Réécriture de Graphes Admissibles : Confluence et Stratégie

R. Echahed et J. C. Janodet
IMAG-LSR, CNRS
B.P.72 F-38402 Grenoble Cedex
$\{echahed, janodet\}$@$imag.fr$

Abstract: Nous considérons la réécriture de graphes en tant que sémantique opérationnelle des langages "équationnels". Nous nous plaçons dans le cadre des signatures avec constructeurs. Nous commençons par caractériser un sous-ensemble de graphes dits *graphes admissibles*. Un graphe est admissible si aucun de ses cycles ne contient de fonction définie. Ensuite, nous considérons la relation de réécriture sur les graphes admissibles et montrons sa confluence ainsi que sa confluence modulo "la bisimulation". Enfin, nous définissons une stratégie de réécriture pour les graphes admissibles qui calcule toujours des rédex nécessaires (*needed*, en anglais).

1 Introduction

Plusieurs définitions de la réécriture de graphes existent dans la littérature (cf. par exemple [5, 8, 15] pour un survol de ces différentes approches). Dans cet article, nous considérons la réécriture de graphes comme un moyen de calcul symbolique permettant l'évaluation de fonctions définies par des règles de réécriture, sur des structures de données représentées par des graphes (cycliques) (e.g., [13, 1]). La figure 1 montre un échantillon de telles structures de données. Le graphe H est un élément du type `Humain` défini par le constructeur `personne` de profil `personne : Nom, Age, Humain` $\rightarrow$ `Humain` où `Nom` et `Age` dénotent d'autres types de données. Pour cet exemple, il semble

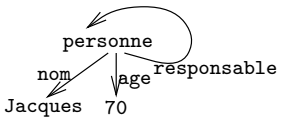

FIG. 1 – *Le graphe H représente un personnage qui s'appelle Jacques, 70 ans, et qui est responsable de lui-même (parce qu'il est majeur).*

naturel d'introduire les fonctions d'accès `nom`, `age` et `responsable` définies par les règles suivantes :

```
          nom( personne(x_nom, y_age, z_resp) ) -> x_nom
          age( personne(x_nom, y_age, z_resp) ) -> y_age
  responsable( personne(x_nom, y_age, z_resp) ) -> z_resp
```

Journées du GDR Programmation.
12, 13, 14 novembre 1997.
Rennes.

Cependant, de telles règles, dites "effondrantes" (*collapsing rules*, en anglais) parce que leurs membres droits sont des variables, ne permettent pas, en général, d'obtenir une relation de réécriture confluente. Nous rappelons ci-dessous l'exemple classique de règles effondrantes induisant une relation de réécriture non confluente.

Exemple 1 Considérons les règles `u(x)` $\rightarrow$ `x` et `v(x)` $\rightarrow$ `x` et les graphes g_1, g_2 et g_3 de la figure 2. En utilisant la définition de la réécriture de graphes (cf. définition 11), on peut réécrire g_1 de deux manières, soit vers g_2 en utilisant la première règle, soit vers g_3 en utilisant la seconde règle. Mais il n'existe pas de graphe g_4 tel que g_2 et g_3 se réécrivent tous les deux vers g_4.

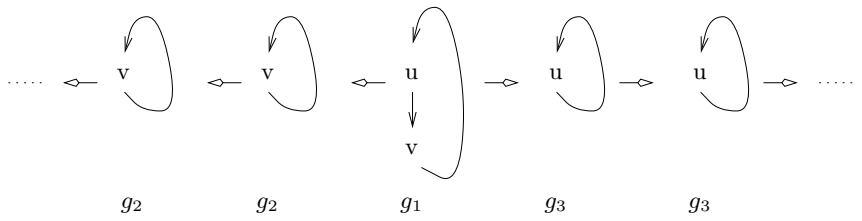

FIG. 2 –

Dans le cadre des systèmes de réécriture de graphes orthogonaux, la confluence a été établie pour des systèmes admettant au plus une règle effondrante. En présence de plusieurs règles effondrantes, la confluence est établie modulo l'égalité de certains graphes [10].

Nous nous plaçons ici dans le contexte de programmes avec constructeurs. Nous nous intéressons à la relation de réécriture définie sur l'ensemble de graphes dits *admissibles*. Un graphe est admissible si aucun de ses cycles ne contient de fonction définie. Par exemple, les graphes H, `nom(`H`)`, `nom(responsable(`H`))` sont des exemples de graphes admissibles. Nous montrons que la relation de réécriture considérée est confluente sur cet ensemble, et cela même en présence de plusieurs règles effondrantes. D'autre part, nous montrons que la relation de réécriture est confluente modulo la bisimulation. On dit que deux graphes sont *bisimilaires* s'ils représentent le même arbre infini quand on les déplie. Ce résultat est utile pour montrer la complétude de la surréduction de graphes [6].

Lorsqu'une relation de réécriture est confluente, l'évaluation des expressions peut être réalisée de manière déterministe en utilisant des stratégies de choix de "rédex". De telles stratégies ont fait l'objet de plusieurs études dans le cadre des systèmes de réécriture de termes finis ou infinis (cf. par exemple [12, 9, 14, 11]). Dans [2], une stratégie calculant des rédex nécessaires a été développée dans le cadre des systèmes de réécriture de termes avec constructeurs. Nous étendons cette dernière stratégie à la réécriture de graphes admissibles et nous montrons qu'elle préserve les mêmes bonnes propriétés sur les graphes admissibles que sur les termes finis.

