sigma_1 = \sigma'_1 \circ \tau$ et $\sigma_2 = \sigma'_2 \circ \tau$. Comme τ est un unificateur le plus général de g par rapport à π, τ est une substitution constructeur. Donc $\tau(g_{|h(o)})$ contient moins de nœuds fonctionnels que g. Donc $(\tau(g_{|h(o)}), \mathcal{T}') \sqsubset (g, \mathcal{T})$. Aussi, par hypothèse d'induction, σ'_1 et σ'_2 sont disjointes sur $\mathcal{V}_{\tau(g_{|h(o)})}$, i.e., il existe une variable $y \in \mathcal{V}_{\tau(g_{|h(o)})}$ tel que $\sigma'_1(y)$ et $\sigma'_2(y)$ ne soient pas unifiables. Comme $y \in \mathcal{V}_{\tau(g_{|h(o)})}$, il existe un nœud $n \in \mathcal{N}_{(g_{|h(o)})}$ et une variable $x \in \mathcal{V}_{(g_{|h(o)})}$ tels que $\mathcal{L}_{(g_{|h(o)})}(n) = x$ et $y \in \mathcal{V}_{\tau(n:x)}$). Il est clair que $\sigma'_1(y)$ est un sous-graphe de $\sigma'_1(\tau(n : x))$, c'est-à-dire de $\sigma_1(n : x)$. De même, $\sigma'_2(y)$ est un sous-graphe de $\sigma_2(n : x)$. Comme $\sigma'_1(y)$ et $\sigma'_2(y)$ ne sont pas unifiables, on déduit que $\sigma_1(n : x)$ et $\sigma_2(n : x)$ ne sont pas unifiables. De plus, $x \in \mathcal{V}_{(g_{|h(o)})}$ et $\mathcal{V}_{(g_{|h(o)})} \subseteq \mathcal{V}_g$. Par conséquent, σ_1 et σ_2 sont disjointes sur $\mathcal{V}_g$.

Cas 3 : $h(o)$ est un nœud variable de g :
Posons $\mathcal{L}_g(h(o)) = y$. τ est un unificateur le plus général de g par rapport à π et h est un homomorphisme allant de π vers $\tau(g)$. De plus, o et $h(o)$ sont étiquetés avec des variables. Donc il existe

un nœud $n \in \mathcal{N}_g$ et une variable $x \in \mathcal{V}_g$ tels que $\mathcal{L}_g(n) = x$, $h(o) \in \mathcal{N}_{\tau(n:x)}$ et $y \in \mathcal{V}_{\tau(n:x)}$. Puisque $pattern(\mathcal{T}_i) = \pi[o \leftarrow p : c_i(o_1 : x_1, \ldots, o_n : x_n)]$ pour tout $i \in 1..k$, il est clair que $pattern(\mathcal{T}_i)$ s'unifie avec g pour tout $i \in 1..k$. Par conséquent, $\lambda(g, \mathcal{T}) = \bigcup_{i \in 1..k} \lambda(s, \mathcal{T}_i)$. Par hypothèse d'induction, si (p_1, R_1, σ_1) et (p_2, R_2, σ_2) sont deux triplets distincts de $\lambda(g, \mathcal{T}_i)$ pour un certain $i \in 1..k$, alors σ_1 et σ_2 sont disjointes sur $\mathcal{V}_g$. Soient $i \neq j$, $(p_1, R_1, \sigma_1) \in \lambda(g, \mathcal{T}_i)$ et $(p_2, R_2, \sigma_2) \in \lambda(g, \mathcal{T}_j)$ deux triplets distincts de $\lambda(g, \mathcal{T})$. Par construction d'un arbre de définition, il existe deux homomorphismes $h_i : \pi \rightarrow pattern(\mathcal{T}_i)$ et $h_j : \pi \rightarrow pattern(\mathcal{T}_j)$. $pattern(\mathcal{T}_i)$ et $pattern(\mathcal{T}_j)$ ont deux constructeurs différents aux nœuds $h_i(o)$ et $h_j(o)$. Comme $(p_1, R_1, \sigma_1) \in \lambda(g, \mathcal{T}_i)$, il existe un homomorphisme $h'_i : \pi \rightarrow \sigma_1(g)$. De même, il existe un homomorphisme $h'_2 : \pi \rightarrow \sigma_2(g)$. Comme $h_i(o)$ et $h_j(o)$ sont étiquetés par deux constructeurs différents, $h'_1(o)$ et $h'_2(o)$ sont aussi étiquetés par deux constructeurs différents. Donc $\sigma_1(n : x)$ et $\sigma_2(n : x)$ ne sont pas unifiables. Par conséquent, σ_1 et σ_2 sont disjointes sur $\mathcal{V}_g$. □

Preuve du Théorème 6.2.2 : Soient g un terme-graphe admissible, c et c' deux graphes constructeurs et $g \overset{+}{\blacktriangleright\!\!\rightsquigarrow}_{\Lambda,\sigma} c$ et $g \overset{+}{\blacktriangleright\!\!\rightsquigarrow}_{\Lambda,\sigma'} c'$ deux Λ-dérivations de surréduction distinctes. Nous commençons par prouver le théorème lorsque ce sont les premiers pas des dérivations $g \overset{+}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma} c$ et $g \overset{+}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'} c'$ qui diffèrent, i.e., nous supposons que ces dérivations sont de la forme $g \blacktriangleright\!\!\rightsquigarrow_{\sigma_1} g_1 \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'_1} c$ et $g \blacktriangleright\!\!\rightsquigarrow_{\sigma_2} g_2 \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'_2} c'$ et nous supposons que les triplets $(p_1, R_1, \sigma_1) \in \Lambda(g)$ et $(p_2, R_2, \sigma_2) \in \Lambda(g)$ sont différents. D'après le lemme 6.2.3, les substitutions σ_1 et σ_2 sont disjointes sur $\mathcal{V}_g$. Donc il existe un nœud $n \in \mathcal{N}_g$ et une variable $x \in \mathcal{V}_g$ tels que $\mathcal{L}_g(n) = x$ et $\sigma_1(x)$ et $\sigma_2(x)$ ne soient pas unifiables. Puisque $\sigma = \sigma'_1 \circ \sigma_1$ et $\sigma' = \sigma'_2 \circ \sigma_2$, il est clair que $\sigma(x)$ et $\sigma'(x)$ ne sont pas non plus unifiables. Donc σ et σ' sont disjointes sur $\mathcal{V}_g$.

Nous nous attaquons maintenant au cas général, en supposant que les dérivations de surréduction que nous considérons sont de la forme $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\theta} g' \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'_1} c$ et $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\theta} g' \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'_2} c'$ et en supposant que les premiers pas des dérivations $g' \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'_1} c$ et $g' \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\sigma'_2} c'$ sont différents. Dans ce cas, nous avons montré que σ'_1 et σ'_2 étaient disjointes sur $\mathcal{V}_{g'}$. Donc il existe une variable $y \in \mathcal{V}_{g'}$ telle que $\sigma'_1(y)$ et $\sigma'_2(y)$ ne soient pas unifiables. Il y a deux cas à étudier selon que $y \in \mathcal{V}_g$ ou qu'il existe $x \in \mathcal{V}_g$ telle que $y \in \mathcal{V}_{\theta(x)}$. Premier cas, si $y \in \mathcal{V}_g$, alors $y \notin \mathcal{D}\theta$. Donc $\sigma(y) = (\sigma'_1 \circ \theta)(y) = \sigma'_1(y)$ et $\sigma'(y) = (\sigma'_2 \circ \theta)(y) = \sigma'_2(y)$. Puisque $\sigma'_1(y)$ et $\sigma'_2(y)$ ne sont pas unifiables, nous concluons que $\sigma(y)$ et $\sigma'(y)$ ne sont pas unifiables, c'est-à-dire que σ et σ' sont disjointes sur $\mathcal{V}_g$. Deuxième cas, nous supposons qu'il existe $x \in \mathcal{V}_g$ telle que $y \in \mathcal{V}_{\theta(x)}$ et $\sigma'_1(y)$ et $\sigma'_2(y)$ ne soient pas unifiables. Comme $\sigma(x) = \sigma'_1(\theta(x))$, $\sigma'(x) = \sigma'_2(\theta(x))$ et $y \in \mathcal{V}_{\theta(x)}$, il est clair que $\sigma(x)$ et $\sigma'(x)$ ne sont pas unifiables, donc que σ et σ' sont disjointes sur $\mathcal{V}_g$. □

6.2.2 Longueur des dérivations engendrées

La restriction de Λ aux termes du premier ordre développe toujours les plus petites dérivation de surréduction [AEH94a]. Aussi, lorsque nous avons conçu Λ, nous espérions montrer que les dérivations de surréduction engendrées par $\overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\Lambda}$ étaient de longueurs plus petites que toute autre dérivation de surréduction. C'est malheureusement faux :

Exemple 6.2.4 Soit $g =$ `n1 :A(n2 :B(n3 :X,n4 :C),n5 :B(n6 :Y,n4)` un terme-graphe admissible et `(R) l1 :B(l2 :C,l3 :Z)` → `l3 :Z` une règle de réécriture. Le lecteur peut vérifier que $\Lambda(g) =$ `(n2,R,`σ_1`)` où $\sigma_1 =$ {`X` ↦ `p :C`}. On a donc $g \blacktriangleright\!\!\rightsquigarrow_{\Lambda,\sigma_1} g'$ avec $g' =$ `p1 :A(p2 :C,p3 :B(n6 :Y,p2))`. De plus, comme $\Lambda(g') =$ `(n5,R,`σ_2`)` avec $\sigma_2 =$ {`Y` ↦ `q :C`}, on déduit que $g' \blacktriangleright\!\!\rightsquigarrow_{\Lambda,\sigma_2} c$ où $c =$ `m1 :A(m2 :C,m2)`. Par conséquent, il existe une Λ-dérivation $g \overset{*}{\blacktriangleright\!\!\rightsquigarrow}_{\Lambda,\sigma} c$ où $\sigma =$ {`X` ↦ `m :C`, `Y` ↦ `m :C`}. Cette Λ-dérivation est composée de

deux Λ-pas de surréduction. Pourtant, on peut aussi effectuer la dérivation $g \blacktriangleright\!\!\rightsquigarrow_{[\mathtt{n2},\mathtt{R},\sigma]} c$ qui est, elle, composée d'*un seul* pas de surréduction. □

Néanmoins, nous obtenons le résultat suivant :

Theorème 6.2.5 Soient SP un WAGRS, $\blacktriangleright$ une stratégie de compression, g un terme-graphe admissible, c un graphe constructeur et θ une substitution constructeur tels que $g \overset{n}{\rightsquigarrow}_{\theta} c$ soit une dérivation *quelconque* de surréduction *non compressante* de longueur n. Alors il existe un graphe constructeur s et une substitution σ tels que $g \overset{m}{\blacktriangleright\!\!\rightsquigarrow}_{\bar{\Lambda},\sigma} s$, $s \dot{\leq} c$, $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$ et $m \leq n$. ◇

Preuve : Par hypothèse, $g \overset{n}{\rightsquigarrow}_{\theta} c$ est une dérivation de surréduction non compressante de longueur n. Dans ce cas, on déduit de la preuve du théorème de cohérence (Théorème 5.3.4) qu'il existe une dérivation de réécriture $\theta(g) \overset{n}{\rightarrow} c'$ de longueur n telle que $c' \sim c$. Comme $\theta(g) \overset{n}{\rightarrow} c'$, il existe une $\bar{\Phi}$-dérivation de réécriture $\theta(g) \overset{p}{\rightarrow}_{\bar{\Phi}} c''$ de longueur p telle que $p \leq n$ et $c'' \sim c'$, d'après la Proposition 6.1.13. Utilisons maintenant la Proposition 6.1.14 avec $S_1 = s_1 = g$ et $T_1 = t_1 = \theta(g)$. Il est clair qu'il existe un $\mathcal{D}$-monomorphisme $h_1 : S_1 \rightarrow T_1$ tel que $h_1(s_1) = t_1$ et $(\sigma_{h_1})_{|\mathcal{V}_{s_1}}$ soit une substitution constructeur. Comme c'' est un graphe constructeur et $\theta(g) = t_1 \overset{p}{\rightarrow}_{\bar{\Phi}} c''$ est une dérivation de réécriture de longueur p, il existe un graphe constructeur s et deux substitutions σ et η tels que $s_1 = g \overset{m}{\blacktriangleright\!\!\rightsquigarrow}_{\bar{\Lambda},\sigma} s$, $m \leq p$, $\eta(s) \doteq c''$ et $\eta \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$. Comme $\eta(s) \doteq c''$, $c'' \sim c'$ et $c' \sim c$, on déduit que $s \dot{\leq} c$. Comme $\eta \circ \sigma \doteq \sigma_{h_1}\ [\mathcal{V}_{S_1}]$, $S_1 = g$ et $\sigma_{h_1} = \theta$, on infère que $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. Enfin, comme $m \leq p$ et $p \leq n$, on conclut que $m \leq n$. □

6.2.3 Nécessité des Λ-pas de surréduction compressante

Dans [AEH94a], les auteurs montrent que les Λ-pas de surréduction de termes du premier ordre sont des pas *nécessaires* de surréduction. Nous généralisons leur définition dans le cas de la Λ-surréduction de graphes admissibles :

Définition 6.2.6 (Pas de surréduction nécessaire)
Soient SP un cGRS, $\blacktriangleright$ une stratégie de compression et g un terme-graphe admissible. Un pas de surréduction $g \blacktriangleright\!\!\rightsquigarrow_{[p,R,\sigma]} g'$ est *un plus haut pas nécessaire de surréduction*[5] ssi pour toute substitution constructeur η telle que $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$, $\eta(g_{|p})$ est un plus haut rédex nécessaire de $\eta(g)$. ◇

Nous obtenons le résultat suivant :

Theorème 6.2.7 Soient SP un ISGRS, $\blacktriangleright$ une stratégie de compression, g un terme-graphe admissible et η une substitution constructeur tels que $\eta(g)$ soit $\mathcal{C}$-normalisable. Alors :

1. Il existe un unique triplet $(p, R, \sigma) \in \Lambda(g)$ tel que $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$.
2. $\eta(g_{|p})$ est un plus haut rédex nécessaire de $\eta(g)$.

