PORGY : réécriture et visualisation de graphes dynamiques

Bruno Pinaud, Guy Melançon

PORGY : réécriture et visualisation de graphes dynamiques

Bruno Pinaud, Guy Melançon

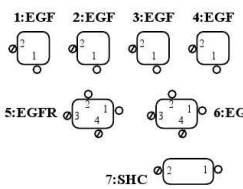
CNRS UMR 5800 LaBRI, INRIA Bordeaux Sud-Ouest
Campus Université Bordeaux I, 351 Cours de la Libération, 33405 Talence Cedex
{bruno.pinaud,guy.melancon}@{labri.fr, inria.fr}
http://www.labri.fr http://www.inria.fr/bordeaux

Résumé. Cet article présente les premiers résultats sur la visualisation et la manipulation interactive d'un système de réécriture de graphes. Nous sommes amenés à nous pencher sur la visualisation de graphes dont la topologie évolue au cours du temps selon des modifications dictées par des règles de réécriture. Le système doit non seulement de montrer le graphe qui évolue au cours du temps, mais il apparaît ici comme un atelier complet permettant d'étudier le système de réécriture lui-même. Il s'agit d'amener la visualisation en appui à l'étude du système de réécriture pour permettre de comprendre son comportement et d'identifier ses propriétés comme par exemple la convergence des calculs ou les configurations bloquantes. Outre les questions relatives au dessin des graphes qui font encore l'objet de travaux, le système se penche sur les problèmes d'identification de motifs et d'historique des réécritures. Nous abordons aussi les questions plus techniques relatives à la structure interne du système.

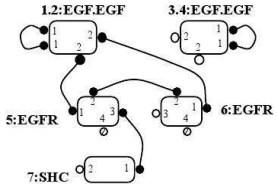
1 Introduction

Les systèmes de réécriture sont d'abord apparus en informatique comme un artefact permettant d'étudier les objets de l'informatique eux-mêmes. Plus généralement, la réécriture est un modèle de calcul qui est aussi utilisé en algèbre, en logique et en linguistique (Terese, 2000). Il s'agit de transformer des objets syntaxiques (mots, termes, lambda-termes, programmes, preuves, graphes, ...) en appliquant des règles. Des exemples classiques d'utilisation de la réécriture sont la simplification d'une expression algébrique, la définition de la grammaire formelle d'un langage de programmation (ou d'une langue naturelle), l'optimisation de code dans les compilateurs ou bien encore la gestion des courriers électroniques dans dans le logiciel *sendmail* où les entêtes des courriers sont manipulées par des systèmes de réécriture. Dans cet article, nous nous intéressons à la réécriture de graphes en bio-informatique sur la modélisation des réseaux d'interactions de polypeptides. Les sommets du graphe représentent les complexes bio-moléculaires qui s'agencent les uns aux autres au gré des réactions biochimiques. Les règles de réécriture permettent alors de tenir compte de la réalité biochimique et de contraindre les façons dont les molécules peuvent s'agencer, exigeant par exemple la présence de certaines configurations avant de lier deux molécules (Andrei, 2008). La figure 1 est un exemple d'un tel système. À partir de l'état initial (figure1-a) du modèle, on peut appliquer une suite de règles de réécriture pour par exemple essayer de trouver une configuration finale

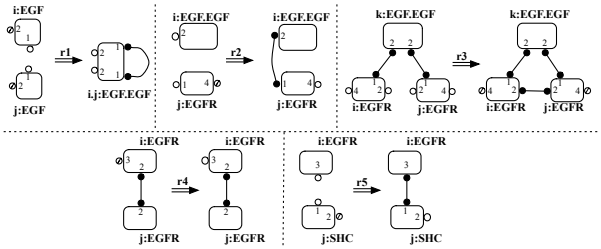
biologiquement valide sur laquelle plus aucune règle n'est applicable (figure1-b). La réécriture apparaît aussi utile pour décrire et simuler des systèmes autonomes (Andrei et Kirchner, 2009; Kephart et Chess, 2003). Un tel système est par exemple un réseau d'échanges de messages qui selon des règles fournies par l'administrateur peut s'auto-gérer (échanger des messages), s'auto-configurer (évolution de la structure du réseau pour assurer un fonctionnement optimal), s'auto-optimiser (par exemple, éviter la saturation d'un serveur), s'auto-réparer (gérer un serveur qui tombe en panne) et s'auto-protéger (gérer les messages indésirables).



(a) État initial.



(b) Un état final stable possible.



(c) Les règles de réécriture associées. Seuls les éléments qui jouent un rôle dans les règles sont représentés.

FIG. 1 – *Modélisation des interactions et agencements de complexes moléculaires par la réécriture (extrait de Andrei (2008)).*

De manière plus formelle, un système de réécriture est un ensemble de règles de réécriture de la forme $A \rightarrow B$ où A et B sont des graphes. Une telle règle s'applique à un graphe G si celui-ci contient au moins une instance du membre gauche A, c'est-à-dire un sous-graphe isomorphe à A. Le graphe G se réécrit alors en un nouveau graphe G', obtenu en remplaçant l'instance de A par une instance du membre droit B correspondant. On connaît aussi la réécriture de mots (séquence de lettres) ou de termes (expression parenthésées), dont la réécriture de graphes est une généralisation – qui gagne aussi en pouvoir d'expression.