Cet article se décompose de la manière suivante. Nous rappelons succinctement les définitions de base sur les graphes dans la section suivante. La section 3 introduit le cadre des systèmes de réécriture de graphes avec constructeurs. Nous nous intéressons ensuite aux problèmes de la confluence et de la confluence modulo la bisimulation dans la section 4. Enfin, nous exposons une stratégie de réécriture optimale dans la section 5. Les définitions précises ainsi que les preuves pourront être consultées dans [7].

2 Préliminaires sur les graphes

De nombreuses notations sont utilisées pour étudier la réécriture de graphes. Nous commençons par préciser les définitions et les notations que nous utilisons et qui sont conformes avec [4].

Une *signature multi-sortée* $\Sigma = \langle S, \Omega \rangle$ consiste en un ensemble S de sortes, et une famille $\Omega = \uplus_{(w,s)\in S^* \times S} \Omega_{w,s}$ d'ensembles de symboles d'opérations. On note $f : s_1 \dots s_n \to s$ 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* $s_1 \dots s_n, s$.

Un graphe est composé d'un ensemble de nœuds et de flèches entre ces nœuds. Chaque nœud est étiqueté par un symbole d'opération ou par une variable. Soient $\mathcal{X} = \uplus_{s\in S} \mathcal{X}_s$ une famille S-indicée de variables, et $\mathcal{N} = \uplus_{s\in S} \mathcal{N}_s$, une famille S-indicée de nœuds. Nous supposons que $\mathcal{X}$ et $\mathcal{N}$ sont fixés dans la suite.

Définition 1 Un *graphe* g défini sur $\langle \Sigma, \mathcal{N}, \mathcal{X} \rangle$ est un quadruplet $g = \langle \mathcal{N}_g, \mathcal{E}_g, \mathcal{S}_g, \mathcal{R}_g \rangle$, où $\mathcal{N}_g$ est un ensemble de *nœuds*, $\mathcal{E}_g : \mathcal{N}_g \to \Omega \cup \mathcal{X}$ est une *fonction d'étiquetage* associant un symbole d'opération ou de variable à tout nœud de g, $\mathcal{S}_g : \mathcal{N}_g \to \mathcal{N}_g^*$ est une *fonction successeur* associant une séquence (éventuellement vide) de nœuds à tout nœud de g, et $\mathcal{R}_g$ est un ensemble de nœuds distingués qu'on appelle *racines*: $\mathcal{R}_g \subseteq \mathcal{N}_g$. De plus, nous imposons trois conditions de bonne définition. (i) Les graphes sont bien typés : un nœud n est de même sorte que son étiquette $\mathcal{E}_g(n)$, et ses successeurs $\mathcal{S}_g(n)$ sont compatibles avec l'arité de $\mathcal{E}_g(n)$. (ii) Les graphes sont connexes : pour tout nœud $n \in \mathcal{N}_g$, il existe une racine $r \in \mathcal{R}_g$ et un chemin entre r et n. Un chemin C d'un nœud n_0 vers un nœud n_k est une séquence de nœuds $C = [n_0, n_1, \dots, n_k]$ telle que $k \geq 1$ et n_i soit un successeur de n_{i-1} $\forall i \in 1..k$. (iii) Soit $\mathcal{V}(g)$ l'ensemble des variables de g. Pour tout $x \in \mathcal{V}(g)$, il existe un nœud *unique* $n \in \mathcal{N}_g$ tel que $\mathcal{E}_g(n) = x$. Un graphe (éventuellement cyclique) qui n'a qu'une seule racine s'appelle un *terme-graphe*. □

Exemple 2 Dans la figure 3, nous donnons deux exemples de graphes. Leurs racines sont désignées par des flèches sans antécédent. Le graphe de gauche, G, représente une liste cyclique alternant des 2 et des 1. G est un terme-graphe, puisqu'il n'a qu'une racine $\mathcal{R}_G = \{\texttt{x1}\}$. Les nœuds de G sont $\mathcal{N}_G = \{\texttt{x1}, \dots, \texttt{x5}\}$, sa fonction d'étiquetage est définie par $\mathcal{E}_G(\texttt{x1}) = \mathcal{E}_G(\texttt{x5}) = \texttt{cons}$, $\mathcal{E}_G(\texttt{x2}) = \texttt{+}$, $\mathcal{E}_G(\texttt{x3}) = \texttt{succ}$, et $\mathcal{E}_G(\texttt{x4}) = \texttt{0}$, et sa fonction successeur est définie par $\mathcal{S}_G(\texttt{x1}) = \texttt{x2.x5}$, $\mathcal{S}_G(\texttt{x2}) = \texttt{x3.x3}$, $\mathcal{S}_G(\texttt{x3}) = \texttt{x4}$, $\mathcal{S}_G(\texttt{x4}) = \varepsilon$, $\mathcal{S}_G(\texttt{x5}) = \texttt{x3.x1}$. Le graphe de droite, T, représente la règle de réécriture classique `succ(x)+y` → `succ(x+y)` (cf. définition 8). T a deux racines $\mathcal{R}_T = \{\texttt{l1,r1}\}$.

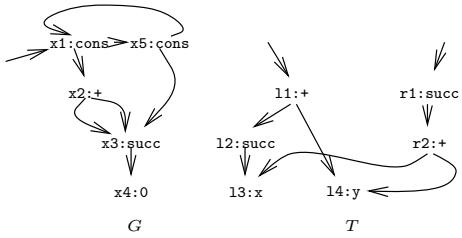

FIG. 3 – *Deux exemples de graphes.*

Les dessins et la définition formelle des graphes ne sont pas toujours pratiques pour donner des exemples. Nous utilisons ci-après une notation linéaire pour écrire les terme-graphes.

