Interactive layout and handling of mathematical formulas in structured documents

Hanane Naciri, Laurence Rideau

Affichage et manipulation interactive de formules mathématiques dans les documents structurés

Hanane Naciri — Laurence Rideau

INRIA Sophia Antipolis
BP. 93, 06902 Sophia Antipolis Cedex - France.
{Hanane.Naciri,Laurence.Rideau}@sophia.inria.fr

RÉSUMÉ. Afficher des formules mathématiques et interagir avec ces formules sont des atouts primordiaux pour les outils informatiques dédiés aux mathématiques. Dans cet article, nous faisons un bilan des outils existants puis nous décrivons FIGUE, moteur d'affichage interactif incrémental et bidimensionnel, développé à l'INRIA, pour obtenir une bibliothèque dédiée au développement d'éditeurs de documents structurés et d'interfaces graphiques. Enfin nous montrons un exemple d'utilisation de FIGUE, dans le cadre du développement de preuves mathématiques sur ordinateur.

ABSTRACT. Tools dedicated to mathematics need to display formulas and to interact with them. In this paper, we present a summary of existing tools, then we describe FIGUE, an incremental two dimensional layout engine, developed at INRIA, to get a specialized toolbox for building customized editors and graphical user interfaces. Finally we give an exemple of interface using FIGUE to develop mathematical proofs on computer.

MOTS-CLÉS formules mathématiques, outils de formatage, MathML, édition structurée, document numérique, interaction homme machine.

KEYWORDS: mathematical formulas, formatting tools, MathML, structure edition, electronic document, man-machine interface.

1. Introduction

La production de documents scientifiques et en particulier le développement d'environnements graphiques pour les mathématiques nécessitent la possibilité d'exprimer et de manipuler une grande diversité d'objets comme du texte simple, des formules mathématiques, des schémas, etc.. Des problèmes particuliers se posent pour les formules mathématiques qui ont une structure bidimensionnelle et parfois complexe : fractions, matrices, intégrales, ou même exposants ou indices.

Nous distinguons deux familles de systèmes de production de documents scientifiques :

– les systèmes d'édition de documents en mode non-interactif (*batch-oriented processors*), comme par exemple le système TEX [33] qui permet un formatage de grande qualité ;

– les éditeurs WYSIWYG (*What You See Is What You Get*), comme par exemple AMAYA [61], éditeur WYSIWYG de pages Web permettant entre autre une édition structurée de formules mathématiques.

Ces derniers éditeurs offrent un environnement graphique qui aide l'utilisateur à visualiser son document pendant toute la phase de création. Ils permettent de manipuler dynamiquement des objets graphiques contrairement aux outils de la première famille qui nécessitent la transcription du document à l'aide d'un langage qui sera interprété ultérieurement pour l'affichage. Le travail présenté ici appartient à cette deuxième catégorie. La section suivante présente le contexte scientifique et les divers outils proposés dans cette catégorie.

1.1. Editeurs WYSIWYG :contexte et solutions existantes

Parmi les éditeurs WYSIWYG qui intègrent les formules mathématiques citons Edimath [45, 46], Publisher [37], FrameMaker [17], Grif [48] et Word [2]. D'autre part, plusieurs éditeurs mathématiques ont été développés pour permettre l'édition des expressions mathématiques usuelles : la formule éditée et formatée peut ensuite être copiée en tant qu'image dans d'autres applications. Parmi les éditeurs connus de ce type, citons MacEqn [38], MathWriter [16], Expressionist [30], MathType [52] et EquationBuilder [57].

Les systèmes dédiés à la production de document, comme Grif [48] ou comme INForm [21], ajoutent les avantages de l'édition structurée à l'approche WYSIWYG. Contrairement aux autres systèmes déjà cités, ces systèmes séparent la représentation du document de son contenu logique, ce qui permet une interaction avec les objets du document. L'éditeur Euromath [14], basé sur Grif, offre un environnement de travail pour les mathématiciens utilisant un modèle de données uniforme. L'auteur du document doit seulement s'occuper du contenu, l'affichage et la structure étant manipulés par le système. Cepen-

dant, ces systèmes de production de documents manipulent uniquement des expressions mathématiques éditées par l'utilisateur et non pas des formules dont la structure n'est pas connue d'avance, comme dans le cas où de telles expressions seraient générées par un système de calcul symbolique extérieur. Des environnements graphiques pour des systèmes de calcul symbolique doivent alors manipuler à la fois des expressions saisies par l'utilisateur et celles générées par le système de calcul.

Des expérimentations d'interface d'outils mathématiques, ont tout d'abord été menées dans le domaine du calcul formel : interface spécifique à Maple [51], ou plus généralement interface pour le calcul symbolique [28], dont les idées ont ensuite été largement reprises [55, 29] : les systèmes de calcul formel (Maple, Mathematica, ...) sont aujourd'hui dotés d'interfaces très conviviales et puissantes.

La plupart de ces travaux (en particulier les travaux de Kajler [28]) s'appuient sur l'édition structurée qui est apparue dans les années 80. Elle a été appliquée tout d'abord aux outils d'aide à la programmation (comme le Cornell Synthesizer [58] ou Centaur [11]). Basés sur des descriptions syntaxiques et sémantiques d'un langage donné, ces outils avaient comme objectifs la génération d'environnements de programmation spécifiques au langage décrit. Ces environnements dotés d'interfaces graphiques puissantes incluaient un système d'édition structurée, des interpréteurs et divers autres outils sémantiques comme la transformation de programmes ou la traduction. Plus tard, le savoir-faire en matière d'édition structurée spécifique à ces outils d'aide à la programmation a été appliqué à la construction d'environnements pour les systèmes de preuves [59, 8, 10], qui sont des exemples d'outils symboliques manipulant des mathématiques autres que les outils de calcul formel.

1.2. Objectifs à atteindre et choix pour le développement de notre système

En général, tout outil pour les mathématiques doit assurer un affichage performant de formules mathématiques (bidimensionnel, incrémental, facilement personnalisable). Le développement d'outils de formatage génériques et portables participe à améliorer la qualité d'affichage et d'interaction dans les environnements WYSIWYG pour les mathématiques. De tels outils de formatage doivent non seulement résoudre le problème de l'affichage d'une diversité d'objets comme les formules mathématiques, mais également fournir un moyen simple et naturel pour interagir avec ces objets. D'autre part, ces outils doivent aussi pouvoir permettre de partager les données mathématiques avec d'autres applications et de les diffuser sur Internet en adoptant par exemple le format standard pour les expressions mathématiques MathML (dialecte XML pour les formules mathématiques)

Notre objectif est donc de fournir une librairie de fonctions permettant d'afficher et de manipuler des données comportant des mathématiques, mais aussi offrant des fonc-

tionnalités de partage et de transfert de ces données mathématiques. Cette librairie, libre d'usage, distribuée avec ses codes source et écrite dans un langage portable et accepté par une majorité d'utilisateurs potentiels, doit fournir les briques de base pour la construction d'outils de formatage, de manipulation et de partage sur l'Internet de documents électroniques incluant des mathématiques.

C'est dans cette optique que nous avons développé le système FIGUE, dont le noyau est un module de formatage et d'affichage interactif, auquel nous avons ajouté du support MathML pour le partage de données et l'affichage sur le Web. Pour des questions de portabilité et de visibilité nous avons choisi de le développer en JAVA et nous distribuons gratuitement et librement les classes JAVA et le code source (voir [22]).

La suite de cet article décrira les principales composantes de ce système : les caractéristiques de FIGUE pour manipuler des objets structurés, en particulier les formules mathématiques, seront expliquées en détail ; ensuite, nous présenterons le standard MathML et comment il est intégré à FIGUE ; enfin, nous montrerons le rôle de FIGUE pour le développement d'environnements graphiques pour des systèmes de calcul symbolique.

2. Affichage et manipulation d'objets structurés

2.1. Présentation de la bibliothèque FIGUE

FIGUE est un module de formatage et d'affichage interactif et portable utilisé dans le développement d'outils WYSIWYG comme le développement d'éditeurs de documents structurés ou plus généralement des systèmes de production de documents. FIGUE permet de présenter les objets structurés et en particulier les formules mathématiques de façon conviviale (avec une bonne qualité d'affichage) et offre des interactions variées à la souris comme la sélection de sous-expressions afin de pouvoir les manipuler dynamiquement (évaluation, simplification, modification, génération de code, etc). D'autre part, FIGUE est un module indépendant, offrant la possibilité de présenter graphiquement des objets structurés, édités par les utilisateurs en suivant une syntaxe suffisamment naturelle et non ambiguë, ou produits par des calculs de systèmes symboliques ; il permet aussi de les manipuler dynamiquement et les modifier. FIGUE est donc, en particulier, un excellent candidat pour le développement d'interfaces pour les mathématiques. Il est aujourd'hui utilisé dans l'équipe LEMME[1] de l'INRIA Sophia-Antipolis pour le développement d'outils interactifs et d'éditeurs d'objets structurés comme les programmes, les formules mathématiques ou les preuves sur ordinateur.

A l'origine, une version préliminaire de FIGUE, intégrée au système CENTAUR, a été utilisée pour le développement de CAS/PI [28], une interface utilisateur générique pour

1. Logiciels et Mathématiques

les systèmes de calcul formel. Actuellement, FIGUE (écrit totalement en JAVA) est utilisé pour le développement du système PCOQ [43, 47], une interface graphique conçue pour le système de preuves COQ [25]. Il a aussi été utilisé dans le projet SmartTools [53] pour le développement d'outils interactifs génériques.

Dans la suite de cette section nous présentons en détails les caractéristiques de FIGUE pour manipuler et afficher des objets structurés, en particulier les formules mathématiques : les structures de boîtes, les constructeurs graphiques, les algorithmes de formatage et d'affichage, les possibilités d'incrémentalité, la gestion de sélection à la souris et les mécanismes d'interaction.

2.2. Structures de boîtes

FIGUE se base sur la notion de boîtes [33] pour représenter tous les objets structurés du document. Il manipule un arbre de boîtes qu'il formate puis affiche. Cet arbre de boîtes peut être obtenu de différentes manières (comme par exemple à partir d'une description XML). Dans l'usage que nous en faisons couramment, il est produit par une description d'un langage abstrait, appelé PPML (*Pretty Printing Meta Language*) [26]. Ce formalisme PPML était proposé par le système CENTAUR pour exprimer l'affichage des données (objets) structurées.