Nombre de questions se posent dès lors que l'on considère un système de réécriture.

Terminaison On peut typiquement chercher à savoir si les calculs se *terminent*. En d'autres mots, quel que soit le graphe de départ, peut-on garantir que l'application d'une suite de règles de réécriture se termine en un graphe sur lequel aucune règle n'est applicable ?

Confluence La nature non-déterministe de la réécriture fait qu'on peut appliquer plusieurs règles au même objet ou plusieurs fois la même règle sur des sous-graphes différents, obtenant ainsi des résultats différents.

> Pour un système de réécriture donné, établir si un tel système est *confluent* revient à vérifier la propriété du diamant : étant donné deux graphes G_1, G_2 obtenus par réécriture (application d'une règle R) d'un même graphe G, existe t-il des suites de réécriture $G_1 \rightarrow \ldots \rightarrow H$ et $G_2 \rightarrow \ldots \rightarrow H$ transformant G_1 et G_2 en un même graphe H ?

Un système de réécriture qui vérifie à la fois la propriété de terminaison et la propriété de confluence est dit *convergent*, ouvrant la voie à la définition de forme normale (pour la classe de graphes – de mots ou de termes – considérée). D'autres questions pourraient encore être mentionnées mais nous éloigneraient de l'objet de notre article.

C'est bien de la visualisation d'un tel système que nous nous préoccupons. Une plate-forme d'étude de systèmes de réécriture basée sur la visualisation doit notamment aider l'expert à développer une intuition sur le système concernant les questions de convergence et de confluence. En ciblant la réécriture sur telle ou telle règle, on pourra peut-être arriver à comprendre la combinatoire sous-jacente et ainsi voir poindre le ferment d'une preuve. Par exemple, pour implémenter une politique de sécurité dans un système de communications, en visualisant l'effet des réécritures, un utilisateur pourra valider la cohérence des règles régissant le comportement global du système.

Cette article présente les premiers travaux de conception et développement de la plate-forme visuelle de réécriture PORGY [1]. La section 2 introduit les problèmes liés à la réécriture de graphes en terme de visualisation et manipulation de graphes dynamiques. La section 3 présente les différents composants visuels de la plate-forme et la section 4 leur implémentation. La section 5 présente quelques exemples où la visualisation permet de résoudre facilement des problèmes formellement difficiles pour un expert en réécriture. Enfin, nous concluons cet article dans la section 6 par des perspectives de recherche et développement.

2 Réécriture et systèmes complexes

La visualisation de graphes dynamiques a encore été peu étudiée et s'est pour l'essentiel penchée sur le cas de graphes dont la topologie évolue sur un intervalle de temps (discret) donné. Le travail de Frishman et Tal (2008) gère le cas de graphes dont on connaît à l'avance les évolutions, et propose un dessin (dynamique) basé sur un calcul de positions contraintes pour les sommets à l'aide d'un modèle masses-ressorts adaptés. Loubier et Dousset (2007) propose une approche où le temps, associé aux quadrans de l'espace de dessin, vient contraindre la position des sommets dans le dessin du graphe.

Le cas que nous considérons ici est à notre connaissance nouveau et à plusieurs titres :

- La dynamique des graphes ne tient pas au passage du temps, et qui plus est, à une suite de modifications complètement connues à l'avance. C'est l'application de règles sur des sous-graphes qui régit les modifications possibles que peuvent connaître les graphes.
- La visualisation du système requiert non seulement de pouvoir visualiser le graphe à un moment donné et suivre ses évolutions, mais elle exige de pouvoir visualiser l'ensemble des réécritures possibles vers lesquelles peuvent évoluer un graphe en lui appliquant les suites autorisées de règles – si on s'intéresse, par exemple, plus particulièrement à la confluence des règles.

1. Travaux effectués dans le cadre du projet d'équipe associée INRIA PORGY, `http://gravite.labri.fr/?Projects_%2F_Grants:Porgy`.

- L'environnement de visualisation doit offrir une vue sur l'historique des réécritures.
- Ces opérations débordent hors du cadre strict du dessin de graphes et exigent de mobiliser les techniques et algorithmes de recherche de motifs sur les graphes.
- Enfin, l'environnement que nous nous proposons de développer exige la mise en place d'une structure de données dédiée, puissante et versatile pour se plier aux exigences énumérées ci-dessus.

On comprend bien pourquoi la visualisation de réécriture de graphes pose ces questions. Prenons le cas du système de réécriture modélisant les règles d'interactions de polypeptides au gré des réactions biochimiques (Fig. 1). S'il peut être intéressant de rejouer les différentes phases d'assemblages, les questions auxquelles la visualisation apporte une aide précieuse sont toutes autres. Il s'agit de permettre à l'expert de comprendre si les sites actifs (ceux qui sont ouverts à un assemblage) se situent à des endroits privilégiés, et à quel moment au cours du processus ? Y-a-t'il une ou plusieurs logiques d'assemblage – auquel cas il y aurait confluence – et si oui les temps de calculs sont-ils comparables ? Enfin, nombre de questions sur le processus biologique se traduisent en autant de questions sur la réécriture, que la simulation et la visualisation doivent pouvoir éclairer.