Autrement dit, les triplets calculés par Λ engendrent des plus hauts pas nécessaires de surréduction. ◇

Ce théorème repose sur le lemme suivant dont les origines se trouve dans [AEH94b, Theorem 1] :

Lemme 6.2.8 Soient SP un ISGRS et g un terme-graphe admissible.

[5] *outermost-needed narrowing step* en anglais.

- Si $\Lambda(g) \ni (p, R, \sigma)$, alors pour toute substitution η telle que $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$, et pour toute dérivation de réécriture de $\eta(g)$ vers un terme-graphe de racine constructeur, un descendant de $\eta(g_{|p})$ est réécrit à la racine, en un ou plusieurs pas, en un terme-graphe de racine constructeur et $\eta(g_{|p})$ est un plus haut rédex de $\eta(g)$.
- Si $\Lambda(g)$ n'est pas définie, alors g ne peut pas être surréduit en un terme-graphe de racine constructeur.

$\diamond$

Preuve : Soit m le plus haut nœud fonctionnel à gauche dans g et $\mathcal{T}$ un arbre de définition de l'étiquette de m dans g. Dans ce cas, $\Lambda(g) = \lambda(g_{|m}, \mathcal{T})$. Il est clair que le lemme est vrai pour $\Lambda(g)$ ssi il est vrai pour $\lambda(g_{|m}, \mathcal{T})$. Aussi, nous supposons sans perte de généralité que g est un terme-graphe admissible de racine fonctionnelle et que $\mathcal{T}$ est un pdt dont le motif s'unifie avec g. On note $\sqsubset$ l'ordre nœthérien défini par $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ ssi le graphe g_1 a moins d'occurrences d'opérations définies que le graphe g_2, ou alors $g_1 = g_2$ et $\mathcal{T}_1$ est un sous-arbre propre de $\mathcal{T}_2$. On montre le lemme par induction nœthérienne sur $\sqsubset$. Nous étudions les différents cas de la définition de $\lambda(g, \mathcal{T})$:
Cas de base : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle et $\mathcal{T} = rule(\pi \to r)$ où $\pi \to r$ est une variante d'une règle de $\mathcal{R}$. Comme π et g sont unifiables, $\lambda(g, \mathcal{T})$ est défini et $\lambda(g, \mathcal{T}) = \{(\mathcal{R}\text{oot}_g, \pi \to r, \sigma)\}$ où σ est un unificateur le plus général de g par rapport à π. Soit η une substitution telle que $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$. g est un terme-graphe de racine fonctionnelle donc $\eta(g)$ est aussi un terme-graphe de racine fonctionnelle. Donc dans toute dérivation de réécriture de $\eta(g)$ vers un terme-graphe de racine constructeur, un descendant de $\eta(g_{|\mathcal{R}\text{oot}_g})$ doit être réécrit à la racine en un terme-graphe de racine constructeur. D'autre part, σ est un unificateur le plus général de g par rapport à π, donc π filtre $\sigma(g)$ à la racine. Comme $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$, il existe une substitution θ telle que $\theta \circ \sigma \doteq \eta\ [\mathcal{V}_g]$. Comme π filtre $\sigma(g)$ à la racine, on infère que π filtre $\theta(\sigma(g))$ à la racine. Comme $\theta(\sigma(g)) \doteq \eta(g)$, on déduit de la Proposition 3.5.1 que π filtre $\eta(g)$ à la racine. Donc $\eta(g_{|\mathcal{R}\text{oot}_g})$ est plus haut rédex de $\eta(g)$.
Cas d'induction : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle, $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, o est un nœud variable de π, o est de sorte ζ, $c_1, \ldots, c_k$ $(k > 0)$ sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt de motif $\pi[o \leftarrow p : c_j(o_1 : x_1, \ldots, o_n : x_n)]$ tel que n soit le nombre d'arguments de c_j, $x_1, \ldots, x_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds. Comme π et g s'unifient et comme π est un motif, on déduit qu'il existe un unificateur le plus général τ de g par rapport à π ainsi qu'un homomorphisme $h : \pi \to \tau(g)$. Trois cas sont à étudier, selon l'étiquette du nœud $h(o)$ dans $\tau(g)$.
Cas 1 : $h(o)$ est un nœud constructeur :
Dans ce cas, il y a deux possibilités. Si il existe $i \in 1..k$ tel que $pattern(\mathcal{T}_i)$ et g s'unifient, alors $\lambda(g, \mathcal{T}) \supseteq \lambda(g, \mathcal{T}_i)$. Par hypothèse d'induction, le lemme est vrai pour $\lambda(g, \mathcal{T}_i)$ donc il est aussi vrai pour $\lambda(g, \mathcal{T})$. Si il n'existe pas $i \in 1..k$ tel que $pattern(\mathcal{T}_i)$ et g s'unifient, alors l'opération $\mathcal{L}_g(\mathcal{R}\text{oot}_g)$ est incomplètement définie par le système de réécriture $\mathcal{R}$. Donc g ne peut pas être surréduit en un terme-graphe de racine constructeur et $\lambda(g, \mathcal{T})$ n'est pas défini.
Cas 2 : $h(o)$ est un nœud fonctionnel de g :
Posons $\mathcal{L}_g(h(o)) = f$. Soit $\mathcal{T}'$ l'arbre de définition de f. On distingue le cas où $\lambda(\tau(g_{|h(o)}), \mathcal{T}')$ est défini et le cas où $\lambda(\tau(g_{|h(o)}), \mathcal{T}')$ n'est pas défini.
Cas 2.1 : Supposons que $\lambda(\tau(g_{|h(o)}), \mathcal{T}') \ni (p, R, \sigma')$, où p est un nœud fonctionnel de $\tau(g_{|h(o)})$, R est une variante d'une règle de $\mathcal{R}$ et σ' est une substitution. Alors $\lambda(g, \mathcal{T})$ est défini et $\lambda(g, \mathcal{T}) \ni (p, R, \sigma)$ avec $\sigma = \sigma' \circ \tau$. Soit η une substitution telle que $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$. Nous montrons que dans toute dérivation de réécriture de $\eta(g)$ vers un terme-graphe de racine constructeur, un descendant de $\eta(g_{|h(o)})$ doit être réécrit à la racine en un terme-graphe de racine constructeur.

Supposons que $\eta(g)$ puisse se réécrire à la racine en un terme-graphe de racine constructeur. Dans ce cas, g lui-même peut se surréduire à la racine en un terme-graphe de racine constructeur (en utilisant par exemple la substitution η). Par définition d'un ISGRS, toutes les règles $l \to r$ de $\mathcal{R}$ dont le membre gauche s'unifie avec g sont représentées par une feuille $rule(l' \to r')$ de $\mathcal{T}$, où $l' \to r'$ est une variante de $l \to r$. Par construction d'un arbre de définition, si $l \to r$ est représentée par une feuille de $\mathcal{T}$, alors

π filtre l à la racine, i.e., il existe un homomorphisme $h' : \pi \to l$. Par hypothèse, $\mathcal{T}$ est un pdt dont la racine est étiquetée par le symbole *branch* et pas par le symbole *rule*. Donc π filtre l sans être un renommage de l. Ceci implique que $h'(o)$ est un nœud constructeur de l. Pourtant, dans le cas que nous considérons, nous supposons que $h(o)$ est un nœud fonctionnel de g. Donc il n'existe pas de règle de $\mathcal{R}$ dont le membre gauche s'unifie avec g. Par conséquent, g ne peut pas se surréduire à la racine en un terme-graphe de racine constructeur et dans toute dérivation de surréduction de g en un terme-graphe de racine constructeur, il existe un pas de surréduction qui se fait au nœud $h(o)$. De ce fait, on déduit qu'il est impossible de réécrire $\eta(g)$ à la racine en un terme-graphe de racine constructeur (sinon g lui-même pourrait se surréduire à la racine en un terme-graphe de racine constructeur). De plus, dans toute dérivation de réécriture de $\eta(g)$ qui contient un pas de réécriture à la racine, un descendant de $\eta(g_{|h(o)})$ doit être réécrit à la racine en un terme-graphe de racine constructeur.

Nous montrons maintenant que dans toute dérivation de réécriture de $\eta(g_{|h(o)})$ vers un terme-graphe de racine constructeur, un descendant de $\eta((g_{|h(o)})_{|p})$ est réécrit à la racine en un terme-graphe de racine constructeur. Comme τ est un unificateur le plus général de g par rapport à π, τ est une substitution constructeur, donc $\tau(g_{|h(o)})$ a moins d'opérations définies que g. Par conséquent, $(\tau(g_{|h(o)}), \mathcal{T}') \sqsubset (g, \mathcal{T})$. Donc par hypothèse d'induction, pour toute substitution η' telle que $\sigma' \dot{\leq} \eta'\ [\mathcal{V}_{\tau(g_{|h(o)})}]$, et pour toute toute dérivation de réécriture de $\eta'(\tau(g_{|h(o)}))$ vers un terme-graphe de racine constructeur, un descendant de $\eta'(\tau((g_{|h(o)})_{|p}))$ est réécrit en un terme-graphe de racine constructeur. Par hypothèse, $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$ et $\sigma = \sigma' \circ \tau$. Donc il existe une substitution θ telle que $\theta \circ \sigma' \circ \tau \doteq \eta\ [\mathcal{V}_g]$. Posons $\eta' = \theta \circ \sigma'$. On a $\eta' \circ \tau \doteq \eta\ [\mathcal{V}_g]$. De plus, on a $\sigma' \dot{\leq} \eta'\ [\mathcal{V}_{\tau(g)}]$, donc en particulier $\sigma' \dot{\leq} \eta'\ [\mathcal{V}_{\tau(g_{|h(o)})}]$. On déduit donc de l'hypothèse d'induction que dans toute dérivation de réécriture de $\eta'(\tau(g_{|h(o)})) \doteq \eta(g_{|h(o)})$ vers un terme-graphe de racine constructeur, un descendant de $\eta'(\tau((g_{|h(o)})_{|p})) \doteq \eta((g_{|h(o)})_{|p})$ est réécrit en un terme-graphe de racine constructeur.

Nous avons prouvé que dans toute dérivation de réécriture de $\eta(g)$ vers un terme-graphe de racine constructeur, un descendant de $\eta(g_{|h(o)})$ doit être réécrit à la racine en un terme-graphe de racine constructeur. De plus, dans toute dérivation de réécriture de $\eta(g_{|h(o)})$ vers un terme-graphe de racine constructeur, un descendant de $\eta((g_{|h(o)})_{|p}) = \eta(g_{|p})$ est réécrit à la racine en un terme-graphe de racine constructeur. Donc par transitivité, dans toute dérivation de réécriture de $\eta(g)$ vers un terme-graphe de racine constructeur, un descendant de $\eta(g_{|p})$ est réécrit à la racine en un terme-graphe de racine constructeur.

D'autre part, $\eta'(\tau((g_{|h(o)})_{|p})) \doteq \eta(g_{|p})$ est un plus haut rédex de $\eta'(\tau(g_{|h(o)})) \doteq \eta(g_{|h(o)})$ par hypothèse d'induction. Comme π filtre $\tau(g)$ à la racine, π filtre $\eta'(\tau(g))$ à la racine. Comme $\eta'(\tau(g)) \doteq \eta(g)$, on déduit de la Proposition 3.5.1 que π filtre $\eta(g)$ à la racine. π est un motif et o est un nœud variable de π, donc il existe un chemin allant de $\mathcal{R}\text{oot}_{\eta(g)}$ vers $\mu(o)$ dans $\eta(g)$ et tous les nœuds de ce chemin sont constructeurs, sauf $\mathcal{R}\text{oot}_{\eta(g)}$ et $h(o)$. Nous avons montré que $\eta(g)$ n'était pas un rédex. Donc il n'existe pas de rédex sur le chemin allant de $\mathcal{R}\text{oot}_{\eta(g)}$ vers $h(o)$. Comme $\eta(g_{|p})$ est un plus haut rédex de $\eta(g_{|h(o)})$, on conclut que $\eta(g_{|p})$ est un plus haut rédex de $\eta(g)$.

Cas 2.2 : Supposons que $\lambda(\tau(g_{|h(o)}), \mathcal{T}')$ ne soit pas défini. Alors $\lambda(g, \mathcal{T})$ n'est pas défini, lui non plus. De plus, par hypothèse d'induction, $\tau(g_{|h(o)})$ ne peut pas être surréduit (donc réécrit) en un terme-graphe de racine constructeur. De même que dans le cas 2.1, g ne peut pas être surréduit à la racine en un terme-graphe de racine constructeur et dans toute dérivation de réécriture de $\eta(g)$ vers un terme-graphe de racine constructeur, un descendant de $\eta(g_{|h(o)})$ doit être réécrit à la racine en un terme-graphe de racine constructeur. Comme $\tau(g_{|h(o)})$ ne peut pas être réécrit en un terme-graphe de racine constructeur, $\eta(g)$ ne peut pas non plus être réécrit en un terme-graphe de racine constructeur. Si g pouvait se surréduire en un terme-graphe de racine constructeur, alors $\eta(g)$ pourrait se réécrire en un terme-graphe de racine constructeur, ce qui est impossible. Donc g ne peut pas être surréduit en un terme-graphe de racine constructeur.