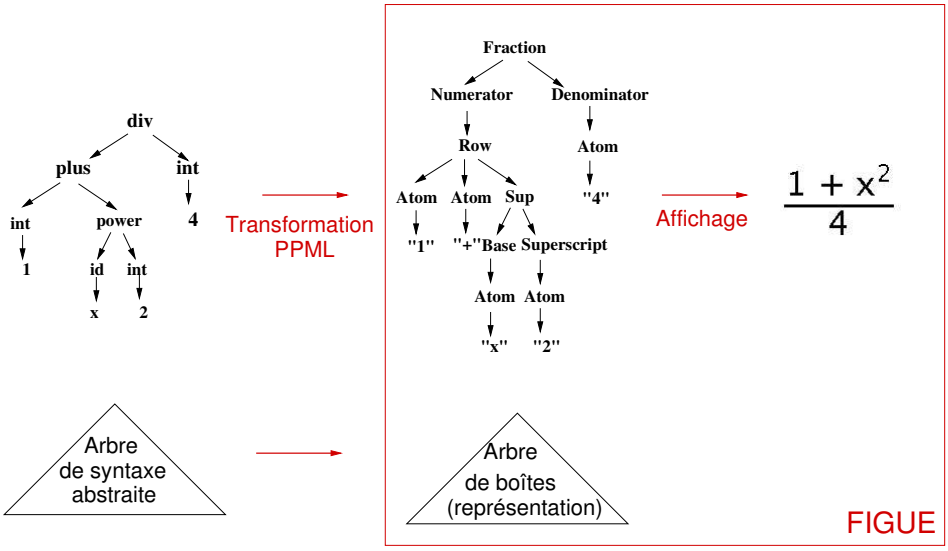


Figure 1. *Transformation d'un arbre de syntaxe abstraite représentant une formule mathématique en un arbre de boîtes* FIGUE*, à l'aide de règles* PPML.

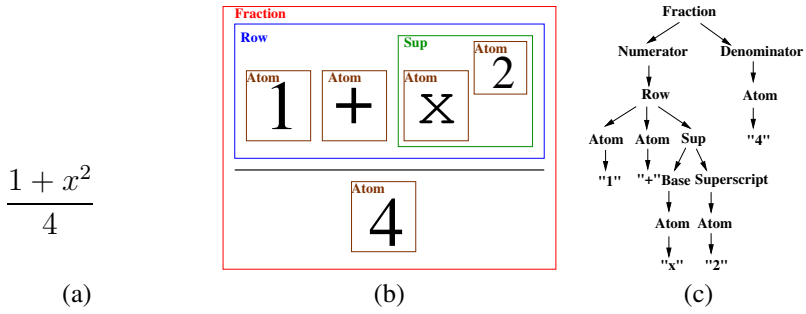
Une description PPML est une suite de règles où chaque règle associe à une structure logique de données sa structure de boîtes ou de représentation décrivant le formatage correspondant. PPML transforme les objets représentés sous forme d'arbre de syntaxe abstraite (suivant un formalisme) en un arbre de boîtes FIGUE correspondant à l'affichage : pour chaque noeud de l'arbre de syntaxe, PPML cherche et applique la règle de transformation correspondante. La figure 1 montre un exemple de transformation d'un arbre de syntaxe abstraite représentant une formule mathématique en un arbre de boîtes FIGUE.

Prenons un exemple simple d'une règle PPML décrivant comment l'opérateur binaire d'addition sera affiché :

$$\text{plus}(*x, *y) \quad \rightarrow \quad [\text{<Row>} \ *x \ \text{"+"} \ *y]$$

On associe à un arbre de syntaxe abstraite dont la racine est un "plus" ayant deux fils $*x$ et $*y$, une boîte composée d'un constructeur graphique *Row* (voir section suivante) et trois éléments $*x$, $+$ et $*y$. En partie droite de la règle, $*x$ et $*y$ sont des appels récursifs de l'affichage sur les fils de l'opérateur plus. Ils seront séparés par la chaîne "+" qui est un atome (une feuille de l'arbre de boîtes). Le constructeur *Row* permet de formater ses fils en mode horizontal et applique des règles de coupure si la boîte englobante dépasse la largeur de la zone d'affichage.

Les objets du document sont représentés sous forme d'arbre de boîtes, où chaque nœud est une boîte (constructeur graphique) englobant toutes ses boîtes filles et les feuilles sont des atomes (unités basiques comme les chaînes de caractères). La figure 2 montre un exemple simple de représentation en boîtes d'une formule mathématique.



(a) (b) (c)

Figure 2. *(a) Formule mathématique. (b) Ensemble de boîtes correspondantes. (c) Représentation sous forme d'arbre de boîtes.*

2.3. Constructeurs graphiques

FIGUE offre, par défaut, quelques constructeurs graphiques. Chaque constructeur prend une liste de boîtes ou des atomes en argument et les formate selon un algorithme de formatage spécifique. Les constructeurs graphiques de base de formatage en FIGUE sont : *Atom*, *Horizontal*, *Vertical* et *Paragraphe*. *Atom* est un constructeur de base qui produit des feuilles de l'arbre de boîtes (atomes qui n'ont pas de fils). *Horizontal* formate en mode horizontal la liste de ses fils. Il prend en paramètre l'espace entre deux éléments. L'alignement se fait au niveau du point de sortie d'un élément et du point d'entrée de l'élément suivant (voir figure 3). *Vertical* dispose ses fils en mode vertical. Il prend en paramètre l'espace consécutif entre deux lignes et l'indentation (i.e. un décalage horizontal de tous les éléments par rapport au premier). *Paragraphe* met ses éléments en mode horizontal tant qu'il reste assez de marge à droite de la page. Sinon il retourne à la ligne et dispose à nouveau ses éléments horizontalement (*Paragraphe* correspond au constructeur *Row* décrit dans la section précédente)

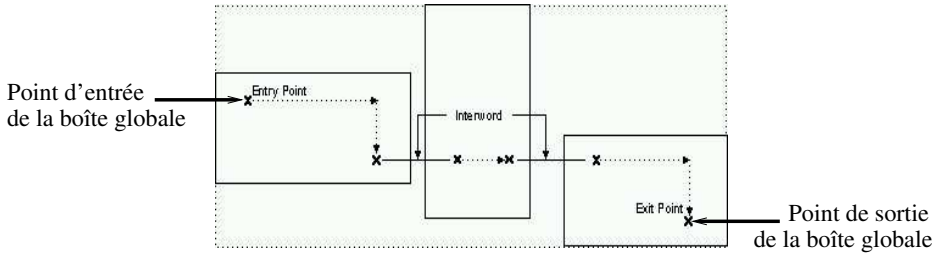


Figure 3. *Le constructeur Horizontal dispose ses fils horizontalement. Pour son alignement avec les autres boîtes, il a le même point d'entrée que son premier fils et le point de sortie de son dernier fils.*

En utilisant ces constructeurs de base, nous arrivons seulement à gérer une édition linéaire des documents (comme des programmes). Ce résultat n'est pas suffisant pour manipuler des formules mathématiques qui sont naturellement bidimensionnelles (racine carrée, matrice,intégrale,. . .). Pour cela, nous avons introduit dans FIGUE de nouveaux constructeurs pour les formules mathématiques (*Racine*, *Matrice*, *Puissance*, . . .). Ces constructeurs mathématiques disposent leurs fils selon la formule mathématique correspondante et au besoin dessinent les symboles nécessaires. Par exemple, le constructeur *Racine* dessine le signe racine autour de sa première boîte fille en tenant compte de sa taille (voir figure 4) et dispose son deuxième fils de façon à avoir une racine n-ième.

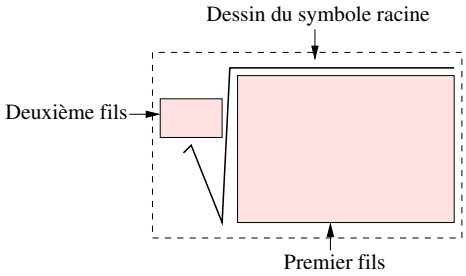


Figure 4. *Disposition des boîtes et le dessin de symboles pour le constructeur racine (Le formatage de la racine n-ième).*

2.4. Formatage et affichage bidimensionnels

Le formatage et l'affichage au niveau de FIGUE sont bidimensionnels, par opposition aux méthodes employées dans certaines interfaces qui obligent à linéariser les notations. Ils correspondent aux notations employées pour les documents manuscrits ou typographiés. FIGUE permet la disposition de boîtes dans un espace à deux dimensions (voir figure 5) en respectant la spécificité d'étirement de certains constructeurs en fonction de leur contenu (racine, intégrale, fraction)[2].

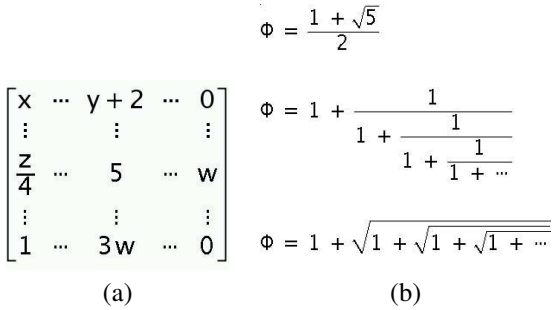


Figure 5. *Exemples de formatage bidimensionnel d'objets. (a) Disposition des éléments d'une matrice. (b) Affichage de formules mathématiques de structures différentes.*

Le formatage en FIGUE manipule directement l'arbre de boîtes. Cette structure d'arbre mémorise les relations de dépendances (la hiérarchie d'inclusion) entre les boîtes (objets graphiques) à afficher. L'algorithme de formatage parcourt cet arbre en mettant à jour les propriétés (ou attributs) graphiques des boîtes : origine, taille, alignement (voir figure 3).

2. Dans cet article, les exemples montrés sont des copies d'écran et ne sont pas fabriqués de toutes pièces : ils gagnent donc en authenticité mais peuvent perdent en qualité de rendu.

Les propriétés graphiques d'une boîte sont déterminées par la nature de son constructeur graphique et en fonction des propriétés de ses boîtes filles.

