Université de Nice - Sophia Antipolis
UFR SCIENCES

École Doctorale STIC

THÈSE

pour obtenir le titre de
Docteur en Sciences
de l'Université de Nice - Sophia Antipolis

Spécialité: Informatique

présentée et soutenue par

Hanane NACIRI

Conception et réalisation d'outils pour l'interaction homme machine dans les environnements de démonstrations mathématiques

Thèse dirigée par **Laurence RIDEAU**

soutenue le 11 Décembre 2002

Jury

M.	Roger	Marlin	Université de Nice	Président
Mme	Maylis	Delest	LABRI Bordeaux	Rapporteur
M.	Vincent	Quint	INRIA Rhône-Alpes	Rapporteur
M.	Norbert	Kajler	Ecole des Mines de Paris	Examinateur
M.	Stéphane	Lavirotte	Université de Nice	Examinateur
Mme	Laurence	Rideau	INRIA Sophia Antipolis	Directeur

À mon père, je dédie ce vers du célèbre poète arabe Ahmad Chawqi (احمد شوقي)

و ما نيل المطالب بالتمني و لكن تاخد الدنيا غلابا

et à ma mère celui-ci

الام مدرسة اذا اعددتهآ اعددت شعبا طيب الاعراق

Remerciements

Un grand merci à Laurence pour son encadrement, pour l'intérêt et l'attention qu'elle a porté à mon travail de thèse et pour son soutien constant.

Je remercie Vincent Quint et Maylis Delest d'avoir accepté de rapporter ma thèse et pour leurs commentaires constructifs.

Je remercie également les membres du jury Roger Marlin, Norbert Kajler et Stéphane Lavirotte pour leur présence et l'attention qu'ils ont porté à mon travail.

Je remercie tous les membres de l'équipe LEMME pour leur gentillesse, leurs précieux conseils et pour la bonne ambiance de travail qui règne, notament *François Lecoq* et *le bon gros William.*

Enfin, je remercie ma famille et tous mes amis pour leur affection et leur aide de tous les instants.

Table des matières

Table des figures

Introduction

Contexte de la thèse

Mon travail de thèse a été effectué au sein de l'équipe LEMME[1] de l'INRIA[2] Sophia Antipolis. Les principaux thèmes de recherche de l'équipe LEMME sont :

- **les environnements de preuves**
  Le but de ce thème est d'étudier les outils mécaniques de recherche et de vérification de preuves pour faciliter leur utilisation par des ingénieurs et des mathématiciens dans la production de logiciels et de théories mathématiques formelles;
- **la théorie des types et la formalisation de théories mathématiques**
  Les buts de ce thème sont de développer la théorie des types, et d'étudier comment des théories mathématiques où interviennent de nombreux types d'objets peuvent être représentées dans le calcul des constructions inductives (CCI en abrégé), de façon à être aussi lisibles et utilisables que possible par quelqu'un qui en a une connaissance scolaire ou académique;
- **les implémentations certifiées d'algorithmes de calcul scientifiques**
  Pour obtenir des programmes certifiés, une approche inverse de celle généralement utilisée est proposée : plutôt que de chercher à prouver des propriétés d'un programme existant (en formalisant sa sémantique), il est proposé de produire des programmes dont la correction découle de celle de leur processus de création;
- **la sémantique des langages de programmation**
  Les algorithmes intervenant dans l'implantation des langages de programmation font également partie du champ d'investigation de LEMME. Pour ces algorithmes, on se repose généralement sur la description sémantique d'un langage, et les propriétés que l'on cherche à établir pour un algorithme sont soit qu'il préserve la sémantique des programmes (s'il s'agit d'un algorithme de transformation ou d'optimisation) soit que les programmes qu'il produit sont exempts de certains comportements indésirables (s'il s'agit d'un compilateur ou d'un vérificateur de programmes). Pour classifier ce type d'algorithmes et de vérification, on parle de preuves en sémantique des langages de programmation;

1. Logiciels et Mathématiques
2. Institut National de Recherche en Informatique et en Automatique

Le but de cette thèse se place dans le premier champ d'activité “les environnements de preuves“ et plus précisément dans le cadre du développement d'outils pour l'interaction homme-machine dans les environnements de démonstrations mathématiques. Afficher du texte mathématique et interagir avec les formules mathématiques sont des atouts primordiaux pour les outils dédiés aux mathématiques. Nous nous sommes intéressés en particulier à produire un outil d'affichage interactif pour présenter et manipuler des documents contenant à la fois du texte et des formules mathématiques.

Objectifs à atteindre

Tout outil pour les mathématiques se doit d'assurer un affichage performant des formules mathématiques (bidimensionnel, incrémental, facilement personnalisable). Le développement d'outils de formatage génériques et portables contribue à améliorer la qualité d'affichage et d'interaction dans les environnements WYSIWYG [3] 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 le Web en adoptant par exemple le format standard pour les expressions mathématiques MathML [4] (dialecte XML [5] pour les formules mathématiques).

Notre objectif est de fournir une librairie de fonctions permettant d'afficher et de manipuler des données comportant des mathématiques, mais aussi offrant des fonctionnalités 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 le Web 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 bidimensionnel 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 [30]).

Jusqu'à présent la diffusion des documents scientifiques se fait en général en anglais, mais avec l'internationalisation, ces documents commencent à circuler dans différentes langues. Un grand besoin d'outils est apparu pour manipuler ces documents dans différentes langues, systèmes d'écriture ou conventions culturelles. En particulier, nous avons besoin d'outils de formatage multilingues. Un autre objectif fixé est de pouvoir afficher et manipuler les formules mathématiques dans différentes langues et systèmes d'écriture, en particulier l'arabe. Dans le cas des documents mathématiques arabes, les formules peuvent

3. What You See Is What You Get: environnement d'édition interactif dans lequel l'utilisateur travaille directement sur l'apparence finale du document.
4. Mathematical Markup Language
5. Extensible Markup Language

être écrites dans le sens inverse du texte arabe et contiennent à leur tour du texte ou des symboles arabes (mélange de la direction "droite à gauche" et de la direction "gauche à droite" dans la même formule mathématique). Avoir une approche générale pour manipuler les formules mathématiques dans un contexte bidirectionnel (mélange des directions d'écritures) nous permet de présenter des explications de preuves mathématiques en arabe dans le système PCOQ. Les implémentations actuelles de MathML permettent uniquement l'affichage de formules mathématiques de gauche à droite. Notre objectif est d'utiliser ce travail comme une base d'expérience pour définir l'usage de la norme MathML pour l'affichage bidirectionnel de formules mathématiques.

Résultats obtenus

Nous résumons ici les principaux résultats obtenus dans ce travail de thèse. Ces résultats sont divisés en trois grandes parties :

- Premièrement, l'extension de la librairie FIGUE pour présenter et manipuler des documents contenant à la fois du texte et des formules mathématiques.
- Deuxièmement, la migration de nos outils vers la technologie Web : implanter MathML.
- Troisièmement, la présentation des formules mathématiques dans différentes langues et systèmes d'écriture comme l'arabe.

1 - Extension de la librairie FIGUE

A l'origine FIGUE (version JAVA) n'offrait qu'une édition linéaire des documents (comme des programmes) à l'aide de ses boîtes graphiques de base de formatage (paragraphe, boîte horizontale et verticale). Cette fonctionnalité n'est pas suffisante pour manipuler des formules mathématiques qui sont naturellement bidimensionnelles (racine, matrice, puissance ...). Pour cela, nous avons étendu et amélioré FIGUE pour pouvoir présenter et manipuler des documents contenant à la fois du texte et des formules mathématiques. Nous avons étudié et implémenté les algorithmes de formatage de différents constructeurs mathématiques.

Notre version de FIGUE offre un moyen simple et naturel pour interagir avec des objets structurés (textes et formules mathématiques), édités par les utilisateurs, ou produits par des calculs des systèmes symboliques. FIGUE est aujourd'hui utilisé dans notre équipe LEMME pour le développement du système PCOQ [2], une interface graphique conçue pour le système de preuves COQ [35]. Il permet de présenter d'une façon conviviale des preuves développés sous PCOQ. On trouvera quelques exemples de preuves développés en PCOQ avec différentes notations mathématiques dans [68]. Nous avons expérimenté dans le cadre d'un stage d'ingénieur l'utilisation de FIGUE pour le développement d'une interface graphique commune pour deux systèmes de calcul symbolique BERNINA et SHASTA. Cette expérience nous a permis de tester FIGUE dans un autre contexte et d'améliorer la présentation des notations mathématiques.

2 - Migration vers la technologie Web

Pour nous conformer aux technologies Web dans le cadre des environnements de preuves, nous avons développé un module permettant de faire le lien entre FIGUE et le standard MATHML. Ce module permet d'afficher du MathML dans FIGUE et il offre aussi la possibilité de générer du code MathML à partir de l'affichage de FIGUE. Cette fonctionnalité nous permet de communiquer les preuves (buts et scripts) sur le Web en utilisant des navigateurs compatibles avec MATHML-*présentation*[6]. Inversement, FIGUE peut être utilisé comme moteur d'affichage MATHML-*présentation* par d'autres applications. Nos programmes permettent aussi de manipuler des données présentées par MATHML-*contenu*. Il peut avoir alors le rôle de passerelle entre les interfaces des applications et le Web.

Dans [54] nous décrivons comment utiliser FIGUE pour l'affichage des principaux éléments MathML2.0; on trouvera aussi à cette adresse les codes sources correspondants. Une expérience d'utilisation de FIGUE comme composant d'affichage de MathML pour différentes applications est en cours : nous collaborons avec le professeur Paul Wang [36] de l'université de Kent State aux États Unis pour l'utilisation de FIGUE dans leur interface Dragonfly pour l'accès IAMC[7] pour afficher et manipuler des formules encodées en MathML(voir IAMC Framework Project [36]).

3 - Présentation des formules mathématiques dans différentes langues

Nous avons aussi étudié les problèmes que pose l'affichage des formules mathématiques dans différentes langues et systèmes d'écriture, en particulier en l'arabe. Dans le cadre de cette thèse, nous proposons une approche générale pour manipuler les formules mathématiques dans un contexte bidirectionnel (mélange des directions d'écriture) et nous avons étendu FIGUE pour implémenter cette technique. Le résultat de ce travail est utilisé pour présenter des explications de preuves mathématiques en arabe dans le système PCOQ. Les explications de preuves mathématiques en langue naturelle [25] étaient jusqu'à présent uniquement en français ou en anglais.