Définition 2 Dans la grammaire suivante, la variable A (resp. n) parcourt l'ensemble $\Omega \cup \mathcal{X}$ (resp.

$\mathcal{N}$). L'expression linéaire d'un terme-graphe est définie par :

$$\textsc{Graphe} \quad ::= \quad n{:}A(\textsc{Graphe},\ldots,\textsc{Graphe}) \mid n$$

□

Exemple 3 Avec cette syntaxe, le terme-graphe G de la figure 3 s'écrit :
$G =$ `x1:cons(x2:+(x3:succ(x4:0),x3),x5:cons(x3,x1))`.

Nous explicitons maintenant sur des exemples les notions de sous-graphe et de remplacement nécessaires pour la définition d'un pas de réécriture. Le *sous-graphe* d'un graphe g au nœud p, noté $g_{|p}$, se construit en considérant p comme une racine, et en éliminant tous les nœuds inaccessibles depuis p dans g. Par exemple, le sous-graphe de G (cf. fig. 3) au nœud `x2` est défini par $G_{|\texttt{x2}} =$ `x2:+(x3:succ(x4:0),x3)`.

Concernant le remplacement, la figure 4 montre les étapes de calcul du remplacement par D du sous-graphe de G au nœud `x2`. D'abord, comme G et D partagent des nœuds (`x3` et `x4`), il faut considérer G et D non pas comme deux terme-graphes, mais comme un seul graphe avec deux racines (H_1 de la figure 4). Ensuite, on redirige toutes les flèches pointant sur `x2` dans G vers la racine `r1` de D (H_2) ; cette opération s'appelle une *redirection de pointeurs*. Enfin, on supprime tous les nœuds inaccessibles depuis la racine de G. Le graphe résultant du *remplacement par D du sous-graphe de G au nœud* `x2` se note $G[\texttt{x2} \leftarrow D]$.

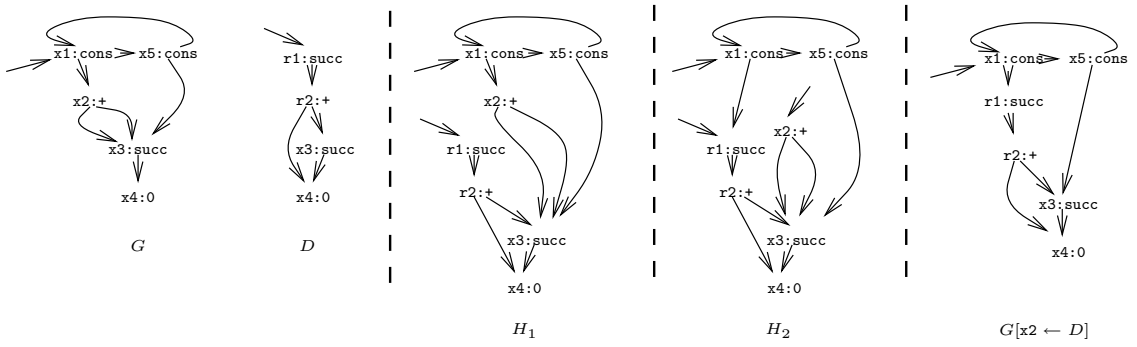

FIG. 4 – Calcul d'un remplacement.

Remarque : Le calcul de $g[p \leftarrow d]$ peut avoir des résultats surprenants dans certains cas. Prenons par exemple $g =$ `l1:+(l2:0,l3:0)`, $d =$ `r1:+(l1:+(l2:0,l3:0),r2:0)`, et $p =$ `l1`. Alors, le lecteur peut vérifier que $g[p \leftarrow d] =$ `r1:+(r1,r2:0)` et donc que $g[p \leftarrow d] \neq d$. Autrement dit, le calcul de $g[p \leftarrow d]$ peut modifier d.

Nous rappelons ci-dessous la notion d'homomorphisme de graphes. Ils jouent presque le même rôle que les substitutions pour des termes du premier ordre.

Définition 3 Soient g_1, g_2 deux graphes. Un *homomorphisme* $h : g_1 \to g_2$ de g_1 vers g_2 est une application de $\mathcal{N}_{g_1}$ vers $\mathcal{N}_{g_2}$ telle que pour tout $n \in \mathcal{N}_{g_1}$, si $\mathcal{E}_{g_1}(n) \notin \mathcal{X}$, alors $\mathcal{E}_{g_2}(h(n)) = \mathcal{E}_{g_1}(n)$, $\mathcal{S}_{g_2}(h(n)) = h(\mathcal{S}_{g_1}(n))$[1], et $\mathcal{R}_{g_2} = h(\mathcal{R}_{g_1})$[2], et si $\mathcal{E}_{g_1}(n) \in \mathcal{X}$, alors $h(n) \in \mathcal{N}_{g_2}$. On dit qu'un homomorphisme h est un $\mathcal{V}$-homomorphisme si pour tout nœud étiqueté par une variable, soit x, son image par h est aussi étiquetée par la même variable x. □

1. i.e., si $\mathcal{S}_{g_1}(n) = \varepsilon$, alors $\mathcal{S}_{g_2}(h(n)) = \varepsilon$, et si $\mathcal{S}_{g_1}(n) = n_1 \ldots n_k$, alors $\mathcal{S}_{g_2}(h(n)) = h(n_1) \ldots h(n_k)$.
2. c'est-à-dire, si $\mathcal{R}_{g_1} = \{r_1, \ldots, r_k\}$, alors $\mathcal{R}_{g_2} = \{h(r_1), \ldots, h(r_k)\}$.