L'algorithme de formatage prend aussi en compte les paramètres de la zone d'affichage, i.e la largeur de la page (pour que les boîtes soient visibles) et les différents paramètres liés au contexte graphique de chaque boîte (les multiples polices de caractères utilisées).

Chaque constructeur a son propre algorithme de formatage, déterminant comment il disposera ses fils par rapport à son origine, ce qui ensuite déterminera l'alignement de sa boîte englobante avec l'ensemble (avec ses points d'entrée et de sortie - voir figure 3). Un soin particulier est accordé à l'efficacité des algorithmes de formatage des constructeurs mathématiques. Par exemple, la disposition correcte des éléments d'une matrice requiert un algorithme en plusieurs itérations[3].

Une fois le formatage effectué, chaque boîte FIGUE est capable de s'afficher elle-même dans un contexte graphique (polices de caractères, couleur, arrière-plan, coordonnées). L'algorithme d'affichage parcourt simplement l'arbre de boîtes pour afficher les boîtes en fonction de leur contexte graphique et éventuellement dessiner de nouveaux symboles ou des caractères typographiques (comme par exemple le symbole racine - voir figure 4).

2.5. Incrémentalité

En mode interactif, nous devons minimiser le coût de reformatage dû à la mise à jour ou à la sélection d'une ou plusieurs boîtes. Chaque modification d'une boîte (contexte, taille, origine,. . .) doit être propagée dans l'ensemble de l'arbre de façon optimale afin de mettre à jour seulement les boîtes affectées. Le formatage dans FIGUE peut être effectué de façon incrémentale, i.e. quand une boîte est déplacée ou modifiée, il ne recalcule que les propriétés modifiées (positionnement, origine, taille, rang des fils) des boîtes concernées. Ce formatage incrémental conduit à un affichage incrémental des objets du document où seules les zones modifiées ou touchées par la modification sont réaffichées.

Dans la version actuelle de FIGUE, l'algorithme de formatage incrémental est relativement simple. En mettant à jour la structure de l'arbre de boîtes, les modifications sont propagées vers la racine de père en père. Cet algorithme utilise les propriétés particulières de chaque boîte. Par exemple, dans le cas d'une boîte qui dispose ses fils horizontalement, nous ne reformatons qu'à partir du premier fils modifié (les autres fils seront poussés vers la droite, voir figure 6). Dans FIGUE, il est possible de spécialiser l'algorithme de reformatage pour chaque boîte.

3. La position d'un terme de la matrice n'est connu qu'après avoir déterminé, par une recherche plus globale, la largeur de chaque colonne de la matrice et la hauteur de chaque ligne (voir figure 5a).

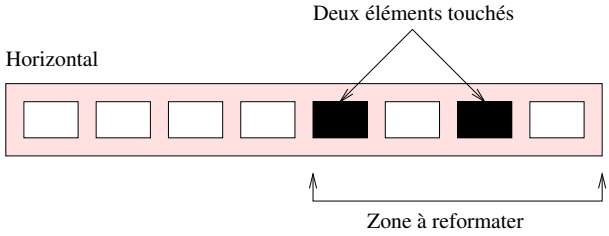


Figure 6. *Reformatage incrémental de la boîte Horizontal.*

En cas de débordement, si la boîte n'est pas fractionnable (comme dans le cas de la boîte de type *Horizontal*), un ordre de coupure et de reformatage est propagé vers le haut dans la structure d'arbre de boîtes ce qui peut nuire à l'efficacité du reformatage incrémental. Concrètement cela signifie que l'utilisateur doit éviter d'utiliser des boîtes non sécables et préferer des boîtes de type *Paragraphe* lorsque c'est possible (i.e. garder le constructeur *Horizontal* pour les cas où la coupure est impossible ou inesthétique : ponctuations, caractères spéciaux, mots collés, ...).

L'incrémentalité dépend de la sémantique (ou la nature) de chaque constructeur graphique. Pour chaque constructeur, il faut se poser la question suivante : quels sont les éléments à reformater ou à déplacer si on modifie une boîte fille de ce constructeur. Nous avons étudié l'algorithme d'affichage incrémental de chaque constructeur FIGUE et en particulier les constructeurs mathématiques en essayant de proposer des euristiques acceptables à l'usage, c'est à dire qui minimisent le nombre de calculs, et les mouvements sur l'écran. Par exemple, dans le cas du constructeur *Matrice*, nous retaillons la colonne de l'élément modifié et nous appliquons une translation horizontale aux colonnes suivantes.

2.6. Interaction et sélection

La plupart des interfaces modernes et interactives offrent la possibilité de sélectionner à la souris les objets affichés afin de les manipuler dynamiquement (en utilisant le copier-coller entre autres).

FIGUE offre un puissant mécanisme d'interaction permettant de manipuler dynamiquement et d'interagir à la souris avec les objets affichés (calcul, simplification de formules mathématiques, génération de code, ...). Nous distinguons deux niveaux d'interaction (voir figure 7). Le premier niveau d'interaction se situe entre l'utilisateur et les objets du document affiché par l'interface graphique et le deuxième niveau d'interaction se situe entre l'interface graphique et des systèmes extérieurs effectuant des traitements et des calculs (des systèmes de calcul symbolique ou autres).

FIGUE permet à l'utilisateur de sélectionner à la souris les objets affichés grâce à son mécanisme de sélection directement lié à la structure de l'arbre de boîtes. Il traduit auto-

Figure 7. *Deux niveaux d'interaction existent en* FIGUE. *L'utilisateur peut sélectionner des objets du document. La structure de ces objets est utilisée pour communiquer ensuite une requête à un système de calcul.*

matiquement les coordonnées de la souris sur l'écran en un emplacement dans la structure d'arbre de boîtes et retrouve ainsi le sous-arbre correspondant aux objets sélectionnés (voir figure 8). Une fois que la boîte (objet graphique) est sélectionnée par l'utilisateur, il est possible d'obtenir le sous-arbre de la structure syntaxique sous-jacente correspondant à cette boîte[4]. Le lien entre la structure de représentation des objets du document et leur contenu syntaxique est maintenu par le système, ce qui offre un mécanisme puissant d'interaction structurée.

La sélection peut être simple (un seul élément) ou multiple (plusieurs éléments sélectionnés à la fois). A chaque sélection, une action peut être appliquée, comme dessiner la plus petite boîte englobante contenant l'objet sélectionné. Par exemple si on a l'expression mathématique $(a * c) + b$ et qu'on sélectionne le symbole $*$, la sous-expression $a * c$ sera automatiquement sélectionnée.

Dans le cas d'une sélection multiple de boîtes, il n'y a pas forcément de sous-arbre directement associé. Il est néanmoins possible de formuler des heuristiques qui permettent une sélection multiple structurée, comme par exemple en déterminant la plus petite sous-expression contenant tous les objets sélectionnés. Si on reprend l'expression mathématique $(a * c) + b$ et qu'on sélectionne cette fois a et c, la sous-expression $a * c$ sera sélectionnée, car elle représente la plus petite sous-expression englobant les objets sélectionnés. Par contre, si on sélectionne c et b alors l'expression $(a * c) + b$ sera sélectionnée entièrement.

Malheureusement, dans le cas où l'on veut sélectionner la diagonale d'une matrice, cette méthode sélectionne toute la matrice (la plus petite structure contenant tous les éléments sélectionnés). Toutefois, il est possible d'affiner la méthode de sélection et de

4. On détermine la règle PPML qui est à l'origine de l'affichage de cette boîte. Ensuite, le sous-arbre sélectionné correspond alors au schéma syntaxique de la partie gauche de la règle PPML

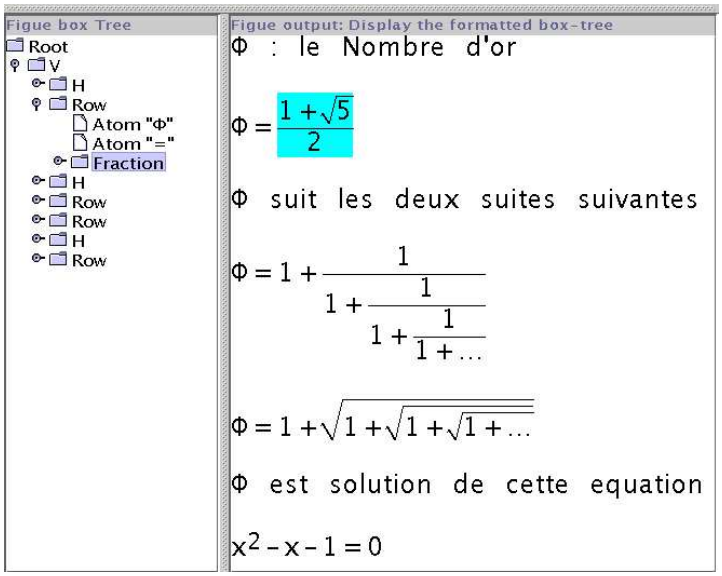


Figure 8. *Sélection des objets affichés par l'utilisateur dans la structure du document (partie droite), et calcul de leurs emplacements dans la structure d'arbre de boîtes par* FIGUE *(partie gauche).*

construire une sous-liste *virtuelle* des éléments de la diagonale sur laquelle l'interface pourra travailler comme si cette sous-liste existait vraiment dans l'objet structuré sous-jacent.

Il est également possible d'effectuer des opérations plus *intelligente* qu'un copier-coller, comme par exemple développer une expression mathématique. La figure 9 illustre ce mécanisme d'interaction plus complexe. L'utilisateur veut développer la sous-expression mathématique $(a - b)^2$ en utilisant l'interface graphique. Premièrement, l'utilisateur sélectionne la sous-expression à développer. Ensuite, l'interface récupère la structure syntaxique de cette sous-expression et la communique à un système de calcul symbolique via un protocole de communication défini (ce qui demande de traduire la structure de la sous-expression selon ce protocole). Après traitement, ce système renvoie le résultat du développement sous forme d'arbre de syntaxe. L'interface remplacera l'ancienne sous-expression par ce résultat. Pour cela, l'interface utilisera les règles de transformation PPML spécifiques pour obtenir le sous-arbre de boîtes correspondant. Les mécanismes de formatage et d'affichage incrémentaux sont alors exploités (voir section 2.5).