Cas 3 : $h(o)$ est un nœud variable de g :
τ est un unificateur le plus général de g par rapport à π. De plus, o et $h(o)$ sont étiquetés avec des variables. Donc il existe un nœud q étiqueté par une variable dans g tel que $h(o) \in \mathcal{N}_{\tau(g_{|q})}$. Puisque

$pattern(\mathcal{T}) = \pi[o \leftarrow p : c_j(o_1 : x_1, \ldots, o_n : x_n)]$ pour tout $i \in 1..k$, il est clair que $pattern(\mathcal{T}_i)$ s'unifie avec g pour tout $i \in 1..k$. Par conséquent, $\lambda(g, \mathcal{T}) \supseteq \lambda(g, \mathcal{T}_i)$ pour tout $i \in 1..k$. Par hypothèse d'induction, le lemme est vérifié par $\lambda(g, \mathcal{T}_i)$ pour tout $i \in 1..k$. Donc il l'est aussi par $\lambda(g, \mathcal{T})$. □

Preuve du Théorème 6.2.7 : Soient g un terme-graphe admissible et η une substitution constructeur tels que $\eta(g)$ soit $\mathcal{C}$-normalisable. On déduit du Lemme 6.1.11 qu'il existe un triplet $(p, R, \sigma) \in \Lambda(g)$ tel que $\sigma \dot{\leq} \eta\ [\mathcal{V}_g]$. On doit montrer que ce triplet est unique. Supposons qu'il existe deux triplets distincts (p_1, R_1, σ_1) et (p_2, R_2, σ_2) tels que $\sigma_1 \dot{\leq} \eta\ [\mathcal{V}_g]$ et $\sigma_2 \dot{\leq} \eta\ [\mathcal{V}_g]$. Alors il existe deux substitutions θ_1 et θ_2 telles que $\theta_1 \circ \sigma_1 \doteq \eta\ [\mathcal{V}_g]$ et $\theta_2 \circ \sigma_2 \doteq \eta\ [\mathcal{V}_g]$. En conséquence, pour toute variable $x \in \mathcal{V}_g$, $\theta_1(\sigma_1(x)) \doteq \theta_2(\sigma_2(x))$. Aussi, σ_1 et σ_2 ne sont pas des substitutions disjointes, ce qui est impossible d'après le Lemme 6.2.3. Nous devons encore montrer que $\eta(g_{|p})$ est un plus haut rédex nécessaire de $\eta(g)$. C'est une conséquence immédiate du Lemme 6.2.8. Aussi, nous concluons que $g \blacktriangleright\!\!\rightsquigarrow_{[p,\, R,\, \sigma]} g'$ est un plus haut pas nécessaire de surréduction et donc que les triplets calculés par Λ engendrent des plus hauts pas nécessaires de surréduction. □

6.3 Surréduction parallèle compressante de graphe

Dans la définition d'un pas de surréduction compressante, nous utilisons des pas de réécriture simples. Or la relation de *réécriture parallèle* (cf. Définition 4.3.4) est beaucoup plus efficace. Aussi, on peut définir une nouvelle relation de surréduction compressante :

Définition 6.3.1 (Surréduction parallèle compressante de graphe)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un cGRS, $\blacktriangleright$ une stratégie de compression, g_1 un terme-graphe admissible, g_2 un graphe, σ une substitution $R_1, \ldots, R_n$ n règles de réécriture de $\mathcal{R}$ et $p_1, \ldots, p_n$ n nœuds distincts de g_1. On définit le *pas de surréduction parallèle compressante de g_1 en g_2 aux nœuds $p_1, \ldots, p_n$ avec les règles $R_1, \ldots, R_n$ et la substitution σ* en posant $g_1 \blacktriangleright\!\!\Downarrow_{[p_1, R_1] \ldots [p_n, R_n], \sigma} g_2$ ssi il existe un graphe g_1' tel que $\blacktriangleright(\sigma(g_1)) = g_1'$ avec le $\mathcal{V}$-homomorphisme $h : \sigma(g_1) \rightarrow g_1'$ et $g_1' \twoheadrightarrow_{[h(p_1), R_1] \ldots [h(p_n), R_n]} g_2$. ◇

6.3.1 Définition de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}}, \bar{\Phi}}$

Dans la définition du pas de surréduction parallèle compressante, plusieurs paramètres ne sont pas fixés. En effet, il faut exhiber des nœuds et des règles de réécriture pour effectuer les pas de réécriture parallèle. Pour cela, nous utilisons la stratégie de réécriture $\bar{\Phi}$ (cf. Définition 4.3.14).

Quant aux calculs des substitutions, nous pourrions utiliser celles qui apparaissent dans les triplets calculés par $\bar{\Lambda}$. Mais ces substitutions sont parfois redondantes. En effet, nous avons vu dans l'Exemple 6.1.5 que $\bar{\Lambda}(g) = \{(\mathtt{n2}, \mathtt{R3}, \sigma_1), (\mathtt{n5}, \mathtt{R4}, \sigma_2), (\mathtt{n7}, \mathtt{R4}, \sigma_2), (\mathtt{n6}, \mathtt{R7}, Id)\}$ où les substitutions σ_1 et σ_2 sont définies par $\sigma_1 = \{\mathtt{x} \mapsto \mathtt{r1} : \mathtt{a}, \mathtt{y} \mapsto \mathtt{r2} : \mathtt{a}\}$ et $\sigma_2 = \{\mathtt{y} \mapsto \mathtt{r3} : \mathtt{a}\}$. Il est clair que σ_1 est moins général que σ_2. Aussi, il n'est pas forcément utile de considérer σ_1 tout de suite pour faire un pas de surréduction ; l'affectation de la variable `x` sera éventuellement faite plus tard dans le calcul.

Bref, nous améliorons le calcul des substitutions de $\bar{\Lambda}$ en définissant la *stratégie parallèle de surréduction* $\bar{\bar{\Lambda}}$:

Définition 6.3.2 (Stratégie $\bar{\bar{\Lambda}}$)

Soient SP un WAGRS et g un terme-graphe admissible. $\bar{\bar{\Lambda}}$ est la fonction partielle telle que :

$$\bar{\bar{\Lambda}}(g) = \left\{ \sigma_{|\mathcal{V}_g} \text{ telle que } \left| \begin{array}{l} \exists (p, R, \sigma) \in \bar{\Lambda}(g), \\ (\forall (q, S, \theta) \in \bar{\Lambda}(g), \\ \text{si } \theta \dot{\leq} \sigma \ [\mathcal{V}_g] \text{ et } \theta \not\doteq Id \ [\mathcal{V}_g], \\ \text{alors } \sigma \doteq \theta \ [\mathcal{V}_g]) \\ \text{et} \\ (\exists C \in \mathcal{P}\text{aths}_g(\mathcal{R}\text{oot}_g, p), \\ \forall (q, S, \theta) \in \bar{\Lambda}(g), \\ \text{si } \theta \dot{\leq} \sigma \ [\mathcal{V}_g] \text{ et } q \in C, \\ \text{alors } \sigma \doteq \theta \ [\mathcal{V}_g]) \end{array} \right. \right\} / \doteq$$

◇

Dans la définition de $\bar{\bar{\Lambda}}(g)$, la première condition sélectionne les substitutions les moins instanciées parmi celles de $\bar{\Lambda}(g)$ qui ne sont pas l'identité, en plus de l'identité s'il existe un triplet (p, R, Id) dans $\bar{\Lambda}(g)$. La seconde condition permet d'éliminer les substitutions qui sont en dessous de l'identité dans tous les chemins du graphe. On remarque que cette condition est différente de celle donnée dans [AEH97a] du fait du partage de sous-graphes. Enfin, l'ensemble $\bar{\bar{\Lambda}}(g)$ est défini à la bisimilarité près : si A dénote un ensemble de substitutions, $A/\doteq$ dénote l'ensemble "quotient" de A par $\doteq$ qui est constitué de représentants de A au renommage et à la bisimilarité près.

Définition 6.3.3 ($\bar{\bar{\Lambda}}$-surréduction parallèle compressante)
Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un WAGRS et $\blacktriangleright$ une stratégie de compression. La relation de $\bar{\bar{\Lambda}}$-*surréduction parallèle compressante* $\blacktriangleright\Downarrow_{\bar{\bar{\Lambda}}, \bar{\Phi}}$ est définie par :

$$g_1 \blacktriangleright\Downarrow_{\bar{\bar{\Lambda}}, \bar{\Phi}, \sigma} g_2 \iff \sigma \in \bar{\bar{\Lambda}}(g_1), \blacktriangleright(\sigma(g_1)) = g_1' \text{ et } g_1' \dashrightarrow_{\bar{\Phi}(g_1')} g_2$$

◇

Exemple 6.3.4 Considérons de nouveau l'Exemple 6.1.5. Nous calculons maintenant $\bar{\bar{\Lambda}}(g)$. Dans la Figure 6.8, nous représentons les positions relatives des triplets de $\bar{\Lambda}(g)$ dans g. Ce "graphe" n'a pas de définition formelle mais il facilite la compréhension du calcul de $\bar{\bar{\Lambda}}(g)$. On rappelle que les substitutions σ_1 et σ_2 sont définies par $\sigma_1 = \{\mathtt{x} \mapsto \mathtt{r1} : \mathtt{a}, \mathtt{y} \mapsto \mathtt{r2} : \mathtt{a}\}$ et $\sigma_2 = \{\mathtt{y} \mapsto \mathtt{r3} : \mathtt{a}\}$.

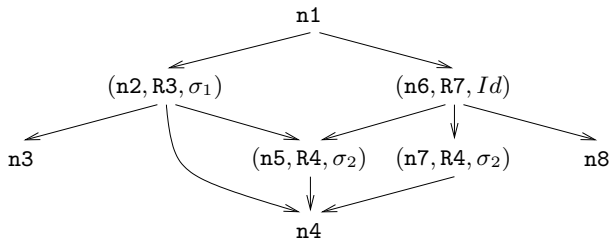
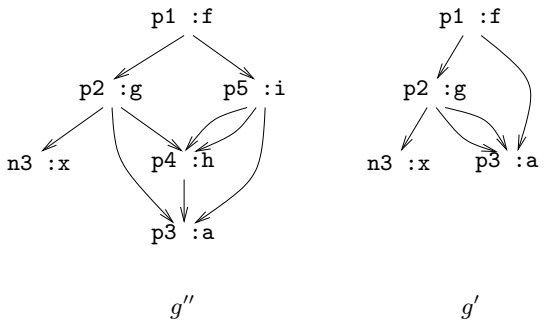


FIG. 6.8:

Calculer $\bar{\bar{\Lambda}}(g)$ consiste à effacer les triplets $(p, R, \sigma) \in \bar{\Lambda}(g)$ tels que $\sigma \notin \bar{\bar{\Lambda}}(g)$. $(\mathtt{n2}, \mathtt{R3}, \sigma_1)$ doit être éliminé parce que $\sigma_2 \dot{\leq} \sigma_1 \ [\mathcal{V}_g]$. $(\mathtt{n7}, \mathtt{R4}, \sigma_2)$ doit être effacé parce que le seul chemin allant de $\mathtt{n1}$ vers $\mathtt{n7}$ (c'est-à-dire, $[\mathtt{n1}, 2, \mathtt{n6}, 2, \mathtt{n7}]$) contient le nœud $\mathtt{n6}$ et $(\mathtt{n6}, \mathtt{R7}, Id) \in \bar{\Lambda}(g)$ et

$Id \dot{\leq} \sigma_2\ [\mathcal{V}_g]$. On conserve $(\texttt{n5}, \texttt{R4}, \sigma_2)$ puisqu'il existe un chemin $C = [\texttt{n1}, 1, \texttt{n2}, 3, \texttt{n5}]$ tel que pour tout $(q, R, \theta) \in \bar{\Lambda}(g)$, si $q \in C$ (par exemple, n2), alors $\sigma_2 \dot{\leq} \theta\ [\mathcal{V}_g]$. On conserve le triplet $(\texttt{n6}, \texttt{R7}, Id)$ pour les mêmes raisons. Par conséquent, nous concluons que $\bar{\bar{\Lambda}}(g) = \{Id, \sigma_2\}$.

Pour finir, calculons le $\bar{\bar{\Lambda}}$-pas de surréduction parallèle compressante maximale commençant avec g avec la substitution σ_2. Le lecteur peut vérifier que $\sigma_2(g) =$ `n1 :f(n2 :g(n3 :x,r3 :a,n5 :h(r3)),n6 :i(n5,n7 :h(r3),n8 :a)`. De plus $\blacktriangleright\!\!\gg(\sigma_2(g)) = g''$ où $g'' =$ `p1 :f(p2 :g(n3 :x,p3 :a,p4 :h(p3)),p5 :i(p4,p4,p3))` (cf. Figure 6.9). D'après la Définition 4.3.14, $\bar{\Phi}(g'') = Outer(g'', S)$ où



g'' g'

FIG. 6.9:

$$\begin{aligned} S &= \bar{\varphi}(g''_{|\texttt{p1}}, \mathcal{T}^1_{\texttt{f}}) \cup \bar{\varphi}(g''_{|\texttt{p1}}, \mathcal{T}^2_{\texttt{f}}) \\ &= \bar{\varphi}(g''_{|\texttt{p2}}, \mathcal{T}_{\texttt{g}}) \cup \bar{\varphi}(g''_{|\texttt{p5}}, \mathcal{T}^1_{\texttt{i}}) \cup \bar{\varphi}(g''_{|\texttt{p5}}, \mathcal{T}^2_{\texttt{i}}) \cup \bar{\varphi}(g''_{|\texttt{p5}}, \mathcal{T}^3_{\texttt{i}}) \\ &= \emptyset \cup \bar{\varphi}(g''_{|\texttt{p4}}, \mathcal{T}_{\texttt{h}}) \cup \bar{\varphi}(g''_{|\texttt{p4}}, \mathcal{T}_{\texttt{h}}) \cup \{(\texttt{p5}, \texttt{R7})\} \\ &= \{(\texttt{p4}, \texttt{R4}), (\texttt{p5}, \texttt{R7})\} \end{aligned}$$