Exemple 4 Dans la figure 5, on donne un exemple d'homomorphisme h allant du graphe $L =$ `l1:+(l2:succ(l3:x),l4:y)` vers le sous-graphe de G au nœud `x2`. Formellement, h est une application de $\mathcal{N}_L$ vers $\mathcal{N}_{(G_{|\texttt{x2}})}$ définie par $h(\texttt{l1}) = \texttt{x2}$, $h(\texttt{l2}) = h(\texttt{l4}) = \texttt{x3}$, et $h(\texttt{l3}) = \texttt{x4}$. h n'est pas un $\mathcal{V}$-homomorphisme puisque $\mathcal{E}_L(\texttt{l3}) = \texttt{x}$ alors que $\mathcal{E}_G(h(\texttt{l3})) = \mathcal{E}_G(\texttt{x4}) \neq \texttt{x}$.

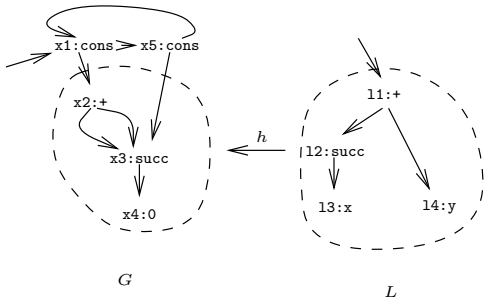

FIG. 5 – *Homomorphisme entre deux graphes.*

La définition d'un pas de réécriture nécessite la définition du *filtrage*.

Définition 4 Étant donné un graphe g, un terme-graphe l, et un nœud n de g, on dit que l *filtre* g *au nœud* n, noté $l \leq g_{|n}$, s'il existe un homomorphisme $h : l \to g_{|n}$. h est appelé *filtre de* l *sur* g *au nœud* n. □

Exemple 5 L'homomorphisme h de la figure 4 est un filtre de L sur G au nœud `x2`.

Enfin, la définition d'un pas de réécriture nécessite la définition de *l' "application" d'un homomorphisme* $h : g_1 \to g_2$ *sur un graphe* g. Calculer $h(g)$ consiste à remplacer tous les sous-graphes communs entre g et g_1 par les sous-graphes correspondants dans g_2.

Exemple 6 Considérons l'homomorphisme $h : L \to G_{|\texttt{x2}}$ de l'exemple 4. Soit un nouveau graphe $R =$ `r1:succ(r2:+(l3:x,l4:y))`. Calculer $h(R)$ consiste à remplacer tous les sous-graphes communs à R et L par les sous-graphes correspondants dans $G_{|\texttt{x2}}$. Ici, R et L partagent les sous-graphes `l3:x` et `l4:y`. Les graphes correspondants par h dans G sont respectivement `x4:0` et `x3:succ(x4:0)`. On obtient donc $h(R) =$ `r1:succ(r2:+(x4:0,x3:succ(x4)))`, soit le graphe D de la figure 4.

En dehors des notions de filtrage et d'application d'un homomorphisme sur un graphe, les homomorphismes sont aussi utilisés pour définir des *relations d'égalités* sur les graphes. Nous avons déjà utilisé l'*égalité syntaxique* des graphes, notée $=$. Nous définissons ci-dessous deux nouvelles égalités.

Définition 5 Soient g_1 et g_2 deux graphes. On dit que g_1 *et* g_2 *sont égaux au renommage des nœuds près*, noté $g_1 \sim g_2$ s'il existe un $\mathcal{V}$-homomorphisme bijectif allant de g_1 vers g_2. On dit que g_1 *et* g_2 *sont bisimilaires*, noté $g_1 \doteq g_2$, s'il existe un graphe g' et deux $\mathcal{V}$-homomorphismes $h_1 : g_1 \to g'$ et $h_2 : g_2 \to g'$. □

Remarque : Dans la définition de la bisimulation de graphes, les homomorphismes h_1 et h_2 sont des $\mathcal{V}$*-homomorphismes*. Ainsi, les variables de g_1 et g_2 sont préservées dans g'. Par conséquent, deux graphes qui n'ont pas le même ensemble de variables ne peuvent pas être bisimilaires. La

bisimulation peut être définie autrement. En effet, on peut montrer que deux termes-graphes g_1 et g_2 sont bisimilaires s'ils représentent le même arbre (infini) quand on les déplie [3].

Exemple 7 Dans la figure 6, les graphes de gauche et de droite sont bisimilaires, parce qu'il existe un $\mathcal{V}$-homomorphisme entre chacun d'eux et le graphe du milieu.

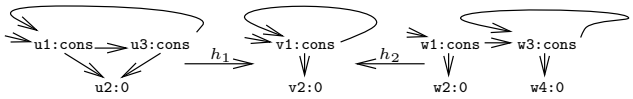

FIG. 6 – *Trois graphes bisimilaires.*

3 Système de réécriture de graphes avec constructeurs

Nous nous plaçons maintenant dans le cadre des signatures avec constructeurs pour des raisons pratiques. Une *signature avec constructeurs* $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ consiste en un ensemble S de sortes, un ensemble $\mathcal{C}$ de constructeurs, et un ensemble $\mathcal{D}$ de fonctions définies tels que $\langle S, \mathcal{C} \uplus \mathcal{D} \rangle$ soit une signature, et $\mathcal{C} \cap \mathcal{D} = \emptyset$.