L'interaction et la sélection sont des fonctionnalités très importantes et très utiles dans les interfaces homme-machine. Le système PCOQ utilise FIGUE et exploite ce mécanisme

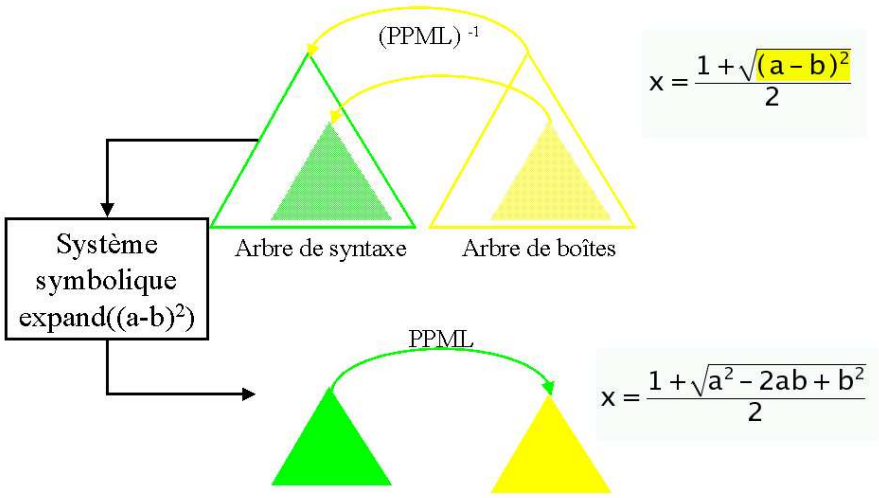


Figure 9. *Le mécanisme d'interaction permet d'obtenir la syntaxe des objets affichés afin de la communiquer au système de calcul symbolique et ensuite de réafficher le résultat du calcul.*

puissant de sélection pour effectuer des preuves par sélection et des transformations de formules (voir section 4.3).

2.7. Cas des formules mathématiques

Parmi les fonctionnalités dans des systèmes dédiés aux mathématiques (section 1), notre développement de FIGUE tente de résoudre les problèmes liés à l'affichage et la manipulation de formules mathématiques. Le traitement des objets mathématiques soulève plusieurs problèmes [55] [3], comme la complexité et la diversité des règles typographiques [7], la gestion des grandes formules, l'efficacité et l'incrémentalité des algorithmes de formatage, la sélection de sous-expressions et l'ambiguïté des notations mathématiques.

Parmi les problèmes rencontrés en FIGUE liés à la complexité des règles typographiques, citons l'exemple du dessin des délimiteurs (tels que $(, \int, \{$) qui sont des symboles mathématiques de taille variable délimitant des expressions mathématiques plus ou moins

grandes. Le dessin de ces symboles doit tenir compte de leur taille et de leur contexte graphique [1]. Pour les dessiner, nous nous sommes inspirés du système METAFONT en LaTeX [31] [32] et des polices *outlines* [24] qui se basent sur les courbes de Bézier[5]. L'algorithme de dessin de chaque symbole calcule l'épaisseur et les paramètres des courbes représentant le contour du caractère en fonction de la taille de la police utilisée et de la hauteur du symbole. Ces symboles mathématiques de tailles variables sont gérés au cas par cas. Arriver à trouver un algorithme général pour résoudre les problèmes du dessin de ces symboles demande d'étudier en profondeur les règles typographiques. Pour l'instant, le résultat obtenu pour l'affichage de ces symboles est satisfaisant (voir figure 10) mais devrait être amélioré par l'intégration de polices vectorielles appropriées dans notre système. Notons ici que notre but est d'obtenir à l'écran des notations mathématiques qui facilitent le travail de l'utilisateur, pas de concurrencer la qualité "papier" d'un document typographié dans les règles de l'art. Cet objectif ne serait de toutes façons pas atteignable tant que la résolution des écrans reste bien inférieure à celle de l'impression.

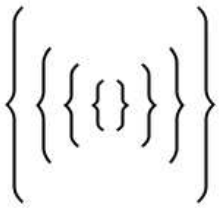


Figure 10. *Dessin des symboles mathématiques de tailles variables dans* FIGUE *: exemple de résultat obtenu pour l'affichage des accolades dans un contexte graphique comportant une grande fonte (taille 28).*

Pour la gestion de grandes formules, plusieurs solutions sont envisageables :

– Affichage à échelle réduite de l'expression mathématique. Cette solution n'assure pas la visibilité de l'expression.

– Appliquer des règles de césure et tenter de visualiser l'expression dans son intégralité. Certaines formules (telle les matrices) supportent mal la césure.

– Utiliser l'élision pour n'afficher que les termes les plus significatifs de l'expression. Le choix des termes à retenir est difficile et nécessite des compétences mathématiques approfondies.

– Fragmentation de l'expression en sous-expressions de tailles plus raisonnables.

5. Ces symboles peuvent être des courbes quelconques, par exemple des polynômes du troisième degré (cubiques) dont les courbes de Bézier sont un cas particulier.

En FIGUE, il est possible d'associer des actions (coupure, réduction d'échelle, ...) aux grandes formules, comme appliquer des règles de coupures automatiques ou afficher les termes jusqu'à une profondeur fixée par l'utilisateur. Dans la suite de notre travail, nous espérons proposer des solutions intéressantes à ce problème difficile.

Un autre défi pour un outil de formatage et d'affichage est de permettre à l'utilisateur d'obtenir une qualité d'impression sur papier le plus proche possible de ce qui est affiché à l'écran. Cet aspect est en cours de développement en FIGUE, avec une génération de PostScript acceptable. La difficulté principale consiste à découper le document en pages sans découper les éléments graphiques (les numérateur et dénominateur d'une fraction doivent rester sur une même page). Cette génération se fait à partir de la structure formatée et est donc indépendante des constructeurs graphiques puisse qu'elle ne prend en compte que les positions des divers éléments graphiques et les actions de dessin. Une étude est en cours pour déterminer si ces travaux peuvent être adaptés à la génération d'autres formats que PostScript comme par exemple SVG, le dialecte XML pour le graphique vectoriel.

2.8. Bilan des fonctionnalités d'affichage et de manipulation

Le noyau de FIGUE offre donc aujourd'hui une panoplie de fonctionnalité pour l'affichage et la manipulation de documents incluant des mathématiques. Son architecture et le fait qu'il soit distribué avec ses codes source, permet, si le besoin s'en fait ressentir, de rajouter de nouveaux constructeurs graphiques, sous réserve d' implanter les algorithmes de formatage et de dessin associés à ces nouveaux constructeurs. En ce qui concerne l'incrémentalité et l'interaction, des méthodes par défaut sont fournies par FIGUE, mais dans des cas très spécifiques ces méthodes peuvent être mal adaptées à la structure graphique ou à la sémantique d'un nouveau constructeur et devront être fournies lors de l'ajout du constructeur (ce fut par exemple le cas pour la matrice pour le reformatage après modification d'un élément ou la selection d'une diagonale).

3. Format d'échange de données : MathML

Dans le but de permettre un échange de données mathématiques entre applications et de l'intégration de nos outils sur l'Internet, nous avons développé un module de compatibilité avec le format standard MathML (dialecte de XML dédié aux expressions mathématiques). Nous détaillons ce module dans cette section.

3.1. Introduction

Un des intérêts majeurs des mathématiques sur ordinateur est de pouvoir partager les données mathématiques avec d'autres utilisateurs, de les échanger entre applications et de les diffuser sur Internet. Cela permet d'atteindre une population plus large d'utilisateurs

potentiels et de partager des résultats (feuilles de calcul pour des systèmes de calcul formel, preuves, ...). Dans ce but, plusieurs formats et protocoles d'échanges de formules mathématiques existent avec des systèmes comme OpenMath [19], MathLink [49] pour le système Mathematica, et plus récemment MathML [36], dédié aux mathématiques sur Internet.

MathML est proposé par le W3C comme un format standard pour les expressions mathématiques remplaçant les divers formats de données actuels. Certains de ces formats existant sont basés sur la syntaxe (LaTeX, notations utilisées dans les systèmes de calcul symbolique tels que Mathematica, Maple V, etc), mais jusqu'ici sur Internet, les formules mathématiques sont affichés en images (sans structure).

MathML est conçu pour servir de support d'échange entre logiciels scientifiques et mathématiques sans avoir pour unique objectif leur représentation graphique. MathML utilise deux types de description des formules : les éléments de présentation et les éléments de description sémantique (contenu). Les éléments de présentation sont destinés à la description bidimensionnelle des expressions mathématiques, alors que les éléments de contenu visent à donner une sémantique aux objets mathématiques (proche de la syntaxe abstraite).

Plusieurs communautés scientifiques s'intéressent au standard MathML : navigateurs Internet, systèmes de calcul, éditeurs structurés et éditeurs de textes. Les navigateurs Web essaient d'introduire dans leurs systèmes les formules mathématiques codées en MathML. Parmi les navigateurs Web qui supporte actuellement MathML, citons le navigateur et éditeur Amaya [61] du W3C, ainsi que les dernières versions de Mozilla, Techexplorer d'IBM et Internet Explorer.

Quelques systèmes de calcul (symbolique ou numérique) utilisent MathML pour décrire leurs résultats de calcul et les échanger avec des interfaces graphiques ou avec d'autres applications mathématiques. Le système Mathematica est capable d'évaluer des expressions MathML et d'afficher le résultat dans son interface graphique. Il peut être utilisé comme un serveur de calcul pour MathML. Une expérience de formalisation des termes de preuves du système COQ [25] utilise MathML pour représenter les formules mathématiques [5]. MathML est également choisi pour représenter dans un environnement graphique les algorithmes du calcul numérique [35] en donnant la possibilité à l'utilisateur de définir ses propres algorithmes et structures de données.

D'autre part, quelques travaux autour des éditeurs de textes intègrent MathML. Le système Omega [44], une généralisation du système TEX, permet la génération automatique de code MathML à partir des formules mathématiques écrites en langage TEX ou Latex. Amaya [61] permet l'édition structurée de MathML sur Internet. MathType [60] est un éditeur d'équations mathématiques et un outil d'affichage pour MathML. WebEq [62] offre une collection d'outils d'édition et d'affichage MathML incluant un éditeur visuel,

un traducteur de WebTeX vers MathML et une *applet* d'affichage interactif des mathématiques sur Internet.