$Outer(g'', S)$ sélectionne les plus hauts nœuds de S dans g'', donc $\bar{\Phi}(g'') = \{(\texttt{p4}, \texttt{R4}), (\texttt{p5}, \texttt{R7})\}$. Le lecteur peut vérifier que $g'' \mathrel{+\!\!+\!\!\rightarrow}_{\bar{\Phi}} g'$ avec $g' =$ `p1 :f(p2 :g(n3 :x,p3 :a,p3),p3)` (cf. Figure 6.9). Donc on conclut que $g \blacktriangleright\!\!\gg\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma_2} g'$. □

6.3.2 Cohérence de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$

Dans cette section, nous montrons le résultat suivant :

Theorème 6.3.5 Soient SP un WAGRS et $\blacktriangleright$ une stratégie de compression. $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ est cohérente sur SP, i.e., pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution θ tels que $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\theta} c$, il existe un terme-graphe constructeur s tel que $\theta(g) \xrightarrow{*} s$ et $s \doteq c$. ◇

Preuve : On montre de manière plus générale que pour tout terme-graphe admissible g_1 et g_2 et pour toute substitution θ tels que $g_1 \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\theta} g_2$, il existe un terme-graphe g'_2 tel que $\theta(g) \xrightarrow{*} g'_2$ et $g'_2 \rhd g_2$. Nous faisons ceci par induction sur la longueur n de la dérivation $g_1 \overset{n}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\theta} g_2$. Le cas $n = 0$ est évident. Soit $g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_2 \overset{n}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\theta} g_3$ une dérivation de longueur $n+1$. Comme $g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_2$, il existe un terme-graphe g'_1 tel que $\blacktriangleright(\sigma(g_1)) = g'_1$ et $g'_1 \mathrel{+\!\!+\!\!\rightarrow}_{\bar{\Phi}} g_2$. Comme $g'_1 \mathrel{+\!\!+\!\!\rightarrow}_{\bar{\Phi}} g_2$, on déduit que $g'_1 \xrightarrow{*} g_2$ d'après la Proposition 4.3.6. Nous avons $\sigma(g_1) \rhd g'_1$ et $g'_1 \xrightarrow{*} g_2$, donc d'après la Proposition 5.4.1, il existe un terme-graphe g'_2 tel que $\sigma(g_1) \xrightarrow{*} g'_2$ et $g'_2 \rhd g_2$. Par ailleurs, nous avons $g_2 \overset{n}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\theta} g_3$, donc il existe un terme-graphe g'_3 tel que $\theta(g_2) \xrightarrow{*} g'_3$ et $g'_3 \rhd g_3$ par hypothèse

d'induction. Comme $g'_2 \rhd g_2$, il est clair que $\theta(g'_2) \rhd \theta(g_2)$. Nous avons $\theta(g'_2) \rhd \theta(g_2)$ et $\theta(g_2) \stackrel{*}{\to} g'_3$, donc d'après la Proposition 5.4.1, il existe un terme-graphe g''_3 tel que $\theta(g'_2) \stackrel{*}{\to} g''_3$ et $g''_3 \rhd g'_3$. Nous avons vu que $\sigma(g_1) \stackrel{*}{\to} g'_2$, donc $\theta(\sigma(g_1)) \stackrel{*}{\to} \theta(g'_2)$. Comme $\theta(\sigma(g_1)) \stackrel{*}{\to} \theta(g'_2)$ et $\theta(g'_2) \stackrel{*}{\to} g''_3$, nous déduisons que $\theta(\sigma(g_1)) \stackrel{*}{\to} g''_3$. Comme $g''_3 \rhd g'_3$ et $g'_3 \rhd g_3$, nous concluons que $g''_3 \rhd g_3$. □

6.3.3 Complétude de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$

Dans ce paragraphe, nous nous attaquons au résultat suivant :

Theorème 6.3.6 Soient SP un WAGRS et $\blacktriangleright$ une stratégie de compression. $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ est complète sur SP, i.e., pour tout terme-graphe admissible g, pour tout graphe constructeur c et pour toute substitution constructeur θ tels que $\theta(g) \stackrel{*}{\to} c$, il existe un graphe constructeur s et une substitution σ tels que $g \stackrel{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta$ $[\mathcal{V}_g]$. ◇

Notre preuve du théorème ci-dessus est proche de celle donnée dans [AEH97b] pour le cas des termes du premier ordre. Elle nécessite plusieurs propositions que nous énonçons maintenant.

Proposition 6.3.7 Soient SP un WAGRS, g un terme-graphe admissible et $\mathcal{T}$ un pdt dont le motif s'unifie avec g. Si $(p, R, \theta) \in \bar{\lambda}(g, \mathcal{T})$, alors $(p, R) \in \bar{\varphi}(\theta(g), \mathcal{T})$. ◇

Preuve : On note $\sqsubset$ l'ordre nœthérien défini par $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ ssi le graphe g_1 a moins d'occurrences d'opérations définies que le graphe g_2, ou alors $g_1 = g_2$ et $\mathcal{T}_1$ est un sous-arbre propre de $\mathcal{T}_2$. On montre par induction nœthérienne sur $\sqsubset$ que si $(p, R, \theta) \in \bar{\lambda}(g, \mathcal{T})$, alors $(p, R) \in \bar{\varphi}(\theta(g), \mathcal{T})$. Nous étudions les différents cas de la définition de $\bar{\lambda}(g, \mathcal{T})$:
Cas de base : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle et $\mathcal{T} = rule(\pi \to r)$ tel que $\pi \to r$ soit une variante d'une règle de $\mathcal{R}$. Soit $(p, R, \theta) \in \bar{\lambda}(g, \mathcal{T})$. On déduit de la définition de $\bar{\lambda}$ que $p = \mathcal{R}oot_g$, $R = \pi \to r$ et θ est un unificateur le plus général de g par rapport à π. Comme θ est un unificateur de g par rapport à π, il existe un homomorphisme $h : \pi \to \theta(g)$. Donc π filtre $\theta(g)$ à la racine. Par conséquent, $\bar{\varphi}(\theta(g), \mathcal{T})$ est défini et $\bar{\varphi}(\theta(g), \mathcal{T}) = \{(\mathcal{R}oot_g, \pi \to r)\}$. Donc $(p, R) \in \bar{\varphi}(\theta(g), \mathcal{T})$.
Cas d'induction : Considérons le couple $(g, \mathcal{T})$ où g est un terme-graphe admissible de racine fonctionnelle, $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, o est un nœud variable de π, o est de sorte ζ, $c_1, \ldots, c_k$ $(k > 0)$ sont différents constructeurs de la sorte ζ et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un pdt de motif $\pi[o \leftarrow p : c_j(o_1 : x_1, \ldots, o_n : x_n)]$ tel que n soit le nombre d'arguments de c_j, $x_1, \ldots, x_n$ soient des variables fraîches et $p, o_1, \ldots, o_n$ soient de nouveaux nœuds. Soit $(p, R, \theta) \in \bar{\lambda}(g, \mathcal{T})$. Nous étudions les différents cas de la définition de $\bar{\lambda}(g, \mathcal{T})$ lorsque la racine du motif de $\mathcal{T}$ est étiquetée par *branch*.
Cas 1 : Supposons qu'il existe $i \in 1..k$ tel que g et $pattern(\mathcal{T}_i)$ soient unifiables. Alors $\bar{\lambda}(g, \mathcal{T}) \supseteq \bar{\lambda}(g, \mathcal{T}_i)$. De plus, le triplet (p, R, θ) que nous considérons est dans $\bar{\lambda}(g, \mathcal{T}_i)$ pour un certain $i \in 1..k$. Comme $(g, \mathcal{T}_i) \sqsubset (g, \mathcal{T})$, nous déduisons de l'hypothèse d'induction que $(p, R) \in \bar{\varphi}(\theta(g), \mathcal{T}_i)$. Comme $\bar{\varphi}(\theta(g), \mathcal{T}_i)$ est défini, $pattern(\mathcal{T}_i)$ filtre g à la racine, donc $\bar{\varphi}(\theta(g), \mathcal{T})$ est défini et $\bar{\varphi}(\theta(g), \mathcal{T}) = \bar{\varphi}(\theta(g), \mathcal{T}_i)$. Par conséquent, $(p, R) \in \bar{\varphi}(\theta(g), \mathcal{T})$.
Cas 2 : Supposons qu'il n'existe pas $i \in 1..k$ tel que g et $pattern(\mathcal{T}_i)$ soient unifiables. Soient τ un unificateur le plus général de g par rapport à π et h l'homomorphisme allant de π vers $\tau(g)$. Dans le cas que nous considérons ici, $h(o)$ est étiqueté par une opération définie f. Soit $\mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\}$ une forêt d'arbres de définition de f et soit $S = \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_1) \cup \ldots \cup \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_k)$.

Nous avons choisi $(p, R, \theta) \in \bar{\lambda}(g, \mathcal{T})$, donc par définition de $\bar{\lambda}$, il existe une substitution θ' et un entier $i \in 1..k$ tels que $(p, R, \theta') \in \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_i)$ et $\theta = \theta' \circ \tau$. Comme τ est un unificateur le plus général de g par rapport à π, τ est une substitution constructeur. Donc $\tau(g_{|h(o)})$ a moins d'opérations définies que g, i.e., $(\tau(g_{|h(o)}), \mathcal{T}'_i) \sqsubset (g, \mathcal{T})$. Aussi, par hypothèse d'induction, $\bar{\varphi}(\theta'(\tau(g_{|h(o)})), \mathcal{T}'_i)$ est défini et $(p, R) \in \bar{\varphi}(\theta'(\tau(g_{|h(o)})), \mathcal{T}'_i)$. Comme $\theta = \theta' \circ \tau$ et comme $h(o)$ est un nœud fonctionnel de g, donc de $\theta(g)$, on infère que $(p, R) \in \bar{\varphi}(\theta(g)_{|h(o)}, \mathcal{T}'_i)$.

Par hypothèse, $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$. De plus, il existe un homomorphisme h allant de π vers $\tau(g)$. Comme $\theta = \theta' \circ \tau$, il est clair qu'il existe un homomorphisme $h' : \pi \to \theta(g)$. Comme $h(o)$ est étiqueté par une opération définie f, $h'(o)$ est aussi étiqueté par f. Aussi $\bar{\varphi}(\theta(g), \mathcal{T})$ est défini et comme $\mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\}$ est une forêt d'arbres de définition de f, on déduit que $\bar{\varphi}(\theta(g), \mathcal{T}) = \bar{\varphi}(\theta(g)_{|h(o)}, \mathcal{T}'_1) \cup \ldots \cup \bar{\varphi}(\theta(g)_{|h(o)}, \mathcal{T}'_k)$ par définition de $\bar{\varphi}$. Nous avons vu que $(p, R) \in \bar{\varphi}(\theta(g)_{|h(o)}, \mathcal{T}'_i)$. Donc il est clair que $(p, R) \in \bar{\varphi}(\theta(g), \mathcal{T})$. □

Proposition 6.3.8 Soient SP un WAGRS, g un terme-graphe admissible et θ une substitution constructeur tels que $\bar{\Phi}(g) = \emptyset$ et $\bar{\Phi}(\theta(g)) \neq \emptyset$. Alors l'ensemble $\{\sigma \in \bar{\bar{\Lambda}}(g) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ est non vide. ◇

Preuve : Soit $(p, R) \in \bar{\Phi}(\theta(g))$. Nous déduisons du Lemme 6.1.11 qu'il existe une substitution σ telle que $(p, R, \sigma) \in \bar{\Lambda}(g)$ et $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. Par conséquent, l'ensemble $A = \{(p, R, \sigma) \in \bar{\Lambda}(g) \mid \sigma \dot{\leq} \theta\ [\mathcal{V}_g]\}$ n'est pas vide. Nous affirmons que $\sigma \not\doteq Id\ [\mathcal{V}_g]$ pour tout $(p, R, \sigma) \in \bar{\Lambda}(g)$. En effet, d'après la définition de $\bar{\Lambda}$, si n est le plus haut nœud fonctionnel à gauche de g, il existe un arbre de définition $\mathcal{T}$ dont le motif s'unifie avec $g_{|n}$ tel que $(p, R, \sigma) \in \bar{\lambda}(g_{|n}, \mathcal{T})$. De plus, d'après la Proposition 6.3.7, $(p, R) \in \bar{\varphi}(\sigma(g_{|n}), \mathcal{T})$, donc $\bar{\Phi}(\sigma(g)) \neq \emptyset$. Si $\sigma \doteq Id\ [\mathcal{V}_g]$, alors $\sigma(g)$ et g sont égaux au renommage de nœuds et des variables près. Comme $\bar{\Phi}(\sigma(g)) \neq \emptyset$, nous déduisons que $\bar{\Phi}(g) \neq \emptyset$, ce qui est impossible par hypothèse. Par conséquent, il n'existe pas de triplet $(p, R, \sigma) \in \bar{\Lambda}(g)$ tel que $\sigma \doteq Id\ [\mathcal{V}_g]$. Donc l'ensemble $A = \{(p, R, \sigma) \in \bar{\Lambda}(g) \mid \sigma \dot{\leq} \theta\ [\mathcal{V}_g]\}$ est égal à l'ensemble $B = \{(p, R, \sigma) \in \bar{\Lambda}(g) \mid \sigma \dot{\leq} \theta\ [\mathcal{V}_g] \text{ et } \sigma \not\doteq Id\ [\mathcal{V}_g]\}$. Comme $A \neq \emptyset$ et $A = B$, nous déduisons que $B \neq \emptyset$. Pour finir, $\bar{\bar{\Lambda}}(g)$ choisit un ensemble de substitutions minimales par rapport à $\dot{\leq}$ parmi celles intervenant dans les triplets de $\bar{\Lambda}(g)$. De plus, $\bar{\bar{\Lambda}}(g)$ restreint le domaine de ces substitutions à $\mathcal{V}_g$. Enfin, toutes les substitutions σ apparaissant dans les triplets de B satisfont $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. Comme l'ensemble $B = \{(p, R, \sigma) \in \bar{\Lambda}(g) \mid \sigma \dot{\leq} \theta\ [\mathcal{V}_g] \text{ et } \sigma \not\doteq Id\ [\mathcal{V}_g]\}$ est non vide, nous concluons que l'ensemble $C = \{\sigma \in \bar{\bar{\Lambda}}(g) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ est non vide. □

Proposition 6.3.9 Soient SP un WAGRS, g un terme-graphe admissible et θ une substitution constructeur tels que l'ensemble $A = \{\sigma \in \bar{\bar{\Lambda}}(g) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ soit vide. Alors $\bar{\Phi}(\theta(g)) \subseteq \bar{\Phi}(g)$. ◇

Preuve : Soit $(p, R) \in \bar{\Phi}(\theta(g))$. Nous affirmons que $(p, R) \in \bar{\Phi}(g)$. En effet, si $(p, R) \in \bar{\Phi}(\theta(g))$, alors il existe une substitution η telle que $(p, R, \eta) \in \bar{\Lambda}(g)$ et $\eta \dot{\leq} \theta\ [\mathcal{V}_g]$, d'après le Lemme 6.1.11. Il y a deux cas à étudier.