Les implémentations actuelles de MathML permettent uniquement l'affichage de formules mathématiques de gauche à droite. Nous collaborons avec le professeur Stephen Watt, invité expert du groupe de travail W3C[8] autour de MathML, dans le but de définir l'usage de la norme MathML pour l'affichage de droite à gauche et l'affichage bidirectionnel des formules mathématiques. Nous utilisons FIGUE comme support d'implémentation de ces propositions de recommandations.

Les résultats de ces différents travaux de thèse ont fait l'objet des publications sui-

---

6. MathML utilise deux types de description des formules : les éléments de présentation MATHML-*présentation* et les éléments de description sémantique (contenu) MATHML-*contenu*

7. Internet Accessible Mathematical Computation

8. World Wide Web Consortium

vantes :

- une communication à la conférence CARI2000[9] intitulée “Affichage interactif, bidimensionnel et incrémental de formules mathématiques”. Dans cet article [59], nous présentons notre librairie FIGUE et comment nous l’avons étendu pour introduire l’affichage des formules mathématiques.
- un article accepté dans la revue Arima qui sera publié en fin décembre 2002 et qui est aussi accessible en rapport INRIA: ”Affichage et manipulation interactive de formules mathématiques dans les documents structurés”. Cet article [60] décrit comment manipuler les documents structurés contenant des formules mathématiques et présente un exemple d’utilisation des documents structurés pour présenter les preuves formelles.
- une communication à la conférence ASCM 2001[10] (Matsuyama, Japon) intitulée “FIGUE: Mathematical Formula Layout with Interaction and MathML Support”. Cet article [61] présente l’extension de FIGUE pour manipuler des formules MathML.
- une communication au workshop IAMC’2001[11] (London Ontario, Canada), à ISSAC 2001 intitulée “The Mariage of MathML and Theorem Proving”. Cet article [62] décrit notre approche pour utiliser MathML dans le cadre des environnements de preuves formelles.
- une communication à la conférence internationale MathML 2002 (Chicago)”Formal Mathematical Proof Explanations in Natural Language Using MathML: An Application to Proofs in Arabic”. Ce résumé [63] montre comment nous pouvons utiliser notre expérience pour développer des explications de preuves en langue arabe et définir l’usage de MathML pour l’affichage de droite à gauche et l’affichage bidirectionnel des formules mathématiques.
- une communication à la conférence CARI 2002 (Yaoundé, Cameroun) intitulée “Affichage et diffusion sur Internet d’explications en langue arabe de preuves mathématiques”. Cet article [59] présente notre approche générale pour manipuler les formules mathématiques dans un contexte bidirectionnel (mélange des directions d’écriture) et montre comment nous avons étendu FIGUE pour implémenter cette technique.

Plan de lecture

Dans le chapitre 1, nous présentons le contexte scientifique et les divers outils existant ayant un lien avec notre travail de thèse : les éditeurs WYSIWYG de mathématiques et les interfaces graphiques pour les mathématiques. Nous décrivons aussi l’état actuel des mathématiques sur le Web et dans les documents multilingues.

Dans le chapitre 2, nous présentons un modèle de formatage des objets structurés du document. Nous décrivons la structure de boîtes utilisée par ce modèle et les algorithmes de formatage correspondants.

9. Colloque Africain sur la Recherche en Informatique
10. Asian Symposium on Computer Mathematics
11. Internet Accessible Mathematical Computation

Dans le chapitre 3, nous présentons notre moteur d'affichage FIGUE et les différentes techniques utilisées pour son implémentation qui sont basées sur le langage de boîtes décrit dans le chapitre 2. Nous présentons ses principales fonctionnalités : formatage bidimensionnel, incrémentalité, interaction. Enfin, nous montrons comment FIGUE est utilisé actuellement en tant que brique de base de l'interface graphique PCOQ.

Dans le chapitre 4, nous décrivons en détails les algorithmes de formatage pour chaque type de boîtes FIGUE. Notons que ce chapitre est très technique et peut être laissé de côté dans une première lecture.

Dans le chapitre 5, nous décrivons en détail comment MathML est utilisé dans notre système pour représenter les preuves formelles et son rôle pour communiquer les preuves (buts et scripts) sur le Web en utilisant des navigateurs compatibles avec MATHML-*présentation*. Nos programmes permettent aussi de manipuler des données présentées par MATHML-*contenu*.

Enfin, dans le chapitre 6, nous proposons une approche générale pour manipuler les formules mathématiques dans un contexte bidirectionnel couvrant toutes les familles de formules mathématiques (style latin, style égyptien et style marocain). Le résultat de ce travail est utilisé pour présenter des explications de preuves mathématiques en arabe (texte arabe incluant des formules mathématiques) dans le système PCOQ.

Chapitre 1

Contexte scientifique

Contents

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 parfois complexe, comme les exposants, indices, fractions, matrices, et intégrales.

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 le système TEX [44] qui permet un formatage de grande qualité;
- les éditeurs WYSIWYG (*What You See Is What You Get*).

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 (les éditeurs WYSIWYG). La section suivante présente le contexte scientifique et les divers outils existant dans cette catégorie.

1.1 Éditeurs wysiwyg de mathématiques: contexte et solutions existantes

Nous nous intéressons particulièrement aux éditeurs donnant à l'utilisateur un support pour les mathématiques. Certains éditeurs WYSIWYG intègrent directement les formules mathématiques comme Edimath [71, 72], Publisher [70], FrameMaker [31], Grif [73] et Word [98]. 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 [93], MathWriter [24], Expressionist [41], MathType [28], EquationBuilder [84], JOME [29] (il se base sur la technique Java Bean).

Les systèmes dédiés à la production de document, comme Grif [73] ou comme IN-Form [91], 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 [22], 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. Parmi les derniers éditeurs structurés, notons Amaya [92] éditeur de pages web et navigateur Web disposant d'une interface WYSIWYG pour l'édition des pages Web incluant des formules mathématiques.

Cependant, 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. Le système TeXmacs [88], éditeur WYSIWYG de textes scientifiques allie la qualité d'impression de TeX avec une interface conviviale. TeXmacs permet à la fois l'édition de textes TeX et de formules, et il fait aussi l'interface avec plusieurs logiciels libres de calcul numérique ou formel (comme Maxima, Mupad, Pari ...). TeXmacs est un exemple de logiciel libre de traitement de textes qui sait calculer (voir figure 1.1).

En dehors des éditeurs, il existe une autre catégorie d'outils manipulant des formules mathématiques : les interfaces pour outils mathématiques. Des expérimentations de telles interfaces ont tout d'abord été menées dans le domaine du calcul formel : interface spécifique à Maple [75], ou plus généralement interface pour le calcul symbolique [39], dont les idées ont ensuite été largement reprises [80, 40] : 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 [39]) 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 [86] ou Centaur [17]). Basés sur des descriptions syntaxiques et sémantiques d'un langage donné, ces outils

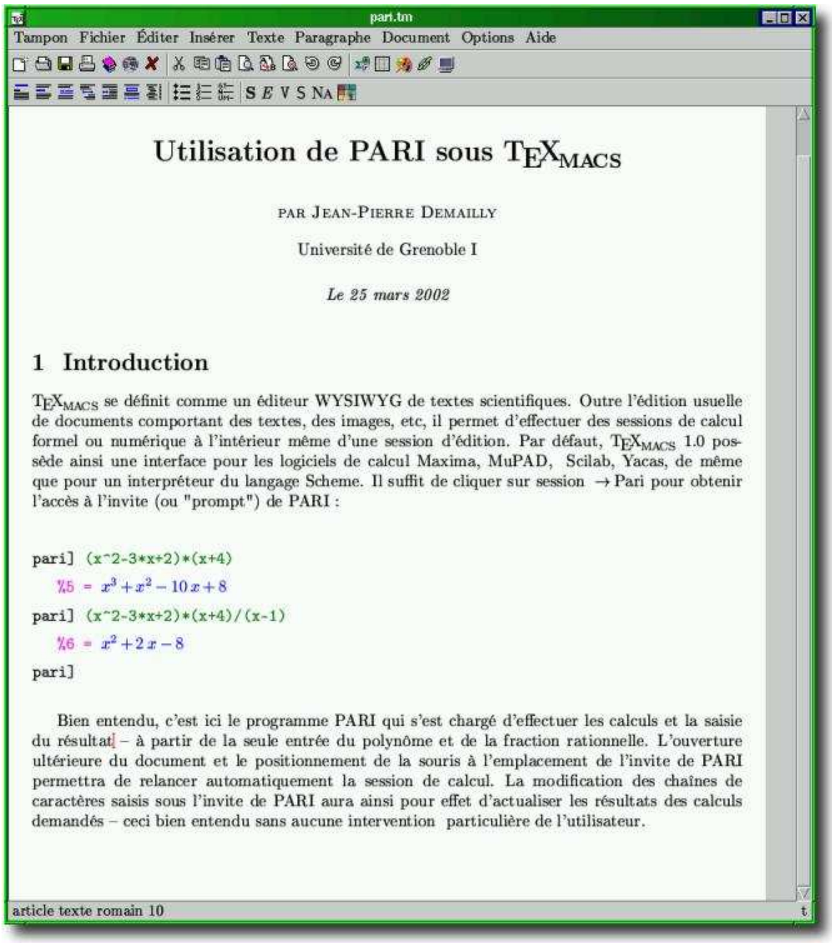


Fig. 1.1 – *Un exemple d'une fenêtre TeXmacs montrant l'interface de TeXmacs avec le système de calcul Pari.*

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 [89, 14, 13], qui sont des exemples d'outils symboliques manipulant des mathématiques autres que les outils de calcul formel.