Exemple 8 La figure 7 montre un exemple de signature avec constructeurs définissant les *listes éventuellement cycliques de naturels*.

```
signature list-of-nat
sorts
   list nat
constructors
   0 : → nat
   succ : nat → nat
   cons : nat list → list
operations
   + : nat nat → nat
   * : nat nat → nat
   sqr : nat nat → nat
   car : list → nat
   cdr : list → list
end signature
```

FIG. 7 – *Signature* `list-of-nat`.

Un graphe g est dit *constructeur* si tous ses nœuds sont étiquetés soit par des variables, soit par des symboles de constructeurs.

Dans la suite de cet article, nous nous intéressons à une classe particulière de graphes dit *admissibles*. Ils correspondent aux graphes les plus utilisés dans la pratique.

Définition 6 Un graphe est admissible sur une signature avec constructeurs s'il ne contient pas de cycle sur un nœud étiqueté par une fonction définie, i.e., il n'existe pas de chemin $[n_0, \ldots, n_k]$ tel que $n_0 = n_k$ et n_0 est étiqueté par une fonction définie. □

Exemple 9 Le graphe G de la figure 2 est admissible. Les graphes `z:cdr(n1:cons(n2:0,z))` et `z:+(z,z)` ne le sont pas.

Définition 7 Un terme-graphe est un *motif* (*pattern*, en anglais) si (i) sa racine est étiquetée par une fonction définie, (ii) tous ses autres nœuds sont étiquetés soit par des variables, soit par des symboles de constructeurs, (iii) il ne contient pas de cycle sur sa racine, et (iv) il existe un chemin unique entre sa racine et tout autre nœud. Autrement dit, un motif est un graphe qui correspond à un terme du premier ordre linéaire dont seule la racine est étiquetée par une fonction définie. □

Définition 8 Une *règle de réécriture* est un graphe e avec deux racines, noté $e = l \rightarrow r$. l (resp. r) est un terme-graphe qu'on appelle *membre gauche* (resp. *membre droit*) de la règle. Nous supposerons toujours que $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. On dit qu'une règle e' est une variante de la règle e si e' est obtenu à partir de e en renommant toutes les variables et tous les nœuds. □

Par définition d'un graphe, les variables apparaissant dans une règle de réécriture sont toujours partagées entre le membre gauche et le membre droit de la règle. Le graphe T de la figure 3 montre un exemple de règle.

La définition suivante introduit la notion de règle de réécriture admissible. Ce type de règles permet à l'ensemble des graphes admissibles d'être stable par réécriture.

Définition 9 Une règle de réécriture $e = l \rightarrow r$ est *admissible* si (i) l est un motif, (ii) r est un terme-graphe admissible, (iii) l n'est pas un sous graphe de r, et (iv) $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. □

On dit que deux règles de réécriture admissibles ne sont pas *superposables* si leur membres gauches ne sont pas unifiables. Cette notion d'unification est celle des termes du premier ordre, puisque les membres gauches des règles sont des motifs.

Définition 10 Un *système de réécriture de graphes (SRG) avec constructeurs* est une paire $SP = \langle \Sigma, \mathcal{R} \rangle$ telle que $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ soit une signature avec constructeurs, et $\mathcal{R}$ soit un ensemble de règles de réécriture admissibles et non superposables. □

Exemple 10 La figure 8 montre un ensemble de règles de réécriture associées à la signature `list-of-nat` de la figure 7. Seuls les nœuds importants sont explicites dans les règles.

```
rules list-of-nat
   0 + n1:y → n1
   succ(n1:x) + n2:y → succ(n1 + n2)
   n1:0 * x → n1
   succ(n1:x) * n2:y → (n1 * n2) + n1
   sqr( n1:0 ) → n1
   sqr( n1:succ(x) ) → n1 * n1
   car( cons(n1:x,l) ) → n1
   cdr( cons(x,n1:l) ) → n1
end rules
```

FIG. 8 – *Règles de la signature* `list-of-nat`.

La réécriture d'un graphe s'effectue en deux étapes. D'abord, on calcule le filtre entre le membre gauche d'une règle et un sous-graphe du graphe à réécrire, et ensuite, on remplace ce sous-graphe par le membre droit instancié de la règle. La définition suivante est conforme avec [4].

Définition 11 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs, g_1 et g_2 deux graphes admissibles, $e = l \rightarrow r$ une variante de règle de réécriture, et p un nœud de g_1. On dit que g_1 *se réécrit en* g_2

au nœud p avec la règle $l \to r$, noté $g_1 \to_{[p,l\to r]} g_2$, si il existe un homomorphisme $h : l \to g_{1|p}$ (i.e., si l filtre g_1 au nœud p) et si $g_2 = g_1[p \leftarrow h(r)]$. Dans ce cas, on dit que $g_{1|p}$ est un *rédex* de g_1. Un graphe est sous *forme normale* s'il ne contient pas de rédex. Un graphe constructeur est nécessairement sous forme normale. On note $\xrightarrow{*}$ la fermeture réflexive et transitive de $\to$ et on parle de *dérivation* de g_1 vers g_2 lorsque $g_1 \xrightarrow{*} g_2$. □

Exemple 11 Considérons le graphe G et la règle $T = L \to R$ de la figure 3, où $L =$ `l1:+(l2:succ(l3:x,l4:y))` et $R =$ `r1:succ(r2:+(l3:x,l4:y))`. D'après la figure 5, L filtre G au nœud `x2` avec l'homomorphisme h. Nous avons vu dans l'exemple 6 que $h(R) =$ `r1:succ(r2:+(x4:0,x3:succ(x4)))`. Enfin, d'après le calcul présenté dans la figure 4, nous avons $G[\texttt{x2} \leftarrow h(R)] =$ `x1:cons(r1:succ(r2:+(x4:0,x3:succ(x4))),x5:cons(x3,x1))`. Appelons G' ce dernier graphe. Par définition, $G \to_{[\texttt{x2}, L\to R]} G'$.