Cas 1 : Supposons qu'il existe $\sigma \in \bar{\bar{\Lambda}}(g)$ telle que $\eta \doteq \sigma\ [\mathcal{V}_g]$:
Comme $\sigma \doteq \eta\ [\mathcal{V}_g]$ et $\eta \dot{\leq} \theta\ [\mathcal{V}_g]$, nous déduisons que $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. Par hypothèse, l'ensemble $\{\sigma \in \bar{\bar{\Lambda}}(g) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ est vide, donc $\sigma \doteq Id$. Comme $\eta \doteq \sigma\ [\mathcal{V}_g]$ et $\sigma \doteq Id$, nous inférons que $\eta \doteq Id\ [\mathcal{V}_g]$.

Soit m le plus haut nœud fonctionnel à gauche dans $\theta(g)$ et soit $\mathcal{F} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ une forêt d'arbres de définition de l'étiquette de m. On a alors $\bar{\Phi}(\theta(g)) = Outer(\theta(g), S)$ avec $S = \bar{\varphi}(\theta(g)_{|m}, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(\theta(g)_{|m}, \mathcal{T}_k)$. Comme $(p, R) \in \bar{\Phi}(\theta(g))$, p est un plus nœud fonctionnel de $\theta(g)$ parmi ceux qui apparaissent dans les couples de S. θ est une substitution constructeur et m est le plus haut nœud fonctionnel à gauche dans $\theta(g)$, donc m est aussi le plus haut nœud fonctionnel à gauche dans g. De plus, on a $\bar{\Lambda}(g) = \bar{\lambda}(g_{|m}, \mathcal{T}_1) \cup \ldots \cup \bar{\lambda}(g_{|m}, \mathcal{T}_k)$. Comme $(p, R, \eta) \in \bar{\Lambda}(g)$, il existe $i \in 1..k$ tel que $(p, R, \eta) \in \bar{\lambda}(g_{|m}, \mathcal{T}_i)$. Aussi, $(p, R) \in \bar{\varphi}(\eta(g_{|m}), \mathcal{T}_i)$, d'après la Proposition 6.3.7. Nous avons vu que $\eta \doteq Id\ [\mathcal{V}_g]$. Donc $\eta(g)$ et g sont égaux au renommage des nœuds et des variables près. Comme $(p, R) \in \bar{\varphi}(\eta(g_{|m}), \mathcal{T}_i)$, nous déduisons que $(p, R) \in \bar{\varphi}(g_{|m}, \mathcal{T}_i)$.

Nous avons vu que p était un plus nœud fonctionnel de $\theta(g)$ parmi ceux qui apparaissent dans les couples de $S = \bar{\varphi}(\theta(g)_{|m}, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(\theta(g)_{|m}, \mathcal{T}_k)$. Nous affirmons que p est aussi un plus nœud fonctionnel de g parmi ceux qui apparaissent dans les couples de $T = \bar{\varphi}(g_{|m}, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(g_{|m}, \mathcal{T}_k)$, c'est-à-dire, $(p, R) \in \bar{\Phi}(g)$ (puisque $\bar{\Phi}(g) = Outer(g, T)$). En effet, dans le cas contraire, il existerait des couples $(q, S) \in T$ tels que q soit au dessus de p dans g. Mais comme ces nœuds q sont des nœuds fonctionnels et que θ est une substitution constructeur, ces nœuds q existeraient aussi au dessus de

p dans $\theta(g)$ et les couples (q, S) appartiendraient tous à S. Aussi, p ne pourrait pas être un plus nœud fonctionnel de $\theta(g)$ parmi ceux qui apparaissent dans les couples de S. Autrement dit, (p, R) ne pourrait pas être un couple de $\bar{\Phi}(\theta(g))$ ce qui est contraire à l'hypothèse. Par conséquent, $(p, R) \in \bar{\Phi}(g)$.
Cas 2 : Supposons que $\eta \not\doteq \sigma\ [\mathcal{V}_g]$ pour tout $\sigma \in \bar{\bar{\Lambda}}(g)$:
Dans ce cas, il existe une substitution $\sigma \in \bar{\bar{\Lambda}}(g)$ telle que $\sigma \not\doteq \eta\ [\mathcal{V}_g]$ et $\sigma \dot{\leq} \eta$, par définition de $\bar{\bar{\Lambda}}$. De plus, comme $\sigma \dot{\leq} \eta$ et $\eta \dot{\leq} \theta\ [\mathcal{V}_g]$ et l'ensemble $\{\sigma \in \bar{\bar{\Lambda}}(g) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ est vide, nous déduisons que $\sigma \doteq Id$. Par conséquent, la substitution $\eta_{|\mathcal{V}_g}$ n'apparaît pas dans $\bar{\bar{\Lambda}}(g)$ parce qu'elle a été effacée avec la seconde clause de la définition de $\bar{\bar{\Lambda}}$: tous les chemins allant de $\mathcal{R}oot_g$ vers p dans g passent par un certain nœud q tel qu'il existe $(q, S, \gamma) \in \bar{\Lambda}(g)$ avec $\gamma \doteq Id\ [\mathcal{V}_g]$. Comme θ est une substitution constructeur et p est un nœud fonctionnel, aucun nouveau chemin allant du nœud $\mathcal{R}oot_g$ vers le nœud p n'a été créé dans $\theta(g)$. Donc tous les chemins allant de $\mathcal{R}oot_{\theta(g)}$ vers p dans $\theta(g)$ passent par les nœuds q définis précédemment. Le calcul de $\bar{\Phi}(\theta(g))$ sélectionne des plus hauts nœuds fonctionnels. Comme tous les nœuds q sont au dessus de p dans $\theta(g)$, nous concluons que $(p, R) \notin \bar{\Phi}(\theta(g))$, ce qui est impossible. Par conséquent, le cas 2 n'est pas possible. □

Nous utilisons les trois propositions précédentes pour prouver le lemme clef suivant :

Lemme 6.3.10 Soient SP un WAGRS, $\blacktriangleright$ une stratégie de compression, g_0 un terme-graphe admissible, c un graphe constructeur et θ une substitution constructeur tels que $\theta \not\doteq Id\ [\mathcal{V}_g]$ et $\theta(g_0) \xrightarrow{*} c$. Alors il existe une dérivation $g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_n$ tels que $n \geq 0$ et soit (1) g_n est un graphe constructeur tel que $g_n \dot{\leq} c$, soit (2) l'ensemble $A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ est non vide. ◇

Preuve : Nous distinguons deux cas, celui où la dérivation $g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots$ est finie et celui où elle est infinie.
Cas 1 : La dérivation $g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots$ est finie :

Soit g_n le dernier terme-graphe de cette dérivation. Comme $g_0 \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_n$, nous déduisons du théorème de cohérence de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ (Théorème 6.3.5) qu'il existe un terme-graphe g'_n tel que $Id(g_0) = g_0 \xrightarrow{*} g'_n$ et $g'_n \doteq g_n$. Comme $g_0 \xrightarrow{*} g'_n$ et $g'_n \doteq g_n$, il est clair que $\theta(g_0) \xrightarrow{*} \theta(g'_n)$ et $\theta(g'_n) \doteq \theta(g_n)$. Par hypothèse, $\theta(g_0) \xrightarrow{*} c$ et c est un graphe constructeur ; de plus, $\theta(g_0) \xrightarrow{*} \theta(g'_n)$. Comme $\rightarrow$ est confluente au renommage des nœuds près (Théorème 3.3.4), nous déduisons qu'il existe un graphe constructeur c' tel que $\theta(g'_n) \xrightarrow{*} c'$ et $c' \sim c$. Comme $\theta(g'_n) \doteq \theta(g_n)$ et $\theta(g'_n) \xrightarrow{*} c'$ où c' est un graphe constructeur, nous déduisons du théorème de confluence de $\rightarrow$ modulo la bisimilarité (Théorème 3.3.7) qu'il existe un graphe constructeur d tel que $\theta(g_n) \xrightarrow{*} d$ et $d \doteq c'$. Il y a deux nouveaux cas à étudier.
Cas 1.1 : Si $\theta(g_n)$ est un graphe constructeur, alors $\theta(g_n) = d$. Comme $c \sim c'$ et $c' \doteq d$ et $d = \theta(g_n)$, nous déduisons que $\theta(g_n) \doteq c$, c'est-à-dire $g_n \dot{\leq} c$.
Cas 1.2 : Si $\theta(g_n)$ n'est pas un graphe constructeur, alors $\theta(g_n) \xrightarrow{+} d$. Comme $\bar{\Phi}$ est $\mathcal{C}$-normalisante (Théorème 4.3.19), on infère que $\bar{\Phi}(\theta(g_n)) \neq \emptyset$. De plus, g_n est le dernier terme-graphe de la dérivation $g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_n$, donc $Id \notin \bar{\bar{\Lambda}}(g_n)$ (sinon, on pourrait faire un nouveau pas $g_n \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_{n+1}$). Comme $Id \notin \bar{\bar{\Lambda}}(g_n)$, on ne peut pas *réécrire* g_n. Aussi, $\bar{\Phi}(g_n) = \emptyset$. Nous avons montré que $\bar{\Phi}(g_n) = \emptyset$ et $\bar{\Phi}(\theta(g_n)) \neq \emptyset$, donc d'après la Proposition 6.3.8, l'ensemble $A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ n'est pas vide.
Cas 2 : La dérivation $g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots$ est infinie :
Supposons qu'il n'existe pas de terme-graphe g_n tel que l'ensemble $A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ soit non vide. Alors, d'après la Proposition 6.3.9, $\bar{\Phi}(\theta(g_n)) \subseteq \bar{\Phi}(g_n)$ pour tout $n \geq 0$. Comme toutes les substitutions utilisées dans la dérivation $g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots$ sont l'identité, on peut voir cette dérivation comme une dérivation alternant des pas de compression et des pas de réécriture calculés par $\bar{\Phi}$: $g_0 \rhd g'_0 \Vdash\!\!\rightarrow_{\bar{\Phi}} g_1 \rhd g'_1 \Vdash\!\!\rightarrow_{\bar{\Phi}} g_2 \ldots$. Nous avons vu que $\bar{\Phi}(\theta(g_n)) \subseteq \bar{\Phi}(g_n)$ pour tout $n \geq 0$. Comme

$g_n \rhd g'_n$, nous inférons que $\bar{\Phi}(\theta(g'_n)) \subseteq \bar{\Phi}(g'_n)$ pour tout $n \geq 0$. En conséquence, il existe une dérivation infinie commençant avec $\theta(g_0)$ qui alterne des pas de compression et des pas de réécriture calculés par $\bar{\Phi}$ de la forme $\theta(g_0) \rhd \theta(g'_0) \leftmapsto_{\bar{\Phi}} \theta(g_1) \rhd \theta(g'_1) \leftmapsto_{\bar{\Phi}} \theta(g_2) \ldots$. Etant donnés trois termes-graphes admissibles u, v et v' tels que $u \rhd v$ et $v \leftmapsto_{\bar{\Phi}} v'$, il existe deux termes-graphes admissibles w et u' tels que $u \leftmapsto_{\bar{\Phi}} w \xrightarrow{*} u'$ et $u' \rhd v'$. Comme il existe une dérivation $\theta(g_0) \rhd \theta(g'_0) \leftmapsto_{\bar{\Phi}} \theta(g_1) \rhd \theta(g'_1) \leftmapsto_{\bar{\Phi}} \theta(g_2) \ldots$, nous déduisons qu'il existe une dérivation $\theta(g_0) \leftmapsto_{\bar{\Phi}} w_0 \xrightarrow{*} u_1 \leftmapsto_{\bar{\Phi}} w_1 \xrightarrow{*} u_2 \ldots$ (avec $u_1 \rhd \theta(g_1), u_2 \rhd \theta(g_2), \ldots$). De plus, la dérivation $\theta(g_0) \rhd \theta(g'_0) \leftmapsto_{\bar{\Phi}} \theta(g_1) \rhd \theta(g'_1) \leftmapsto_{\bar{\Phi}} \theta(g_2) \ldots$ est infinie, donc la dérivation $\theta(g_0) \leftmapsto_{\bar{\Phi}} w_0 \xrightarrow{*} u_1 \leftmapsto_{\bar{\Phi}} w_1 \xrightarrow{*} u_2 \ldots$ est infinie, elle aussi. Mais ceci n'est pas possible. En effet, $\theta(g_0) \xrightarrow{*} c$ et c est un graphe constructeur. Or $\bar{\Phi}$ est $\mathcal{C}$-hypernormalisante (d'après le Théorème 4.3.19), donc une dérivation de réécriture qui alterne des $\bar{\Phi}$-pas de réécriture avec des pas de réécriture quelconques termine nécessairement avec un graphe d tel que $d \doteq c$. La dérivation $\theta(g_0) \leftmapsto_{\bar{\Phi}} w_0 \xrightarrow{*} u_1 \leftmapsto_{\bar{\Phi}} w_1 \xrightarrow{*} u_2 \ldots$ est un exemple d'une telle dérivation de réécriture. Elle termine donc nécessairement avec un graphe d tel que $d \doteq c$. Pourtant nous avons montré qu'elle était infinie. Cette contradiction nous permet de refuter l'hypothèse : il existe nécessairement un terme-graphe admissible g_n tel que l'ensemble $A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \neq\!\!\!\doteq Id\}$ soit non vide. □

Nous aurons aussi besoin de la proposition suivante :

Proposition 6.3.11 Soient SP un WAGRS, $\blacktriangleright$ une stratégie de compression, g un terme-graphe admissible et d un graphe constructeur tels que $g \xrightarrow{*} d$. Il existe un graphe constructeur s tel que $g \blacktriangleright\!\!\Downarrow^{*}_{\bar{\bar{\Lambda}},\bar{\Phi},Id} s$ et $s \dot{\leq} d$. ◇

Preuve : Par induction sur la longueur n de la dérivation $g \xrightarrow{*} d$. Le cas de base $n = 0$ est évident. Soit $n > 0$. Supposons que la propriété soit vraie pour tout $n' < n$. Soit $g \xrightarrow{n} d$ une dérivation de longueur n. Soit g' *n'importe quel graphe* tel que $g \rhd g'$. Comme $g \rhd g'$ et $g \xrightarrow{n} d$ et d est un graphe constructeur, il existe un graphe constructeur d' et une dérivation $g' \xrightarrow{n'} d'$ tels que $d \rhd d'$ et $n' \leq n$, d'après la Proposition 5.4.1. Puisque $g' \xrightarrow{n'} d'$ et d' est un graphe constructeur, il existe un terme-graphe admissible g'' et un graphe constructeur d'' tels que $g' \leftmapsto_{\bar{\Phi}} g''$, $g'' \xrightarrow{n''} d''$, $n'' < n'$ et $d'' \sim d'$, d'après le Lemme 4.3.18.