1.2 Interfaces graphiques pour les mathématiques

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 sont aussi riches de fonctionnalités pour des utilisateurs experts. 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 [79]. La principale caractéristique de cette interface est l'édition en deux dimensions de formules mathématiques. L'interface GI/S [101] 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 tracés 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 [38] permet à un utilisateur expert d'adapter l'interface pour un besoin spécifique et de la connecter à d'autres logiciels externes. Mathematica [19] 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. Le système de calcul formel Maple utilise aussi cette notion de cahier interactif pour représenter ses données. Enfin, citons Emath [5] qui est un composant 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 [23] et IPE [74] (Interactive Proof Editor). Ces deux interfaces ont introduit l'édition structurée pour maintenir et éditer des preuves. CTCOQ [13], une interface utilisateur pour le système de preuves COQ [35], utilise l'environnement de programmation Centaur [17] et a introduit de nouvelles idées d'interaction, comme *proof-by-pointing* [14], *drag-and-drop* [12] 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 [8]. D'autres interfaces graphiques ont été conçues pour d'autres systèmes de preuves : XIsabelle [65] une interface pour le système Isabelle [67], TkHOL [83] interface pour le système HOL [32] et le système graphique JAPE [16]. Ces systèmes dépendent de boîtes à outils graphiques orientées texte et sont moins efficaces pour la manipulation des formules à la souris. Pcoq [2] en cours de développement, reprend les idées de son prédécesseur CTCOQ en introduisant des améliorations au niveau de la manipulation des objets de preuves (voir pour plus de détails la section 3.2.2).

1.3 Les mathématiques sur le Web

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 le Web. 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, ...). L'utilisation du Web par les scientifiques (en recherche et en industrie) augmente chaque jour et il est devenu important de pouvoir inclure des formules mathématiques dans les pages Web. Dans ce but, beaucoup d'efforts ont été consacrés pour fournir un standard des mathématiques sur le Web et aussi pour développer des outils pour manipuler et interagir avec ces formules. D'autre part quelques développements permettent d'effectuer des calculs sur le Web : citons par exemple JavaMath [81] qui est un composant logiciel permettant à des programmes en Java de réaliser des calculs en faisant appel à des moteurs de calculs existants et IAMC [95] qui est un composant distribué facilitant la connection entre des applications mathématiques différentes (suivant l'architecture client/serveur).

Pour pouvoir manipuler des notations mathématiques sur le Web, deux tâches sont essentielles : d'abord il faut avoir un format standard pour représenter les formules mathématiques et ensuite avoir le moyen d'adapter et d'étendre les outils existants (comme les navigateurs Web) pour manipuler et présenter ces données mathématiques.

Standards et protocoles de communication des formules

Pour représenter des notations mathématiques, plusieurs formats et protocoles d'échanges de formules mathématiques existent avec des systèmes comme OpenMath [26], MathLink [97] pour le système Mathematica, et plus récemment MathML [94], dédié aux mathématiques sur le Web. 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, etc), mais jusqu'ici sur le Web, le format le plus couramment utilisé pour les formules mathématiques est le format *image*.

En attendant la maturité de la nouvelle technologie Web autour de MathML, représenter les formules sous forme d'images reste une solution acceptable. Actuellement beaucoup de technologies produisent une très grande qualité graphique des formules mathématiques comme le système MathType [56, 90] qui offre la possibilité de générer des documents en format HTML + GIF à partir des documents mathématiques (utilisant JavaScript et CSS (Cascading Style Sheets)). L'inconvénient majeur de représenter les formules mathématiques en format d'image est la production des formules statiques sur le Web(sans structure) pour lesquelles il est impossible par exemple de changer la taille ou la couleur; de plus ces formules ne peuvent pas être utilisées par d'autres applications (indexées ou manipulées par des systèmes de calcul symbolique).

Actuellement, MathML est le format standard dédié au mathématiques sur le Web.

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).

Techniques utilisées pour manipuler les formules sur le Web

MathML est basé sur la technologie XML et d'autres spécifications W3C comme CSS (Cascading Style Sheets) et il est totalement intégré aux technologies Web. Il est possible aujourd'hui d'utiliser des règles de transformation du langage XSLT [100] pour afficher des formules MathML et d'avoir des notations mathématiques standard [21]. Pour utiliser cette technique, il suffit de copier les feuilles de stylessheet standards [99] pour MathML sur le serveur avec les documents contenant du MathML sans se soucier d'apprendre le langage XSLT.

Pour manipuler des mathématiques sur le Web, plusieurs types d'applications différentes s'intéressent au standard MathML : navigateurs Web, systèmes de calcul, éditeurs structurés et éditeurs de textes. Pour adopter MathML, ces systèmes utilisent différentes techniques de mise en ouvre :

- les plug-ins sont des programmes ajoutés au navigateur Web pour ajouter de nouvelles fonctionnalités, comme par exemple interpréter les balises du langage MathML.
- le contrôle ActiveX est une technologie OLE (Liaison et Incorporation d'Objets) de Microsoft adaptée au HTML (par exemple ajouter un composant pour manipuler des formules).
- les applets Java sont des programmes écrits en Java qui peuvent être insérés à des navigateurs Web pour être exécutés par la suite. Il est possible de produire par exemple une applet Java pour manipuler des formules MathML.
- MathML en natif dans les navigateurs comme dans Amaya [1], ainsi que dans les dernières versions de Mozilla [58].

Nous vous proposons un tour d'horizon des implémentations existantes autour de MathML classées selon les communautés scientifiques. La liste détaillée des implémentations actuels est sur la page Web de W3C (*http://www.w3.org/Math/iandi/*) et les différentes techniques pour manipuler des mathématiques sur le Web sont résumés dans [57].

Parmi les navigateurs Web qui implémentent actuellement MathML, nous avons déjà cité navigateur/éditeur Amaya [1] du W3C, ainsi que les dernières versions de Mozilla [58]; Netscape, Techexplorer d'IBM [85] et Internet Explorer implémentent aussi MathML mais avec d'autres techniques (plug-in, utilisant le moteur d'affichage MathPlayer [55]).

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 [19, 53] est capable d'évaluer des expressions MathML et d'afficher le résultat dans son interface graphique. Le système Maple [51] et MathCad [52] implémentent aussi l'import et l'export du format MathML. Ces systèmes peuvent être utilisés comme des serveurs de calcul pour MathML. Une expérience de formalisation des termes de preuves du système COQ [35] utilise MathML pour représenter les formules mathématiques incluses dans les termes de preuve [7]. MathML est également choisi pour représenter dans un environnement graphique les algorithmes de calcul numérique [50] en donnant la possibilité à l'utilisateur de définir ses propres algorithmes et structures de données.

D'autre part, quelques travaux autour d'éditeurs de textes intègrent MathML. Le système Omega [69, 64], une généralisation du système TEX, permet la génération automatique de code MathML à partir des formules mathématiques écrites en langage TEX ou Latex. D'autres travaux s'intéressent au passage de Latex vers MathML et inversement comme TeX4ht [87] (un interpréteur de DVI hautement spécialisé) ou le système xmltex [20] permettant d'éditer du MathML dans TEX. Amaya [1] permet l'édition structurée de MathML sur le Web. MathType [56, 90] est un éditeur d'équations mathématiques et un outil d'affichage pour MathML. WebEq [96] 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 le Web.

Les preuves formelles sur le Web

Dans le contexte des interfaces graphiques pour les systèmes de preuves, les preuves formelles sont généralement composées de formules logiques (le but à prouver) et des commandes (script) pour découper les buts en sous buts plus simples et construire la preuve du but initial. Les interfaces graphiques pour les systèmes de preuves ont besoin de présenter les formules mathématiques apparaissant à la fois au niveau des commandes et aussi au niveau des instructions. Le développement de preuves sur machine devient plus facile en utilisant des notations mathématiques appropriées : présenter les formules mathématiques d'une manière proche des notations utilisées en littérature mathématique augmente la lisibilité des formules mathématiques et accélère d'une façon importante le processus du développement des preuves.

Une fois que la preuve est complète, sa diffusion représente une étape importante de son cycle de vie. Dans ce cas, une bonne présentation des preuves est souhaitable. Une preuve peut être diffusée dans un but didactique ou dans le cadre d'un contexte industriel, par exemple, pour faire contrôler un développement formel par une troisième partie dans le but d'une certification ou d'une évaluation de sécurité.

Nous avons le choix entre communiquer la forme compilée d'une preuve (comme dans le projet HELM [33]) ou le code source du script. Ces deux choix peuvent être comparés, dans le domaine de la programmation, au choix entre la distribution des librairies compilées ou la distribution du code source. Dans notre contexte, nous avons choisi plutôt de distribuer les scripts de preuves en donnant la possibilité de présenter les commandes en langage naturel proche des preuves standards écrites sur papier par des mathématiciens. Dans les deux cas (script ou forme compilée), les notations mathématiques sont importantes pour augmenter la lisibilité et la compréhension des preuves.

1.4 Les mathématiques dans les documents multilingues

Jusqu'à présent la diffusion des documents scientifiques se faisait en général en anglais, mais avec l'internationalisation, ces documents circulent dans différentes langues. Un grand besoin d'outils pour manipuler ces documents dans différentes langues, systèmes d'écriture ou conventions culturelles est apparu. En particulier, nous avons besoin d'outils de formatage multilingues. Ces outils de formatage permettent d'afficher des documents comportant différents systèmes d'écriture ou langues et gèrent automatiquement les changements de direction en cas de langues écrites de droite à gauche comme l'arabe ou l'hébreu et en cas de langues écrites de haut en bas comme le japonais ou le coréen (voir la figure 1.2).

これは日本語の文章です。 ↓

de droite à gauche en vertical

(a)

← هده جملة باللغة العربية

de droite à gauche

(b)

Fig. 1.2 – *Un texte japonais écrit de droite à gauche dans le sens vertical (a) et un texte arabe écrit de droite à gauche (b). La traduction en français du texte japonais est "Cela est une phrase en anglais" et celle du texte arabe est "Cela est une phrase en langue arabe"*

Knuth [46] propose des algorithmes pour le formatage du texte linéaire mélangeant du texte écrit de droite à gauche et du texte écrit de gauche à droite. Ce travail a été la base du système ArabTEX [47] qui fait partie des outils non-interactifs de formatage multilingues. ArabTEX offre le moyen d'écrire du texte arabe, perse, kurde, pachto (afghane) et d'autres systèmes d'écriture écrits de droite à gauche dans TEX.

