FL-system's Intelligent Cache

Gurvan Le Guernic, Julien Perret

FL-system's Intelligent Cache

Gurvan Le Guernic, Julien Perret
IRISA, Campus universitaire de Beaulieu,
Avenue du Général Leclerc, 35042 Rennes Cedex - France
{Gurvan.Le_Guernic, Julien.Perret}@irisa.fr

Résumé : Cet article pr´esente une application de techniques issues du g´enie logiciel à la mod´elisation d'environnements virtuels. Ces derniers, dès lors qu'ils sont complexes, constituent des masses de donn´ees particulièrement volumineuses et, de ce fait, difficiles à manipuler. De plus, les descriptions exhaustives de ces environnements, c'est-à-dire explicites, sont peu ´evolutives. Pour r´esoudre ces problèmes, il existe des m´ethodes bas´ees sur des systèmes de r´e´ecriture permettant de d´ecrire de façon g´en´erique les objets de l'environnement. Chaque objet est ainsi d´ecrit par un axiome qui est r´e´ecrit lors de la g´en´eration de l'environnement. Lors de cette phase, il est fr´equent de retrouver deux s´equences de r´e´ecriture aboutissant à un même objet. Il est alors possible d'utiliser un m´ecanisme tel que le cache afin d'am´eliorer les performances en profitant des calculs d´ejà effectu´es.
Cependant, se contenter de mettre en correspondance les noms et paramètres des axiomes n'est pas suffisant pour ´etablir l'ensemble des r´e´ecritures identiques. En effet, les r´e´ecritures d'un même axiome avec des paramètres effectifs diff´erents peuvent aboutir au même objet. Le système propos´e dans cet article effectue une analyse dynamique des r´e´ecritures afin d'am´eliorer les performances du cache en d´etectant de tels cas.
La première partie de cet article pr´esente le système de r´e´ecriture, la seconde l'analyse utilis´ee, et la dernière le m´ecanisme propos´e.

Mots-clés : R´ealit´e virtuelle - Systèmes de r´e´ecriture - 3D - Cache 3D - Calcul de d´ependances - Mod´elisation et informatique fondamentale

1 INTRODUCTION

La gestion d'environnements virtuels 3D de grande taille comporte des contraintes propres. Elle n´ecessite le stockage de la description d'un grand nombre d'´el´ements montrant de fortes similitudes structurelles et apparaissant à l'identique ou presque à plusieurs endroits de la scène (on pourra par exemple penser à une rang´ee de bancs ou de lampadaires). La description exhaustive d'un environnement urbain dans un langage 3D (*e.g.* VRML) fige le modèle 3D dans un ´etat unique (ou du moins difficilement maintenable) et n´ecessite une grande capacit´e de stockage. Une solution consiste à utiliser le m´ecanisme de r´e´ecriture appel´e *FL-system* [Marvie, 2003, Marvie, 2005]. Ce système, bas´e sur les *L-systems* [Lindenmayer, 1968, Smith, 1984] et les *grammaires de Chomsky* [Prusinkiewicz, 2001], permet de d´ecrire la structure g´en´erale des principaux types d'´el´ements rencontr´es dans l'environnement urbain tels que bâtiments et arbres. La r´e´ecriture d'un axiome initial à l'aide de la grammaire associ´ee aux ´el´ements urbains d'un type donn´e permet la g´en´eration automatique d'une entit´e urbaine de ce type. Cela permet une description compacte et ´evolutive d'environnements urbains 3D de grande taille. Il est ainsi possible de construire de nombreuse entit´es diff´erentes d'un même type d'objet en ex´ecutant plusieurs r´e´ecritures de l'axiome initial avec des paramètres diff´erents.

L'inconv´enient majeur d'une telle m´ethode est le coût, en temps de calcul, impliqu´e par la phase de r´e´ecriture. Cependant, le surcoût impos´e par cette phase peut être r´eduit. Dans le cas de la g´en´eration d'une ville de grande taille, il est probable qu'un même bâtiment[1] apparaisse deux fois en des lieux diff´erents de la scène. Il est encore plus probable d'avoir de multiples occurrences d'une même fenêtre. Il serait donc int´eressant de pouvoir r´ecup´erer le r´esultat de r´e´ecritures d´ejà effectu´ees. Il est bien sur possible d'ajuster la grammaire pour qu'elle r´eutilise des ´el´ements 3D. N´eanmoins, cette m´ethode pr´esente l'inconv´enient de compliquer la tache du r´edacteur des grammaires et de diminuer la g´en´ericit´e structurelle des règles de r´e´ecriture. Une autre solution consiste à utiliser un cache. Chaque fois qu'un axiome à r´e´ecrire correspond à une clef du cache, l'objet 3D correspondant à cette clef est directement r´eutilis´e.

Dû à la nature param´etrique des axiomes du *FL-system*, un système simple de reconnaissance des clefs du cache se r´evèle insuffisant. Par exemple, un axiome g´en´erateur de bâtiments simples (bat) peut être d´efini à l'aide de 3 paramètres : d pour la distance au point d'observation (valant 1 pour un bâtiment très ´eloign´e), n pour le nombre d'´etages, et s pour le style du bâtiment. Pour n constant, il est probable que cet axiome g´enère le même bâtiment quel que soit s lorsque d vaut 1. Ainsi $\mathtt{bat}(1, 8, s_1)$ et $\mathtt{bat}(1, 8, s_2)$ aboutissent au même objet 3D. Seulement, il n'est pas ´evident pour un cache naïf que $\mathtt{bat}(1, 8, s_1)$ et $\mathtt{bat}(1, 8, s_2)$ doivent être consid´er´es comme ´equivalents.

[1] Nous consid´erons ici deux bˆatiments comme identiques si leurs caract´eristiques sont suffi samment proches pour pouvoir utiliser, pour les deux, un mˆeme repr´esentant, *i.e.* un mˆeme mod`ele g´eom´etrique.

La m´ethode que nous proposons utilise des notions et m´ecanismes similaires à ceux que l'on rencontre dans l'analyse de d´ependances entre les entr´ees et les sorties d'un programme [Pugh, 1989, Abadi, 1996]. Les travaux sur les d´ependances sont très nombreux dans le domaine de la s´ecurit´e informatique, et de nombreux autres problèmes du g´enie logiciel ont de forts liens avec l'analyse de d´ependance [Abadi, 1999]. Dans notre cas, ce type d'analyse permet par exemple de d´etecter que lorsque d est ´egale à 1 le r´esultat final de la r´e´ecriture de l'axiome bat est d´ependant uniquement de ses deux premiers paramètres. Ainsi, le système de cache peut d´etecter automatiquement qu'il doit consid´erer $\mathtt{bat}(1, 8, s_1)$ et $\mathtt{bat}(1, 8, s_2)$ comme ´equivalents.

Ce document pr´esente tout d'abord une version simplifi´ee des *FL-systems*. En section 3, nous introduisons un calcul dynamique des d´ependances appliqu´e aux *FL-systems* permettant d'identifier les paramètres intervenant dans la r´e´ecriture d'un axiome. La section suivante pr´esente l'application de ce calcul à la r´ealisation d'un cache *intelligent* pour la phase de r´e´ecriture des *FL-systems*. Enfin, nous concluons en section 5.

2 FL-SYSTEM

Les *FL-systems* permettent de mod´eliser toutes sortes d'objets g´eom´etriques et, de pr´ef´erence, des objets qui pr´esentent une structure fortement redondante, tels des plantes ou des bâtiments. En effet, bas´es sur une description grammaticale des objets, les *FL-systems* permettent d'utiliser à bon escient la redondance de l'environnement mod´elis´e. Cette r´ep´etition nous permet des descriptions très courtes et g´en´eriques de classes d'objets. Une classe d'objets est caract´eris´ee par un ensemble de paramètres. Une instance d'objet est alors obtenue par la d´efinition d'un ensemble de valeurs associ´ees à ces paramètres.
N´eanmoins, les valeurs donn´ees à l'axiome d'un *FL-system* n'ont pas n´ecessairement d'influence sur toutes les parties de l'objet g´en´er´e. Il est en effet courant qu'une même partie d'un objet se retrouve à l'identique dans un autre objet de la même classe, malgr´e des diff´erences importantes dans les valeurs de leurs paramètres (on pourra ici penser aux feuilles de deux arbres de la même espèce). Dans cette section, nous proposons une grammaire et une s´emantique pour les règles de r´e´ecriture des *FL-systems*.