Supposons que $\bar{\Phi}(g') = \{(p_1, R_1), \ldots, (p_k, R_k)\}$. D'après le Lemme 6.1.11, comme $(p_j, R_j) \in \bar{\Phi}(g')$, il existe une substitution σ_j telle que $(p_j, R_j, \sigma_j) \in \bar{\Lambda}(g')$ et $\sigma_j \dot{\leq} Id\ [\mathcal{V}_{g'}]$, i.e., $\sigma_j \doteq Id\ [\mathcal{V}_{g'}]$. Par définition de $\bar{\Phi}$, tous les nœuds p_j sont des plus hauts nœuds fonctionnels de g', donc $Id \in \bar{\bar{\Lambda}}(g')$. Aussi, le $\bar{\Phi}$-pas de réécriture $g' \leftmapsto_{\bar{\Phi}} g''$ peut être vu comme un $\bar{\bar{\Lambda}}$-pas de surréduction non compressante $g' \Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g''$. Il est clair que $Id(g) = g$. Ci-dessus, nous utilisons un graphe g' quelconque tel que $g \rhd g'$. Posons $g' = \blacktriangleright(Id(g))$. Dans ce cas, nous obtenons $g \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g''$.

Nous avons vu que $g'' \xrightarrow{n''} d''$ est une dérivation telle que $n'' < n'$ et d'' est un graphe constructeur. Comme $n'' < n'$ et $n' \leq n$, nous utilisons l'hypothèse d'induction : il existe un graphe constructeur s tel que $g'' \blacktriangleright\!\!\Downarrow^{*}_{\bar{\bar{\Lambda}},\bar{\Phi},Id} s$ et $s \dot{\leq} d''$. Comme $g \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g''$ et $g'' \blacktriangleright\!\!\Downarrow^{*}_{\bar{\bar{\Lambda}},\bar{\Phi},Id} s$, nous déduisons que $g \blacktriangleright\!\!\Downarrow^{*}_{\bar{\bar{\Lambda}},\bar{\Phi},Id} s$. Comme $s \dot{\leq} d''$ et $d'' \sim d'$ et $d \rhd d'$, nous concluons que $s \dot{\leq} d$. □

Nous avons explicité toutes les propositions et tous les lemmes qui sont utiles pour la preuve du théorème de complétude de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$. Cette preuve se fait par induction sur la taille des substitutions :

Définition 6.3.12 (Taille d'une substitution)
On note $\|g\|$ le nombre de nœuds non variables d'un terme-graphe admissible g (c'est-à-dire le nombre de nœuds constructeurs et de nœuds fonctionnels de g). On appelle *taille d'une*

substitution θ l'entier noté $size(\theta)$ tel que :

$$size(\theta) = \sum_{x \in \mathcal{D}\theta} \| \blacktriangleright\!\!\gg (\theta(x)) \|$$

$\diamond$

Nous sommes prêt pour faire la preuve du théorème de complétude de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$.

Preuve du Théorème 6.3.6 : On montre par induction sur la taille d'une substitution constructeur θ que pour tout terme-graphe admissible g et pour tout graphe constructeur c tels que $\theta(g) \xrightarrow{*} c$, il existe un graphe constructeur s et une substitution σ tels que $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$.
Cas de base : Supposons que $size(\theta) = 0$. Dans ce cas, θ est un simple renommage de variables. Donc si $\theta(g) \xrightarrow{*} c$, nous inférons qu'il existe un graphe constructeur d tel que $g \xrightarrow{*} d$ et $\theta(d) \sim c$. Aussi, d'après la Proposition 6.3.11, il existe un graphe constructeur s tel que $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},Id} s$ et $s \dot{\leq} d$. Comme $\theta(d) \sim c$, nous inférons que $d \dot{\leq} c$. Comme $s \dot{\leq} d$ et $d \dot{\leq} c$, nous déduisons que $s \dot{\leq} c$. Il existe donc bien un graphe constructeur s et une substitution $\sigma = Id$ tels que $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$.
Cas d'induction : Supposons que $size(\theta) > 0$ et que le théorème soit vrai pour toute substitution constructeur de taille strictement inférieure à celle de θ. Comme $size(\theta) > 0$, il est clair que $\theta \not\doteq Id$. Donc d'après le Lemme 6.3.10, il existe une dérivation $g = g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_n$ telle que $n \geq 0$ et soit (1) g_n est un graphe constructeur tel que $g_n \dot{\leq} c$, soit (2) l'ensemble $A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ est non vide.

Cas 1 : Il existe bien un graphe constructeur $s = g_n$ et une substitution $\sigma = Id$ tels que $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} s$, $s \dot{\leq} c$ et $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$.
Cas 2 : Soit $\sigma \in A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$ et g_{n+1} le terme-graphe admissible tels que $g_n \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_{n+1}$. On a donc $g = g_0 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} \cdots \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},Id} g_n \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_{n+1}$. Comme $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_{n+1}$, on déduit du théorème de cohérence de $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ (Théorème 6.3.5) qu'il existe un terme-graphe g'_{n+1} tel que $\sigma(g) \xrightarrow{*} g'_{n+1}$ et $g'_{n+1} \doteq g_{n+1}$. Comme $\sigma \in A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$, nous avons $\sigma \dot{\leq} \theta$. Donc il existe une substitution constructeur θ' telle que $\theta' \circ \sigma \doteq \theta\ [\mathcal{V}_g]$. Comme $\sigma(g) \xrightarrow{*} g'_{n+1}$, nous déduisons que $\theta'(\sigma(g)) \xrightarrow{*} \theta'(g'_{n+1})$. Comme $\theta' \circ \sigma \doteq \theta\ [\mathcal{V}_g]$, il est clair que $\theta'(\sigma(g)) \doteq \theta(g)$. De plus, $\theta'(\sigma(g)) \xrightarrow{*} \theta'(g'_{n+1})$ et $\theta(g) \xrightarrow{*} c$ où c est un graphe constructeur. Comme $\rightarrow$ est confluente modulo la bisimilarité (Théorème 3.3.7), nous déduisons qu'il existe un graphe constructeur u tel que $\theta'(g'_{n+1}) \xrightarrow{*} u$ et $u \doteq c$. Nous avons vu que $g'_{n+1} \doteq g_{n+1}$, donc $\theta'(g'_{n+1}) \doteq \theta'(g_{n+1})$. De plus, nous avons montré qu'il existait un graphe constructeur u tel que $\theta'(g'_{n+1}) \xrightarrow{*} u$. Comme $\rightarrow$ est confluente modulo la bisimilarité (Théorème 3.3.7), nous déduisons qu'il existe un graphe constructeur v tel que $\theta'(g_{n+1}) \xrightarrow{*} v$ et $v \doteq u$. En résumé, nous avons construit une dérivation $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_{n+1}$ et nous avons montré qu'il existait une substitution constructeur θ' et un graphe constructeur v tels que $\theta'(g_{n+1}) \xrightarrow{*} v$. Par définition de θ', nous avons $\theta' \circ \sigma \doteq \theta\ [\mathcal{V}_g]$. De plus, comme $\sigma \in A = \{\sigma \in \bar{\bar{\Lambda}}(g_n) \mid \sigma \dot{\leq} \theta \text{ et } \sigma \not\doteq Id\}$, $\sigma \not\doteq Id$, donc $size(\theta') < size(\theta)$. Aussi, par hypothèse d'induction, nous déduisons qu'il existe un graphe constructeur s et une substitution σ' tels que $g_{n+1} \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma'} s$, $s \dot{\leq} v$ et $\sigma' \dot{\leq} \theta'$. Nous avons $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_{n+1}$ et $g_{n+1} \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma'} s$, donc $g \overset{*}{\blacktriangleright\!\!\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma'\circ\sigma} g_{n+1}$. De plus, $s \dot{\leq} v$, $v \doteq u$ et $u \doteq c$, donc $s \dot{\leq} c$. Enfin, nous devons montré que $\sigma' \circ \sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. Nous avons vu que $\theta' \circ \sigma \doteq \theta\ [\mathcal{V}_g]$. De plus, comme $\sigma' \dot{\leq} \theta'$, il existe une substition θ'' telle que $\theta'' \circ \sigma' \doteq \theta'\ [\mathcal{V}_{g_{n+1}}]$. Par conséquent, $\theta'' \circ \sigma' \circ \sigma \doteq \theta' \circ \sigma \doteq \theta\ [\mathcal{V}_g]$. Aussi, nous concluons que $\sigma' \circ \sigma \dot{\leq} \theta\ [\mathcal{V}_g]$. $\square$

6.4 Propriétés d'optimalité de $\blacktriangleright\!\rightsquigarrow\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$

Nous étudions maintenant quelques propriétés d'optimalité de $\blacktriangleright\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$. Il va de soit que la relation de surréduction parallèle compressante hérite de toutes les propriétés d'optimalité de la surréduction parallèle de termes [AEH97a] :

- $\blacktriangleright\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ ne calcule que des pas nécessaires de surréduction compressante dans le cas des ISGRS (puisque pour ces systèmes de réécriture, $\blacktriangleright\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}} = \blacktriangleright\!\rightsquigarrow_{\bar{\Lambda}}$).
- $\blacktriangleright\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ normalise les termes-graphes sans variable en des graphes constructeurs de manière déterministe.

En plus des propriétés ci-dessus, les structures de graphes induisent de nouvelles améliorations pour la surréduction. En fait, l'implantation de $\bar{\lambda}$ est plus efficace que celle correspondant aux termes. En effet, grâce au partage des sous-expressions dans les structures de graphes, $\bar{\lambda}$ permet d'éviter les calculs redondants qui se font quand $\bar{\lambda}$ doit revisiter plusieurs fois le même sous-graphe partagé. Ce genre d'amélioration n'est pas possible sur des structures d'arbres qui permettent d'implanter les termes.

Par contre, les substitutions calculées par $\blacktriangleright\!\overset{*}{\Downarrow}_{\bar{\bar{\Lambda}},\bar{\Phi}}$ ne sont pas disjointes [AEH97a, Example 8]. De même, les dérivations de surréduction engendrées par $\blacktriangleright\!\gg\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ ne sont pas de longueurs plus petites que toute autre dérivation de surréduction, tout comme celles calculées par $\blacktriangleright\!\overset{*}{\gg}\!\rightsquigarrow_{\Lambda}$ dans le cas des ISGRS (cf. Exemple 6.2.4).

Chapitre 7

Conclusion

Bilan

Les langages logico-fonctionnels sont des langages de programmation de très haut niveau permettant de définir dans un formalisme unifié des types de données, des fonctions et des prédicats (relations). Ces langages offrent aux utilisateurs tous les avantages de la programmation fonctionnelle (réduction efficace d'expressions) et de la programmation logique (variables logiques, définition de relations, résolution de buts). En outre, les langages logico-fonctionnels jouissent des avantages qui découlent de l'intégration de la programmation logique et fonctionnelle comme, par exemple, les définitions mutuellement récursives de fonctions et de prédicats.

Plusieurs langages logico-fonctionnels ont été proposés durant les dix dernières années. Cependant, tous ces langages modélisent les structures de données sous la forme de *termes du premier ordre* (arbres). Cette restriction importante permet de manipuler les types abstraits algébriques mais elle rend difficile l'utilisation de certaines structures de données représentées sous la forme de graphes cycliques. Rappelons l'importance de ces structures dans de nombreux domaines, comme les bases de données, où elles permettent de modéliser simplement les relations sémantiques qui existent entre différents objets (par exemple, les *humains*, comme la famille de Adam, traitée en exemple dans l'introduction, les *villes* ...).

L'objectif de cette thèse était d'introduire les graphes cycliques comme structure de données de base dans les langages logico-fonctionnels. Les retombées de cette étude sont multiples. D'une part, la syntaxe du langage considéré est plus expressive que celles des langages logico-fonctionnels actuels. D'autre part, l'utilisation de graphes plutôt que de termes permet d'améliorer les performances des sémantiques opérationnelles classiques de ces langages. Enfin, compte tenu des performances des implantations de langages fonctionnels à base de graphes, comme CLEAN [PvE93], nos travaux contribueront à l'amélioration de l'efficacité des implantations de langages logico-fontionnels.