3 Visualisation des réécritures et dessin de graphes

Le système de visualisation que l'on se propose de construire (voir la figure 2 pour une vue d'ensemble) doit permettre de visualiser simultanément plusieurs graphes rendant compte du système de réécriture. Convenons d'appeler *modèle* le graphe qui sera réécrit, que l'on notera m_0 à son état initial. Ce modèle sera réécrit en une suite de graphes modèles m_1, puis m_2, *etc*. On voit ainsi apparaître le besoin de maintenir un environnement de visualisation composant des vues sur chacun des composants du système de réécriture.

Les règles de réécriture sont décrites à l'aide de graphes et doivent être visualisées pour aider l'utilisateur à raisonner sur le système, ou encore pour lui permettre de sélectionner une règle à appliquer. Elles impliquent deux graphes, un membre gauche et un membre droit, mis en correspondance par une partie médiane. C'est d'ailleurs cette distinction gauche/droite qui permet de représenter la règle comme un seul et unique graphe. La mise en correspondance des sommets et des arêtes qui doivent être préservés au moment de la réécriture ajoute à cette paire de graphes des arêtes additionnelles qui sont connectées par l'intermédiaire d'une structure médiane appelée "bridge". Le dessin des règles est donc bien un problème de dessin de graphes au sens strict. La lecture de la correspondance gauche/droite peut alors être vue comme un problème de dessin de graphes où l'objectif est d'assurer une lisibilité en limitant par exemple le nombre de croisement d'arêtes (contrairement à la règle représentée sur la figure 2). On a donc un problème de dessin de graphes relativement classique. A la minimisation du nombre de croisements s'ajoute le routage des arêtes pour éviter qu'elles ne chevauchent des sommets (Gansner et al., 1993; Freivalds, 2001). Pour le moment, nous utilisons simplement une version modifiée de l'algorithme "Bubble Tree" (Auber et al., 2004) étendu aux graphes généraux pour dessiner les règles ainsi que les modèles.

Parce que les règles peuvent être appliquées en parallèle, ou parce que plusieurs règles peuvent être appliquées à un même modèle m_i, ou simplement pour connaître l'historique des réécritures, on construit un *arbre de dérivations* rendant compte de toutes les traces des calculs de réécriture. Une suite de règles appliquées à un premier modèle m_0 se lit alors le long

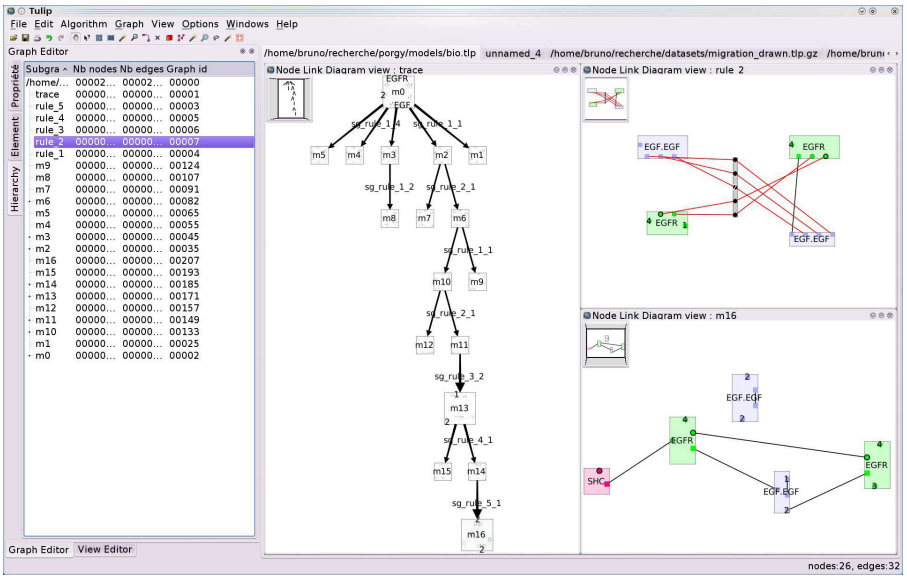
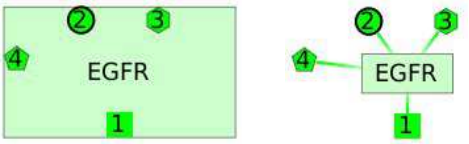