2.1 Le paradigme d'instanciation d'objets géométriques

En informatique graphique, il est courant de vouloir r´eutiliser des objets g´eom´etriques d´ejà d´efinis ailleurs, d'une part pour r´eduire l'espace de stockage, mais aussi le coût d'affichage de ces modèles lors de la navigation. Pour ce faire, Hart [Hart, 1992] propose le paradigme d'*instanciation d'objets géométriques* dans le cadre de la mod´elisation d'objets fractals. Ainsi, à partir de règles simples, il devient possible de r´eutiliser le même arbre pour d´efinir une forêt en comportant des centaines, chaque arbre pouvant lui-même utiliser plusieurs fois la même branche et ainsi de suite. Ce paradigme, aujourd'hui int´egr´e à la plupart des langages de mod´elisation g´eom´etrique (*e.g.* VRML[Marrin, 1997]), am´eliore très sensiblement les performances des applications 3D.
Dans le cadre de la mod´elisation d'environnements naturels, Deussen [Deussen, 1998] propose l'*instanciation approximative* comme une extension du paradigme d'instanciation de Hart. Les objets mod´elis´es proc´eduralement sont projet´es dans l'espace des paramètres de l'axiome et ensuite regroup´es par *clusters*. Chaque *cluster* repr´esente alors une instance approximative de tous les objets qui en font partie, r´eduisant ainsi le nombre d'objets que contient la scène.
Les *FL-systems* permettent l'utilisation de nœuds VRML, et donc l'utilisation du paradigme d'instanciation. Nous proposons ici un m´ecanisme d'instanciation automatique, implicite, et transparent pour l'utilisateur. Ce m´ecanisme est bas´e sur le calcul dynamique des d´ependances dans les règles de r´e´ecriture (cf. section 3).

2.2 La grammaire utilisée

Comme le montre la figure 1, ces règles sont compos´ees d'une partie gauche ($Axiome$), d'une condition ($Cond$), et d'une partie droite ($Terme$). Nous consid´rerons ici que la strat´egie de r´e´ecriture est d´eterministe et r´ecursive à gauche, contrairement aux L-systems dont la r´e´ecriture est parallèle. Cette restriction a pour but de rendre d´eterministe chaque ´etape de la r´e´ecriture d'un axiome. Ceci simplifie la pr´esentation des travaux d´evelopp´es dans cet article, en particulier les th´eorèmes pr´esent´es en section 3. De plus, pour contrôler la profondeur de r´e´ecriture, nous utiliserons un paramètre donn´e à l'axiome. Ainsi, à partir d'un axiome donn´e, une r´e´ecriture engendrera un mot compos´e de terminaux. Les terminaux, dans les *FL-systems*, sont des fonctions et, plus g´en´eralement, des appels à des primitives g´eom´etriques quelconques (VRML, OpenGL ou autres). Pour plus de d´etails sur les strat´egies de r´e´ecriture pouvant être appliqu´ees sur les *FL-systems*, ainsi que sur la nature des mots g´en´er´es, voir [Marvie, 2005].

La figure 2 donne un exemple de *FL-system*. L'axiome initialise les paramètres n, δ, l, w et $color$, qui repr´esentent respectivement la profondeur de r´e´criture[2], un angle de rotation, la longueur d'un segment de branche, le rayon d'une branche et la couleur des feuilles. Les terminaux rotateX et rotateZ tournent le repère local autour des axes X et Z respectivement. Le terminal moveZ d´eplace le repère local le long de l'axe Z, tandis que les crochets [et] empilent et d´epilent le repère local, permettant la cr´eation facile de sous-branches. Hormis pour l'axiome **L**, il existe 2 règles de r´e´ecriture pour tous les axiomes. La première repr´esente la règle de d´ecomposition de l'axiome à proprement parler, l'autre nous assure que lorsque la profondeur de r´e´ecriture est atteinte, l'axiome est remplac´e par un terminal.

[2] La profondeur de r´e´ecriture d'un *FL-system* est son ˆage.

$\omega : true \rightarrow \mathbf{A}(7, 22.5, 10, 1, green)$
$\mathbf{A}(n, \delta, l, w, color) : n > 0 \rightarrow \mathbf{B}(n-1, \delta, l, w, color) \quad \mathsf{rotateZ}(\delta \times 5) \quad \mathbf{B}(n-1, \delta, l, w, color) \quad \mathsf{rotateZ}(\delta \times 5) \quad \mathbf{B}(n-1, \delta, l, w, color)$
$\mathbf{A}(n, \delta, l, w, color) : n = 0 \rightarrow \epsilon$
$\mathbf{B}(n, \delta, l, w, color) : n > 0 \rightarrow [\, \mathsf{rotateX}(\delta) \quad \mathbf{F}(n-1, \delta, l, w, color) \quad \mathbf{L}(n-1, \delta, l, w, color) \quad \mathbf{A}(n-1, \delta, l, w \times 0.707, color)]$
$\mathbf{B}(n, \delta, l, w, color) : n = 0 \rightarrow \epsilon$
$\mathbf{F}(n, \delta, l, w, color) : n > 0 \rightarrow \mathbf{F}(n-1, \delta, l, w, color) \quad \mathbf{L}(n-1, \delta, l, w, color) \quad \mathsf{rotateZ}(\delta \times 5) \quad \mathbf{F}(n-1, \delta, l, w, color)$
$\mathbf{F}(n, \delta, l, w, color) : n = 0 \rightarrow \mathsf{cylinder}(l, w) \quad \mathsf{moveZ}(l)$
$\mathbf{L}(n, \delta, l, w, color) : true \rightarrow [\, \mathsf{rotateX}(-\delta \times 2) \quad \mathsf{leaf}(l, color)\,]$

FIG. 2 – *FL-system* pour un buisson modifi´e à partir d'un *L-system* propos´e par Prusinkiewicz[Prusinkiewicz, 1990]

$Val = \mathbb{R}^+$

$$
\begin{array}{rcl}
Var & ::= & [a-z] \\
NAxiome & ::= & [A-Z][a-z]^\star \\
NTerminal & ::= & [a-z]^\star \\
V & ::= & Var \mid Val \\
Terminal & ::= & NTerminal\ (\ ExPar^\star\ ) \\
OpCond & ::= & = \mid > \\
Cond & ::= & V \quad OpCond \quad V \mid \mathit{true} \\
OpPar & ::= & + \mid - \mid x \\
ExPar & ::= & (\ ExPar \quad OpPar \quad ExPar\ ) \\
 & \mid & V \\
Axiome & ::= & NAxiome\ (\ ExPar^\star\ ) \\
Terme & ::= & Axiome \quad Terme \\
 & \mid & Terminal \quad Terme \mid \epsilon \\
RWrule & ::= & Axiome : Cond \rightarrow Terme
\end{array}
$$

FIG. 1 – Proposition de grammaire pour les règles de r´e´ecriture. *Ad´esigne une liste d'´el´ements de type A.

FIG. 3 – R´esultat de la r´e´ecriture du L-system de la figure 2

L'axiome **A** repr´esente un apex[3] à partir duquel sont cr´e´ees trois branches **B**. Les branches **B** se d´ecomposent en une arête **F**, une feuille **L** et un nouvel apex **A**. Chaque arête **F** est d´ecompos´ee en une arête, une feuille, et une autre arête, simulant ainsi le ph´enomène de croissance des branches. Lorsque la profondeur de r´e´ecriture est nulle, les arêtes **F** sont remplac´ees par des cylindres (dont l'axe de rotation est l'axe Z) grâce au terminal cylinder. Les feuilles **L** sont, quand à elles, remplac´ees par des objets g´eom´etriques leaf.