Considérons maintenant la règle de réécriture `l1:+(l2:0,l3:x)` $\to$ `r1:+(l1,r2:0)`. Cette règle n'est pas admissible puisque la racine de son membre gauche est un nœud du membre droit. En utilisant cette règle, le lecteur peut vérifier que le terme-graphe admissible `x1:+(x2:0,x3:0)` se réécrit en `r1:+(r1,r2:0)`. Ce dernier graphe n'est pas admissible (il contient un cycle sur la fonction définie `+`).

La proposition suivante établit la stabilité de l'ensemble des graphes admissibles par réécriture avec une règle admissible.

Proposition 1 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs, g_1 un graphe admissible, $l \to r$ une variante de règle de $\mathcal{R}$, et n un nœud de g_1. Si il existe un graphe g_2 tel que $g_1 \to_{[n,l\to r]} g_2$, alors g_2 est admissible.

4 Confluence

Dans cette section, nous établissons la confluence et la confluence modulo la bisimulation des SRG avec constructeurs en présence d'un nombre quelconque de règles effondrantes. Le problème de la confluence des SRG n'est pas une extension immédiate de la confluence des systèmes de réécriture de termes, comme nous l'avons vu en introduction (cf. exemple 1).

Définition 12 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. On dit que *la relation de réécriture* $\xrightarrow{*}$ *est confluente* si pour tous graphes *admissibles* g_1, g_2 égaux au renommage des nœuds près ($g_1 \sim g_2$) tels que $g_1 \xrightarrow{*} g_1'$ et $g_2 \xrightarrow{*} g_2'$, il existe deux graphes admissibles g_1'' et g_2'' tels que $g_1' \xrightarrow{*} g_1''$, $g_2' \xrightarrow{*} g_2''$, et g_1'', g_2'' soient égaux au renommage des nœuds près. (cf. figure 9). □

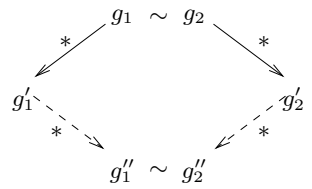

FIG. 9 – *Confluence.*

Proposition 2 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. Alors la relation de réécriture de graphes admissibles est confluente.

Dans [3], les terme-graphes cycliques sont représentés avec des systèmes d'équations. Il est prouvé que tout SRG sans règles superposables est confluent. Ce résultat surprenant est établi en fait pour une relation particulière de réécriture différente de celle considérée ici.

Nous considérons maintenant le problème de la confluence modulo la bisimulation.

Définition 13 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. On dit que *la relation de réécriture* $\stackrel{*}{\rightarrow}$ *est confluente modulo la bisimulation* $\doteq$ si pour tous graphes admissibles et bisimilaires g_1, g_2 tels que $g_1 \stackrel{*}{\rightarrow} g_1'$ et $g_2 \stackrel{*}{\rightarrow} g_2'$, il existe deux graphes admissibles et bisimilaires g_1'' et g_2'' tels que $g_1' \stackrel{*}{\rightarrow} g_1''$ et $g_2' \stackrel{*}{\rightarrow} g_2''$ (cf. figure 10). □

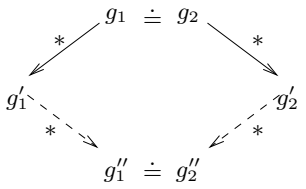

FIG. 10 – *Confluence modulo la bisimulation.*

Proposition 3 Soit $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. Alors la relation de réécriture de graphes admissibles est confluente modulo la bisimulation.

5 Une stratégie de réécriture de graphes admissibles

Pour effectuer un pas de réécriture sur un graphe, il faut exhiber un nœud où une règle de réécriture doit s'appliquer. Un tel nœud se détermine avec une stratégie de réécriture.

Définition 14 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. Une *stratégie de réécriture de graphe* est une fonction partielle $\mathcal{S}$ qui prend un graphe admissible et retourne un couple (p, R) où p est un nœud de g, R est une règle de réécriture, et g se réécrit au nœud p avec la règle R. On note $g \rightarrow_{\mathcal{S}} g'$ si $\mathcal{S}(g) = (p, R)$ et $g \rightarrow_{[p,R]} g'$. On note $\stackrel{*}{\rightarrow}_{\mathcal{S}}$ la fermeture réflexive et transitive de $\rightarrow_{\mathcal{S}}$ et on parle de $\mathcal{S}$*-dérivation* de g vers g' lorsque $g \stackrel{*}{\rightarrow}_{\mathcal{S}} g'$. Une stratégie $\mathcal{S}$ est *normalisante* si pour tout graphe g qui admet une forme normale constructeur c, il existe un graphe constructeur c' et une $\mathcal{S}$-dérivation de g vers c' telle que c et c' soient égaux au renommage des nœuds près. □

Une stratégie normalisante efficace a été définie dans [2] pour les systèmes de réécriture de termes avec constructeurs. Nous nous en inspirons ici pour définir une stratégie sur les SRG avec constructeurs, qui reste normalisante sur les graphes admissibles.