L'algorithme bidirectionnel d'Unicode [15] représente un standard général pour traiter le changement de direction dans du texte linéaire. Il regroupe les résultats obtenus par des expériences précédentes autour du formatage du texte multilingue. Dans le cas WYSIWYG, plusieurs problèmes sont posés pour le traitement du texte multilingues et sont décrits dans [11, 10]. Le système ditroff/ffortid [82] est un exemple de système WYSIWYG pour le formatage du texte arabe.

L'algorithme bidirectionnel d'Unicode traite uniquement le changement de direction du texte sans objets complexes comme les formules mathématiques. ArabTEX et son extension arabemath [48, 49] permettent d'écrire des formules mathématiques arabes orientées de droite à gauche (le style des formules actuellement en Egypte, voir figure 1.4) dans TEX. Dans [46], les auteurs proposent une approche d'implémentation permettant de présenter des formules mathématiques de gauche à droite dans du texte écrit de droite à gauche et de gérer le changement de direction (cela représente pour eux un bon exemple pour étudier le comportement de l'algorithme bidirectionnel avec des imbrications d'objets structurés). Par contre, le cas de formules écrites dans le sens inverse du texte et qui peuvent contenir aussi du texte ou des symboles n'est pas traité par les deux études citées ci-dessus.

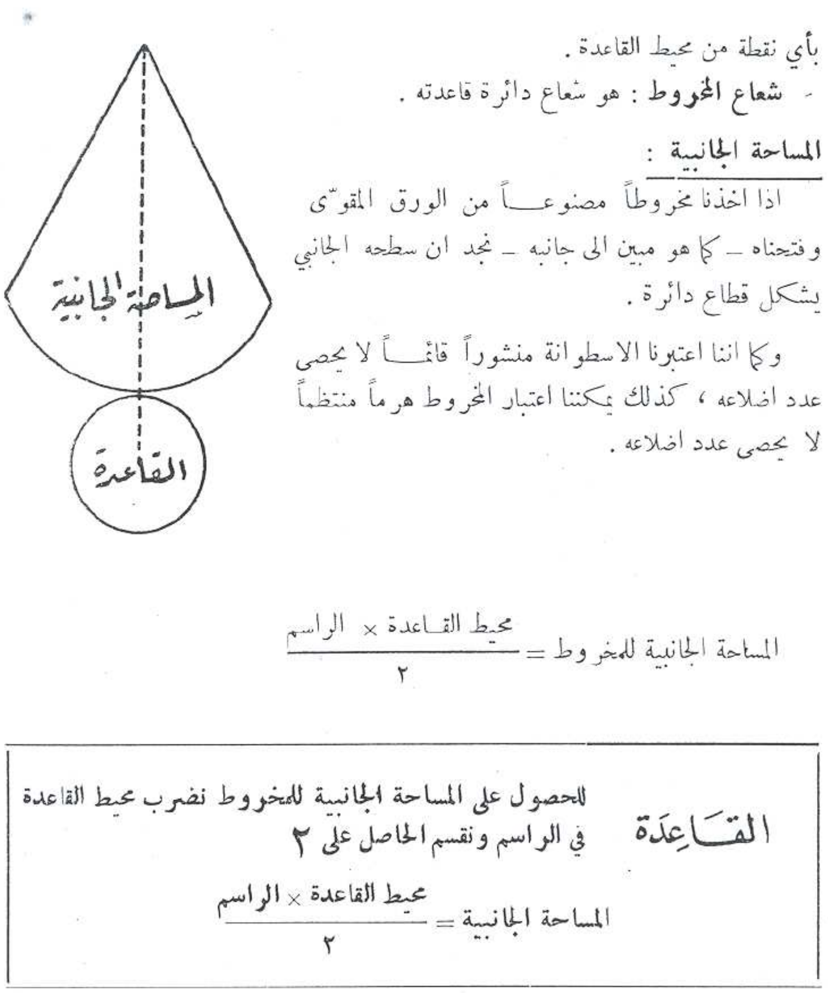
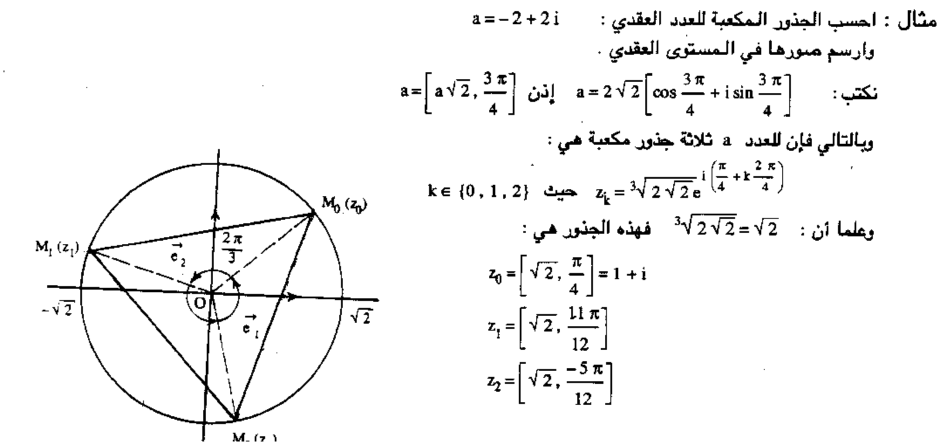


FIG. 1.3 – *Un texte mathématique en arabe orienté de droite à gauche suivant le style égyptien. Cet exemple est pris d'un livre de mathématiques pour niveau collège.*

Nous nous sommes intéressés à étudier le cas des formules mathématiques écrites dans le sens inverse du texte arabe qui à son tour contiennent du texte arabe (voir figure 1.4). Ce modèle de formules est utilisé actuellement au Maroc et en Algérie au niveau secondaire. Les règles d'affichage bidirectionnel de ces formules sont différentes de celles appliquées pour du texte simple. Cela est dû à la nature bidimensionnelle des formules mathématiques et à l'importance des relations spatiales dans les notations mathématiques.



مثال : احسب الجذور المكعبة للعدد العقدي : $a=-2+2i$

وارسم صورها في المستوى العقدي .

نكتب : $a=2\sqrt{2}\left[\cos\frac{3\pi}{4}+i\sin\frac{3\pi}{4}\right]$ إذن $a=\left[a\sqrt{2},\frac{3\pi}{4}\right]$

وبالتالي فإن للعدد a ثلاثة جذور مكعبة هي :

$z_k=\sqrt[3]{2\sqrt{2}}\,e^{i\left(\frac{\pi}{4}+k\frac{2\pi}{4}\right)}$ حيث $k\in\{0,1,2\}$

وعلما أن : $\sqrt[3]{2\sqrt{2}}=\sqrt{2}$ فهذه الجذور هي :

$z_0=\left[\sqrt{2},\frac{\pi}{4}\right]=1+i$

$z_1=\left[\sqrt{2},\frac{11\pi}{12}\right]$

$z_2=\left[\sqrt{2},\frac{-5\pi}{12}\right]$

FIG. 1.4 – *Des formules mathématiques extraites d'un livre de mathématique pour niveau baccalauréat au Maroc. Le texte est écris en arabe et les formules sont écrites de gauche à droite.*

Chapitre 2

Modèle de formatage des objets structurés

Contents

Afin de produire une image cohérente à partir des données brutes d'un document (morceaux de texte, formules mathématiques, graphiques, ...), il est nécessaire de positionner et d'ordonner ces objets. Nous appelons ce processus le **formatage**. Par analogie, le formatage s'apparente à un jeu de construction (ici le but est de construire l'image du document) à partir des différentes briques (ici les données brutes).

Dans ce chapitre, nous traitons uniquement le problème du formatage sans nous soucier de la structure logique du document. Prenons l'exemple d'une preuve mathématique composée de théorèmes et de lemmes : alors que le rôle du formatage est de présenter cette preuve sous forme de paragraphes avec un style graphique approprié (fontes, marges, interlignes, ...) la structure logique, quant à elle, n'est constituée que du lien abstrait entre les différentes parties de la preuve (un théorème est composé d'un énoncé et d'une preuve).

2.1 Structure de boîtes

Les documents que nous étudions se décomposent en objets structurés : pour ces documents structurés, le formatage se base sur la notion de boîtes [44]. Les objets du document sont représentés sous forme d'arbre de boîtes, où chaque nœud est une boîte englobant toutes ses boîtes filles et où les feuilles sont des entités atomiques (par exemple, les chaînes de caractères). Cette structure d'arbre rappelle non seulement les relations de dépendance (hiérarchie d'inclusion) mais aussi les relations spatiales entre les boîtes (objets graphiques) à afficher. Dans cet arbre, chaque boîte doit disposer ses boîtes filles selon le formatage désiré, calculer sa propre taille (en fonction de la taille des boîtes filles) et enfin décider de son alignement avec les autres boîtes du document. La figure 2.1 montre un exemple simple de représentation en boîtes d'une formule mathématique.

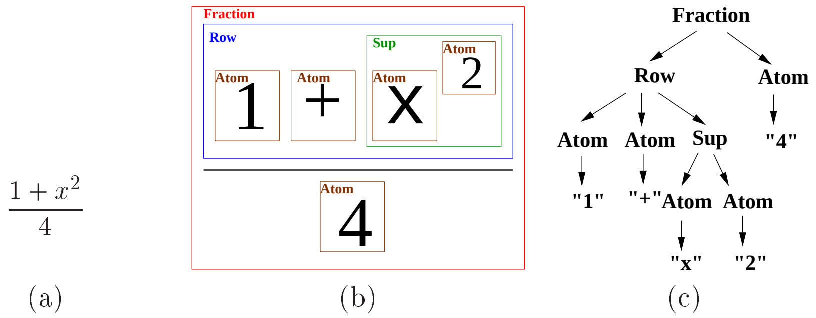


FIG. 2.1 – *(a) Formule mathématique. (b) Ensemble de boîtes correspondantes. (c) Représentation sous forme d'arbre de boîtes.*

Tout au long de ce mémoire de thèse, nous utilisons un vocabulaire précis pour décrire les relations de hiérarchie dans la structure d'arbre de boîtes. La figure 2.2 contient un modèle d'arbre de boîtes. Le vocabulaire qui est employé pour les boîtes de cet arbre est le suivant :

- b_i est une *boîte fille* de la boîte englobante B.
- B est le *père* de la boîte b_i.
- b_i est le frère suivant $b_{(i-1)}$.
- $b_{(i-1)}$ est le frère précédant b_i.
- b_i est le frère de toutes les boîtes $b_0 \ldots b_n$.

