REECRITURE DE GRAPHES : ORIENTATION BIBLIOGRAPHIQUE

Bruno COURCELLE (+)
Université Bordeaux-I
LaBRI (CNRS, URA 1304)

La réécriture des graphes ne se présente pas actuellement comme une théorie unifiée. Les nombreuses définitions existantes reflètent la diversité des motivations. Le présent texte propose un classement des différentes définitions et une bibliographie brièvement commentée qui est introductive et non exhaustive. On y trouvera des textes récents ainsi que des articles de synthèse où le lecteur trouvera les références aux articles plus anciens.

Situation par rapport aux réécritures de termes

Les graphes généralisent les termes qui généralisent les mots. Les systèmes de réécriture de graphes généralisent les systèmes de réécriture de termes *sans variables* qui sont eux-mêmes, en un certain sens, des systèmes semi-thueiens. Les systèmes de réécriture de graphes ne permettent pas (dans l'état actuel des définitions existantes) de *dupliquer des sous-graphes* comme peuvent le faire (pour les sous-termes) les systèmes de réécriture de termes, et ceci grâce aux variables.

Pourquoi réécrire des graphes ?

Les motivations sont très nombreuses, pour la raison que les graphes permettent de modéliser une grande variété d'objets et de situations. On entendra le terme "graphe" dans un sens très large incluant les hypergraphes avec toutes leurs variantes, orientées et/ou étiquetées, ainsi que les graphes dessinés. Tous les graphes seront finis.

Parmi les motivations de la réécriture des graphes, on peut citer, par ordre en gros chronologique d'apparition

(+) Adresse postale : 351 Cours de la Libération, 33405 TALENCE Cedex, France;
Courrier électronique : **courcell@labri.u-bordeaux.fr**

(1) la définition générative de familles de graphes, d'où l'on peut tirer des preuves de propriétés de graphes par récurrence sur les suites de réécritures qui les engendrent;

(2) la génération et la reconnaissance de dessins; les applications concernent entre autres, la reconnaissance des formes et la modélisation du développement de structures végétales ou cellulaires;

(3) l'implantation efficace des langages fonctionnels et du λ-calcul au moyen de graphes représentant des termes avec partages;

(4) l'extension aux graphes des techniques et des résultats de la théorie des langages formels (grammaires, automates, équations, congruences syntaxiques), laquelle englobe déjà depuis longtemps la description des ensembles d'arbres (et de termes, les termes sont assimilés à des arbres);

(5) la construction d'algorithmes efficaces de reconnaissance de familles de graphes;

(6) des applications à la conception assistée par ordinateur;

(7) la modélisation du calcul distribué.

Absence d'une définition la plus générale

On pourrait observer qu'il n'y a pas *une* définition des graphes mais une multitude, correspondant aux usages très divers évoqués ci-dessus, et donc qu'il ne saurait exister *une* définition de la réécriture des graphes. Mais même si l'on se restreint à une classe bien précise de graphes, par exemple celle des graphes orientés avec étiquettes sur les arcs et les sommets, la difficulté n'en demeure pas moins de définir une notion "la plus générale" réécriture de graphe. La raison en est la suivante. Soit une règle disant que l'on a le droit de remplacer G par H. Soit un graphe M où G apparaît comme sous-graphe induit. Ecrivons $M = C[G]$ où C représente le "contexte" d'une occurrence de G dans M. Le problème est de définir précisément $C[H]$ c'est à dire le résultat de la substitution dans M de H à cette occurrence de G, c'est à dire de spécifier précisément comment H "se plonge'" dans le contexte C, autrement dit, comment les sommets de H seront liés à ceux de C. Il est clair que cette difficulté est inexistante dans les cas des mots (systèmes semi-thueiens) et des termes (même pour des règles avec variables).

La difficulté précédemment évoquée est technique. Une seconde difficulté, plus fondamentale, est l'*absence de sémantique*.

En effet, dans le cas des termes, qui désignent ou peuvent toujours désigner, des valeurs ou des fonctions, la réécriture n'est autre le remplacement d'un sous-terme par un terme équivalent, c'est à dire ayant la même sémantique. Cela assure un fondement solide à toutes les définitions, même si, à l'utilisation, l'on ne préoccupe pas de sémantique. Les graphes "généraux" n'ont pas de sémantique. Certains graphes en ont, par exemple les schémas de programmes, les graphes dirigés acycliques utilisés en compilation, les graphes représentant des λ-termes. Les notions de réécriture correspondantes en découlent ou, tout au moins, leurs définitions sont "validables" en termes de sémantique. Mais l'absence de "sémantique générale" des graphes rend arbitraires les définitions "les plus générales" que l'on peut être tentés d'imaginer.