La figure 4 montre les diff´erences entre la strat´egie de r´e´ecriture parallèle des *L-systems* et celle propos´ee ici pour les *FL-systems*. N´eanmoins, en s'assurant que les termes situ´es en partie droite des règles d´ecr´ementent le paramètre n de l'axiome, le r´esultat de la r´e´ecriture complète du *FL-system* et du *L-system* ´equivalent après n d´erivations successives sont identiques. La figure 3 montre une interpr´etation g´eom´etrique de la r´e´ecriture de cet exemple.

2.3 La sémantique utilisée

Pour pouvoir utiliser un cache pour les *FL-systems*, nous avons besoin d'une s´election des règles de r´e´ecriture d´eterministe. Nous proposons ici une formalisation de la s´emantique utilis´ee pour la r´e´ecriture des *FL-systems*. Tout d'abord, nous d´efinissons $\mathcal{R}$ comme la liste (ordonn´ee) des règles de r´e´ecriture d'un *FL-system* donn´e. Soit $\mathbf{A}(v^*)$ un axiome tel que $v^* \in \mathbb{R}^*$. Nous avons alors la s´emantique donn´ee en figure 5.

$$
\frac{x \in \text{Terminaux}^\star \qquad \mathbf{A}(p^*) : c_* \wedge c \rightarrow \boldsymbol{T} \ = \ \mathtt{first}(\mathcal{R}, \mathbf{A}(v^*))}{x\ \mathbf{A}(v^*)\ y \quad \rightarrow_{\mathcal{R}} \quad x\ \mathtt{eval}(\boldsymbol{T}[v^*/p^*])\ y}
$$

FIG. 5 – S´emantique des règles de r´e´ecriture

Nous d´efinissons $\mathtt{first}(\mathcal{R}, \mathbf{A}(\vec{v}))$ la fonction qui retourne la règle de r´e´ecriture "$\mathbf{A}(\vec{p}) : c_* \wedge c \rightarrow \boldsymbol{T}$" où :
- "$\mathbf{A}(p^*) : c \rightarrow \boldsymbol{T}$" = r_i est la première règle de $\mathcal{R}$ s'appliquant (*i.e.* la condition "c" est vraie) à $\mathbf{A}(v^*)$
- c_* est la conjonction des n´egations des condition des règles pr´ec´edant r_i dans $\mathcal{R}$ et s'appliquant à un axiome de nom **A**

$X[a/b]$ retourne le r´esultat de la substitution dans X de toutes les occurrences de b par a. Ainsi,

[3] En biologie, apex est le nom donn´e au sommet d'un organe.

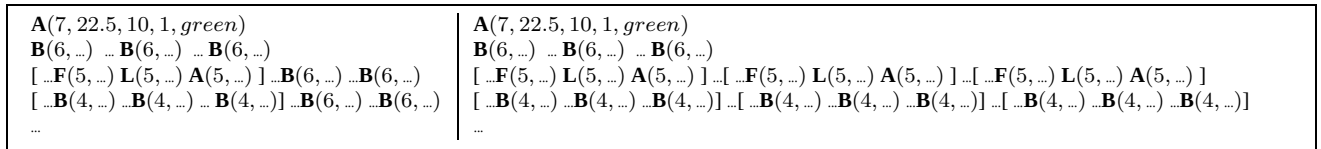

FIG. 4 – Comparaison des strat´egies de r´e´ecriture entre les *FL-systems* propos´es (à gauche) et les *L-systems* (à droite).

$(\mathbf{A}(x+3, y))[(1,0)/(x,y)]$ rend $\mathbf{A}(1+3, 0)$. La fonction $\texttt{eval}(\boldsymbol{T})$ parcourt le terme $\boldsymbol{T}$ et ´evalue les expressions y apparaissant pour obtenir les paramètres num´eriques de l'´etape de r´e´ecriture suivante. Par exemple, $\texttt{eval}(\mathbf{A}(1+3, 0))$ retourne $\mathbf{A}(4, 0)$.

Exemple 2.1: Un autre exemple de *FL-system*

$$\omega : true \rightarrow \mathbf{A}(1, 2)$$
$$\mathbf{A}(m, n) : m = 0 \rightarrow \epsilon$$
$$\mathbf{A}(m, n) : n > 0 \rightarrow \mathbf{A}(m, n-1)$$
$$\mathbf{A}(m, n) : n = 0 \rightarrow \epsilon$$

Étant donn´e le *FL-system* de l'exemple 2.1, l'axiome $\mathbf{A}(1, 2)$ se r´e´ecrit en $\mathbf{A}(1, 1)$ avec $\texttt{first}(\mathcal{R}, \mathbf{A}(1, 2))$ ´egal :

$$\mathbf{A}(m, n) : \neg(m = 0) \wedge (n > 0) \rightarrow \mathbf{A}(m, n-1)$$

Dans cet exemple, la condition $m \neq 0 \wedge n > 0$ donne la d´ependance vis-à-vis des variables m et n de la règle appliqu´ee. Si la n´egation de $m = 0$ n'´etait pas pr´esente, l'importance de la valeur de m ne serait pas explicite. Toujours dans ce même exemple, la fonction `eval` remplace le paramètre m par sa valeur (1), et ´evalue l'expression "$n - 1$" à 1.

Le paragraphe pr´ec´edent aborde la d´ependance qui existe entre la valeur des paramètres d'un axiome et le r´esultat final de sa r´e´ecriture. La section suivante propose d'aller plus loin dans l'´etude de cette d´ependance.

3 CALCUL DYNAMIQUE DES DÉPENDANCES

Le système de cache pr´esent´e dans cet article met à profit la propri´et´e de *joignabilit´e* de certains termes dans un système de r´e´ecriture [Dershowitz, 1990]. Si deux termes se r´e´ecrivent en un même terme sous forme normale (*i.e.* compos´e uniquement de terminaux) alors ces deux termes sont dit *joignables*. Dans cette section, nous exposons la m´ethode employ´ee afin de calculer dynamiquement un ensemble, aussi grand que possible, de termes joignables avec le terme en cours de r´e´ecriture. La m´ethode employ´ee s'inspire de travaux sur la confidentialit´e de programmes s´equentiels [Le Guernic, 2005]. Elle se base sur la modification de la s´emantique de r´e´ecriture afin de manipuler des *valeurs étiquetées*. La r´e´ecriture d'un axiome, param´etr´e par des *valeurs étiquetées*, retourne alors un terme sous forme normale, lui-même marqu´e par une ´etiquette. Cette dernière d´ecoule des ´etiquettes des paramètres de l'axiome qui ont influenc´e cette r´e´ecriture.

3.1 Adaptation de la grammaire

La simple modification, donn´ee en figure 6, de notre grammaire initiale (figure 1) permet d'int´egrer les valeurs ´etiquet´ees. $\mathcal{P}(\mathbb{L}abel)$ est l'ensemble des ensembles de $\mathbb{L}abel$. Les valeurs (Val) sont maintenant des paires not´ees "$e : n$". Le deuxième ´el´ement de cette paire (n), aussi appel´e valeur num´erique, appartient à $\mathbb{R}$ ("n" est la valeur utilis´ee dans la section 2). Le premier ´el´ement (e) est un ensemble de *labels* formant une *étiquette*. Chaque ´etiquette de l'axiome initial[4] est compos´ee d'un unique label. Chacun de ces labels n'est utilis´e qu'une seule fois (*cf.* exemple 3.1).