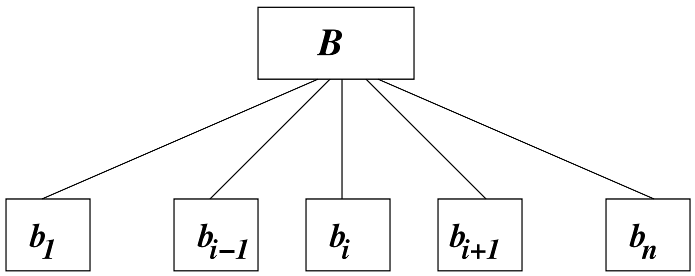


FIG. 2.2 – *Exemple d'arbre de boîtes : un père et ses boîtes filles.*

Notre but est de fournir un langage permettant de décrire comment chaque boîte calcule sa taille, comment elle dispose ses boîtes filles et comment elle détermine son alignement avec l'ensemble des objets du document. Une boîte possède les propriétés (ou attributs) suivantes :

– une origine (les coordonnées des origines des boîtes filles sont relatives à cet origine). L'origine d'une boîte donne sa position par rapport à son père.
– un rectangle englobant (taille de la boîte). Ce rectangle est décrit par la largeur (*Width*), la hauteur (*Height*), le coin du haut du rectangle à gauche (*UpperLeft*) et le coin d'en bas du rectangle à droite (*BottomRight*)
– un alignement défini par deux points : un point d'entrée (*EntryPoint*) qui est l'origine de la boîte et le point de sortie (*ExitPoint*). Ces deux points permettent de s'aligner par rapport aux autres boîtes.

De plus, chaque boîte possède ses propres propriétés sémantiques de formatage bi-dimensionnel qui servent à calculer sa taille, à gérer le mode interactif, et à gérer la coupure des lignes en cas de dépassement de la largeur de page. La figure 2.3 montre la représentation graphique d'une boîte avec les notations qui sont utilisées dans la suite de ce chapitre.

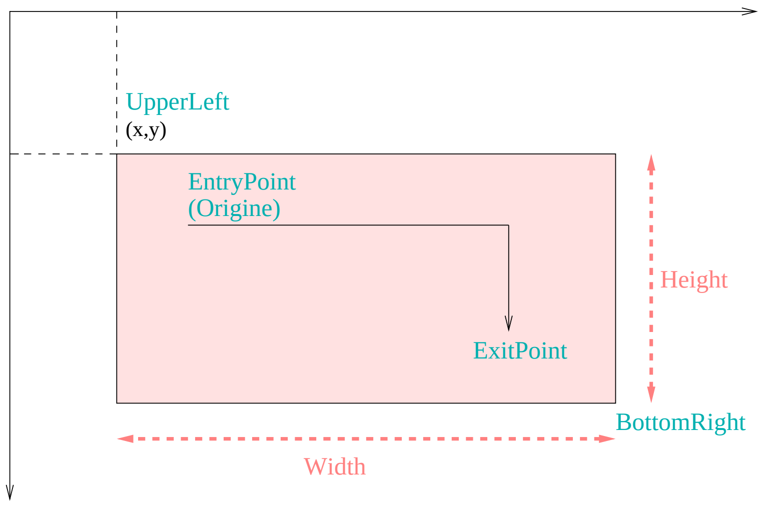


FIG. 2.3 – *Description graphique d'une boîte montrant ses principaux attributs.*

Les feuilles de l'arbre de boîtes constituent des entités atomiques (elles n'ont pas de filles). Dans le cas des documents mathématiques, ces entités atomiques correspondent en général à des chaînes de caractères (éventuellement composées d'un seul caractère). Chaque boîte de ce type calcule sa propre taille en fonction de sa propre fonte et de la chaîne de caractères qui la compose. De plus, les points d'entrée et de sortie sont alignés par rapport à la ligne de référence (voir la figure 2.4).

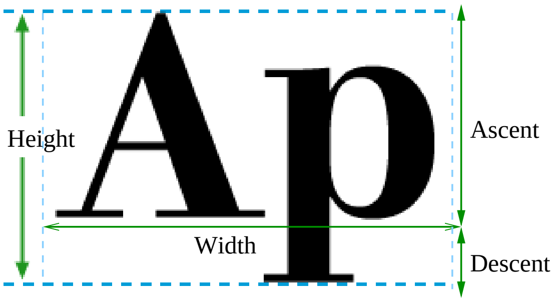


FIG. 2.4 – *Le calcul de la taille de cette boîte dépend des propriétés de la fonte choisie.* $UpperLeft = (0, Ascent)$ $BottomRight = (Width, Descent)$ $Height = Ascent + Descent$ $Width = f(fonte, chaineCaracteres)$

2.2 Algorithmes de formatage

Chaque boîte doit disposer ses boîtes filles, calculer sa taille, et décider de son alignement. Tout ceci correspond au formatage de la boîte. Le formatage prend aussi en compte les paramètres de la zone d'affichage, ici la largeur de la page (pour que les boîtes soient visibles). La hauteur de la page de formatage est toujours supposée infinie et la vision entière du document peut être gérée au moyen d'une barre d'ascenseur (ce qui diffère du contexte d'impression). Par contre un changement de la largeur de la page implique un reformatage du document. Dans la suite, nous décrivons l'algorithme général de formatage des boîtes, puis le formatage incrémental, l'intéractivité et enfin l'affichage des boîtes.

2.2.1 Algorithme général

Les algorithmes de formatage manipulent la structure d'arbre de boîtes. Ils prennent aussi en paramètre la largeur de page. L'algorithme général de formatage que nous proposons appelle récursivement le formatage des boîtes filles et à chaque nouvelle étape, vérifie qu'il reste assez d'espace pour disposer une nouvelle boîte (suivant la marge à droite restante)[1]. Dans le cas contraire, l'algorithme applique d'autres alternatives jusqu'à l'obtention d'un espace suffisant. Chaque boîte spécifie ses propres alternatives, par exemple, un paragraphe peut proposer le retour à la ligne et une boîte horizontale peut proposer de réduire sa taille en appliquant l'élision sur une ou plusieurs de ses boîtes filles. Les alternatives proposées par une boîte sont nécessairement compatibles avec le type de la boîte. Par exemple, une boîte *table* peut interdire les coupures au niveau de ses lignes. Chaque boîte fixe un ordre de choix pour ses alternatives et définit aussi comment chacune de ces alternatives (actions) sera exécutée. La liste des alternatives possibles est :

– Réduction de la distance entre les éléments horizontaux.

1. Ici, nous traitons uniquement le cas du formatage du document orienté de gauche à droite. Dans le chapitre 6, nous montrons comment l'algorithme de formatage se comporte dans le cas des documents orientés de droite à gauche.

- Réduction de l'indentation.
- Élision de la boîte entière.
- Élision de certaines boîtes filles.
- Coupure des lignes.

Nous avons besoin à chaque étape de cet algorithme de connaître la distance qui reste à couvrir. Par conséquent, nous introduisons une structure de *distance* indiquant l'espace restant. De plus, nous avons besoin de garder trace du décalage entre la boîte courante et la boîte qui précède, ainsi que le décalage entre la boîte courante et son père. Le choix d'une alternative peut se baser sur ces décalages : dans le cas d'une boîte horizontale, les décalages permettent de choisir la boîte fille sur laquelle on appliquera l'élision. Ces deux structures *distance* et *decalages* sont mises à jour à chaque nouvelle itération dans l'algorithme général.

Dans la suite nous décrivons les étapes de l'algorithme général de formatage. La description de ces étapes est la suivante :

Soit *distance* la structure représentant la distance restant à couvrir.
Soit *alternative* la structure représentant les actions à appliquer dans le cas d'une distance négative.
Soit *decalages* la structure gardant l'historique des décalages des boîtes filles par rapport à leur père et aussi par rapport à leur frère précédent.
Soit n le nombre de boîtes filles
Pour i **allant de** 1 à n **faire**
Début

- Vérifier si la distance restante est positive ($distance > 0$)
- **Si** ($distance < 0$) **alors** Choisir l'une des alternatives en fonction de l'historique des décalages déjà calculés jusqu'à l'obtention d'une distance positive.
- Formater la boîte b_i
- Mettre à jour la structure de *decalage* pour prendre en compte la dépendance entre la boîte b_i et son père et aussi son frère (dépendance entre les structures)

Fin
Une fois toutes les boîtes filles formatées, la boîte courante calcule sa propre taille et elle dispose ses boîtes selon le formatage désiré.

Le calcul de la taille pour chaque type de boîtes sera décrit en détail dans le chapitre 4.

2.2.2 Incrémentalité

En mode interactif, nous devons minimiser le temps de reformatage dû à la mise à jour ou à la sélection d'une ou plusieurs boîtes. Chaque modification d'une boîte (taille, origine, style, ...) doit être propagée dans l'ensemble de l'arbre de façon optimale afin de mettre à jour seulement les boîtes affectées. Ce formatage incrémental conduit à un affichage incrémental des objets du document où seules les zones modifiées ou touchées par la modification sont réaffichées. Reformater uniquement une zone d'un document permet de gagner en temps de positionnement de toutes les boîtes mais parfois le résultat peut être moins optimisé que le formatage complet du document.

L'algorithme de formatage incrémental que nous proposons est relativement simple. A chaque opération de mise à jour (ajouter une boîte, enlever une boîte ou remplacer une boîte), l'algorithme marque toutes les boîtes concernées par cette modification et ensuite il calcule la zone à reformater. Nous utilisons les propriétés des boîtes pour éviter un reformatage total : quand une boîte est marquée, on marque son père et parmi ses boîtes filles, uniquement les boîtes filles concernées par cette modification. Par exemple, dans le cas d'une boîte horizontale, nous ne reformatons qu'à partir de la première boîte fille modifiée (voir figure 2.5).