Prenons pour fixer les idées la définition suivante qui concerne les graphes aussi bien que les hypergraphes et permet déjà de faire un certain nombre de choses intéressantes:

(1) par "graphe", on entendra graphe ou hypergraphe *orienté* dont seuls les arcs et hyperarcs sont étiquetés;

(2) une règle est de la forme $G \to H$ où les graphes G et H sont munis chacun d'une suite (éventuellement vide) de sommets deux à deux distincts, que l'on appellera les *sources*;

(3) cette règle s'utilise ainsi: on réécrit M en N si et seulement si $M = C \ {}_k G$ et $N = C \ {}_k H$ où k est la *composition parallèle* des graphes avec k sources (c'est une sorte de concaténation) définie ainsi:

> si G et G' sont deux graphes à k sources, $G \ {}_k \ G'$ est le graphe sans source obtenu à partir de deux copies disjointes de G et de G' dont on fusionne les sources de même rang. (Les sources fusionnées forment un *k-séparateur* de $G \ {}_k \ G'$).

Cette définition n'est pas la plus générale: elle ne permet pas "d'éclater" un sommet, c'est-à-dire de remplacer un sommet par un graphe.

Classification

On propose la classification suivante des réécritures de graphes:

1.- Engendrer
    1.1 par des grammaires "context-free"

1.2 par des grammaires non "context-free"
2.- Reconnaître
3.- Calculer
4.- Modéliser

Le quadruple critère *engendrer/reconnaître/calculermodéliser* correspond à des motivations différentes; le critère "*context-free" /non "context-free"* correspond à des moyens et à des cadres théoriques différents.

1. Engendrer

1.1 Grammaires "context-free"

Les grammaires HR ("Hyperedge Replacement") s'obtiennent à partir des règles de réécriture de graphes avec sources définies ci-dessus en imposant que dans chaque règle $G \rightarrow H$, le membre gauche G est un unique hyperarc orienté, muni d'une étiquette *non-terminale,* dont tous les sommets sont des sources. Ces grammaires sont "context-free" en un sens axiomatique précisé ci-dessous.

[1] BAUDERON M., COURCELLE B., Graph expressions and graph rewritings, Mathematical Systems Theory **20** (1987) 83-127

[2] HABEL A., Hyperedge replacement : grammars and languages, L.N.C.S. **643**, 1992

[3] COURCELLE B., Graph rewriting : An algebraic and logic approach, in "Handbook of Theoretical Computer Science , Volume B", J. Van Leeuwen ed., Elsevier,1990, pp.193-242

[4] COURCELLE B., Graph grammars, monadic second-order logic and the theory of graph minors, in "Graph Structure Theory", N. Robertson, P. Seymour eds., Contemporary Mathematics **147**, AMS, 1993, pp. 565-590.

Les grammaires VR ("Vertex Replacement'') sont plus complexes à définir: elles sont fondées sur le remplacement d'un sommet par un graphe. Contrairement aux remplacements d'hyperarcs évoqués ci-dessus, ces rempla-cements de sommets ne donnent pas toujours des grammaires "context-free". On peut donner des conditions suffisantes pour qu'une telle grammaire soit "context-free".

[5] ENGELFRIET J., ROZENBERG G., Graph grammars based on node rewriting: an introduction to NLC graph grammars, Proceedings of the 4th International Workshop on Graph Grammars, L.N.C.S. **532** (1991) 12-23.

[6] COURCELLE B., ENGELFRIET J., ROZENBERG G., Handle-rewriting hypergraph grammars, J. Comput. System Sci., **46** (1993) 218-270.

Le terme "context-free" défini dans [7] qualifie certaines grammaires mais ne définit pas une classe particulière. Une grammaire est "context-free" si les réécritures portant sur des non-terminaux distincts peuvent être permutées. Cette condition permet de représenter par des *arbres de dérivation* les classes d'équivalence des dérivations relativement à l'équivalence engendrée par les permutations des réécritures portant sur des non-terminaux distincts. Par ailleurs, une autre condition permet de caractériser les ensembles engendrés comme les composantes de la plus petite solution d'un système canoniquement associé à la grammaire. Il en résulte que le bon cadre formel pour l'étude des grammaires "context-free" est l'Algèbre Universelle et plus précisément la théorie des systèmes d'équations récursives dont on prend les plus petites solutions (voir [8]). Notons que l'ensemble des graphes finis, ainsi que l'ensemble des graphes planaires finis (sans étiquette) ne sont engendrables par aucune grammaire "context-free" connue: c'est une différence marquante avec le cas des mots puisque chaque langage X^*, pour X fini, est algébrique.

[7] COURCELLE B., An axiomatic definition of context-free rewriting and its application to NLC graph grammars, Theoretical Computer Science **55** (1987) 141-181.