Pour introduire les graphes dans les langages logico-fonctionnels, nous avons modélisé les programmes sous la forme de systèmes de réécriture de graphes cycliques avec constructeurs (cGRS) et nous avons étudié les relations de réécriture et de surréduction qu'ils induisent. Ces relations n'ont pas les mêmes propriétés sur les graphes cycliques que sur les termes du premier ordre. Par exemple, les systèmes de réécriture orthogonaux de graphes ne sont pas confluents alors que les systèmes de réécriture orthogonaux de termes le sont.

Notre principale contribution est la mise en évidence d'une classe de graphes cycliques particuliers que nous appelons les graphes *admissibles*. Ces graphes cycliques sont ceux qu'on

utilise en programmation pour modéliser les données du monde réél. De plus, ils permettent de retrouver les "bonnes" propriétés de la relation de réécriture et de surréduction engendrée par les systèmes de réécriture usuels de termes.

Ainsi, nous avons montré que la relation de réécriture engendrée par les systèmes de réécriture *faiblement admissibles* de graphes (WAGRS) était confluente et confluente modulo la bisimilarité par rapport aux graphes admissibles. De plus, nous avons montré que la relation de réécriture engendrée par les systèmes *faiblement admissibles avec bisimilarité* était confluente modulo la bisimilarité par rapport aux graphes admissibles lorsque cette relation est nœthérienne. Nous conjecturons qu'elle le reste lorsque cette relation n'est pas nœthérienne.

Ces propriétés de confluence permettent d'une part, de faire des calculs déterministes, et d'autre part, de faire ces calculs de manière efficace. Pour cela, nous avons étendu deux stratégies de réécriture de termes, définies dans [Ant92], aux graphes admissibles et nous avons montré que notre extension préservait leurs propriétés de normalisation et d'optimalité. L'étude de ces stratégies nous a conduit à définir une nouvelle relation de réécriture de graphes, la réécriture *parallèle*. Enfin, comme conséquence de notre étude, nous retrouvons un résultat connu pour les termes du premier ordre [O'D77], à savoir la $\mathcal{C}$-normalisation de la stratégie consistant en la réécriture parallèle de tous les plus hauts rédex d'un graphe admissible (*parallel outermost strategy* en anglais).

De même que pour la réécriture, les graphes admissibles nous ont permis de définir une relation de surréduction (compressante) qui est cohérente et complète par rapport à la réécriture dans le cadre des WAGRS. Cette relation de surréduction est un mécanisme de base pour résoudre des buts avec un programme logico-fonctionnel. Pour améliorer l'efficacité de ce calcul, nous avons étendu trois stratégies optimales de surréduction de termes aux graphes admissibles : la surréduction nécessaire [AEH94a], la surréduction faiblement nécessaire [AEH97a] et la surréduction parallèle [AEH97a]. Nous avons montré que les relations de surréduction compressante qu'elles induisent étaient cohérentes et complètes dans le cadre des WAGRS. Puis nous avons énoncé plusieurs propriétés d'optimalité les concernant.

L'ensemble de ces résultats tend à prouver que les graphes admissibles sont de très bons candidats pour représenter les types de données du monde réel dans les langages logico-fonctionnels. Ces résultats ont donné lieu à plusieurs communications en France [EJ99c, EJ97c] et à l'étranger [EJ98a, EJ99b, EJ98b].

Travaux futurs

Plusieurs voies sont possibles pour poursuivre ce travail. Tout d'abord, les langages logico-fonctionnels actuels ne permettent pas de manipuler directement les pointeurs. Cette restriction n'a pas d'incidence dans les langages à base de termes, mais elle rend difficile l'évolution d'une structure de données représentée par un graphe cyclique lors de l'exécution d'un programme (par exemple, l'insertion d'un élément dans une liste circulaire). Nous souhaitons donc étendre les systèmes de réécriture de graphes classiques en permettant à l'utilisateur de faire des redirections explicites de pointeurs.

Ensuite, les programmes dans les langages logico-fonctionnels nécessitent souvent d'utiliser des règles de réécriture *conditionnelles* de graphes, c'est-à-dire des règles avec des conditions d'application qui en restreignent l'utilisation. La réécriture conditionnelle de termes a fait l'objet de plusieurs études dont les principaux résultats sont regroupés dans [Klo92, Chapter 11]. La surréduction conditionnelle, elle, a fait l'objet d'une étude "systématique" dans [MH92].

Peu de résultats sont établis sur la réécriture et la surréduction conditionnelle des graphes : la seule référence que nous ayions trouvée dans la littérature est [Ohl97]. Pourtant, les perpectives de ce champ d'investigation sont assez vastes.

Enfin, nous avons commencé un travail de programmation d'un interpréteur d'un langage logico-fonctionnel à base de graphes. Ce travail vise à montrer l'implémentabilité de nos stratégies de réécriture et de surréduction et donc à valider l'ensemble de nos résultats en pratique.

Bibliographie

[AEH94a] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Proc. of the 21st ACM Symposium on Principles of Programming Languages (POPL'94)*, pages 268–279, Portland, 1994.

[AEH94b] S. Antoy, R. Echahed, and M. Hanus. An optimal narrowing strategy. Internal report (draft), IMAG, June 1994. Long version of [AEH94a]. To appear in the Journal of the ACM.

[AEH97a] S. Antoy, R. Echahed, and M. Hanus. Parallel evaluation strategies for functional logic languages. In *Proc. of the International Conference on Logic Programming (ICLP'97)*, pages 138–152, Portland, 1997. MIT Press.

[AEH97b] S. Antoy, R. Echahed, and M. Hanus. Parallel evaluation strategies for functional logic languages. Internal report (draft), IMAG, September 1997. Long version of [AEH97a].

[AK96] Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3-4), 1996.

[AKP97] Z. M. Ariola, J. W. Klop, and D. Plump. Confluent rewriting of bisimilar term graphs. *Electronic Notes in Theoretical Computer Science*, 7, 1997. Available at URL : `http://www.elsevier.nl/inca/publications/store/5/0/5/6/2/5/`.

[AL88] H. Aït-Kaci and P. Lincoln. LIFE : a natural language for natural languages. MCC technical report ACA-ST-074-88, DEC Paris Research Laboratory, 1988.

[ALN87] H. Aït-Kaci, P. Lincoln, and R. Nasr. LEFUN : logic, equations and functions. In *Proc. of the 4th IEEE International Symposium on Logic Programming (ISLP'87)*, pages 17–23, San Francisco, 1987.

[AN86] H. Aït-Kaci and R. Nasr. LOGIN : a logic programming language with built-in inheritance. *Journal of Logic Programming*, 3 :185–215, 1986.

[AN89] H. Aït-Kaci and R. Nasr. Integrating logic and functional programming. *Lisp and Symbolic Computation*, 2 :51–89, 1989.

[Ant92] S. Antoy. Definitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming (ALP'92)*, pages 143–157. LNCS 632, September 1992.

[AP92] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. Technical report PRL-11, DEC Paris Research Laboratory, 1992. Available at URL : `http://www.isg.sfu.ca/life/`.

[Bar84] H. P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*. North-Holland, 1984.

[BE86] D. Bert and R. Echahed. Design and implementation of a generic, logic and functional programming language. In *Proc. of the European Symposium on Programming (ESOP'86)*, pages 119–132. LNCS 213, 1986.

[BE95] D. Bert and R. Echahed. On the operational semantics of the algebraic and logic language LPG. In *Data Types Specification*, pages 132–152. LNCS 906, 1995.

[BER94] D. Bert, R. Echahed, and J. C. Reynaud. Reference manual of the LPG specification language and environment. Technical report SCOP-LSR, IMAG, 1994.

[BKW92] A. Bockmayr, S. Krischer, and A. Werner. An optimal narrowing strategy for general canonical systems. In *Proc. of the 3rd International Workshop on Conditional Term Rewriting Systems*, pages 483–497. LNCS 656, 1992.

[BMST99] R. Bardohl, M. Minas, A. Schürr, and G. Taentzer. Application of graph transformation to visual languages. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformations : Applications, Langages and Tools*. World Scientific, 1999. To appear.

[BS94] F. Baader and J. H. Siekmann. Unification theory. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, 1994.

[BvEG+87] H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. LNCS 259, 1987.

[CER79] V. Claus, H. Ehrig, and G. Rozenberg, editors. *Proc. of the 1st International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*. LNCS 73, 1979.

[CKvC83] A. Colmerauer, H. Kanoui, and M. van Caneghem. Prolog, bases théoriques et développement actuels. *TSI*, 2.4 :271–312, 1983.

[Col82] A. Colmerauer. Prolog and infinite trees. *Logic Programming*, 1982.

[Col90] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7) :69–90, July 1990.

[Cor93] A. Corradini. Term rewriting in CT_Σ. In *Proc. of the 4th International Joint Conference CAAP-FASE (TAPSOFT'93)*, pages 468–484. LNCS 668, April 1993.

[Cor96] A. Corradini. Concurrent graph and term graph rewriting. In *CONCUR'96 : Concurrency Theory*, pages 438–464. LNCS 1119, August 1996.

[Cou83] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25 :95–169, 1983.

[CR36] A. Church and J. B. Rosser. Some properties of conversion. *Transactions of AMS*, 39 :472–482, 1936.

[Dau89] M. Dauchet. Simulation of turing machines by a left-linear rewrite rule. In *Proc. of the 3rd International Conference on Rewriting Techniques and Applications (RTA'89)*, pages 109–120. LNCS 355, 1989.

[DG89] J. Darlington and Y. Guo. Narrowing and unification in functional programming - an evaluation mechanism for absolute set abstraction. In *Proc. of the 3rd International Conference on Rewriting Techniques and Applications (RTA'89)*, pages 92–108. LNCS 355, 1989.

[dGL86] D. de Groot and G. Lindstrom, editors. *Logic Programming, Functions, Relations, and Equations*. Prentice Hall, 1986.

[DJ90] N. Dershowitz and J. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Formal Methods and Semantics*, volume B, chapter 6, pages 243–320. North Holland, Amsterdam, 1990.

[Dör95] H. Dörr. *Efficient Graph Rewriting and Its Implementation*. Springer Verlag, LNCS 922, 1995.

[Ech88] R. Echahed. On completeness of narrowing strategies. In *Proc. CAAP'88*, pages 89–101. LNCS 299, 1988.

[Ech90] R. Echahed. Sur l'intégration des langages algébriques et logiques. Thèse de l'Institut National Polytechnique de Grenoble, 1990.

[Ech92] R. Echahed. Uniform narrowing strategies. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming (ALP'92)*, pages 259–275, Volterra, Italy, September 1992.

[EEKR99] H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations : Applications, Langages and Tools*, volume 2. World Scientific, 1999. To appear.

[EJ97a] R. Echahed and J. C. Janodet. Introducing graphs in functional logic programming languages. Internal report, IMAG, 1997. Long version of [EJ98a, EJ98b].

[EJ97b] R. Echahed and J. C. Janodet. On constructor-based graph rewriting systems. Technical report 985-1, IMAG, 1997. Long version of [EJ97c]. Available at URL : `ftp://ftp.imag.fr/pub/LEIBNIZ/PMP/c-graph-rewriting.ps.gz`.

[EJ97c] R. Echahed and J. C. Janodet. Réécriture de graphes admissibles : Confluence et stratégie. In *Actes des Journées du* GDR *Programmation*, pages 77–89, Rennes, November 1997.

[EJ98a] R. Echahed and J. C. Janodet. Admissible graph rewriting and narrowing. In *Proc. of Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, pages 325–340. MIT Press, June 1998.

[EJ98b] R. Echahed and J. C. Janodet. On admissible graph rewriting and narrowing. In *Proc. of 7th International Workshop on Functional and Logic Programming (WFLP'98)*, April 1998.

[EJ98c] R. Echahed and J. C. Janodet. On weakly orthogonal constructor-based graph rewriting. Internal report, IMAG, 1998. Long version of [EJ99b]. Available at URL : `ftp://ftp.imag.fr/pub/LEIBNIZ/PMP/wa-c-graph-rewriting.ps.gz`.

[EJ99a] R. Echahed and J. C. Janodet. Collapsing graph narrowing. Internal report, IMAG, May 1999. Available at URL : `ftp://ftp.imag.fr/pub/LEIBNIZ/PMP/wa-collapsing-narrowing.ps.gz`.

[EJ99b] R. Echahed and J. C. Janodet. Parallel admissible graph rewriting. In *Recent Developments in Algebraic Development Techniques*, pages 121–135. LNCS 1589, 1999.

[EJ99c] R. Echahed and J. C. Janodet. Parallel graph narrowing. In *Proc. of 8th International Workshop on Functional and Logic Programming (WFLP'99)*, pages 269–281, June 1999.

[EKL90] H. Ehrig, M. Korff, and M. Löwe. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In *Proc. of the 4th International Workshop on Graph Grammars and their Application to Computer Science*, pages 24–37. LNCS 532, 1990.

[EKR91] H. Ehrig, H. J. Kreowsky, and G. Rozenberg, editors. *Proc. of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*. LNCS 532, 1991.

[ELN+92] G. Engels, C. Lewerentz, M. Nagl, W. Schäfer, and A. Schürr. Building integrated software development environments Part 1 : Tool specification. *ACM Transactions on Software Engineering and Methodology*, 1(2) :135–167, April 1992.

[EM85] H. Ehrig and B. Mahr. Fundamentals of algebraic specification 1 : equations and initial semantics. In *Monograph in Theoretical Computer Science. An EATCS Series*, volume 6. Springer, 1985.

[ENR83] H. Ehrig, M. Nagl, and G. Rozenberg, editors. *Proc. of the 2nd International Workshop on Graph-Grammars and Their Application to Computer Science*. LNCS 153, 1983.

[ENRR87] H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors. *Proc. of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*. LNCS 291, 1987.