FIG. 2 – *Une vue d'ensemble de la plate-forme de réécriture sous Tulip utilisant l'exemple de la figure 1. Plusieurs réécritures ont déjà été effectuées. Le graphe en bas à droite (m16) est un état du modèle. L'arbre à sa gauche est l'arbre de dérivations (ou* `trace` *en anglais) qui permet de se souvenir de l'ensemble des opérations effectuées. Le graphe situé en haut à droite est une représentation de la règle r2.*

d'un chemin depuis la racine de l'arbre de dérivations vers une feuille. Ainsi, il nous faut proposer un algorithme permettant de visualiser cet arbre au fil de son évolution. Nous utilisons pour l'instant un algorithme de dessin classique (Buchheim et al., 2002). Selon le système de réécriture étudié, on peut s'attendre à devoir manipuler des arbres de grande taille, exigeant vraisemblablement l'utilisation d'artifices visuels et la gestion semi-automatique des repliements/déploiements de sous-arbres (Herman et al., 1998; Lee et al., 2006). Afin de permettre à l'utilisateur de suivre le calcul directement en visualisant l'arbre de dérivations, les modèles sont placés et dessinés directement dans les sommets de l'arbre – sans exclure la possibilité de les visualiser dans une fenêtre annexe. Suivant le paradigme des "small multiples" (Tufte, 1990), ces dessins répétés de modèles peuvent faciliter leur comparaison. Le dessin des modèles doit de plus se plier à des contraintes de lisibilité : le modèle m' obtenu d'un modèle antérieur m par réécriture doit être dessiné de manière à préserver la carte mentale de l'utilisateur (Misue et al., 1995; Saffrey et Purchase, 2008), et pourquoi pas en faisant ressortir les régions du graphe qui ont été l'objet de la réécriture (Fig. 7).

4 Visualisation, réécriture et architecture logicielle

Conjointement au travail centré sur le dessin pour la représentation et la visualisation des différents graphes manipulés, nous avons conçu un premier prototype du système dont les



(a) Un portnode tel qu'il est représenté à l'écran.

(b) La structure de ce portnode.

FIG. 3 – *Description de la structure interne d'un portnode et sa visualisation.*

grandes lignes sont esquissées ci-dessous. Nous utilisons pour cela la plate-forme Tulip (Auber, 2003). Le développement d'une application sous Tulip nécessite le développement de modules additionnels. Chaque module gère un aspect particulier du problème. Néanmoins, Tulip n'est pas conçu pour gérer les graphes dynamiques, la réécriture et les différentes opérations qui en découlent. Nous avons donc dû mettre au point une organisation interne spécifique des données pour remédier à cela. De plus, pour avoir un système qui soit le plus générique possible, nous utilisons un modèle de graphe particulier appelé "portgraphes".

4.1 Les portgraphes

Un portgraphe est un graphe composé de sommets appelés des "portnodes". Chaque portnode possède des ports et une arête relie deux portnodes uniquement par l'intermédiaire des ports (Figure 2, le modèle courant situé en bas à droite). Ce formalisme est inspiré des complexes moléculaires formés dans les réseaux d'interactions protéines-protéines dans un réseau biochimique : une protéine est caractérisée par un ensemble de petites zones à sa surface, appelée sites, et deux protéines ne peuvent interagir que par l'intermédiaire de ces sites si leur état est compatible (Andrei, 2008). Une protéine et ses sites sont alors modélisés par un sommet avec des ports et une liaison entre deux protéines est caractérisée par une arête entre deux ports. Ce modèle s'avère suffisamment générique pour gérer de nombreux autres exemples notamment en chimie (les ports représentent les liaisons entre deux atomes), en réseaux (les ports représentent directement les ports de communications des serveurs) ou bien encore dans les réseaux d'interactions (Lafont, 1990) (les ports représentent l'interface des sommets).

D'un point de vue pratique sous Tulip, un portnode est un sommet classique, appelé centre, sur lequel les ports, qui sont aussi des sommets classiques, sont connectés chacun par une arête. Chaque port est caractérisé par un état. Ainsi, les informations concernant le portnode (notamment son nom) sont conservées par le centre et les ports conservent chacun leur état et leurs connexions avec les portnodes voisins. En jouant sur la taille du centre et des effets de transparence fournis par Tulip, on peut effectuer un rendu visuel qui permet de masquer à l'utilisateur la structure sous-jacente (figure 3).

4.2 Modélisation de la plate-forme sous Tulip

Une étape de réécriture est une application d'une règle R donnée sur un modèle m. Cette réécriture s'effectue en deux étapes, chacune réalisée par un module additionnel pour Tulip.

On commence par chercher les instances du membre gauche R_l de R dans m puis une instance donnée de R_l est remplacée par une instance du membre droit R_r.

La première étape est réalisée grâce à une implémentation d'un algorithme de recherche d'isomorphismes de graphe dans un sous-graphe librement inspirée des travaux de Ullmann (1976) et Cordella et al. (2004). Ces algorithmes ont notamment été adaptés pour tenir compte de l'utilisation du modèle des portgraphes et des contraintes de la réécriture. Pour préparer l'étape suivante chaque instance de R_l est sauvegardée en tant que sous-graphe de m. Les port-nodes qui composent les sous-graphes doivent être ordonnés. Dans l'exemple de la figure 1, l'algorithme trouve pour la règle $r1$ 12 instances du membre gauche. Nous conservons aussi dans chaque sous-graphe ainsi créé les informations nécessaires pour établir une correspondance avec les sommets de R_l.