Les travaux sur FIGUE ayant été engagés bien avant l'essor du Web, et en particulier la mise en place des différentes normes comme MathML ou le DOM (*Document Object Model*), FIGUE a précédé ces différentes normes et implémente ses propres objets comme l'objet structuré FIGUE (à comparer au DOM). Toutefois l'architecture de FIGUE se prête bien à la collaboration avec ces normes et en particulier à l'addition du support de MathML.

Dans le cadre de FIGUE, nous avons choisi dans un premier temps de traiter uniquement les éléments de présentation de MathML mais nous envisageons par la suite d'étudier l'intégration des éléments de description sémantique dans notre système. Cette intégration devrait être simplifiée par la philosophie de notre système qui s'appuie sur la syntaxe abstraite des objets manipulés (très proche des éléments de type *contenu* de MathML) et devrait être indépendante de FIGUE puisque le *contenu* ne concerne pas directement l'affichage. FIGUE permet l'affichage des objets mathématiques MathML et leur manipulation. FIGUE peut alors être utilisé comme base pour le développement d'éditeur d'objets structurés MathML, dans le genre d'Amaya. Il aura alors le rôle de passerelle entre les interfaces des applications et Internet.

3.2. Implémentation de MathML dans FIGUE

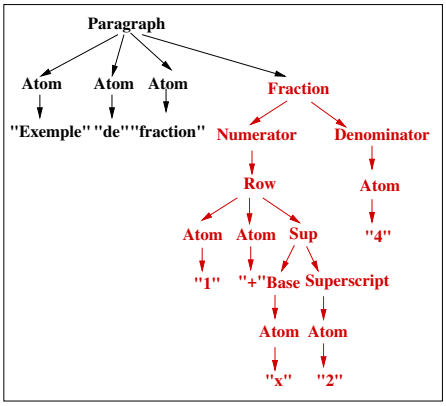
FIGUE implémente du MathML : il permet d'afficher les éléments de présentation MathML et de les manipuler. Inversement, il est possible de générer du MathML à partir des objets mathématiques affichés par FIGUE même si ces objets ont été crées indépendamment de MathML. Tout ceci fait que MathML peut être utilisé par FIGUE comme format d'importation et d'exportation de données et aussi comme format d'échange avec d'autres logiciels mathématiques.

FIGUE, comme MathML, utilise une notation structurée pour représenter les objets mathématiques. La correspondance entre les boîtes FIGUE et les éléments de présentation MathML se fait donc naturellement. Chaque élément de présentation MathML correspond à une boîte (constructeur graphique) FIGUE décrivant comment ses fils doivent être disposés et affichés. La figure 11 montre un exemple du constructeur *Fraction* qui dispose le numérateur au-dessus du dénominateur et sépare les deux par une barre de fraction : la figure 11 montre d'abord le code source XML incluant la représentation de la fraction en MathML (figure 11a), l'arbre de boîtes FIGUE correspondant (figure 11b) et son affichage par FIGUE (figure 11c).

Nous avons développé en FIGUE un module permettant l'interprétation des documents structurés XML décrivant les boîtes FIGUE et incluant des expressions MathML. Il s'agit premièrement d'effectuer l'analyse syntaxique de l'ensemble du document XML suivant une grammaire spécifiée par sa DTD (*Document Type Definition*) afin d'en extraire l'arbre

```
<xml version="1.0" encoding="UTF–8" standalone="no"?>
 <!DOCTYPE Facade SYSTEM "FigueMathML.dtd">
 <Facade xmlns="http://www–sop.inria.fr/lemme/figue"
         xmlns:m="http://www.w3.org/1998/Math/MathML">
 <Root>
 <P>
 <Atom Value="Exemple">
 <Atom Value="de">
 <Atom Value="fraction">
 <math xmlns="http://www.w3.org/1998/Math/MathML">
   <mfrac>
     <mrow>
       <mn> 1 </mn>
       <mo> + </mo>
       <msup>
         <mi> x </mi>
         <mn> 2 </mn>
       </msup>
     </mrow>
     <mn> 4 </mn>
   </mfrac>
 </math>
 </P>
 </Root>
 </Facade>
```

(a)



(b)

Exemple de fraction $\frac{1 + x^2}{4}$

(c)

Figure 11. *Exemple de fraction : (a) encodage XML+MathML, (b) arbre de boîtes* FIGUE*, (c) résultat du formatage.*

DOM (*Document Object Model*) décrivant la structure de ce document XML. Ensuite, notre module traduit l'arbre DOM en un arbre de boîtes FIGUE qui pourra ensuite être affiché par FIGUE (voir figure 12). Ce module assure un lien bidirectionnel entre la structure d'arbre de boîtes FIGUE et la structure d'arbre DOM du document XML. Ayant le chemin d'un noeud dans l'un des arbres, il est possible d'obtenir le chemin correspondant dans l'autre arbre ; i.e si on sélectionne une expression affichée, il est possible d'obtenir l'emplacement de l'élément MathML correspondant dans l'arbre DOM et vice versa. Ceci permet d'obtenir le source XML associé à un élément affiché.

Nous avons testé deux approches pour implémenter ce module en FIGUE (voir figure 13) :

– traduire directement en code Java les objets mathématiques MathML en des objets structurés FIGUE (figure 13a) ;

– traduire à l'aide des règles de transformation paramétrées les objets mathématiques MathML en des objets structurés FIGUE (figure 13b).

La première implémentation de ce module permet de traduire directement l'arbre DOM du document contenant des éléments MathML en un arbre de boîtes FIGUE (voir figure 13a). Pour chaque élément MathML, nous associons le sous-arbre de boîtes FIGUE correspondant et nous assurons, par codage, le lien entre les deux structures.

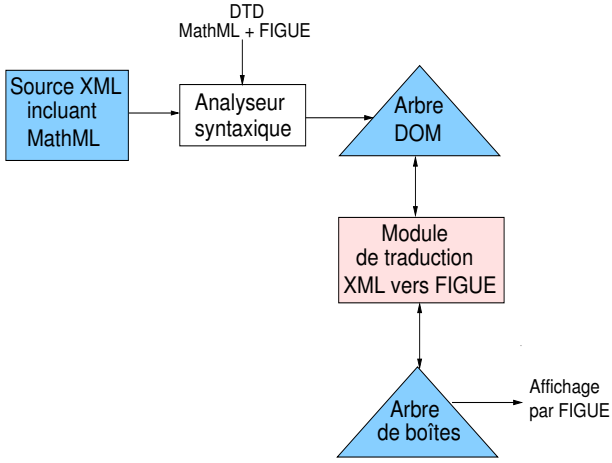


Figure 12. *Module d'interprétation de MathML vers* FIGUE *et vice versa.*

La deuxième implémentation applique des règles de transformation d'un protocole nommé XPPML[6] pour traduire chaque élément MathML (ou plus généralement des éléments XML) en un élément FigueML (élément XML décrivant une boîte FIGUE) qui sera ensuite à son tour transformé en une boîte FIGUE pour être affiché (voir figure 13b). Ces règles XPPML décrivent au moyen d'un document XML les règles de conversion d'un arbre source en un autre arbre résultat. Notons qu'une implémentation plus récente permet de passer directement de MathML vers la structure FIGUE en utilisant les règles XPPML, ce qui permet de minimiser le coût des transformations intermédiaires et obtenir une meilleure efficacité. Toutefois la transformation MathML vers FigueML reste un exemple d'utilisation de XPPML pour traduire une structure XML vers une autre structure XML.

Le compilateur des règles XPPML s'appuie sur les idées du compilateur de règles PPML (voir section 2.2).

6. Pour les lecteurs connaissant XML, notons que XPPML est une forme de XSLT, qui permet de transformer des fichiers XML

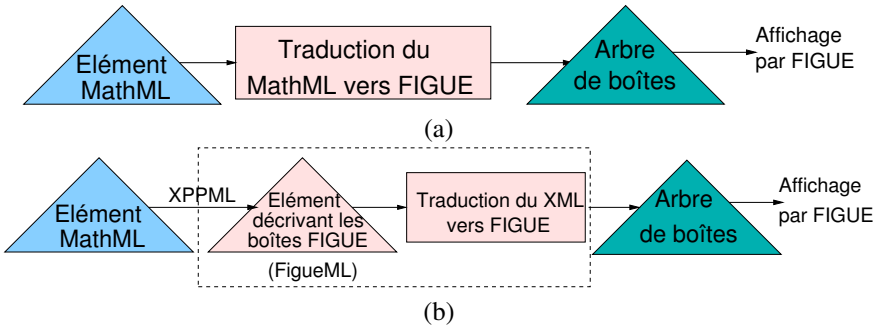


Figure 13. *Deux implémentations possibles pour interpréter du MathML dans* FIGUE. *(a) Schéma de traduction directe d'un élément MathML en un objet structuré* FIGUE. *(b) Schéma de traduction par utilisation de règles XPPML sur un élément MathML pour obtenir un objet structuré* FIGUE.

Il maintient le lien entre les objets graphiques FIGUE et leur structure MathML. Il est possible d'obtenir l'élément MathML correspondant à l'objet graphique sélectionné. Cette fonctionnalité est très utile en mode interactif où nous avons besoin de manipuler les objets affichés. Un autre choix serait d'utiliser le langage XSLT pour transformer les éléments MathML vers notre structure de boîtes. Cependant, à notre connaissance, les processeurs XSLT existants à ce jour ne sont pas incrémentaux. En effet, ils recalculent l'ensemble de l'arbre source pour déterminer le sous-arbre correspondant à l'objet sélectionné. La figure 14 montre quelques règles XPPML utilisées pour traduire le précédent exemple de fraction codé en MathML (voir figure 11) en un arbre de boîtes FIGUE. La première règle XPPML de cet exemple associe à l'élément MathML *mfrac* l'élément représentant la boîte de fraction en FIGUE.

La deuxième implémentation est plus flexible pour l'utilisateur, malgré sa complexité apparente. Elle donne la possibilité de paramétrer la traduction des objets mathématiques MathML vers des objets FIGUE à l'aide des règles de transformation. L'utilisateur peut spécifier ses propres règles de traduction et les modifier selon ses besoins.