Exemple 3.1: Exemple d'´etiquetage initial
$\mathbf{A}(\{l1\} : 7, \{l2\} : 22.5, \{l3\} : 10, \{l4\} : 1, \{l5\} : \textit{green})$

$$\mathbb{E}tiquette = \mathcal{P}(\mathbb{L}abel)$$
$$Val = (\mathbb{E}tiquette \times \mathbb{R}^+)$$

FIG. 6 – Grammaire des valeurs ´etiquet´ees

3.2 Une nouvelle sémantique

La figure 7 pr´esente la nouvelle s´emantique utilis´ee pour la r´e´ecriture de termes dont les valeurs sont ´etiquet´ees. Cette s´emantique pr´esente de grandes similitudes avec celle pr´esent´ee dans la section 2. Il est facile de se convaincre que, mis à part les ´etiquettes, ces deux s´emantiques r´e´ecrivent un axiome de la même façon.

$$\frac{\begin{array}{c} x \in \text{Terminaux}^\star \\ \mathbf{A}(p^*) : c_* \wedge c \rightarrow \boldsymbol{T} \ = \ \texttt{first}(\mathcal{R}, \mathbf{A}(v^*)) \\ e_c = (\texttt{var}(c_* \wedge c))[\texttt{lab}(v)^*/p^*] \\ e_{op} = (\texttt{var}(\texttt{op}(\boldsymbol{T})))[\texttt{lab}(v)^*/p^*] \\ e' = e \cup e_c \cup e_{op} \end{array}}{e : \ x\ \mathbf{A}(v^*)\ y \quad \rightarrow_{\mathcal{R}} \quad e' : \ x\ \texttt{eval}(\boldsymbol{T}[v^*/p^*])\ y}$$

FIG. 7 – Nouvelle s´emantique des règles de r´e´ecriture

Dans cette nouvelle s´emantique, chaque terme est ´egalement ´etiquet´e. Cette ´etiquette identifie l'ensemble des paramètres qui ont influenc´e la r´e´ecriture. Afin de calculer la nouvelle ´etiquette d'un terme lors de sa r´e´ecriture, nous introduisons trois nouvelles fonctions :

`op` retourne la liste des op´erations contenues dans son argument. Par exemple, "$\texttt{op}(\mathbf{A}(x, x+y, x+1))$" rend la liste $[x + y, x + 1]$.

[4] On appelle axiome initial, l'axiome servant de point de d´epart `a la r´e´ecriture.

`var` retourne l'ensemble des variables de son argument. Ainsi "$\texttt{var}(x > y)$" rend en r´esultat l'ensemble $\{x, y\}$.

`lab` extrait l'´etiquette d'une valeur. Par exemple, "$\texttt{lab}(e : 3)$" retourne l'´etiquette e.

La fonction `eval` est ´egalement modifi´ee afin de calculer avec des valeurs ´etiquet´ees. Lors de l'´evaluation d'un paramètre sous forme d'op´eration, `eval` fait abstraction des ´etiquettes et effectue l'op´eration à partir des valeurs num´eriques. Une nouvelle ´etiquette vide est attribu´ee au r´esultat. Par exemple, $\texttt{eval}(\mathbf{A}((e:1) + 3))$ retourne $\mathbf{A}(\emptyset : 4)$.
Lors de la r´e´ecriture d'un terme t dans un terme t' ($e : t \rightarrow_{\mathcal{R}} e' : t'$), la nouvelle ´etiquette (ℓ) est obtenue en faisant l'union des labels de l'ancienne ´etiquette avec les labels des ´etiquettes e_c et e_{op}, où :

e_c contient les labels des paramètres influençant la s´election de la règle de r´e´ecriture pour cette ´etape de l'´evaluation.

e_{op} contient les labels des paramètres qui sont intervenus dans le calcul d'une op´eration à cette ´etape.

3.3 Propriétés de la sémantique

Afin d'´enoncer certaines propri´et´es de la s´emantique, une relation de projection entre paramètres est tout d'abord d´efinie. Cette d´efinition s'appuie sur une repr´esentation des paramètres d'un axiome sous la forme d'une liste de valeurs ´etiquet´ees. Cette projection s'exprime en fonction d'une ´etiquette qui restreint les paramètres pris en compte. On note "$P_1 \overset{e}{\hookrightarrow} P_2$" la projection d'une liste de paramètres P_1 dans une autre liste de paramètres P_2 de façon restreinte à l'´etiquette e. Pour toutes listes de paramètres P_1 et P_2, $P_1 \overset{e}{\hookrightarrow} P_2$ si et seulement si les paramètres, dont l'´etiquette dans P_1 est un sous-ensemble de e, ont la même valeur num´erique dans P_1 et dans P_2. Une d´efinition r´ecursive plus formelle de cette notion est donn´ee dans la d´efinition 3.1.

Définition 3.1 ($P_1 \overset{e}{\hookrightarrow} P_2$)
– $[\,] \overset{e}{\hookrightarrow} [\,]$ *est vrai*
– $[e_1 : x \mid RP_1] \overset{e}{\hookrightarrow} [\_ : y \mid RP_2]$ *est vrai ssi :*
  1. $x = y$ *ou* $e_1 \not\subseteq e$
  2. $RP_1 \overset{e}{\hookrightarrow} RP_2$

Trois nouvelles fonctions sont ´egalement introduites :

`labellise` prend en argument une liste de valeurs num´eriques (n^*) et une liste d'´etiquettes de même longueur (e^*). Elle retourne une liste de valeurs ´etiquet´ees v^* de même longueur que n^*. v^* est construite de telle sorte que, pour tout i, $v_i = e_i : n_i$. Par exemple, $\texttt{labellise}([e_1, e_2], [5, 1])$ retourne la liste $[e_1 : 5, e_2 : 1]$.

`strip` prend en argument un terme ***T*** construit à partir de valeurs ´etiquet´ees. Elle retourne un terme ***T'*** qui correspond au terme ***T*** dont toutes les ´etiquettes ont ´et´e supprim´ees. Ainsi, $\texttt{strip}(\mathbf{A}(e_1 : 2, e_2 : 7))$ retourne $\mathbf{A}(2, 7)$.

$\boldsymbol{T}[v^*/e^*]^{\mathcal{E}}$ est une fonction de substitution bas´ee sur les ´etiquettes. Le r´esultat de la substitution $\boldsymbol{T}[v^*/e^*]^{\mathcal{E}}$ est le terme ***T*** dans lequel les valeurs dont l'´etiquette apparaît à la position i dans la liste e^* ont ´et´e remplac´ees par la valeur à la position i dans v^*. Par exemple, $\mathbf{A}(e_1 : 2, e_2 : 7)[(e_a : 3, e_b : 4)/(e_2, e_3)]^{\mathcal{E}}$ est le terme $\mathbf{A}(e_1 : 2, e_a : 3)$.

Théorème 3.1 (Stabilité une étape) [5]
*Pour tout étiquette e_i, axiome **A**, liste de valeurs numériques n^* et d'étiquettes e_v^* telles que $\texttt{labellise}(n^*, e_v^*) = v^*$ avec v^* paramètres de **A**, et ensemble de règles $\mathcal{R}$, si :*

$$e_i : \mathbf{A}(v^*) \rightarrow_{\mathcal{R}} e : \boldsymbol{T}$$

alors, pour toute liste de valeurs v_2^* telle que $v^* \overset{e}{\hookrightarrow} v_2^*$:

$$\_ : \mathbf{A}(v_2^*) \rightarrow_{\mathcal{R}} \_ : \boldsymbol{T'}$$

avec $\texttt{strip}(\boldsymbol{T'}) = \texttt{strip}(\boldsymbol{T}[v_2^*/e_v^*]^{\mathcal{E}})$.