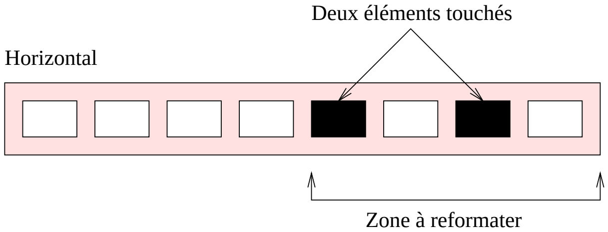


FIG. 2.5 – *Reformatage incrémental de la boîte Horizontal.*

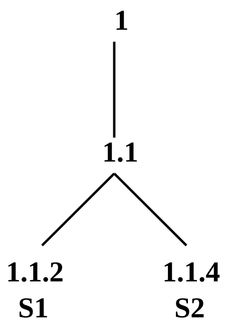
Nous utilisons une structure de chemin d'arbre *path* mémorisant les opérations de mise à jour à appliquer sur l'ensemble de l'arbre de boîtes. Chaque noeud du *path* possède un rang et une liste d'opérations relatives à la mise à jour (par exemple enlever une boîte). Le rang indique l'emplacement dans l'arbre de boîtes où la modification sera appliquée. Pour appliquer les modifications sur l'arbre de boîtes, nous parcourons en parallèle la structure *path* et la structure de boîtes. Il ne faut pas oublier de mettre à jour les autres paramètres de formatage comme le décalage entre les boîtes ou la distance qui reste à couvrir après chaque modification de boîtes.

2.2.3 Sélection

La plupart des interfaces modernes et interactives offrent la possibilité de sélectionner à la souris les objets affichés afin de les manipuler dynamiquement (par exemple, copier-coller). Nous proposons ici une technique de sélection qui est directement liée à la structure de l'arbre de boîtes. Elle traduit automatiquement les coordonnées de la souris en un emplacement (sous forme de chemin) dans la structure des boîtes et retrouve ainsi le sous-arbre correspondant aux objets sélectionnés. Le chemin d'une sélection est représenté par une structure de *path* où les noeuds indiquent l'endroit des boîtes à sélectionner et aussi le type des opérations de sélection à effectuer.

Une sélection a une couleur, un background et un degré de priorité par rapport à d'autres types de sélections. A chaque sélection, une action peut être appliquée, comme dessiner la plus petite boîte englobante contenant l'objet sélectionné. Il est possible de vouloir appliquer certaines opérations sur les sélections comme modifier, inverser ou réduire une sélection.

La figures 2.6 montre un exemple de path pour la sélection de deux boîtes différentes. Le chemin dans l'arbre de la première sélection nommée "sélection-rouge" est *1.1.2* et la deuxième sélection "sélection-blue" est *1.1.4*.



S1= Selection(
nom="selection-rouge",
couleur="rouge",
background="blanc",
priorité="Medium")

S2= Selection(
nom="selection-bleu",
couleur="bleu",
background="blanc",
priorité="High")

FIG. 2.6 – *Le chemin de la sélection de deux boîtes différentes dans la structure d'arbre de boîtes.*

2.2.4 Affichage

Une fois le formatage effectué, le style graphique (polices de caractères, couleur, arrière-plan, coordonnées) et les coordonnés de toutes les boîtes sont déjà définis. Chaque boîte est capable alors de s'afficher elle même avec son style graphique.

L'algorithme d'affichage parcourt simplement l'arbre de boîtes pour demander aux boîtes de s'afficher suivant leur style graphique et éventuellement dessiner de nouveaux symboles ou des caractères typographiques (comme par exemple le symbole racine). L'algorithme traduit aussi les coordonnées des boîtes relatives à leur père en coordonnées absolues et rafraîchit la zone d'affichage.

La sortie standard de l'affichage des boîtes peut être une sortie sur écran (affichage sur écran) ou une sortie pour l'impression (par exemple une sortie PostScript) ou une sortie de texte ASCII. Nous distinguons trois types de sorties : sorties graphiques, sorties textes et sorties impressions. Dans le même type de sortie PostScript, une sortie graphique SVG (*Scalable Vector Graphics*) est possible à partir de l'arbre de boîtes déjà formatées. Pour chaque type de sortie des paramètres graphiques spécifiques sont pris en compte comme le découpage en *pages* et le calcul du numéro des pages dans le cas d'une sortie Postscript. Dans le cadre de ce travail de thèse, nous nous sommes consacrés uniquement à l'affichage sur écran. D'autres travaux sur la sortie PostScript et sur la sortie SVG ont été réalisés indépendamment par notre équipe.

2.3 Discussion

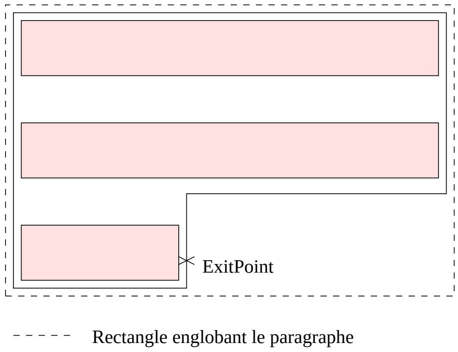
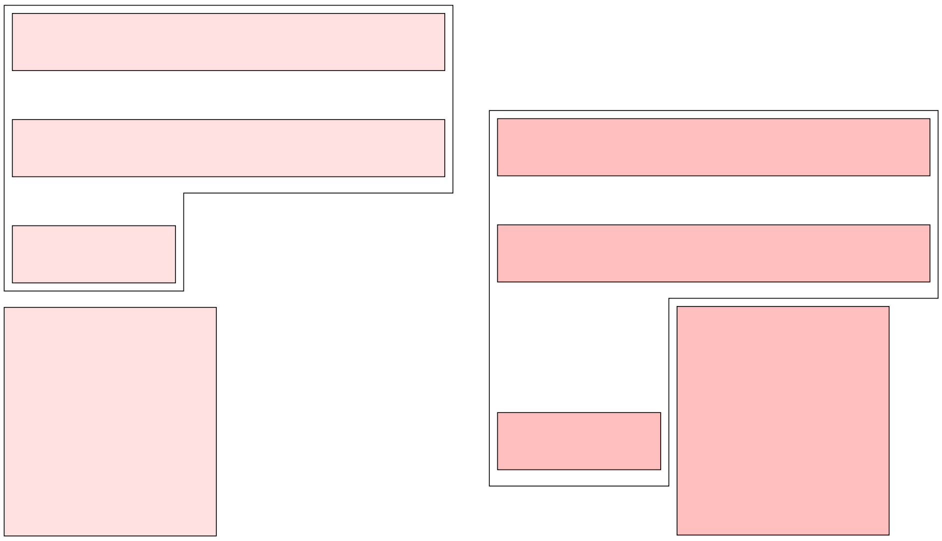


FIG. 2.7 – *La forme géométrique d'un paragraphe différente d'un rectangle.*

Dans le modèle que nous proposons, les boîtes ont une forme rectangulaire et déterminent leur alignement avec les autres boîtes du document en précisant leur point d'entrée et de sortie. En général, le rectangle de la boîte englobe tout son contenu. Tel est le cas de la plupart des boîtes mathématiques (fraction, racine, matrice ..). Mais d'autres types de boîtes, comme le paragraphe, peuvent avoir une forme géométrique différente d'un rectangle (une forme d'un paragraphe est illustrée dans la figure 2.7). En suivant notre modèle, la boîte englobant ce paragraphe est un rectangle mais son point de sortie se trouve à la fin de la dernière ligne. Si on se place uniquement au niveau du formatage du texte simple, ce cas ne pose pas de problème puisqu'il est possible d'aligner d'autres boîtes de texte sans qu'il n'y ait de chevauchement entre ces boîtes. Mais dans le cas où on mélange du texte avec des objets bidimensionnels comme les formules mathématiques, il peut y avoir chevauchement si la boîte mathématique est trop grande. Actuellement, pour contourner ce problème, nous faisons en sorte de ne pas donner d'ordre, en amont de notre moteur d'affichage, qui risquerait de poser problème. Par exemple, avec de grandes formules en forçant le passage à la ligne (on n'écrit pas dans le *trou* du paragraphe) ou en pratiquant l'élision.

L'algorithme de formatage de boîtes proposé ne traite pas ce problème. Une solution simple qui peut être envisagée est de détecter le chevauchement de boîtes et décider de déplacer la dernière boîte à droite ou faire un retour à la ligne pour positionner la dernière boîte (le résultat peut ne pas être très satisfaisant, voir figure 2.8.a). Une solution plus complexe est d'aligner la dernière ligne du paragraphe avec la boîte suivante et non pas aligner tout le paragraphe avec la boîte suivante (voir figure 2.8.b). Cela contredit notre conception dans laquelle le formatage d'une boîte dépend uniquement de ses boîtes filles et non pas de ses frères. Pour améliorer notre algorithme, nous envisageons d'introduire de nouvelles heuristiques pour traiter le mélange des boîtes de texte avec des boîtes mathématiques.



(a) (b)

FIG. 2.8 – *Solutions au problème de chevauchement de boîtes: (a) retour à la ligne, (b) la dernière ligne du paragraphe est décalée.*

Chapitre 3

Expérimentation et techniques employées avec FIGUE

Contents

FIGUE développé dans notre équipe de recherche, est un module de formatage et d'affichage interactif et portable utilisé dans le développement d'outils WYSIWYG comme les éditeurs de documents structurés ou plus généralement les 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).

Une version préliminaire de FIGUE (en Le-Lisp), intégrée au système CENTAUR, a été utilisée pour le développement de CAS/PI [38, 39], une interface utilisateur générique pour les systèmes de calcul formel. Actuellement, FIGUE (écrit totalement en JAVA) est utilisé pour le développement du système PCOQ [2], une interface graphique conçue pour le système de preuves COQ [35].

A l'origine FIGUE ne permettait que l'édition linéaire des documents (comme les programmes) à l'aide de ses boîtes graphiques de base (paragraphe, boîte horizontale et boîte verticale). Ce n'était pas suffisant pour manipuler des formules mathématiques qui sont naturellement bidimensionnelles (racine, matrice, puissance ...).