[ER94] G. Engels and G. Rozenberg, editors. *Proc. of the 5th International Workshop on Graph-Grammars and Their Application to Computer Science*. LNCS 1073, 1994.

[ET96] H. Ehrig and G. Taentzer. Computing by graph transformation : a survey and annotated bibliography. *Bulletin of the EATCS*, 59 :182–226, June 1996.

[Fay79] M. J. Fay. First-order unification in an equational theory. In *Proc. of the 4th International Workshop on Automated Deduction*, pages 161–167, Austin, Texas, 1979. Academic Press.

[Fri85] L. Fribourg. SLOG : a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. of the IEEE International Symposium on Logic Programming*, pages 172–184, Boston, 1985.

[FRW89] W. M. Farmer, J. D. Ramsdell, and R. J. Watro. Redex capturing in term graph rewriting. Technical report M89-36, The MITRE Corporation, 1989.

[FW99] M. Fröhlich and M. Werner. daVinci Homepage. Available at URL : `http://www.informatik.uni-bremen.de/~davinci/`, 1999.

[GKK+87] J. A. Goguen, C. Kirchner, H. Kirchner, A. Mégrelis, J. Meseguer, and T. Winkler. An introduction to OBJ 3. In *LNCS 308*, pages 258–263. Springer Verlag, 1987.

[GLMP91] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel LEAF : a logic plus functional language. *The Journal of Computer and System Sciences*, 42 :139–185, 1991.

[GM86] J. A. Goguen and J. Meseguer. EQLOG : equality, types, and generic modules for logic programming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming, Functions, Relations, and Equations*, pages 295–363. Prentice Hall, 1986.

[HAK+99] M. Hanus, S. Antoy, H. Kuchen, F. J. López-Fraguas, J. J. Moreno-Navarro, and F. Steiner. Curry : An integrated functional logic language. Available at `http://www-i2.informatik.rwth-aachen.de/~hanus/curry/report.html`, January 13 1999. Version 0.5.

[Han90] M. Hanus. Compiling logic programs with equality. In *Proc. of the 2nd International Workshop on Programming Language Implementation and Logic Programming*, pages 387–401. LNCS 456, 1990.

[Han94] M. Hanus. The integration of functions into logic programming : from theory to practice. *Journal of Logic Programming*, 19&20 :583–628, 1994.

[HdBA93] M. Holsheimer, R. A. de By, and H. Aït-Kaci. A database interface for complex objects. Technical report PRL-27, DEC Paris Research Laboratory, 1993. Available at URL : `http://www.isg.sfu.ca/life/`.

[HH82] G. Huet and J. M. Hullot. Proofs by induction in equational theories with constructors. *JCSS*, 25 :239–266, 1982.

[HL91] G. Huet and J. J. Lévy. Computations in orthogonal term rewriting systems. In J. L. Lassez and G. Plotkin, editors, *Computational logic : Essays in honour of Alan Robinson*. MIT Press, Cambridge, MA, 1991.

[HO80] G. Huet and D. Oppen. Equations and rewrite rules - a survey. *Formal language Theory, perspective and open problems*, pages 348–405, 1980.

[HP94] A. Habel and D. Plump. Graph unification and matching. In *Proc. of the 5th International Workshop on Graph Grammars and their Application to Computer Science*, pages 387–401. LNCS 1073, 1994.

[HP96] A. Habel and D. Plump. Term graph narrowing. *Mathematical Structures in Computer Science*, 6 :649–676, 1996.

[HP99] A. Habel and D. Plump. Complete strategies for term graph narrowing. In *Recent Developments in Algebraic Development Techniques*. LNCS 1589, 1999.

[Hue76] G. Huet. Résolution d'équations dans les langages d'ordre 1, 2, ..., ω. Thèse de l'Université de Paris VII, 1976.

[Hue80] G. Huet. Confluent reductions : abstract properties and application to term rewriting systems. *Journal of the ACM*, 27(4) :797–821, 1980.

[Hul80a] J. M. Hullot. Canonical forms and unification. In *Proc. of the 5th Conference on Automated Deduction (CADE'87)*, pages 318–334. LNCS 87, 1980.

[Hul80b] J. M. Hullot. Compilation de formes canoniques dans les théorie équationnelles. Thèse de l'Université de Paris-Sud Centre d'Orsay, 1980.

[JK91] J. P. Jouannaud and C. Kirchner. Solving equations in abstract algebras : a rule-based survey of unification. In J. L. Lassez and G. Plotkin, editors, *Computational Logic : Essays in Honor of Alan Robinson*. MIT Press, 1991.

[Kah94] W. Kahl. Can functional programming be liberated from the applicative style ? In B. Pehrson and I. Simon, editors, *Proc. of the IFIP 13th World Computer Congress - Volume I*, volume A-51, pages 330–335. IFIP Transactions, 1994. Available at URL : `http://diogenes.informatik.unibw-muenchen.de:8080/kahl/HOPS/`.

[KB70] D. E. Knuth and P. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KB91] S. Krischer and A. Bockmayr. Detecting redundant narrowing derivations by the LSE-SL reducibility test. In *Proc. of the International Conference on Rewriting Techniques and Applications (RTA'91)*. LNCS 488, 1991.

[Ken95] J. R. Kennaway. Infinitary rewriting and cyclic graphs. *Electronic notes in Theoretical Computer Science 2*, 1995. Available at URL : `http://www.elsevier.nl/locate/tcs`.

[KKSV94] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Transactions on Programming Languages and Systems*, 16(3) :493–523, 1994.

[KKSV95] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. Transfinite reduction in orthogonal term rewriting systems. *Information and Computation*, 119(1) :18–38, 1995.

[Klo92] J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–112. Oxford University Press, 1992.

[Kni89] K. Knight. Unification : a multidisciplinary survey. *ACM Computing Surveys*, 21(1) :93–124, 1989.

[KP98] H. J. Kreowski and D. Plump. APPLIGRAPH : First annual progress report. Technical report, available at URL : `http://www.informatik.uni-bremen.de/grp/ag-ti/appligraph/`, 1998.

[KP99] H. J. Kreowski and D. Plump. APPLIGRAPH : Second annual progress report. Technical report, available at URL : `http://www.informatik.uni-bremen.de/grp/ag-ti/appligraph/`, 1999.

[Kri96] M. R. K. Krishna Rao. Completeness results for basic narrowing in non-copying implementations. In *Proc. of the Joint International Conference and Symposium on Logic Programming (JICSLP'96)*, pages 393–407. MIT press, 1996.

[LK99] W. Lux and H. Kuchen. An efficient abstract machine for Curry. In *8th International Workshop on Functional and Logic Programming*, pages 171–181. Laboratoire LEIBNIZ - Institut IMAG, 1999.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, Berlin, 1987. 1st edition : 1984.

[Llo94] J. W. Lloyd. Combining functional and logic programming languages. In *Proc. of the International Logic Programming Symposium (ILPS'94)*, pages 43–57, 1994.

[MH92] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing (extended abstract). In *Proc. of the 3rd International Conference on Algebraic and Logic Programming (ALP'92)*, pages 244–258, Volterra, Italy, September 1992. Available at URL : `http://SunSITE.Informatik.RWTH-Aachen.de/dblp/db/indices/a-tree/m/Middeldorp:Aart.html`.

[Mid90] A. Middeldorp. Modular properties of term rewriting systems. PhD Thesis. Centrale Huisdrukkerij Vrije Universiteit, 1990.

[MKLR90] J. J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodríguez-Artalejo. Lazy narrowing in a graph machine. In *Proc. of the 2nd International Conference on Algebraic and Logic Programming (ALP'90)*, pages 298–317. LNCS 463, 1990.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. on Programming Languages and Systems*, 4(2) :258–282, April 1982.

[MR92] J. J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic programming with functions and predicates : the language BABEL. *Journal of Logic Programming*, 12 :191–223, 1992.

[Nag96] M. Nagl, editor. *Building Tightly Integrated Software Development Environments : the IPSEN Approach*. LNCS 1170, 1996.

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2) :223–243, 1942.

[O'D77] M. J. O'Donnell. *Computing in Systems Described by Equations*. LNCS 58, 1977.

[Ohl97] E. Ohlebusch. Conditional term graph rewriting. In *Proc. of the 6th International Joint Conference ALP'97-HOA'97*, pages 144–158, Southampton, 1997. LNCS 1298.

[Pey87] S. L. Peyton Jones. *The Implementation of Functional Programming Langages*. Prentice-Hall, 1987.

[Plu94] D. Plump. Critical pairs in term graph rewriting. In *Proc. of Mathematical Foundations of Computer Science (MFCS'94)*, pages 556–566. LNCS 841, 1994.

[Plu97] D. Plump. Simplification orders for term graph rewriting. In *Proc. of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS'97)*, pages 458–467. LNCS 1295, August 1997.

[Plu98a] D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2. World Scientific, 1998.

[Plu98b] D. Plump. Terminaison of graph rewriting is undecidable. *Fundamenta Informaticae*, 33(2) :201–209, February 1998.

[PRO99] PROGRESS Group. PROGRESS Homepage. Available at URL : `http://www-i3.informatik.rwth-aachen.de/research/progres/`, 1999.

[PvE93] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.

[Red85] U. S. Reddy. Narrowing as the operational semantics of functional languages. In *Proc. of the IEEE International Symposium on Logic Programming*, pages 138–151, Boston, 1985.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12 :23–41, 1965.

[Roz97] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations : Foundations*, volume 1. World Scientific, 1997.

[RS82] J. A. Robinson and E. E. Sibert. LOGLISP : motivation, design and implementation. In K. L. Clark and S. A. Tarnlund, editors, *Logic Programming*, pages 299–314. Academic Press, 1982.

[Sla74] J. R. Slagle. Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *Journal of the ACM*, 21(4) :622–642, 1974.

[Smo95] G. Smolka. An Oz primer. Technical report, DFKI, 1995. Available at URL : `http://www.dfki.uni-sb.de/pas/f2w.cgi?pss/oz-cpe-e`.

[SPvE93] M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, editors. *Term Graph Rewriting. Theory and Practice*. J. Wiley & Sons, Chichester, UK, 1993.

[SR93] R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic : beyond strong sequentiality. *Information and Computation*, 104(1) :78–109, May 1993.

[SS86] L. Sterling and E. Shapiro, editors. *The Art of Prolog*. MIT Press, 1986.

[Yam93] H. Yamanaka. Graph narrowing and its simulation by graph reduction. Research report IIAS-RR-93-10E, Institute for Social Information Science, Fujitsu Laboratories LDT, June 1993.

[You89] J. H. You. Enumerating outer narrowing derivations for constructor-based term rewriting systems. *Journal of Symbolic Computation*, 7 :319–341, 1989.

Sur les types de données dans les langages logico-fonctionnels : Réécriture et surréduction des graphes admissibles.

Les langages logico-fonctionnels sont des langages de programmation de très haut niveau permettant de définir dans un formalisme unifié des types de données, des fonctions et des prédicats (relations). Plusieurs propositions de langages logico-fonctionnels ont été faites mais toutes se restreignent à des calculs basés sur les *termes du premier ordre*. Cette restriction permet de programmer avec des types abstraits algébriques mais elle rend difficile la manipulation des structures de données du monde réel, modélisées sous la forme de *graphes cycliques*. L'objectif de cette thèse est donc d'introduire les graphes cycliques comme structure de données de base des langages logico-fonctionnels. Pour cela, nous voyons les programmes comme des systèmes de réécriture de graphes cycliques et nous étudions les relations de réécriture et de surréduction qu'ils induisent (sémantique opérationnelle).

Une propriété importante de la réécriture concerne la confluence : elle exprime le déterminisme des calculs effectués. De nombreux résultats de confluence existent pour la réécriture de termes mais ils ne s'étendent généralement pas aux graphes cycliques. Nous mettons en évidence une classe de graphes cycliques particuliers, les graphes admissibles, pour laquelle nous donnons une preuve de confluence de la réécriture. Concernant la relation de surréduction, nous en proposons une définition puis nous montrons que ce calcul est cohérent et complet par rapport à celui de la réécriture dans le cadre des graphes admissibles. Nous étudions ensuite plusieurs stratégies de réécriture et de surréduction de graphes admissibles, c'est-à-dire des algorithmes permettant d'éliminer des calculs inutiles ou redondants. Nous montrons que nos stratégies sont optimales selon de nombreux critères dépendants des systèmes de réécriture considérés.

On Data Types in Functional Logic Programming Languages : Admissible Graph Rewriting and Narrowing.

Functional logic languages are very high level programming languages which allow to define in a uniform way data types, functions and predicates (relations). Several propositions of functional logic languages have been done but they are based on *first-order terms* computations. This restriction allows to program with algebraic abstract data types but is not appropriate to manipulate real-world data types, as they are modeled with *cyclic graphs*. The aim of this thesis is thus to introduce cyclic graphs as basic data structure in functional logic languages : we consider the programs as cyclic graph rewriting systems and we study the rewriting and narrowing relations they induce (operational semantics).

An important property of rewriting concerns confluence : it expresses the determinism of the computations. Many results of confluence exist for term rewriting but they do not hold for cyclic graph rewriting in general. We characterize a class of particular cyclic graphs, the admissible graphs, and prove that the admissible graph rewriting relation is confluent. Concerning the narrowing relation, we propose a definition and prove that this calculus is sound and complete w.r.t. the admissible graph rewriting relation. Then we study several admissible graph rewriting and narrowing strategies, i.e., algorithms allowing to eliminate useless or redundant calculus. We show that our strategies are optimal w.r.t. many criteria depending on the graph rewriting system which is considered.

Spécialité : "Informatique : Systèmes et Communications."
Mots-Clefs : Langages de programmation, langages logico-fonctionnels, types de données, graphes admissibles, réécriture de graphes, surréduction de graphes, stratégies de réécriture, stratégies de surréduction.

Laboratoire Leibniz-Imag, 46 Av. Félix Viallet, 38100 Grenoble.