Le th´eorème 3.1 stipule que, si un axiome **A** param´etr´e par v^* se r´e´ecrit en une ´etape dans le terme ***T***, alors le même axiome, param´etr´e par une liste de valeurs dans laquelle se projette v^* (restreint par l'´etiquette g´en´er´ee), se r´e´ecrit en une ´etape dans le terme ***T'***. Les ´etiquettes mises à part, ce terme ***T'*** correspond au terme ***T*** dans lequel les valeurs ´etiquet´ees apparaissant dans v^* ont ´et´e remplac´ees par les valeurs correspondantes des paramètres utilis´es pour la r´e´ecriture de ***T'***.
Soit $\rightarrow_{\mathcal{R}}^{\dagger}$ la fermeture transitive de $\rightarrow_{\mathcal{R}}$ aboutissant à un terme sous forme normale. Le corollaire 3.2 est la sp´ecialisation au cas où $e = \emptyset$ de la g´en´eralisation du th´eorème 3.1 à $\rightarrow_{\mathcal{R}}^{\dagger}$.

Corollaire 3.2 (Reconstructibilité) [6]
*Pour tout axiome **A**, liste de valeurs numériques n^* et d'étiquettes e_v^* telles que $\texttt{labellise}(n^*, e_v^*) = v^*$, avec v^* paramètres de **A**, et ensemble de règles $\mathcal{R}$, si :*

$$\emptyset : \mathbf{A}(v^*) \rightarrow_{\mathcal{R}}^{\dagger} e : \boldsymbol{NT}$$

alors, pour toute liste de valeurs v_2^* telle que $v^* \overset{e}{\hookrightarrow} v_2^*$:

$$\_ : \mathbf{A}(v_2^*) \rightarrow_{\mathcal{R}}^{\dagger} \_ : \boldsymbol{NT'}$$

avec $\texttt{strip}(\boldsymbol{NT'}) = \texttt{strip}(\boldsymbol{NT}[v_2^*/e_v^*]^{\mathcal{E}})$.

3.4 Interprétation des étiquettes résultants de la réécriture d'un axiome

Dans la suite de cet article, afin de simplifier la pr´esentation, les ´etiquettes attribu´ees aux paramètres de l'axiome initial[7] sont index´ees sur les entiers. Ainsi, e_1 est l'´etiquette du premier paramètre, e_2 celle du deuxième, et ainsi de suite. Chacune de ces ´etiquettes est

[5] La preuve du th´eor`eme 3.1 d´ecoule de l'aspect d´eterministe de la s´emantique utilis´ee. En effet, `a chaque ´etape de r´e´ecriture, seul l'axiome le plus `a gauche peut ^etre r´e´ecrit. De plus, les ´etiquettes des valeurs intervenant dans l'´evaluation de la condition de la r`egle de r´e´ecriture retourn´ee par la fonction `first` font partie de l'´etiquette e g´en´er´ee lors de l'´etape de r´e´ecriture. Donc toute r´e´ecriture en une ´etape du m^eme axiome avec des param`etres dans lesquels se projettent les param`etres v^* utilise la m^eme r`egle de r´e´ecriture.

[6] La preuve du corollaire 3.2 s'obtient par sp´ecialisation au cas o`u $e_i = \emptyset$ du corollaire ´equivalent o`u e_i n'est pas restreint `a l'ensemble vide ; ce dernier corollaire ´etant prouv´e par r´ecurrence sur le nombre d'´etapes de la r´e´ecriture.

[7] On appelle axiome initial, l'axiome servant de point de d´epart `a la r´e´ecriture.

compos´ee d'un label unique. Ces labels sont index´es de la même façon que les ´etiquettes. Pour tout entier strictement positif i, $e_i = \{l_i\}$. L'exemple 3.2 pr´esente l'ensemble $\mathcal{R}$ des règles de r´e´ecriture qui sont utilis´ees dans cette partie.

Exemple 3.2: Ensemble des règles des axiomes **A** et **B**

$\mathbf{A}(c, d, x) : c = 0 \rightarrow \mathbf{B}(d, x)$
$\mathbf{A}(c, d, x) : c > 0 \rightarrow \mathtt{v}(d + c)\ \mathbf{A}(c - 1, d, x)$
$\mathbf{B}(d, x) : d = 0 \rightarrow \mathtt{t}()$
$\mathbf{B}(d, x) : d > 0 \rightarrow \mathtt{u}(x)\ \mathbf{B}((d - 1), x)$

Soit e l'´etiquette associ´ee au terme sous forme normal r´esultant de la r´e´ecriture de l'axiome **C** ($\emptyset : \mathbf{C}() \rightarrow^{\dagger}_{\mathcal{R}} e : \boldsymbol{NT}$). e contient les labels de tous les paramètres qui ont influenc´es la s´election des règles lors de la r´e´ecriture de **C** ; et donc la structure g´en´erale[8] du terme sous forme normale obtenu. Par exemple, l'´evaluation de l'axiome $\mathbf{B}(e_1 : 0, e_2 : 3)$ ext´eriorise[9] uniquement l'´etiquette de son premier paramètre et rend le terme sous forme normale $\mathtt{t}()$. Celui-ci n'ayant pas de terminaux avec paramètre, le corollaire 3.2 implique que tout axiome de la forme $\mathbf{B}(\_ : 0, \_)$ se r´e´ecrit sous forme normale en $\mathtt{t}()$. La r´e´ecriture de l'axiome $\mathbf{A}(e_1 : 0, e_2 : 0, e_3 : 3)$ fonctionne de façon similaire. Le r´esultat de sa r´e´ecriture est donn´e dans l'exemple 3.3. L'interpr´etation du r´esultat de l'´evaluation de l'axiome $\mathbf{A}(e_1 : 1, e_2 : 1, e_3 : 1)$ est plus complexe. Le r´esultat, donn´e dans l'exemple 3.3, montre l'ext´eriorisation des ´etiquettes des deux premiers paramètres uniquement. Il est donc possible d'en d´eduire que la r´e´ecriture sous forme normale de tout axiome de la forme $\mathbf{A}(\_ : 1, \_ : 1, \_)$ rend un terme ayant pour structure g´en´erale $\mathtt{v}(\ )\ \mathtt{u}(\ )\ \mathtt{t}()$. Plus pr´ecis´ement, le paramètre du terminal v ayant une ´etiquette vide, le corollaire 3.2 implique que le r´esultat a pour forme $\mathtt{v}(\_ : 2)\ \mathtt{u}(\_)\ \mathtt{t}()$. De plus, ´etant donn´e que le paramètre de u a pour ´etiquette celle du troisième paramètre de l'axiome initial, ce même corollaire implique que la r´e´ecriture sous forme normale de tout axiome de la forme $\mathbf{A}(\_ : 1, \_ : 1, \_ : X)$ rend un terme ayant pour forme $\mathtt{v}(\_ : 2)\ \mathtt{u}(\_ : X)\ \mathtt{t}()$.

Exemple 3.3: R´esultats d'´evaluations
Le tableau suivant pr´esente des exemples d'´evaluation des axiomes **A** et **B** de l'exemple 3.2.

Axiome	R´esultat
$\mathbf{B}(e_1 : 0, e_2 : 3)$	$e_1 : [\mathtt{t}()]$
$\mathbf{A}(e_1 : 0, e_2 : 0, e_3 : 3)$	$\{l_1, l_2\} : [\mathtt{t}()]$
$\mathbf{A}(e_1 : 1, e_2 : 1, e_3 : 1)$	$\{l_1, l_2\} : [\mathtt{v}(\emptyset : 2); \mathtt{u}(e_3 : 1); \mathtt{t}()]$

4 APPLICATION À UN SYSTÈME DE CACHE

Comme pr´esent´e en introduction, le but de ce travail est d'acc´el´erer la r´e´ecriture des *FL-systems* pr´esent´es en section 2 à l'aide d'un m´ecanisme de cache appel´e *cache intelligent*. Dans un cache standard, la paire (clef,valeur) g´en´er´ee par une r´e´ecriture est très simple. La clef correspond a l'axiome initial. Elle inclue le nom de l'axiome et la valeur effective de ses paramètres. La valeur associ´ee à cette clef est le terme normal obtenu lors de la r´e´ecriture. Cependant, il arrive qu'un paramètre n'intervienne pas dans la r´e´ecriture de l'axiome, soit pour une raison de niveau de d´etails pris en compte, soit parce que le paramètre intervient uniquement au moment du rendu de l'environnement virtuel. Ainsi, avec les règles propos´ees dans l'exemple 3.2 et comme indiqu´e dans la section 3.4, les axiomes $\mathbf{B}(0, 3)$ et $\mathbf{B}(0, 0)$ se r´e´ecrivent de la même façon. En utilisant un calcul de d´ependance comme celui qui est pr´esent´e en section 3, il est possible d'am´eliorer la couverture de la clef de façon à ce qu'une concordance soit d´eclench´ee dans le cache pour deux axiomes qui ne diffèrent que par la valeur de paramètres n'intervenant pas dans la r´e´ecriture de ces axiomes. Cette section pr´esente tout d'abord la m´ethode de g´en´eration des clefs ´etendues et des valeurs associ´ees. L'utilisation de ces paires (clef,valeur) est ensuite expliqu´ee et prouv´ee correcte au regard de la s´emantique donn´ee dans la figure 7. Enfin, la question de la gestion des paires (clef,valeur) est abord´ee.