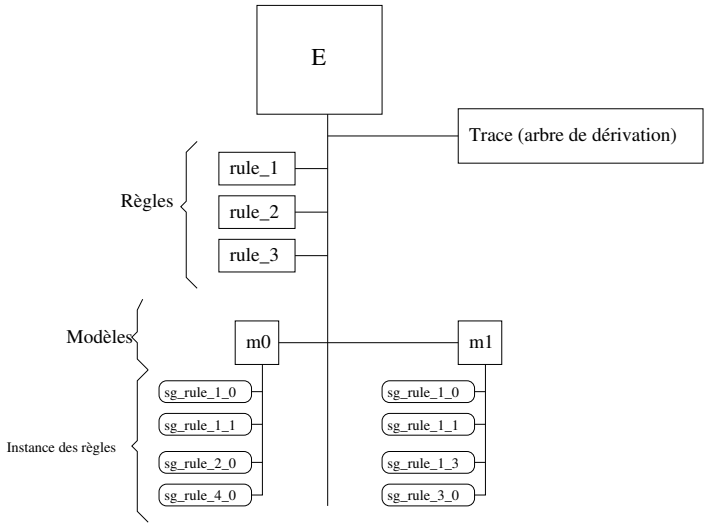


FIG. 4 – *Hiérarchie de sous-graphes Tulip nécessaire pour gérer les aspects dynamiques et la conservation de l'historique de la réécriture. Le graphe racine, appelé E, est imposé par Tulip. Chaque règle (rule_1, rule_2 et rule_3) et les différents modèles ($m0$ et $m1$) calculés sont décrits dans des sous-graphes de E. L'historique des réécritures est conservé dans un dernier sous-graphe (Trace). Les sous-graphes associés aux différents modèles (sg_rule_1_0, sg_rule_1_1, ...) représentent une instanciation d'un membre gauche d'une règle. L'analyse du nom de ces sous-graphes nous permet de reconstruire aisément les différentes actions effectuées par l'utilisateur. Par exemple, le sous-graphe sg_rule_1_1 associé à $m1$ représente la première instance possible (le dernier chiffre 1) de la règle rule_1.*

Ensuite, dans une deuxième étape, l'utilisateur choisit une instance précédemment créée (sous-graphe SG de m) de R_l dans m. Un clone $m1$ de m est réalisé. Le contenu de SG est supprimé de $m1$ et remplacé par l'instance correspondante de R_r dans m_1. Enfin, l'état des ports et des arêtes qui n'interviennent pas dans la règle est restauré. La complexité de ces différentes opérations et la structure interne de Tulip nous ont conduit à utiliser la hiérarchie de sous-graphes présentée dans la figure 4.

Pour terminer cette étape de réécriture, un sommet qui représente $m1$ est alors ajouté à l'arbre de dérivations. Ce nouveau sommet est relié par une arête au sommet qui représente le modèle qui a permis de le créer. Afin de pouvoir reconstruire plus tard la dynamique et analyser le système, nous conservons sur l'arête l'identifiant du sous-graphe SG. Ainsi, en effectuant un simple parcours de cet arbre, nous pouvons facilement retrouver les enchaînements d'opérations effectuées par l'utilisateur. Les sommets de l'arbre de dérivations sont appelés dans Tulip des "méta-sommets". Ce sont en fait des sommets qui englobent un sous-graphe. Un dessin du sous-graphe englobé reste visible à l'intérieur du méta-sommet (figure 5).

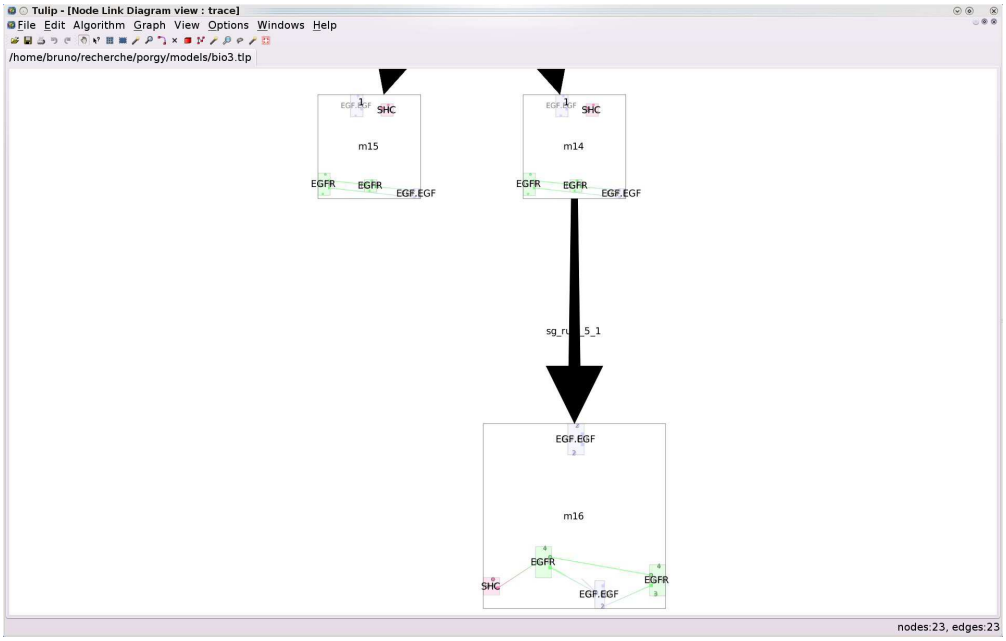


FIG. 5 – *Zoom sur une partie d'un arbre de dérivations. Chaque sommet est un état du modèle. On voit dans les sommets un dessin du modèle concerné.*