3.3. Conclusion

Le schéma de la figure 15 résume les deux niveaux de collaboration entre nos outils de développement d'éditeurs et MathML : la collaboration entre la structure d'affichage de FIGUE et les éléments de présentation MathML (partie droite de la figure 15) et la collaboration entre la structure de syntaxe abstraite des objets manipulés et les éléments de type contenu de MathML (partie gauche de la figure 15). Dans le cadre de cet article, nous avons uniquement détaillé la collaboration entre FIGUE et MathML. FIGUE offre le support de MathML : il affiche et manipule des éléments de présentation MathML et génère du MathML à partir des objets mathématiques affichés par FIGUE.

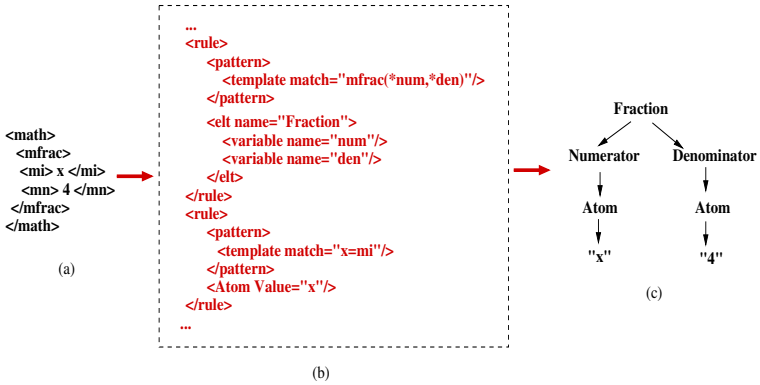


Figure 14. *Un exemple de traduction d'un élément MathML en un objet structuré* FIGUE *par application de règles XPPML. (a) élément MathML. (b) règles XPPML. (c) objet structuré* FIGUE.

L'ajout du support MathML dans FIGUE permet de l'utiliser comme composante d'affichage pour le développement d'éditeurs MathML et plus généralement pour des applications nécessitant la manipulation d'éléments MathML. Il fournit aussi le moyen d'obtenir automatiquement une version *Web* du document affiché qui peut être facilement diffusé sur Internet.

4. Application : développement d'interfaces graphiques

Comme nous l'avons vu précédemment, FIGUE est une composante dédiée à l'affichage interactif d'objets structurés. Il peut donc être utilisé pour le développement d'interfaces pour les mathématiques et en particulier pour des systèmes de calcul symbolique. Dans cette section, nous faisons d'abord un tour d'horizon des interfaces graphiques pour les systèmes de calcul symbolique existants. Ensuite, nous présentons notre approche générique pour le développement de telles interfaces. Enfin, nous présentons en détails l'interface graphique PCOQ qui suit cette approche générique et qui utilise FIGUE comme module d'affichage interactif.

4.1. Interfaces graphiques existantes

La plupart des systèmes de calcul symbolique offre la possibilité à l'utilisateur d'effectuer des traitements symboliques interactifs à travers une interface graphique (simplification, factorisation, preuve de théorème, ...). Ces interfaces conviviales et intuitives permettent de rendre les systèmes de calcul symbolique directement accessibles aux débutants mais aussi riches de fonctionnalités pour des utilisateurs experts.

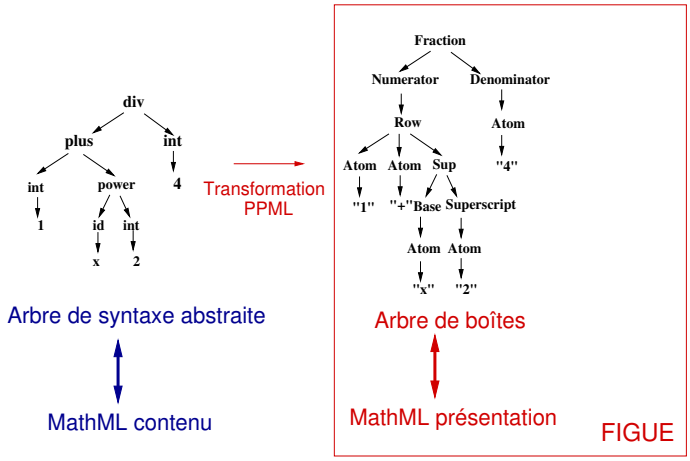


Figure 15. *Deux niveaux de collaboration entre nos outils et MathML : d'une part entre la structure de syntaxe abstraite et MathML contenu (gauche) et d'autre part entre la structure d'affichage de* FIGUE *et MathML présentation (droite).*

Les systèmes de calcul symbolique nécessitent donc des interfaces graphiques puissantes pour atteindre une population plus large d'utilisateurs. Dans ce but, plusieurs expérimentations d'interfaces d'outils mathématiques ont tout d'abord été faites dans le domaine du calcul formel puis pour les systèmes de preuves.

La première interface graphique réelle dans le domaine du calcul formel (interface pour le système Reduce) est MathScribe [54]. La principale caractéristique de cette interface est l'édition en deux dimensions de formules mathématiques. L'interface GI/S [63] de même type que MathScribe est une interface graphique pour le système Macsyma. Elle utilise une structure pour l'affichage qui n'est pas forcément liée à la structure de données des systèmes de calcul formel. Cela permet un mécanisme de sélection permettant par exemple la sélection des termes consécutifs d'une expression linéaire, comme la sélection de $b + c*$ dans l'expression $a - b + c * d$ même si $b + c*$ n'a pas de signification mathématique. L'interface Milo (intégrée plus tard dans FrameMaker) est basée sur la notion de document intégrant dans la même fenêtre du texte, des formules mathématiques et des traces de courbes. Milo est la première interface implémentant la manipulation directe d'expressions mathématiques. Parmi les prototypes d'interfaces graphiques génériques et distribués, CAS/PI [27] permet à un utilisateur expert d'adapter l'interface pour un besoin spécifique et de la connecter à d'autres logiciels externes. Mathematica [13] est un système de calcul algébrique conçu sous la forme d'un noyau algébrique et d'une interface. Mathematica a apporté une nouvelle notion de cahier interactif ou "note book" qui étend le modèle de l'interface-document en permettant une navigation type hypertexte.

Enfin, citons Emath [4] est une composante réutilisable et paramétrable pour l'édition de formules mathématiques qui peut être intégrée dans des interfaces de calcul formel.

Pour les systèmes de preuves, les premiers efforts pour la conception des interfaces ont commencé avec le système Nuprl [15] et IPE [50] (Interactive Proof Editor). Ces deux interfaces ont introduit l'édition structurée pour maintenir et éditer des preuves. CTCOQ [10], une interface utilisateur pour le système de preuves COQ [25], utilise l'environnement de programmation Centaur [11] et a introduit de nouvelles idées d'interaction, comme *proof-by-pointing* [8], *drag-and-drop* [9] et la gestion du script. Notons que ces outils spécifiques à l'interface CTCOQ ont ensuite été repris dans le cas d'outils génériques comme Proof General [6]. D'autres interfaces graphiques ont été conçues pour des systèmes de preuves existant : XIsabelle [41] une interface pour le système Isabelle [42], TkHOL [56] interface pour le système HOL [23] et le système graphique JAPE [12]. Ces systèmes dépendent de boîtes à outils graphiques orientées texte et sont moins efficaces pour la manipulation des formules à la souris. Pcoq [47] en cours de développement, reprend les idées de son antécédent CTCOQ en introduisant des améliorations au niveau de la manipulation des objets de preuves (voir pour plus de détails la section 4.3).

4.2. Approche générique pour le développement des interfaces graphiques

Dans notre approche générique pour le développement des interfaces graphiques pour des systèmes de calcul symbolique, l'interface est totalement séparée du noyau du calcul. Les principaux modules participant au développement de ces interfaces graphiques sont :

– un protocole d'échange de données entre l'interface graphique et le système de calcul symbolique ;

– une édition et une manipulation structurées des données ;

– un formatage et un affichage interactifs des objets structurés, facilement personnalisable par l'utilisateur.

La figure 16 montre notre schéma général pour le développement des interfaces graphiques utilisant FIGUE. L'interface et le système de calcul symbolique sont deux processus indépendants qui communiquent via un protocole. Ce protocole s'occupe du transfert des données, principalement des arbres, et traduit la structure des objets manipulés par l'interface en une structure de données du système de calcul symbolique et vice versa (exemple de protocole [20]). L'interface graphique utilise un module d'édition structurée offrant la possibilité à l'utilisateur de saisir des commandes et un analyseur produisant des objets structurés à partir des données saisies en suivant un formalisme (définition par des types abstraits d'un langage). Les objets structurés peuvent être sauvegardés dans un fichier, communiqués au système de calcul symbolique pour effectuer des calculs ou traduit en une structure de représentation (arbre de boîtes) pour pouvoir les afficher par notre moteur d'affichage FIGUE. Le lien entre les deux structures (arbre de syntaxe abstraite et

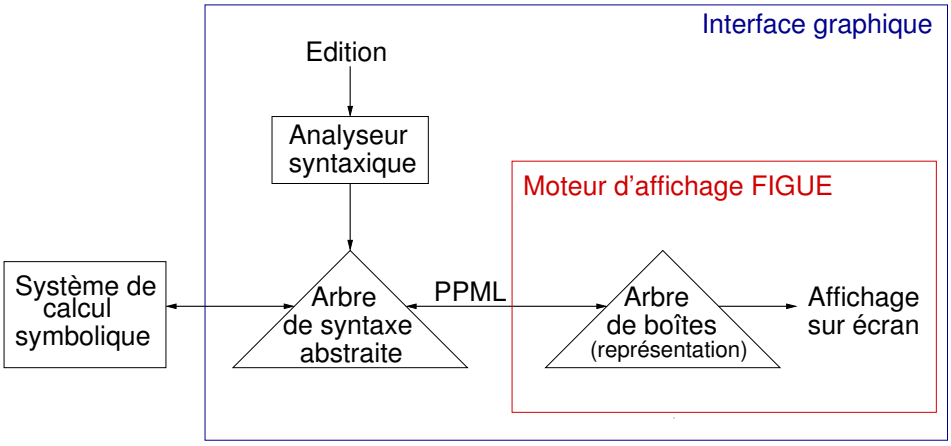


Figure 16. *Le schéma général des interfaces graphiques utilisant* FIGUE *pour les systèmes de calcul symbolique.*

arbre de boîtes FIGUE) se fait grâce aux règles de traduction PPML (voir section 2.2). Ce lien permet un mécanisme puissant de sélection d'objets manipulés par l'interface.