4.1 Génération des clefs et valeurs

Comme expos´e en section 3, à partir de l'´evaluation d'un axiome param´etr´e **A**, il est possible de d´eterminer automatiquement la forme normale r´e´ecrite d'un ensemble $\mathcal{A}$ d'axiomes de même nom (**A**) mais param´etr´es par des valeurs diff´erentes. Afin d'utiliser ce r´esultat dans un système de cache, il reste à d´efinir une m´ethode pour extraire automatiquement une clef caract´erisant l'ensemble $\mathcal{A}$. Il convient ´egalement de d´eterminer la valeur qui doit être associ´ee à cette clef.
Pour ce faire, deux nouvelles fonctions sont d´efinies :

`gen` prend en argument une liste de valeurs ´etiquet´ees v^* et une ´etiquette e. Elle renvoie une liste de valeurs non-´etiquet´ees. L'´el´ement à la position i est la valeur num´erique de v_i[10] si l'´etiquette de v_i est un sous-ensemble de e, et $\_$ sinon. Par exemple, $\mathtt{gen}([\{l_1\} : 1, \{l_2\} : 1, \{l_3\} : 1], \{l_1, l_2\})$ renvoie $[1, 1, \_]$.

`ffonc` prend en argument un terme sous forme normale ***T*** dont les terminaux le composant sont param´etr´es par des valeurs ´etiquet´ees. Elle renvoie un terme sous forme normale ***T'*** dont les terminaux le composant sont param´etr´es par des valeurs non-´etiquet´ees. $\mathtt{t}_j$ est le terminal à la position j dans ***T***, et $\mathtt{t}'_j$ celui à la même position dans ***T'***. $\mathtt{t}'_j$ est construit à partir de $\mathtt{t}_j$ en remplaçant chaque valeur $e : v$ par $\#_i$ si l'´etiquette e est compos´ee uniquement du label l_i et v sinon. Par exemple, $\mathtt{ffonc}(\mathtt{v}(\emptyset : 2)\ \mathtt{u}(\{l_3\} : 1)\ \mathtt{t}())$ renvoie "$\mathtt{v}(2)\ \mathtt{u}(\#_3)\ \mathtt{t}()$".

Lors d'une r´e´ecriture sous forme normale d'un axiome **A** param´etr´e par les valeurs ´etiquet´ees v^* :

$$\emptyset : \mathbf{A}(v^*) \rightarrow^{\dagger}_{\mathcal{R}} e : \boldsymbol{NT}$$

[8] *i.e.* abstraction faite des param`etres des terminaux.

[9] On dit qu'une r´e´ecriture ext´eriorise une ´etiquette si cette ´etiquette se retrouve dans l'´etiquette du terme sous forme normale obtenu.

[10] v_i est l'´el´ement `a la position i dans v

le m´ecanisme de cache g´enère la clef $\mathbf{A}(\mathtt{gen}(\%, e))$ et y associe la valeur $\mathtt{ffonc}(\boldsymbol{NT})$. L'exemple 4.1 montre les paires (clef,valeur) g´en´er´ees par les ex´ecutions de l'exemple 3.3.

Exemple 4.1: Mise en cache

En utilisant les ´evaluations pr´esent´ees dans l'exemple 3.3, le tableau suivant pr´esente les couples (clef,valeur) mis en cache :

Clef	Valeur
$\mathbf{B}(0, \_)$	$[\mathtt{t}()]$
$\mathbf{A}(0, 0, \_)$	$[\mathtt{t}()]$
$\mathbf{A}(1, 1, \_)$	$[\mathtt{v}(2); \mathtt{u}(\#_3); \mathtt{t}()]$

4.2 Utilisation du cache

Afin de formaliser l'utilisation du cache, une relation de concr´etisation entre listes d'´el´ements appartenant à $\mathbb{R}^+ \cup \{\_\}$ est introduite dans la d´efinition 4.1. Une liste l_c est une concr´etisation d'une liste de même longueur l_a, not´e $l_c \succ l_a$, si et seulement si tout ´el´ement appartenant à $\mathbb{R}^+$ dans l_a est ´egal à l'´el´ement à la même position dans l_c.

Définition 4.1 ($l_c \succ l_a$)
– $[\,] \succ [\,]$ *est vrai*
– $[x \mid RL_c] \succ [y \mid RL_a]$ *est vrai ssi :*

1. $y = x \quad$ *ou* $\quad y = \_$
2. $RL_c \succ RL_a$

On d´efinit ´egalement une fonction not´ee $\boldsymbol{T}[\![\vec{v}]\!]^\sharp$. $\boldsymbol{T}$ est un terme sous forme normale et v^* est une liste de valeurs non-´etiquet´ees. Cette fonction substitue dans $\boldsymbol{T}$ les identifiants introduits par la fonction $\mathtt{ffonc}$ $(\#_i)$ par la valeur à la position i dans la liste de valeurs v^*.
Le cache est utilis´e de la façon suivante. On considère la r´e´ecriture de l'axiome $\mathbf{A}$ param´etr´e par les valeurs non-´etiquet´ees v_c^*. Si il existe une paire $(\mathbf{A}(v_a^*), \boldsymbol{T})$ telle que $v_c^* \succ v_a^*$, alors le cache rend directement le terme sous forme normale $\boldsymbol{T}[v_c^*]^\sharp$. La correction du m´ecanisme est prouv´ee par le th´eorème 4.3 ; lui-même prouv´e à l'aide des lemmes 4.1 et 4.2 ainsi que du corollaire 3.2. Le r´esultat de l'application du m´ecanisme de cache à la r´e´ecriture de l'axiome $\mathbf{A}(1, 1, 5)$, lorsque $\mathbf{A}(1, 1, 3)$ a d´ejà ´et´e r´e´ecrit, se trouve dans l'exemple 4.2.

Exemple 4.2: Utilisation du cache

Dans cet exemple, le cache contient les paires (clef, valeur) donn´ees dans le tableau de l'exemple 4.1. Entre autre, il contient la paire $(\mathbf{A}(1, 1, \_), [\mathtt{v}(2); \mathtt{u}(\#_3); \mathtt{t}()])$. La liste de valeur $(1, 1, 5)$ est une concr´etisation de la liste $(1, 1, \_)$. Ainsi, pour l'axiome $\mathbf{A}(1, 1, 5)$, le cache rend le terme sous forme normale $[\mathtt{v}(2); \mathtt{u}(5); \mathtt{t}()]$.