5 Scénarios d'utilisation et interaction

La visualisation du système de réécriture assemble donc différentes fonctionnalités constituant un tableau de bord pour étudier le système et simuler les réécritures. La visualisation permet typiquement de répondre à des questions qui, sans elle, sont formellement difficiles. Par exemple, il est difficile de déterminer un (plus petit) ancêtre commun à deux graphes issus de réécriture [2], alors que la visualisation de l'arbre de dérivation rend cette question triviale.

Grâce à la visualisation, le suivi des réécritures repose bien souvent sur des tâches relativement simples. Par exemple, on peut, vouloir mettre en valeur une molécule sur tout ou

2. C'est là une question duale de la propriété de confluence qui pose la question de l'existence d'un successeur commun à deux graphes issus d'un même troisième.

partie de l'arbre de dérivation ou plus simplement chercher à savoir quand une molécule est modifiée, détruite ou créée. Ces opérations de base en réécriture nécessitent d'inspecter l'effet des transformations locales appliquées au graphe et d'étudier la combinatoire du système de réécriture. Une bonne gestion des structures internes de Tulip ainsi qu'un rendu visuel adapté donne à l'utilisateur un moyen rapide et facile pour trouver une réponse. Un sommet qui n'est pas modifié entre deux modèle consécutifs m et $m1$ est strictement identique dans les deux modèles (c'est le même objet d'un point de vue informatique). Un sommet qui est modifié (ou créé) par une réécriture est détruit dans m et un nouveau est créé dans $m1$. Le mécanisme de sélection de sommets propre à la plate-forme Tulip donne alors une solution immédiate à ce problème de suivi. Il suffit de sélectionner un portnode dans un modèle pour le voir apparaître en surbrillance dans les différents sommets de l'arbre de dérivations (figure 6).

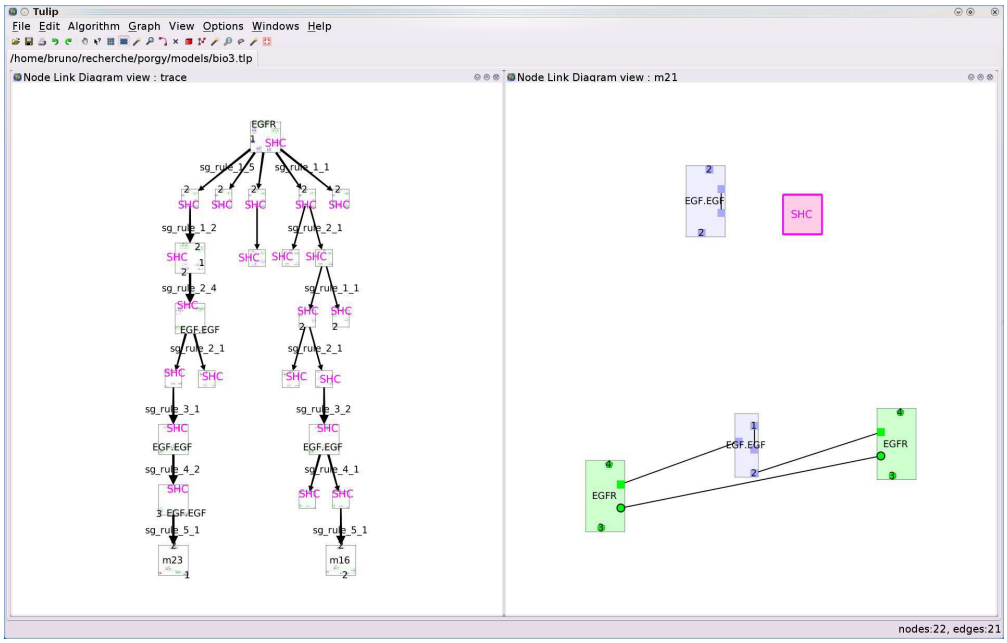


FIG. 6 – *La bio-molécule nommée SHC est sélectionnée dans le modèle courant (graphe de droite). Le sommet associé est automatiquement sélectionné par Tulip dans tous les autres modèles où il existe. Il n'est pas sélectionné dans* $m23$ *et* $m16$*. La molécule a donc subi une réécriture lors de la création de ces modèles.*

De manière équivalente, l'utilisateur peut chercher à visualiser les parties du modèle qui sont concernées par la réécriture par exemple dans le cadre d'une application de règles en parallèle. Selon les applications, il est bien souvent important qu'il n'y ait pas de chevauchement dans les instanciations des membres gauches des règles dans le modèle. La figure 7 est un exemple d'une telle fonctionnalité.