Définition 15 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. Soient g et g' deux graphes admissibles et $B = g \stackrel{*}{\rightarrow} g'$ une dérivation de réécriture. Soit q un nœud de g étiqueté par une fonction définie. On dit que q est un *nœud résiduel* par B si q reste un nœud de g', et dans ce cas, on appelle *descendant de* $g_{|q}$ par B le sous-graphe $(g')_{|q}$. On dit qu'un rédex u de racine q dans g est *nécessaire* si dans toute dérivation de g vers une forme normale constructeur, un descendant de $g_{|q}$ est réécrit au nœud q. On dit qu'un nœud q, étiqueté par une fonction définie, est un *plus haut nœud* de g (*outermost*, en anglais), si q est une racine de g ou s'il existe un chemin allant d'une racine de g vers q qui ne passe pas par un nœud $p \neq q$ étiqueté par une fonction définie. Enfin, on

dit qu'un rédex u de racine q dans g est un *plus haut rédex* de g si q est une racine de g, ou s'il existe un chemin allant d'une racine de g vers q qui ne passe pas par un nœud $p \neq q$ tel que $g_{|p}$ soit un rédex. □

Remarque : Les notions de plus haut nœud et de plus haut rédex sont bien définies dans le cadre des graphes admissibles, i.e., dans un graphe admissible, si p et q sont deux nœuds étiquetés par des fonctions définies tels que p soit plus haut que q, alors q ne peut pas être plus haut que p.

Notre stratégie de réécriture utilise dans sa définition une structure hiérarchique appelée *arbre de définition* [2], dont les feuilles sont les règles d'un SRG avec constructeurs utilisées pour définir une fonction donnée. Dans la définition suivante, les symboles *branch* et *rule* sont des fonctions non interprétées utilisées pour construire les nœuds d'un arbre de définition.

Définition 16 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. On dit que $\mathcal{T}$ est un *arbre de définition partiel*, ou *adp*, de *motif* π dans les deux cas suivants :

1. $\mathcal{T} = rule(\pi \to r)$, où $\pi \to r$ est une variante de règle de $\mathcal{R}$.
2. $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, où o est un nœud de π étiqueté par une variable de sorte s, $c_1, \ldots, c_k$ $(k > 0)$ sont différents constructeurs de la sorte s, et pour tout $j \in 1..k$, $\mathcal{T}_j$ est un adp de motif $\pi[o \leftarrow p : c_j(o_1 : X_1, \ldots, o_n : X_n)]$, où n est le nombre d'arguments de c_j, $X_1, \ldots, X_n$ sont de nouvelles variables, et $p, o_1, \ldots, o_n$ sont de nouveaux nœuds.

On dit que $\mathcal{T}$ est un *arbre de définition* d'une fonction f si $\mathcal{T}$ est un adp *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 de nouvelles variables, $p, o_1, \ldots, o_n$ sont de nouveaux nœuds, et pour toute règle $l \to r$ de $\mathcal{R}$ telle que l soit de la forme $f(g_1, \ldots, g_n)$, il existe une feuille $rule(l' \to r')$ de $\mathcal{T}$ telle que $l' \to r'$ soit une variante de $l \to r$. On dit qu'un SRG avec constructeurs est *inductivement séquentiel* si pour chacune de ses fonctions définies f, il existe un arbre de définition de f. □

Exemple 12 Considérons la définition de l'addition définie par le SRG suivant :

```
x1:+(x2:0,x3:x) -> x3
x4:+(x5:succ(x6:x),x7:y) -> x8:succ(x9:+(x6,x7))
```

L'arbre de définition $\mathcal{T}_+$ de l'opération `+` est

$$\begin{aligned}\mathcal{T}_+ = branch(\ & \texttt{l1:+(l2:X,l3:Y)},\\ & \texttt{l2},\\ & rule(\texttt{l4:+(l5:0,l6:U)} \to \texttt{l6}),\\ & rule(\texttt{l7:+(l8:succ(l9:V),l10:W)} \to \texttt{l11:succ(l12:+(l9,l10))}))\end{aligned}$$

Nous définissons maintenant la stratégie de réécriture Φ sur les graphes admissibles pour les SRG inductivement séquentiels. Appliquée à un graphe admissible g, Φ retourne quand c'est possible un couple (p, R) où p est un nœud de g et R est une règle de réécriture telle que g se réécrive au nœud p avec la règle R. $\Phi(g)$ utilise une seconde fonction φ qui prend deux arguments : un terme-graphe et un adp.

Définition 17 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs. Soit g un graphe admissible. On définit la fonction partielle Φ en posant $\Phi(g) = \varphi(g_{|p}, \mathcal{T}_f)$ où p est un plus haut nœud de g, étiqueté par un symbole de fonction définie f, et $\mathcal{T}_f$ est un arbre de définition de l'opération f.