Lemme 4.1 (Correction de la clef) *Pour toute étiquette e et toutes listes de valeurs étiquetées v_c^* et v_a^* :*

$$v_c^* \succ \mathtt{gen}(v_a^*, e) \Rightarrow v_a^* \overset{e}{\frown} v_c^*$$

Lemme 4.2 (Correction de la valeur) *Pour tout terme sous forme normale **NT**, liste de valeurs étiquetées v_2^*, et liste d'étiquettes e^* telle que l'étiquette à la position i est composée d'un label unique indicé à la valeur i :*

$$\mathtt{strip}(\boldsymbol{NT}[v_2^*/e^*]^{\mathcal{E}}) = (\mathtt{ffonc}(\boldsymbol{NT}))[v_2^*]^\sharp$$

Théorème 4.3 (Correction du cache) *Pour tout axiome **A**, liste de valeurs numériques de longueur k n_1^* et n_2^*, et liste d'étiquettes de longueur k e^* telle que l'étiquette à la position i est composée d'un label unique indicé à la valeur i, si :*

1. $\emptyset : \boldsymbol{A}(\mathtt{labellise}(n_1^*, e^*)) \rightarrow_{\mathcal{R}}^{\dagger} e_f : \boldsymbol{NT}$
2. $\emptyset : \boldsymbol{A}(\mathtt{labellise}(n_2^*, \_)) \rightarrow_{\mathcal{R}}^{\dagger} \_ : \boldsymbol{NT'}$
3. $n_2^* \succ \mathtt{gen}(\mathtt{labellise}(n_1^*, e^*), e_f)$

alors :

$$\mathtt{strip}(\boldsymbol{NT'}) = (\mathtt{ffonc}(\boldsymbol{NT}))[n_2^*]^\sharp$$

4.3 Gestion du cache

Comme pour tout système de cache, les performances du m´ecanisme propos´e sont fortement d´ependantes de la gestion des paires (clef,valeur) par le cache. En effet, si le cache contient trop de paires, le système risque de passer plus de temps à essayer de faire correspondre l'axiome à r´e´ecrire avec une clef du cache qu'à r´e´ecrire l'axiome en question. La r´esolution de ce problème s'opère à deux niveaux. Tout d'abord, il convient de limiter le nombre de paires (clef,valeur) candidates à l'insertion dans le cache. Enfin, le remplacement de paires pr´esentes dans le cache par de nouvelles doit s'effectuer intelligemment.

Le m´ecanisme de g´en´eration de clefs en produit ´enorm´ement. À chaque niveau de r´e´ecriture d'un axiome, une paire (clef,valeur) est g´en´er´ee. Il convient de limiter cette prolif´eration en ignorant les paires correspondant à un nombre r´eduit d'´etapes de r´e´ecriture. Entre autre, les clefs produites lors d'une r´e´ecriture utilisant une règle dont le membre droit est compos´e exclusivement de terminaux doivent être ignor´ees. Il est ´egalement n´ecessaire de ne pas prendre en compte les paires produites lors de la r´e´ecriture d'axiomes rencontr´es exceptionnellement. Une grande part de ce travail peut être effectu´ee par l'auteur d'un *FL-system* en indiquant les axiomes les plus g´en´eriques, et donc int´eressant a prendre en compte pour la g´en´eration des clefs. Par exemple, un *FL-system* produisant une forêt fera une utilisation intensive de l'axiome g´en´erant un arbre. Celui-ci ´etant, *à priori*, relativement complexe, il est int´eressant de le prendre en compte pour la g´en´eration des paires.

Lors de l'insertion d'une paire dans le cache, les m´ethodes standards utilis´ees dans le domaine des caches permettent d'am´eliorer les performances. De plus, le m´ecanisme introduit dans cet article permet, dans certains cas, d'augmenter les capacit´es du cache sans augmenter sa taille. En effet, lors de l'entr´ee dans le cache d'une paire (c, v), si il existe une paire (c', v') pr´esente dans le cache telle que $c' \succ c$, il est possible de remplacer (c', v') par (c, v) sans perte d'expressivit´e du cache. En effet, la valeur v' peut être r´eg´en´er´ee à partir de v.

$\omega : true \rightarrow \mathbf{I}(30, 6, 22.5, 10, 1, green)$
$\mathbf{I}(f, n, \delta, l, w, color) : f > 0 \rightarrow \mathbf{A}(n,\ \delta + rand(-2, 2) * 0.5,\ l + rand(-10, 10) * 0.1,$
$w + rand(-2, 2) * 0.5,\ color + rand(-10, 10) * 0.1) \qquad \mathbf{I}(f - 1,\ n,\ \delta,\ l,\ w,\ color)$
$\mathbf{I}(f, n, \delta, l, w, color) : f = 0 \rightarrow \epsilon$

FIG. 8 – *FL-system* pour une forêt de buissons utilisant le *FL-system* de la figure 2 (pour simplifier, nous consid´erons ici les couleurs comme des r´eels. $green$ est donc une constante r´eelle.)

4.4 Résultats expérimentaux

Le *FL-system* pr´esent´e en figure 8 ´etend celui de la figure 2. Le paramètre f est ajout´e afin de permettre un dimensionnement facile du nombre de buissons engendr´es par le *FL-system*. Les paramètres suivants sont les mêmes qu'auparavant, mais ils sont ici utilis´es en combinaison avec la fonction $rand(a, b)$ qui d´elivre un entier compris entre a et b (inclus). Cette dernière introduit ainsi du bruit dans les paramètres des buissons g´en´er´es. Un prototype utilisant ce *FL-system* a ´et´e impl´ement´e en *Prolog*. Il simule la r´e´ecriture d'un terme sans cache, avec un cache simple, ou avec le système de cache propos´e dans cet article. Chaque clef du cache simple se compose du nom de l'axiome et des valeurs effectives des paramètres de l'axiome ayant servi à la g´en´eration de cette clef. La figure 9 donne, en fonction du type de cache utilis´e, le temps de r´e´ecriture d'un terme correspondant à trente buissons de structure identique mais de formes et de couleurs diff´erentes, *i.e.* des axiomes A ayant des valeurs diff´erentes pour les paramètre δ, l, w et $color$. L'axe des abscisses repr´esente la profondeur de r´e´ecriture, c'est-à-dire la valeur du paramètre n donn´e aux axiomes, ou l'âge des buissons. Le gain en temps de r´e´ecriture entre la m´ethode propos´ee dans cet article et la m´ethode sans cache est illustr´e par la figure 10. La figure 11, quant à elle, montre ce même gain, mais pour des r´e´ecritures de profondeur de 10 et un nombre de buissons variant de 10 à 30.

On constate sur les figures 9 et 10 que, plus la profondeur augmente, plus notre système de cache s'avère efficace. En effet, par rapport à la r´e´ecriture sans cache, la m´ethode propos´ee ici profite alors de l'augmentation du nombre de termes *joignables*[11] rencontr´es lors de la r´e´ecriture d'un buisson, c'est-à-dire de la r´ep´etition dans sa structure. La figure 11 permet de constater que la grande g´en´ericit´e des clefs produites permet de g´en´erer certains buissons directement à partir du cache, c'est-à-dire de tirer partie de l'augmentation de la probabilit´e de retrouver des objets joignables lorsque la taille de l'environnement augmente. Ceci n'est pas possible avec le cache simple. Par contre, lors d'une r´e´ecriture ne rencontrant pas de termes aboutissant à la même liste d'appel de primitives graphiques, notre système de cache est alors ´equivalent au cache simple, voire l´egèrement plus lent. De fait, les efforts de g´en´eralisation des clefs seraient alors une tâche vaine.

[11] Deux termes sont dits *joignables* si leur r´e´ecriture aboutit au m^eme terme sous forme normale

Enfin, il est à noter que la gestion des m´ecanismes de base du cache par le prototype est très mauvaise. Ceci est refl´et´e par les mauvais r´esultats du cache simple. Le prototype de cache intelligent dispose du même encodage des m´ecanismes de base, ce qui laisse à penser qu'une meilleure impl´ementation permettrait d'obtenir de bien meilleurs r´esultats.