D'autres indices visuels spécifiques au système de réécriture (ou selon le domaine d'applications visé) peuvent être ajoutés à ce tableau de bord. Dans le cas de la réécriture de graphes modélisant les interactions entre bio-molécules, on pourrait être intéressé à suivre les concen-

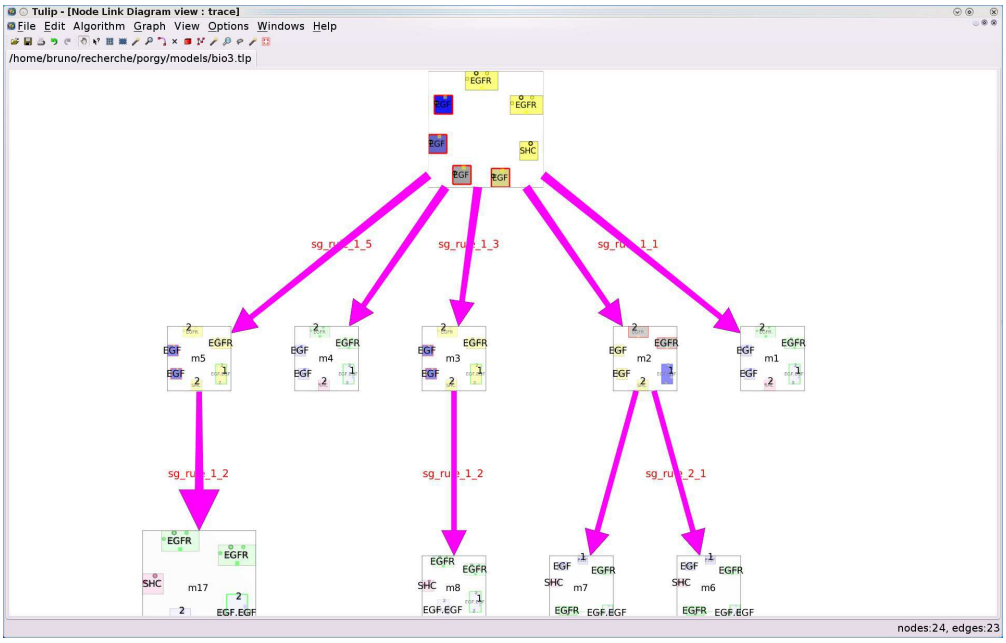


FIG. 7 – *Visualisation des instanciations des membres droits et gauches des règles dans les modèles pour une sélection d'arêtes (ici, toutes). Les couleurs des bordures autour des portnodes indiquent comment le portnode participe à une règle (rouge pour le membre gauche, vert pour le membre droit et bleu si le portnode est dans les deux membres). Les couleurs des portnodes indiquent le nombre de fois où ils ont été reconnus comme faisant partie d'une instance d'un membre gauche. Par exemple, les portnodes de la racine de l'arbre sont identifiés avec 5 couleurs différentes. Donc, certains sont utilisés plusieurs fois pour créer* $m1$ *à* $m5$*.*

trations des diverses bio-molécules (chacune correspondant à un certain type de sommet) au fur et à mesure des réécritures. On peut donc envisager de synchroniser la visualisation avec le calcul d'histogrammes de fréquences (comme fenêtre de visualisation annexe ou directement embarqué dans la visualisation de l'arbre de dérivation reprenant encore une fois le paradigme des "small multiples" (Tufte, 1990), par exemple).

6 Conclusion et perspectives

Le système que nous décrivons revêt un caractère ambitieux tant du point de vue des concepts qu'il met en jeu, de sa mise en œuvre ou de l'impact sur les utilisateurs qu'il s'agisse des utilisateurs finaux comme les biologistes ou des communautés de recherche concernées en informatique. Son architecture est d'ores et déjà posée et s'appuie sur la plate-forme Tulip.

De façon générale, les règles de réécriture sont rarement appliquées de façon unitaire par l'utilisateur. L'application des règles est pilotée par une stratégie (qui peut être vue comme un programme) décrite dans un langage adapté (Kirchner et al., 2008). La prochaine étape

majeure du développement de notre système est d'intégrer un tel mécanisme. La stratégie est décrite par un arbre qu'il faudra visualiser et synchroniser avec l'arbre de dérivations. Par exemple, une stratégie composnant en séquence plusieurs règles peut se traduire par une arête qui effectue un raccourci dans l'arbre de dérivation entre deux modèles non consécutifs. Même si l'utilisateur n'est bien souvent pas intéressé par les différents états intermédiaires, ceux-ci peuvent en revanche servir à guider un algorithme de visualisation dynamique de l'évolution du modèle.

Le dessin des modèles et des règles est pour l'instant effectué à l'aide d'adaptations simples d'algorithmes généraux qui n'assurent pas les transitions d'un modèle à ses successeurs de manière optimale (section 3). Aussi, ces algorithmes doivent-ils évoluer. Les premières manipulations du système de réécriture exploitent des mécanismes propres à Tulip ; d'autres interactions nécessiteront des efforts supplémentaires et sont en cours de développement.