L'interface graphique PCOQ, s'appuie sur ce schéma général pour le développement de son environnement de travail graphique et interactif. La section suivante décrit en détails l'architecture de PCOQ et comment elle utilise FIGUE pour l'affichage des commandes et des formules mathématiques manipulées par l'interface.

4.3. PCOQ : un exemple d'utilisation de FIGUE

PCOQ (la version JAVA de CTCOQ [10] en cours de développement [43]) fournit un environnement de travail graphique et interactif pour le système de preuve COQ [25] suivant l'architecture décrite précédemment. Le moteur de preuve et l'interface utilisateur sont deux processus indépendants. Le but principal de PCOQ est d'aider au développement de preuves à grande échelle et de fournir une interface conviviale facilitant la création de preuves pour les utilisateurs du système de preuve COQ. Cette interface possède les caractéristiques suivantes :

– une interface graphique : présentation structurée et colorée des formules et des commandes ;

– des mécanismes d'édition et de présentation structurées : l'environnement fournit les moyens d'éditer structurellement formules et commandes ;

– la preuve par sélection : l'environnement utilise la structure des formules logiques pour aider systématiquement l'utilisateur à guider la preuve par de simples sélections à la souris [8].

L'interface graphique se base sur la boîte à outils pour la manipulation des données structurées AÏOLI et le noyau d'affichage FIGUE. Les données venant de COQ sont transformées en une structure d'arbres. Ensuite, AÏOLI manipule cette structure en utilisant ses trois composantes principales, inspirées des concepts du système CENTAUR [11] :

– VTP (*Virtual Tree Processor*) : une machine abstraite permettant de spécifier et de manipuler des structures d'arbres en suivant un formalisme (définition par des types abstraits d'un langage).

– PPML (*Pretty Printing Meta Language*) : un langage permettant de décompiler un arbre de syntaxe abstraite en un arbre de boîtes. Une spécification PPML est une suite de règles où chaque règle se compose d'une partie gauche représentant un arbre de syntaxe abstraite et une partie droite décrivant le formatage correspondant [26].

– GFXOBJ : une librairie d'objets graphiques (menus, fenêtres,...) permettant le développement d'applications interactives pour les arbres VTP.

AÏOLI, à l'aide de PPML, traduit un arbre de syntaxe abstraite en un arbre de boîtes. A partir de cet arbre de boîtes, FIGUE construit l'arbre de boîtes formaté qui sera affiché par la suite.

La figure 17 montre un exemple de session PCOQ (preuve en COQ que $\sqrt{2}$ n'est pas rationnel : $\sqrt{2} \notin \mathbb{Q}$). Dans cet exemple, différentes notations sont manipulées : texte, formules mathématiques et images. L'utilisateur peut effectuer des preuves par sélection [8], i.e. la sélection d'une formule logique permet la génération automatique de la commande à envoyer à COQ (en fonction du contexte, de la position de l'expression sélectionnée, ...). PCOQ offre aussi la possibilité de sélectionner et de déplacer un terme d'une formule (*drag and drop* [9]) en appliquant la transformation correspondante. Si nous prenons l'exemple simple de l'équation $ax + b = 0$ et que nous déplaçons le paramètre b du côté droit, nous obtenons l'équation équivalente $ax = -b$ (si la théorie mathématique sous-jacente le permet ; i.e. on est dans un anneau).

5. Conclusion

Tout au long de cet article, nous avons fait apparaître les principales fonctionnalités que tout outil pour les mathématiques devrait assurer :

– un affichage performant de formules mathématiques (bidimensionnel, incrémental, facilement personnalisable) ;

– une manipulation simple et naturelle des formules (interactivité) ;

– des moyens d'entrées des données naturels et efficaces (saisie) ;

– une impression donnant un résultat sur papier le plus proche possible de ce qui est affiché sur l'écran tout en offrant une qualité meilleure ;

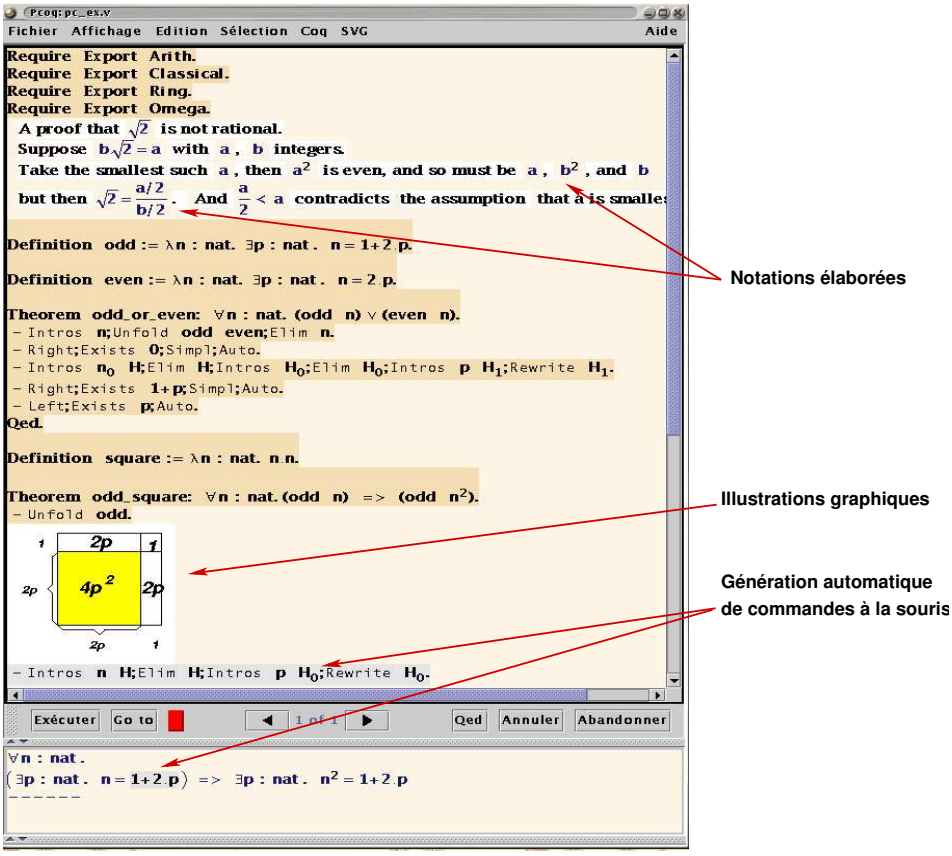


Figure 17. *Une session* PCOQ.

– des moyens de transférer et de communiquer des données.

Lors de nos développements, nous avons essayé de respecter ces différents besoins. Bien que nous nous soyons focalisés sur les aspects *affichage et interaction* d'une part et *transfert et communication* d'autre part, nous étudions aussi comment générer du PostScript dans le but d'obtenir une qualité d'impression sur papier acceptable. En ce qui concerne la *saisie* de formules, le moyen le plus simple que nous ayons à notre disposition est le passage par un format Ascii (saisie au clavier) ; pour cela il faut utiliser une syntaxe suffisamment naturelle et non ambiguë, mais malheureusement différente de ce qui est affiché (syntaxe de LaTeX par exemple). Dans un futur proche, nous voulons aussi donner la possibilité de saisir les formules à travers des menus, en prévoyant des raccourcis clavier pour alléger la tâche des utilisateurs plus expérimentés. Des expériences ont aussi été menées dans le cadre de la reconnaissance de formules manuscrites [34, 40, 39]

saisies grâce à une tablette graphique, mais ces travaux n'ont pas encore été intégrés à notre système.

Le but principal de notre développement de FIGUE est donc essentiellement de produire une bibliothèque de formatage et d'affichage dédiée au développement d'outils interactifs (WYSIWYG) pour les mathématiques comme le développement d'éditeur de documents scientifiques ou des interfaces graphiques pour les systèmes de calcul symbolique. Dans cette optique, notre approche ne se contente pas seulement d'offrir un affichage de qualité d'objets structurés comme les formules mathématiques mais elle offre aussi un moyen puissant et naturel pour interagir et manipuler ces objets actifs du document. Aujourd'hui FIGUE a dépassé le stade de prototype et est utilisé, en particulier, comme brique de base de PCOQ, à qui il fournit un affichage et une interaction très performants, adaptables facilement aux besoins de l'utilisateur.

D'autre part, FIGUE intègre un module permettant l'interprétation et la manipulation des documents structurés XML incluant du MathML, ce qui permet l'échange de formules avec d'autres applications. Pour le futur, nous nous fixons comme objectif d'utiliser ce module de FIGUE pour développer un éditeur d'objets structurés MathML pour le Web, dans le genre de AMAYA. L'avantage de notre approche est d'associer les objets à leur sens sémantique. Contrairement à l'approche de la plupart des éditeurs qui manipulent les objets en tant que texte, notre méthode offre la possibilité de faire des traitements sur les objets affichés hors du contexte du document, comme par exemple, la possibilité de simplifier une formule mathématique d'un document Web.

Enfin, à partir du travail réalisé en PCOQ sur la mise en place d'explications des preuves en langue naturelle [18], une autre direction de recherche est envisageable pour nous. Aujourd'hui, il existe des explications en langue française et anglaise. Dans l'avenir, nous envisageons d'expérimenter l'implémentation des explications en arabe, ce qui permettrait de tester et de pousser dans ses limites FIGUE pour écrire de droite à gauche et même mélanger l'affichage droite-gauche (texte arabe) et l'affichage gauche-droite (mathématiques).

6. Bibliographie

[1] ANDRE J., VATTON I., « Contextual typesetting of mathematical symbols taking care of optical scaling », Technical Report n° RR-1972, 1993, Inria.

[2] ARBOR A., ARBORTEXT M., CORPORATION I., « Word User's Guide », 1993.

[3] ARSAC O., « Interfaces homme machine pour le calcul formel », PhD thesis, Université de Nice Sophia Antipolis, Juillet 1997.