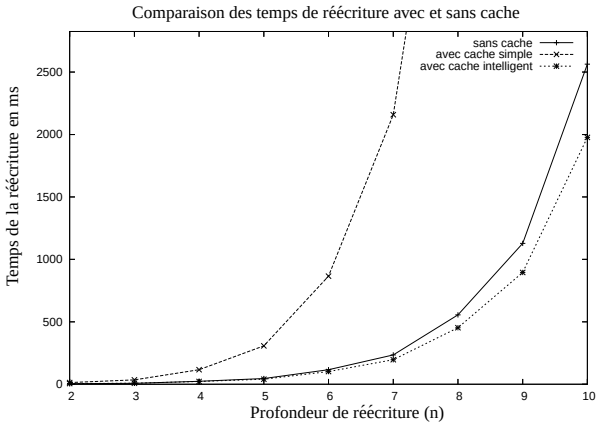

FIG. 9 – Temps de r´e´ecriture pour un nombre de buissons f=30 et une profondeur de n variant de 2 à 10.

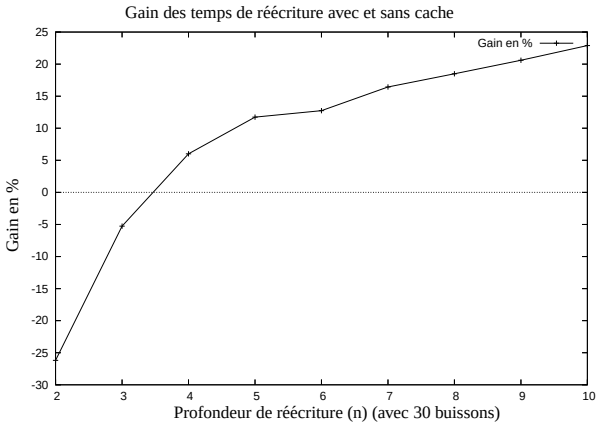

FIG. 10 – Gain en temps de la r´e´ecriture avec cache intelligent pour l'exp´erience de la figure 9

5 CONCLUSION

Dans cet article, nous avons pr´esent´e un système de cache *intelligent* bas´e sur un calcul dynamique des d´ependances. Ce nouveau système de cache pour les *FL-systems* permet une abstraction dans le processus de

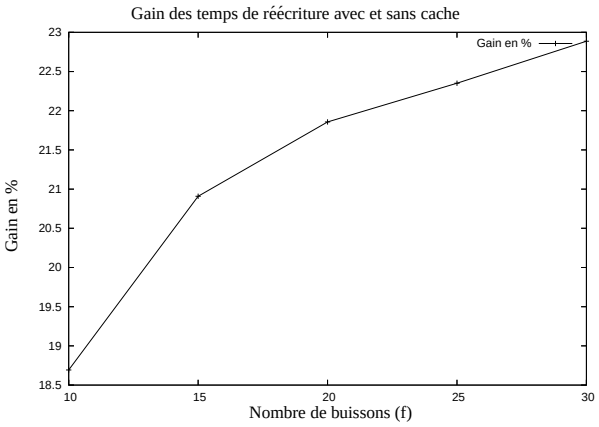

FIG. 11 – Gain de la r´e´ecriture pour une profondeur de n=10 et un nombre de buissons f variant de 10 à 30.

mod´elisation g´eom´etrique : la r´eutilisation des objets g´eom´etriques, jusqu'ici explicite, devient implicite. Cette technique est plus efficace qu'un cache simple grâce à la plus grande g´en´ericit´e des paires (clef,valeur) produites. Il est ainsi possible de g´en´erer la forme normale d'un axiome qui n'a pas encore ´et´e rencontr´e par instanciation de la valeur associ´ee à une clef *proche* de l'axiome à ´evaluer. N´eanmoins, il est important de ne pas calculer des paires (clef,valeur) pour chaque règle de r´e´ecriture, sous peine de faire d´eborder le cache. Une solution à ce problème consiste à identifier, lors de l'´ecriture d'un *FL-system* ou automatiquement par analyse, les règles les plus utilis´ees.

Notre système de cache permet une plus grande souplesse dans la mod´elisation d'environnements très complexes. En effet, consid´erons un *FL-system* repr´esentant un arbre et poss´edant un paramètre $color$ stipulant la couleur des feuilles de celui-ci. Une fois cet arbre utilis´e pour mod´eliser une forêt, la modification de la valeur du paramètre $color$ pourra être prise en compte de façon automatique par le cache. L'objet repr´esentant une feuille sera alors modifi´e pour prendre en compte le changement de couleur, et le reste de la forêt changera de couleur par la même occasion, sans modifier le cache et sans n´ecessiter la moindre r´e´ecriture.

Enfin, le *FL-system Intelligent Cache* constitue une optimisation compl´ementaire à l'instanciation approximative [Deussen, 1998]. En effet, notre technique permet de d´eterminer quels sont les termes dont les r´e´ecritures sont identiques, tandis que l'instanciation approximative permet de regrouper des termes similaires grâce à leur proximit´e dans l'espace des valeurs des paramètres. Notre technique permettrait donc d'am´eliorer le *clustering* en identifiant les paramètres qui n'influencent pas la r´e´ecriture. En effet, ceci revient à r´eduire la dimension de l'espace des paramètres et, de ce fait, à faciliter le *clustering*. Le couplage de ces deux technique constitue donc une perspective int´eressante pour nos prochains travaux.

BIBLIOGRAPHIE

[Abadi, 1999] Abadi M., Banerjee A., Heintze N. et Riecke J. G., A core calculus of dependency. *Proceedings of the ACM Symposium Principles of Programming Languages*, pages 147–160.

[Abadi, 1996] Abadi M., Lampson B. et L´evy J.-J., Analysis and caching of dependencies. *Proceedings of the 1996 ACM International Conference on Functional Programming*, pages 83–91. ACM Press.

[Dershowitz, 1990] Dershowitz N. et Jouannaud J.-P., Rewrite systems. In van Leeuwen J., editor, *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pages 243–320. North-Holland, Amsterdam.

[Deussen, 1998] Deussen O., Hanrahan P., Lintermann B., Měch R., Pharr M. et Prusinkiewicz P., Realistic modeling and rendering of plant ecosystems. *SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286, New York, NY, USA. ACM Press.

[Hart, 1992] Hart J. C., The object instancing paradigm for linear fractal modeling. *Proceedings of the conference on Graphics interface '92*, pages 224–231. Morgan Kaufmann Publishers Inc.

[Le Guernic, 2005] Le Guernic G. et Jensen T., Monitoring information flow. In Sabelfeld A., editor, *Proceedings of the 2005 Workshop on Foundations of Computer Security*. DePaul University. LICS'05 Affiliated Workshop.

[Lindenmayer, 1968] Lindenmayer A., Mathematical models for cellular interactions in development, I & II. *Journal of Theoretic Biology*, 18 :280–315.

[Marrin, 1997] Marrin C., Carey R. et Bell G., A vrml specification. Rapport technique, VRML consortium. http ://www.vrml.org/Sprecifications/VRML97.

[Marvie, 2003] Marvie J.-E., Perret J. et Bouatouch K., Remote interactive walkthrough of city models. *Proceedings of Pacific Graphics*, volume 2, pages 389–393.

[Marvie, 2005] Marvie J.-E., Perret J. et Bouatouch K., Fl-system : A functional l-system for procedural geometric modeling. *The Visual Computer*. To appear.

[Prusinkiewicz, 1990] Prusinkiewicz P., Lindenmayer A., Hanan J. S. et al., *The algorithmic beauty of plants*. Springer-Verlag, New York.

[Prusinkiewicz, 2001] Prusinkiewicz P., Mundermann L., Karwowski R. et Lane B., The use of positional information in the modeling of plants. *Proceedings of Computer Graphics and Interactive Techniques*, pages 289–300. ACM Press.

[Pugh, 1989] Pugh W. et Teitelbaum T., Incremental computation via function caching. *Conference Record of the 16th Annual Symposium on POPL*, pages 315–328.

[Smith, 1984] Smith A. R., Plants, fractals, and formal languages. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 1–10. ACM Press.