Le volet de ces travaux concernant la simulation est une voie qu'il reste encore à explorer. En effet, outre les questions relatives au dessin, cette question met en jeu l'interaction et la visualisation d'indices donnant un retour visuel sur le déroulement de la simulation.

Références

Andrei, O. (2008). *A Rewriting Calculus for Graphs: Applications to Biology and Autonomous Systems*. Thèse de doctorat, Institut national polytechnique de Lorraine.

Andrei, O. et H. Kirchner (2009). A port graph calculus for autonomic computing and invariant verification. In A. Corradini (Ed.), *Proc. of the* 5^{th} *Int. Work. on Computing with Terms and Graphs 2009 (TERMGRAPH 2009)*. Elec. Notes in Theoretical Computer Science, Elsevier.

Auber, D. (2003). Tulip – A huge graph visualization framework. In P. Mutzel et M. Jünger (Eds.), *Graph Drawing Software*, Mathematics and Visualization Series, pp. 105–126. Springer Verlag.

Auber, D., S. Grivet, J.-P. Domenger, et G. Melançon (2004). Bubble tree drawing algorithm. In *Int. Conf. on Computer Vision and Graphics*, pp. 633–641.

Buchheim, C., M. Jünger, et S. Leipert (2002). Improving Walker's algorithm to run in linear time. In *Graph Drawing,* 10^{th} *Int. Symp., GD 2002, revised papers*, Volume 2528 of *LNCS*, pp. 347–364. Springer.

Cordella, L. P., P. Foggia, C. Sansone, et M. Vento (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. on Pattern Analysis and Machine Intelligence 26*(10), 1367–1372.

Freivalds, K. (2001). Curved edge routing. In G. Goos, J. Hartmanis, et J. van Leeuwen (Eds.), 13^{th} *Int. Symp. on Fundamentals of Computation Theory*, Volume 2138 of *LNCS*, pp. 126–137. Springer.

Frishman, Y. et A. Tal (2008). Online dynamic graph drawing. *IEEE Trans. on Visualization and Computer Graphics 14*(4), 727–740.

Gansner, E. R., E. Koutsofios, S. C. North, et K.-P. Vo (1993). A technique for drawing directed graphs. *IEEE Trans. on Software Engineering 19*(3), 214–230.

Herman, I., G. Melançon, et M. Delest (1998). Tree visualisation and navigation clues for information visualisation. *Computer Graphics Forum 17*(2), 153–165.

Kephart, J. O. et D. M. Chess (2003). The vision of autonomic computing. *IEEE Computer 36*(1), 41–50.

Kirchner, C., F. Kirchner, et H. Kirchner (2008). Strategic computations and deductions. In C. Benzmüller, C. E. Brown, J. Siekmann, et R. Statman (Eds.), *Reasoning in Simple Type Theory.*, Volume 17 of *Studies in Logic and the Foundations of Mathematics*, pp. 339–364. College Publications.

Lafont, Y. (1990). Interaction nets. In *Proc. of the* 17^{th} *ACM Symp. on Principles of Programming Languages*, pp. 95–108. ACM Press.

Lee, B., C. S. Parr, C. Plaisant, B. B. Bederson, V. D. Veklser, W. D. Gray, et C. Kotfila (2006). Visualizing Networks with Enhanced Tree Layouts: Can Interaction Tame Complexity? *IEEE Trans. on Visualization and Computer Graphics, Special Issue on Visual Analytics 12*(6), 1414–1426.

Loubier, E. et B. Dousset (2007). Visualisation and analysis of relational data by considering temporal dimension. In *ICEIS 2007* 9^{th} *Int. Conf. on Enterprise Information Systems*, pp. 550–553.

Misue, K., P. Eades, W. Lai, et K. Sugiyama (1995). Layout adjustment and the mental map. *J. of Visual Languages and Computing 6*, 183–210.

Saffrey, P. et H. Purchase (2008). The "mental map" versus "static aesthetic" compromise in dynamic graphs: a user study. In *Proc. of the* 9^{th} *Conf. on Australasian user interface*, pp. 85–94. Australian Computer Society, Inc.

Terese (2000). *Term Rewriting Systems*, Volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Tufte, E. R. (1990). *Envisioning Information*. Graphics Press (8^{th} printing, 2001).

Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *J. of the Association for Computing Machinery 23*(1), 31–42.

Summary

This paper presents the first results aiming at the development of an interactive visualization environment for graph rewriting systems based on the Tulip framework. That is, the system aims at dealing with graphs whose topologies evolve in time according to rewriting rules. As a consequence, our system must not only allow us to visualize the graphs, but must also offer a complete environment to study the rewriting system in itself. In such a setting, the visualization supports the study of the rewriting system, helping the identification of properties such as its confluence or termination, deadlock situations, frequent or rare instances, *etc*. On top of all visualization issues it raises, the system also looks at pattern matching and computing history issues. We also look at issues related to the internal representation of the system required for its visualization.