[8] COURCELLE B., Equivalences and transformations of regular systems; Applications to recursive program schemes and grammars, Theoret. Comput. Sci. **42** (1986) 1-122.

1.2 Génération non "context-free"

Il est facile de définir des grammaires non "context-free" à partir des règles de réécritures de graphes avec sources définies plus haut. Ces grammaires peuvent engendrer tous les ensembles récursivement énumérables de graphes: on ne peut pas en dire grand chose de général et de non trivial à la fois.

[9] UESU T., A system of graph grammars which generates all recursively enumerable sets of labelled graphs, Tsukuba Journal of Mathematics **2** (1978) 11-26.

2. Reconnaître

2.1 Analyse syntaxique

A toute grammaire est associé un problème d'analyse syntaxique, c'est-à-dire de reconnaissance. Il existe des grammaires de graphes ayant des algorithmes d'analyse syntaxique polynomiaux et d'autres pour lesquelles le problème correspondant est NP-complet.

[10] BRANDENBURG F.-J., On polynomial time graph grammars, L.N.C.S. **294** (1988) 227-236.

[11] VOGLER W., Recognizing edge replacement graph languages in cubic time, L.N.C.S. **532** (1991) 676-687.

[12] DREWES F., Recognising *k*-connected hypergraphs in cubic time, Theoret. Comput. Sci. **109** (1993) 83-122.

2.2 Reconnaissance par réduction

Une méthode de reconnaissance des graphes par *réductions successives*, devant atteindre une forme normale appartenant à un ensemble fini spécifié de formes normales acceptables, est présentée dans [13]. Elle ne se fonde pas sur l'analyse syntaxique d'une grammaire mais sur les *congruences syntaxiques finies* qui caractérisent les ensembles *reconnais-sables de graphes* (au sens de [14]). Cette méthode donne lieu à des algorithmes de reconnaissance en temps linéaire. Le principe en est de remplacer autant que possible un sous-graphe du graphe considéré par un autre graphe, plus petit, mais syntaxiquement équivalent. Dans le cas du langage régulier $L = a(b^2)^*c$, la construction donne que L est l'ensemble des mots w dont la forme normale est ac pour le système semi-thueien Noetherien et confluent suivant:

$$\{bbbb \rightarrow bb,\ abb \rightarrow a,\ bbc \rightarrow c\}$$

[13] ARNBORG S., COURCELLE B., PROSKUROWSKI A., SEESE D., An algebraic theory of graph reduction, JACM, **40** (1993) 1134-1164.

[14] COURCELLE B., The monadic second-order logic of graphs I: Recognizable sets of finite graphs, Information and Computation **85** (1990) 12-75.

Notons que de tels systèmes ont été construits "à la main" dans [15] pour les 3-arbres partiels (7 règles) et dans [16] pour les 4-arbres partiels (15 règles). Des systèmes reconnaissant les *k*-arbres partiels existent d'après [13] mais ne sont pas actuellement connus. (Les méthodes permettant en théorie de les construire sont impraticables.)

[15] ARNBORG S., PROSKUROWSKI A., Characterization and recognition of partial 3-trees, SIAM J. Alg. Disc. Math.**7** (1986) 305-314.

[16] SANDERS D., On linear recognition of tree-width at most 4, Preprint, November 1992, Georgia Institute of Technology, Atlanta.

3. Calculer

3.1 Programmation fonctionnelle

L'utilisation de graphes dirigés acycliques pour implanter efficacement les définitions récursives et les réductions en λ-calcul remontent à Vuillemin [17] et à Wadsworth [18] respectivement. Les graphes dirigés acycliques sont essentiels en programmation fonctionnelle. Leur utilisation pose des pro-blèmes théoriques, de définition, de validité par rapport à la sémantique donnée, et des problèmes d'implantation. Les aspects théoriques sont étudiés dans [19, 20].

[17] VUILLEMIN J., Correct and optimal implementation of recursion in a simple programming language, J. Comput. System Sci. **9** (1974) 332-354.

[18] WADSWORTH C., Semantics and pragmatics of the lambda-calculus, Ph.D., Oxford, 1971.

[19] RAOULT J.C., On graph rewritings, Theoret. Comput. Sci. **32** (1984) 1-24.

[20] KENNAWAY R., On "On graph rewritings", Theoret. Comput. Sci. **52**(1987) 37-58 et Errata **61** (1987) 317-320.

[21] CORRADINI A., ROSSI F., Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming, Theoret. Comput. Sci. **109** (1993) 7-48.

[22] MARANGET L., La stratégie paresseuse, Thèse, Université Paris-7, Juillet 1992.

Le langage DACTL [23] est un langage fondé sur la réécriture de graphes éventuellement avec cycles et permetttant l'implantation de langages fonctionnels tels que LISP, HOPE, voire de la réduction optimale des λ-termes définie par Lévy [24] à travers la structure de données proposée par Lamping, revue et corrigée par Gonthier *et al.*[26].