Ici, nous présentons la version de FIGUE que nous avons étendue pour permettre de présenter et de manipuler du texte avec des formules mathématiques. Les techniques utilisées pour implémenter notre système FIGUE sont basées sur le langage de boîtes décrit dans le chapitre 2. Nous présentons les principales fonctionnalités de FIGUE : formatage bidimensionnel, incrémentalité, interaction. Enfin, nous montrons comment FIGUE est utilisé actuellement en tant que brique de base de l'interface graphique PCOQ.

3.1 Présentation de la bibliothèque FIGUE

Le formatage et l'affichage employés par FIGUE sont bidimensionnels, par opposition aux méthodes de certaines interfaces qui obligent à linéariser les notations. FIGUE permet la disposition de boîtes dans un espace à deux dimensions (voir figure 3.1) en respectant la spécificité d'étirement de certaines boîtes en fonction de leur contenu (racine, intégrale, fraction) [1].

$$\begin{bmatrix} x & \cdots & y+2 & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ \frac{z}{4} & \cdots & 5 & \cdots & w \\ \vdots & & \vdots & & \vdots \\ 1 & \cdots & 3w & \cdots & 0 \end{bmatrix}$$

(a)

$$\Phi = \frac{1+\sqrt{5}}{2}$$

$$\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}$$

$$\Phi = 1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \cdots}}}$$

(b)

FIG. 3.1 – *Exemples de formatage bidimensionnel d'objets. (a) Disposition des éléments d'une matrice. (b) Affichage de formules mathématiques de structures différentes.*

3.1.1 Structures de boîtes

FIGUE se base sur la notion de boîtes (voir chapitre 2) 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 (par exemple à partir d'une description XML). Nous reprenons ici l'exemple de la figure 3.2 du chapitre 2 pour montrer la représentation en boîtes d'une formule mathématique.

Dans l'usage que nous en faisons couramment, l'arbre de boîtes est produit par une

1. Dans ce chapitre, 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.

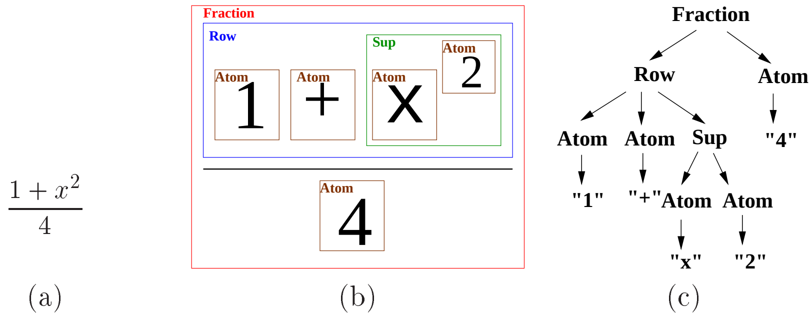


FIG. 3.2 – *(a) Formule mathématique. (b) Ensemble de boîtes correspondantes. (c) Représentation sous forme d'arbre de boîtes.*

description d'un langage abstrait, appelé PPML (*Pretty Printing Meta Language*) [37]. Ce formalisme PPML était proposé par le système CENTAUR pour exprimer l'affichage des données (objets) structurées.

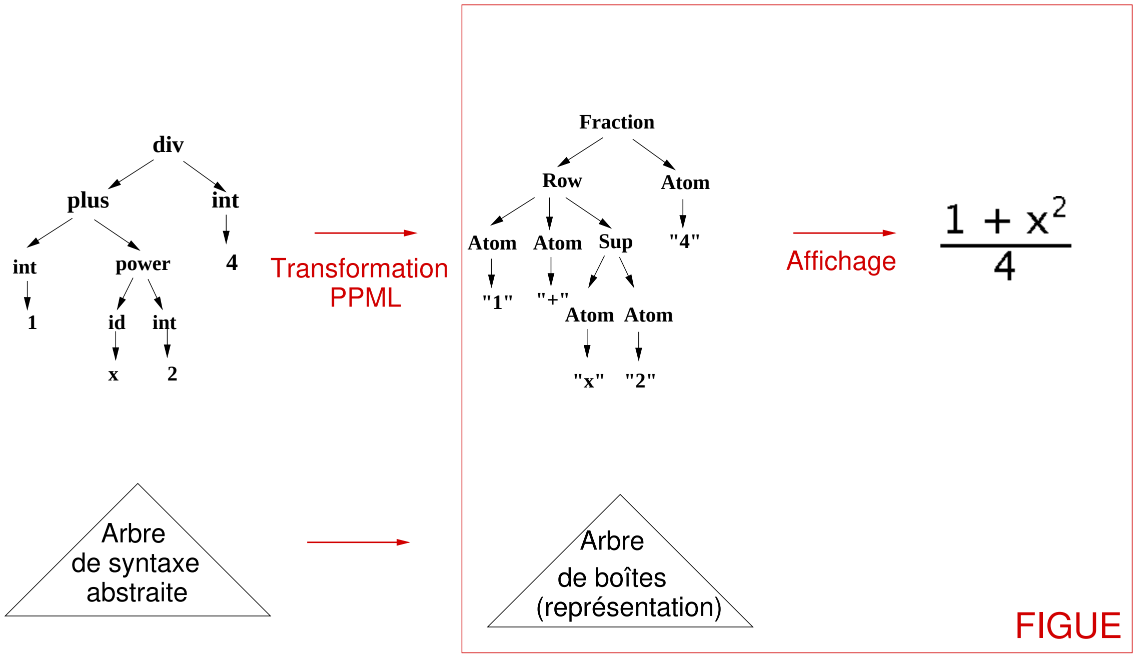


FIG. 3.3 – *Transformation d'un arbre de syntaxe abstraite représentant une formule mathématique en un arbre de boîtes* FIGUE*, à l'aide de règles* PPML.

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 3.3 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'une boîte 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). La boîte *Row* permet de formater ses boîtes filles en mode horizontal et applique des règles de coupure si la boîte englobante dépasse la largeur de la zone d'affichage.

Les règles permettant de transformer l'arbre de syntaxe abstraite de l'exemple de la figure 3.3 en un arbre de boîtes FIGUE sont décrites comme suit :

$$\begin{array}{lll}
\text{div}(*x, *y) & \rightarrow & [<\text{fraction}> \;\; *x \; *y] \\
\text{power}(*x, *n) & \rightarrow & [<\text{sup}> \;\; *x \; *n] \\
\text{int } *x & \rightarrow & [<\text{atom}> \;\; \text{identitypp}(*x)] \\
\text{id } *x & \rightarrow & [<\text{atom}> \;\; \text{identitypp}(*x)]
\end{array}$$

L'utilisateur peut spécifier ses propres règles de transformations PPML pour avoir ses propres notations. Par exemple, un déterminant ayant par défaut un affichage sous forme fonctionnel ($det3.3\ a\ b\ c\ d\ e\ f\ g\ h\ i$) peut être affiché sous forme d'une table (voir figure 5.4).

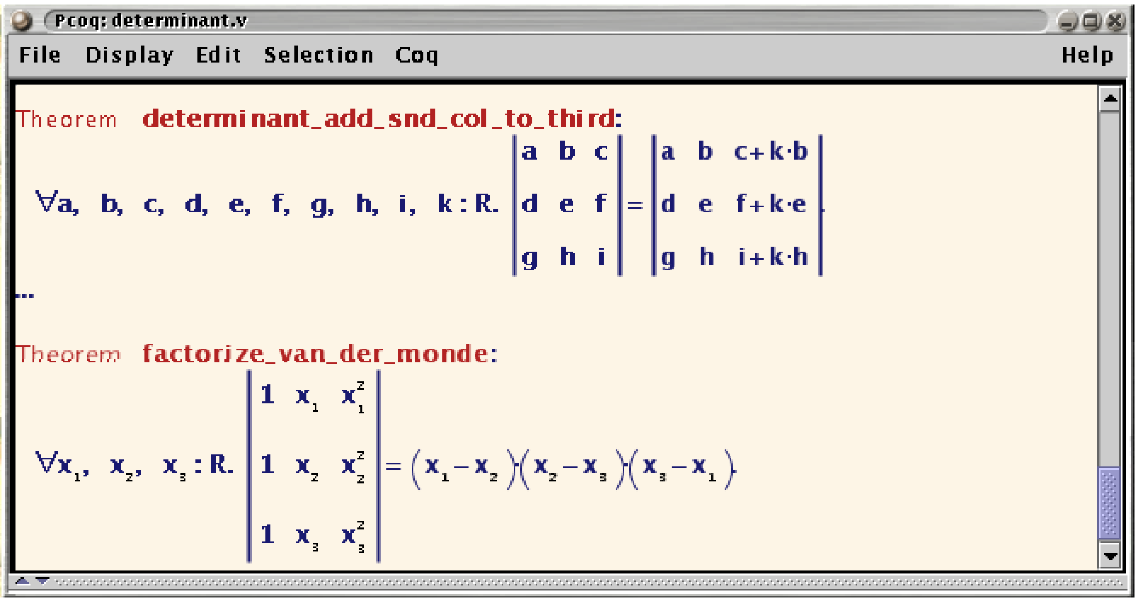


FIG. 3.4 – *Exemple des notations mathématiques: algèbre linéaire.*

3.1.2 Compilateur PPML

Le compilateur PPML est généré à partir de la définition de la syntaxe du langage PPML. L'analyseur syntaxique générique ANTLR[2] [76, 77] est utilisé pour produire le compilateur PPML. La définition de la syntaxe de PPML est décrite en annexe D. La description de PPML est dans *http://www-sop.inria.fr/croap/aioli/doc/aioli.html*.

Pour produire des arbres de boîtes FIGUE à partir de l'arbre de syntaxe abstraite, il faut suivre les étapes suivantes :

- Compiler le fichier de spécification de règles PPML pour obtenir un moteur de règles PPML correspondant à cette spécification.
- Le moteur de règles est appliqué ensuite sur l'arbre source pour produire l'arbre de boîtes. En appliquant ses règles, le moteur de règles indexe l'ensemble des règles appliqué sur chaque noeud de l'arbre et conserve le schéma de correspondance entre la structure de départ et la structure de boîtes. Ce schéma est appelé *skeleton*.
- Le lien entre les deux structures est maintenu par le moteur PPML : ayant le chemin d'une boîte graphique il est possible de retrouver le sous-arbre dans l'arbre de syntaxe à l'aide du schéma *skeleton*.