[4] ARSAC O., DALMAS S., GAËTANO M., « The Design of a Customizable Component to Display and Edit Formulas », DOOLEY S., Ed., *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation (ISSAC-99)*, New York, jul 1999, ACM Press,

p. 283–290.

[5] ASPERTI A., PADOVANI L., COEN C. S., SCHENA I., « Formal Mathematics in MathML », *MathML International Conference*, 2000, extended abstract.

[6] ASPINALL A., « Proof General : A Generic Tool for Proof Development. », 1785 L., Ed., *Tools and Algorithms for the Construction and Analysis of Systems, Proc TACAS 2000*, 2000.

[7] ATTAR P., « Prendre en compte les notations mathématiques dans les documents électroniques », *Document numérique*, vol. 1, n° 2, 1997, p. 147–159.

[8] BERTOT Y., KAHN G., THERY L., « Proof by Pointing », *Lecture Notes in Computer Science*, vol. 789, 1994, page 141.

[9] BERTOT Y., « Direct manipulation of Algebraic Formulae in Interactive Proof Systems », *Workshop on User-Interfaces for Theorem Provers*, Sophia Antipolis, sep 1997.

[10] BERTOT Y., « The CtCoq System : Design and Architecture », *Formal aspects of Computing*, vol. 11, 1999, p. 225-243.

[11] BORRAS P., CLEMENT D., DESPEYROUX T., INCERPI J., KAHN G., LANG B., PASCUAL V., « CENTAUR : The system », Research Report n° 777, Decembre 1987, INRIA, Rocquencourt, France.

[12] BORNAT R., SUFRIN B., « Animating Formal Proof at the Surface : The Jape Proof Calculator », *The Computer Journal*, vol. 42, n° 3, 1999, p. 177–192.

[13] BUCHBERGER B., « *Mathematica :* A System for Doing Mathematics by Computer ? », Technical Report n° 93-50, 1993, RISC-Linz, Johannes Kepler University, Linz, Austria.

[14] CHLEBÍKOVÁ J., « The Euromath System - The Structured Editor for Mathematicians », *Proceedings of the Tenth European TeX Conference*, , 1998, p. 82–93.

[15] CONSTABLE R. L., ALLEN S., H. BROMELY, CLEVELAND W. et al., *Implementing Mathematics with the Nuprl Development System*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

[16] COOKE J. R., SOBEL E. T., *MathWriter : mathematical typesetting with the Macintosh*, Cooke Publications, Ithaca, 1986.

[17] CORPORATION F., JOSE S., USA C., « FrameMaker Reference Manual », 1991.

[18] COSCOY Y., « A Natural Language Explanation for Formal Proofs », RETORÉ C., Ed., *Proceedings of Int. Conf. on Logical Aspects of Computational Liguistics (LACL), Nancy*, vol. 1328, Springer-Verlag LNCS/LNAI, September 1996.

[19] DALMAS S., GAËTANO M., S.WATT, « An OpenMath 1.0 implementation », KÜCHLIN W. W., Ed., *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, July 21–23, 1997, Maui, Hawaii*, New York, NY 10036, USA, 1997, ACM Press, p. 241–248.

[20] DERY A., RIDEAU L., « Distributed programming environments : an example of a message protocol », Technical Report n° RT-0165, 1994, Inria.

[21] VAN EGMOND S., HEEMAN F., VAN VLIET J., « INFORM : an interactive syntax-directed formulae editor », *The Journal of Systems and Software*, vol. 9, n° 3, 1989, p. 169–182.

[22] « Figue », `http ://www-sop.inria.fr/croap/figue`.

[23] GORDON M. J. C., MELHAM T. F., *Introduction to HOL : A theorem proving environment for higher order logic*, Cambridge University Press, 1993.

[24] HERSCH. R. D., *Visual and Technical Aspects of Type*, Cambridge University Press, 1993.

[25] HUET G., KAHN G., PAULIN-MOHRING C., « The Coq Proof Assistant : A Tutorial : Version 6.1 », Technical Report n° RT-0204, 1997, Inria.

[26] JACOBS I., RIDEAU-GALLOT L., « The PPML Manual », Technical report, August 1994, Inria.

[27] KAJLER N., « CAS/PI : a portable and extensible interface for computer algebra systems », WANG P. S., Ed., *Proceedings of ISSAC '92. International Symposium on Symbolic and Algebraic Computation*, New York, NY 10036, USA, 1992, ACM Press, p. 376–386.

[28] KAJLER N., « User interfaces for Symbolic Computation : a Case Study », *Proceedings of the 6th Annual Symposium on User Interface Software and Technology*, New York, NY, USA, Novembre 1993, ACM Press, p. 1–10.

[29] KAJLER N., SOIFFER N., « A Survey of User Interfaces for Computer Algebra Systems », *Journal of Symbolic Computation*, vol. 25, n° 2, 1998, p. 127–160.

[30] KALTOFEN E., WATT S., *Computers and Mathematics*, SpringerVerlag, 1989.

[31] KNUTH D., *Computers and Typesetting*, Addison-Wesley, Reading, 1986.

[32] KNUTH D., « Typesetting *Concrete Mathematics* », *TUGboat*, vol. 10, n° 1, 1989, p. 31–36.

[33] KNUTH D., *The TeX-Book (revised)*, Addison Wesley, 1990.

[34] KOSMALA A., LAVIROTTE S., POTTIER L., RIGOLL G., « On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars », *5th International Conference on Document Analysis and Recognition*, Bangalore, India, sep 1999.

[35] LI J., LETT G. S., « Using MathML to Describe Numerical Computations », *MathML International Conference*, 2000, extended abstract.

[36] « MathML », `http ://www.w3.org/Math/`.

[37] MCCARTHY E., HOLLAND C., LEHMAN J., « The Publisher User Manual », 1987.

[38] OF F., SOFTWARE S., VENABLE, MACEQN D., A COMPUTER, P ROCHESTER, « New York : Software for Recognition Technologies », 1985.

[39] OHTAKE N., FUKUDA R., SUZUKI M., « Optical Recognition and Braille Transcription of Mathematical Documents », *7th International Conference on Computers Helping People with Special Needs ICCHP*, Karlsruhe, 2000.

[40] OKAMURA H., KANAHORI T., CONG W., FUKUDA R., TAMARI F., SUZUKI M., « Handwriting Interface for Computer Algebra Systems », *Fourth Asian Technology Conference in Mathematics*, Guangzhou, 1999, p. 291-300.

[41] OZOLS M. A., CANT A., EASTAUGHFFE K. A., « XIsabelle : A System Description », MCCUNE W., Ed., *Proceedings of the 14th International Conference on Automated deduction*, vol. 1249 de *LNAI*, Berlin, jul 1997, Springer, p. 400–403.

[42] PAULSON L. C., « Isabelle : A Generic Theorem Prover », *Lecture Notes in Computer Science*, vol. 828, 1994, p. xvii + 321.

[43] « Pcoq », `http ://www-sop.inria.fr/croap/pcoq`.

[44] PLAICE O., HARALAMBOUS Y., « Generating MathML and Other ...ML from Omega », *MathML International Conference*, 2000, extended abstract.

[45] QUINT V., « An Interactive System for Mathematical Text Processing », *Technology and Science of Informatics*, vol. 3, n° 2, 1983, p. 169-179.

[46] QUINT V., « Interactive Editing of Mathematics », *Proceedings of the First International Conference on Text Processing Systems*, Boole Press Limited, 1984, p. 55–68.

[47] AMERKAD A., BERTOT Y., POTTIER L. , RIDEAU L., « Mathematics and Proof Presentation in Pcoq », *Proceedings of Workshop Proof Transformation and Presentation and Proof Complexities in connection with IJCAR 2001*,Siena Italy, jun 2001.

[48] QUINT V., VATTON I., BEDOR H., « GRIF : An interactive environment for TEX », DÉSARMÉNIEN J., Ed., *TEX for Scientific Documentation. Second European Conference. Strasbourg, France*, Lecture Notes in Computer Science, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1986, Springer-Verlag, p. 145–158.

[49] RESEARCH W., « MathLink reference guide. », rapport, 1992, Wolfram Research.

[50] RITCHIE B., « The Design and Implementation of an Interactive Proof Editor », PhD thesis, 1988.

[51] ROSIER P., « Développement d'un environnement de programmation Centaur pour Maple », Rapport de stage de DEA Calcul et Déduction, Université de Nice - Sophia Antipolis, 1995, UNSA.

[52] SCIENCE D., « MathType User Manual », 1987.

[53] « SmarTools », `http ://www-sop.inria.fr/oasis/SmartTools/`.

[54] SMITH C. J., SOIFFER N., « MathScribe : A User INterface for Computer Algebra Systems », *Proceedings of the 1986 Symposium on Symbolic and Algebriaic Computation*, jul 1986, p. 7–13.

[55] SOIFFER N., « The Design of a User Interface for Computer Algebra Systems », Technical Report n° CSD-91-626, 1995, University of California, Berkeley.

[56] SYME D., « A New Interface for HOL — Ideas, Issues and Implementation », *Lecture Notes in Computer Science*, vol. 971, 1995, page 324.

[57] TALBOT T., « EquationBuilder User Manual », 1992.

[58] TEITELBAUM T., REPS T., « The Cornell Program Synthesizer : a sunyntax-directed Programming Environment », *Communications of the ACM*, vol. 24, n° 9, 1981, p. 152–168.

[59] THERY L. AND BERTOT L. AND KAHN G., « Real Theorem Provers Deserve Real User-Interfaces », DOOLEY S., Ed., *Proceedings of the Fifth ACM Symposium on Software Development Environments (SDE5)*, Washington D. C, dec 1992, p. 283–290.

[60] TOPPING P., « Using MathType to create TeX and MathML equations », *TUGboat*, vol. 20, n° 3, 1999, p. 184–188.

[61] VATTON I., « W3C's Amaya 4.0 Editor Browser », rapport, 2000, W3C, http ://w3c1.inria.fr/Amaya/.

[62] « WebEQ », `http ://www.webeq.com/`.

[63] YOUNG D. A., WANG P. S., « GI/S : a graphical user interface for symbolic computation systems », *Journal of Symbolic Computation*, vol. 4, n° 3, 1987, p. 365–380.