[23] BARENDREGT H. et al., Term graph reduction, L.N.C.S. **259** (1987) 141-158.

[24] LEVY J.J., Réductions correctes et optimales dans le λ-calcul, Thèse d'Etat, Université Paris-7, 1978.

[25] LAMPING J., An algorithm for optimal lambda-calculus reduction, 7th Symp. on Principles of Programming Languages, 1990, pp. 16-30.

[26] GONTHIER G., ABADI M., J.J. LEVY, The geometry of optimal lambda reduction, 9th Symp. on Principles of Programming Languages, Albuquerque, New Mexico, 1992, pp. 15-26.

Le lecteur trouvera l'essentiel de cette théorie dans le livre suivant:

[27] SLEEP R. *et al.* eds., Term graph rewriting: Theory and Practice, J. Wiley and Sons, Chichester, UK, 1993.

3.2 Calcul distribué

Métivier a introduit des systèmes de réécriture de graphes dont les règles ne modifient pas la structure des graphes auxquels elles s'appliquent mais se bornent à modifier les étiquettes d'arcs ou de sommets, à orienter des arcs non orientés et à modifier les orientations existantes. Chaque modification se fait localement, c'est à dire sur un sous-graphe isomorphe au membre gauche de la règle qui la définit. On obtient ainsi un modèle de calcul *distribué* car deux règles portant sur des sous-graphes disjoints peuvent être appliquées

simultanément. Le calcul est asynchrone et non-déterministe. Chaque forme normale obtenue est un résultat. La puissance d'expression est augmentée pour les graphes de degré non borné au moyen d'un mécanisme de *priorité* d'application des règles en des occurrences où deux membres gauches se chevauchent, ou, de manière équivalente par une notion de *contexte exclu*. Un certain nombre de résultats permettent de délimiter la puissance de calcul de cette méthode. Voir [28-33].

[28] LITOVSKY I., METIVIER Y., Computing trees with graph rewriting systems with priorities, "Tree automata and languages", M. Nivat , A. Podelski eds., Elsevier 1992, 115-139.

[29] LITOVSKY I., METIVIER Y., Computing with graph rewriting systems with priorities, Theoret. Comput. Sci. **115**(1993)191-224.

[30] LITOVSKY I., METIVIER Y., SOPENA E., Definitions and comparisons of local computations on graphs and networks, Mathematical Systems Theory, à paraître, voir aussi L.N.C.S. **629** (1992) 364-373.

[31] LITOVSKY I., METIVIER Y., ZIELONKA W., The power and limitations of local computations on graphs and networks, L.N.C.S. **657** (1993) 333-343.

[32] COURCELLE B., METIVIER Y., Coverings and minors: applications to local computations in graphs, European Journal of Combinatorics, **15** (1994) 127-138.

[33] LITOVSKY I., METIVIER Y., ZIELONKA W., On the recognition of families of graphs with local computations, Information and Computation, à paraître.

4. Modéliser

Les transformations de graphes permettent de décrire la sémantique opérationnelle de différents langages de programmation tels que PROLOG (Corradini et Rossi [21]), les langages d'acteurs (Janssens [34]), "concurrent constraint programs" (Montanari et Rossi [35]).

[34] JANSSENS D., Equivalence of computations in actor grammars, Theoret. Comput. Sci. **109** (1993) 145-180.

[35] MONTANARI U., ROSSI F., Graph rewriting for a partial ordering semantics of concurrent constraint programming, Theoret. Comput. Sci. **109** (1993) 225-256.

La quête de la définition la plus générale

Différents formalismes ont été proposés. Nagl [36] utilise une approche concrète et algorithmique. (Voir aussi [37]).

[36] NAGL M., Graph Gramatiken: Theorie, Implementierung, Anwendungen, Vieweg, Braunschweig, 1979

[37] KREOWSKI H.J., ROZENBERG G., On structured graph grammars, Information Sciences **52** (1990) 185-210 et 221-246.

La formalisation des réécritures de graphes par Ehrig *et al.* [38] dans le cadre de la théorie des catégories et au moyen de sommes amalgamées a fait l'objet d'une littérature abondante. Notons que ces réécritures ne couvrent pas les réécritures utilisées dans les grammaires VR [5,6].

[35] EHRIG H. *et al.*, Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. L.N.C.S. **532** (1991) 24-36.

Réferences générales

On renverra le lecteur aux actes des différents "International Workshop on Graph Grammars and their Applications to Computer Science" parus dans la série *Lecture Notes in Computer Science*, volumes **73** (1979), **153** (1983), **291** (1987) et **532** (1991) auxquels il faut ajouter le numéro spécial de *Theoretical Computer Science*, volume **109** (1993).