Soit g un terme-graphe admissible dont la racine est étiquetée par une fonction définie, et $\mathcal{T}$ un adp tel que le motif π de $\mathcal{T}$ filtre g à la racine. On définit la fonction partielle φ en posant :

$$\varphi(g,\mathcal{T}) = \begin{cases} (\mathcal{R}_g, \pi \to r) & \text{si } \mathcal{T} = rule(\pi \to r); \\ \varphi(g, \mathcal{T}_i) & \text{si } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ et il existe } i \in 1..k \text{ tel que} \\ & \quad \text{le motif de } \mathcal{T}_i \text{ filtre } g \text{ à la racine}; \\ (p, R) & \text{si } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \pi \text{ filtre } g \text{ à la racine par l'homomorphisme } h : \pi \to g, \\ & \quad h(o) \text{ est étiqueté par un symbole de fonction définie } f, \\ & \quad \mathcal{T}' \text{ est l'arbre de définition de } f, \text{ et} \\ & \quad \varphi(g_{|h(o)}, \mathcal{T}') = (p, R). \end{cases}$$

□

Exemple 13 Considérons le graphe admissible $g =$ `x1:cons(x2:+(x3:+(x4:0,x4),x3),x1)`.

$$\begin{array}{rcl} \Phi(g) & = & \varphi(\ \texttt{x2:+(x3:+(x4:0,x4),x3)}, \mathcal{T}_+) \\ & = & \varphi(\ \texttt{x3:+(x4:0,x4)}, \\ & & \quad branch(\texttt{l1:+(l2:X,l3:Y)}, \texttt{l2}, rule(\texttt{l4:+(l5:0,l6:U)} \to \texttt{l6}), \ldots)) \\ & = & \quad (\texttt{x3}, \texttt{l4:+(l5:0,l6:U)} \to \texttt{l6}) \end{array}$$

Lemme 1 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs, f un symbole de fonction définie, $\mathcal{T}$ un adp de f, et g un graphe admissible dont la racine est étiquetée par f. Si $\varphi(g, \mathcal{T}) = (p, R)$, alors dans toute dérivation de g vers un terme-graphe c dont la racine est étiquetée par un symbole constructeur, un descendant de $g_{|p}$ est réécrit à la racine, en un ou plusieurs pas, vers un graphe dont la racine est étiquetée par un constructeur. Par contre, si $\varphi(g, \mathcal{T})$ n'est pas définie, alors g ne peut pas être réécrit vers un terme-graphe constructeur.

Proposition 4 Soient $SP = \langle \Sigma, \mathcal{R} \rangle$ un SRG avec constructeurs, et g un 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 ne peut pas être réécrit vers un graphe constructeur.

De la proposition précédente, on déduit que si un graphe admissible g admet une forme normale constructeur c, et si $\Phi(g) = (p, R)$, alors toute dérivation de g vers c doit nécessairement exécuter un pas de réécriture au nœud p avec la règle R.

Proposition 5 Φ est une stratégie normalisante.

En fait, Φ est plus que normalisante. Elle est *hyper-normalisante*, c'est-à-dire qu' à partir d'un graphe g qui admet une forme normale constructeur c, si une dérivation quelconque D partant de g fait appel de temps en temps à Φ, mais infiniment souvent, alors D termine avec une forme normale constructeur c' tel que $c \sim c'$.

La réécriture de graphe n'introduit pas de duplication de données. Ceci permet d'optimiser les calculs. On obtient pour la stratégie Φ la propriété suivante.

Proposition 6 Soient g un graphe admissible, et c un graphe constructeur tels qu'il existe une dérivation de réécriture $g \xrightarrow{*} c$. Alors, il existe un graphe constructeur c' avec $c \sim c'$ tel que la longueur de la dérivation $g \xrightarrow{*}_{\Phi} c'$ soit plus petite (ou égale à) la longueur de la dérivation $g \xrightarrow{*} c$.

6 Conclusion

Dans cet article, nous nous sommes intéressés à des systèmes de réécriture de graphes avec constructeurs. Nous avons exhibé une classe de graphes cycliques dits admissibles pour laquelle la relation de réécriture est confluente et confluente modulo la bisimulation et ce en présence d'un nombre quelconque de règles effondrantes. Nous avons de plus étudié une stratégie de réécriture qui est normalisante et optimale.

Bien d'autres propriétés de la réécriture de graphes sont intéressantes (par exemple la terminaison, la complétion des systèmes de réécriture de graphes, ...). Notre principale motivation dans ce travail était l'étude des caractéristiques nécessaires pour définir et étudier une relation de surréduction sur les graphes admissibles [6].

Références

[1] Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph transformation for specification and programming. *??*, 1996. In preparation.

[2] S. Antoy. Definitional trees. In *Proc. of the 4th Intl. Conf. on Algebraic and Logic programming*, pages 143–157. Springer Verlag LNCS 632, 1992.

[3] Z.M. Ariola and J.W. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3-4), 1996.

[4] H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. Springer Verlag LNCS 259, 1987.

[5] B. Courcelle. Réécriture de graphes: orientation bibliographique. *On the Web : http://www.labri.u-bordeaux.fr/ courcell/ActSci.html*, 1993.

[6] R. Echahed and J. C. Janodet. Introducing graphs in functional logic languages. Technical report, IMAG, 1997. In preparation.

[7] R. Echahed and J. C. Janodet. On constructor-based graph rewriting systems. Technical report, IMAG, 1997.

[8] H. Ehrig and G. Taentzer. Computing by graph transformation : A survey and annotated bibliography. *Bulletin of the EATCS*, 59:182–226, June 1996.

[9] 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. Previous version: Call by need computations in non-ambiguous linear term rewriting systems, Technical Report 359, INRIA, Le Chesnay, France, 1979.

[10] 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. Previous version: Technical Report CS-R9204, CWI, Amsterdam, 1992.

[11] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. Transfinite reduction in orthogonal term rewriting systems. *Information and Computation*, pages 18–38, 1995.

[12] M. J. O'Donnell. *Computing in Systems Described by Equations*. Springer Verlag LNCS 58, 1977.

[13] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.

[14] R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Information and Computation*, 104(1):78–109, May 1993.

[15] 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.