3.1.3 Boîtes graphiques

A l'origine FIGUE n'offrait que quelques boîtes graphiques. Les boîtes graphiques de base de formatage en FIGUE sont : *Atom*, *Horizontal*, *Vertical* et *Paragraphe*. Chaque boîte prend une liste de boîtes ou des atomes en argument et les formate selon un algorithme de formatage spécifique. *Atom* est une boîte de base qui produit des feuilles de l'arbre de boîtes (atomes qui n'ont pas de boîtes filles). La boîte *Horizontal* formate en mode horizontal la liste de ses boîtes filles. Elle prend en paramètres 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 3.5). La boîte *Vertical* dispose ses boîtes filles en mode vertical. Elle 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, après il retourne à la ligne et dispose à nouveau ses éléments horizontalement.

En utilisant ces boîtes de base, nous arrivons seulement à gérer une édition linéaire des documents (comme des programmes). Cette fonctionnalité n'est pas suffisante pour manipuler des formules mathématiques qui sont naturellement bidimensionnelles (racine carrée, matrice, intégrale,...). Pour cela, nous avons introduit dans FIGUE de nouvelles boîtes pour les formules mathématiques (*Racine*, *Matrice*, *Puissance*, ...). L'architecture extensible de FIGUE, nous a permis d'introduire ces nouvelles boîtes mathématiques sans difficultés.

2. ANTLR (ANother Tool for Language Recognition) est un outil qui permet de définir des analyseurs syntaxiques LL(k). Il offre un langage permettant de produire des compilateurs et des traducteurs à partir des descriptions de grammaire contenant du code C, C++ ou des actions Java.

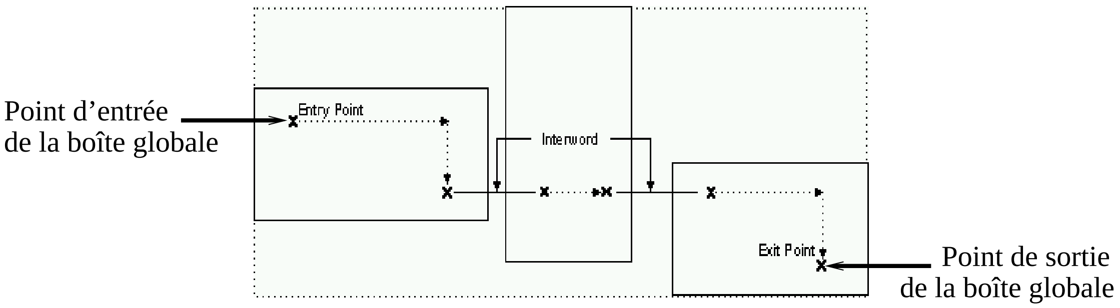


Fig. 3.5 – *La boîte Horizontal dispose ses boîtes filles horizontalement. Pour son alignement avec les autres boîtes, elle a le même point d'entrée que sa première boîte fille et le point de sortie de sa dernière boîte fille.*

Ces boîtes mathématiques disposent leurs boîtes filles selon la formule mathématique correspondante et au besoin dessinent les symboles nécessaires. Par exemple, la boîte *Racine* dessine le signe racine autour de sa première boîte fille en tenant compte de sa taille (voir figure 3.6) et dispose sa deuxième boîte fille de façon à avoir une racine n-ième.

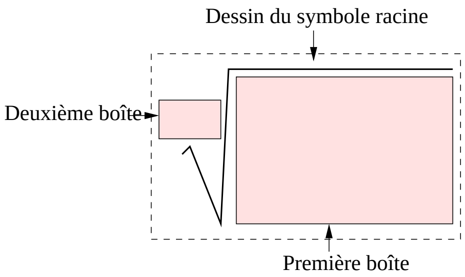


Fig. 3.6 – *Disposition des boîtes et le dessin de symboles pour la boîte racine (Le formatage de la racine n-ième).*

3.1.4 Formatage et affichage bidimensionnels

Pour le formatage, FIGUE manipule directement l'arbre de boîtes. L'algorithme de formatage parcourt cet arbre en mettant à jour les propriétés (ou attributs) graphiques des boîtes : origine, taille, alignement (voir figure 3.5). Les propriétés graphiques d'une boîte sont déterminées par la nature de sa boîte graphique et par les propriétés de ses boîtes filles (voir la description de cet algorithme dans le chapitre 2, page 28). L'algorithme de formatage prend aussi en compte les paramètres de la zone d'affichage, ici 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 boîte dispose de son propre algorithme de formatage qui détermine comment elle place ses boîtes filles par rapport à son origine, et qui détermine ensuite son alignement avec l'ensemble des autres boîtes (ses points d'entrée et de sortie - voir figure 3.5). Un soin particulier est accordé à l'efficacité des algorithmes de formatage des boîtes mathématiques. Par exemple, la disposition correcte des éléments d'une matrice requiert un algorithme en plusieurs itérations [3]. Dans le chapitre 4, les algorithmes spécifiques à chaque type de boîtes FIGUE sont décrits.

Une fois le formatage effectué, chaque boîte FIGUE est capable de s'afficher elle-même dans un style graphique donné (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 style graphique et éventuellement dessiner de nouveaux symboles ou des caractères typographiques (comme par exemple le symbole racine - voir figure 3.6).

3.1.5 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 : lorsqu'une boîte est déplacée ou modifiée, FIGUE 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 (voir section 2.2.2 pour plus de détails).

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

3.1.6 Interaction et sélection

FIGUE offre un puissant mécanisme d'interaction qui permet 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 3.7). 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

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 3.1a).

FIG. 3.7 – *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.*

entre l'interface graphique et des systèmes extérieurs effectuant des traitements et des calculs (des systèmes de calcul symbolique ou autres).

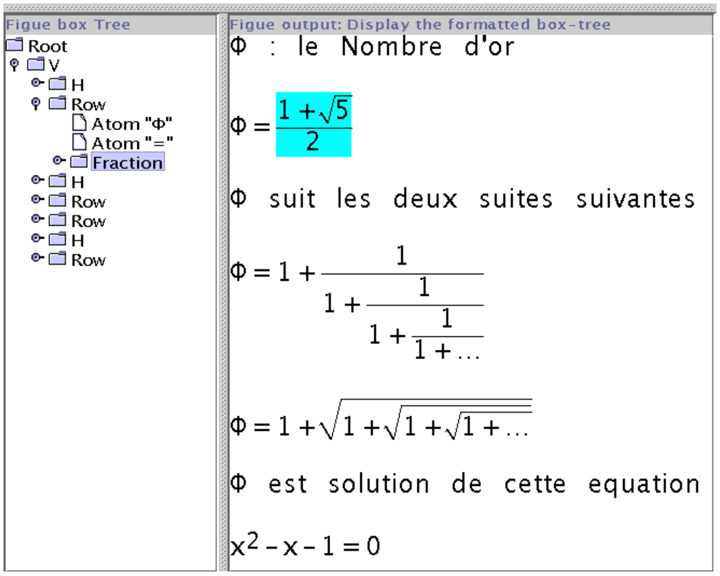


FIG. 3.8 – *Sélection des objets affichés par l'utilisateur dans la structure du document (partie droite), et calcul de leurs emplacements dans la structure d'arbre de boîtes par* FIGUE *(partie gauche).*

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 automatiquement 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 3.8). 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

4. On détermine la règle PPML qui est à l'origine de l'affichage de cette boîte en utilisant la donnée

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 de 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, en déterminant par exemple 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.

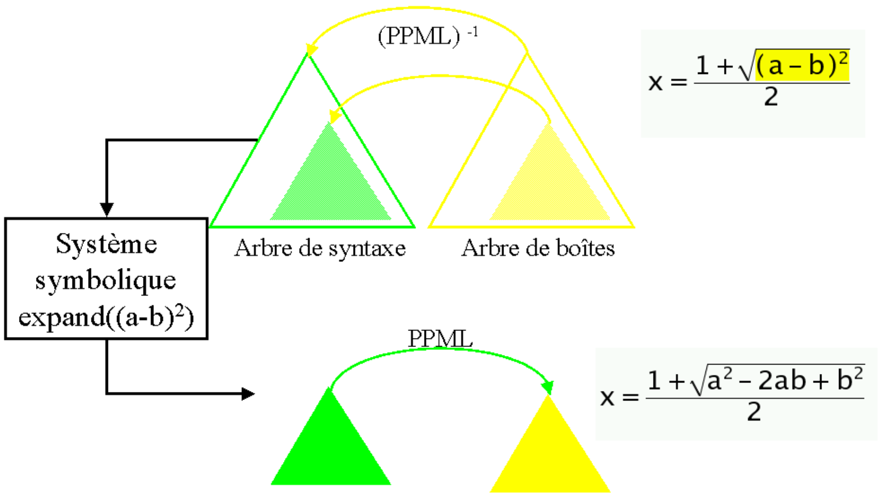


FIG. 3.9 – *Le mécanisme d'interaction permet d'obtenir la syntaxe des objets affichés afin de la communiquer au système de calcul symbolique et ensuite de réafficher le résultat du calcul.*

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 construire une sous-liste *virtuelle* des éléments de la diagonale sur laquelle l'interface pourra travailler

skeleton construite. Ensuite, le sous-arbre sélectionné correspond alors au schéma syntaxique de la partie gauche de la règle PPML comme si cette sous-liste existait vraiment dans l'objet structuré sous-jacent.

Il est également possible d'effectuer des opérations plus *complexes* qu'un copier-coller, comme développer une expression mathématique. La figure 3.9 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 3.1.5).

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 puissant de sélection pour effectuer des preuves par sélection et des transformations de formules (voir section 3.2.2).

3.1.7 Cas des formules mathématiques

Parmi les fonctionnalités dans des systèmes dédiés aux mathématiques (chapitre 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 [80] [4], comme la complexité et la diversité des règles typographiques [9], 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 [3]. Pour les dessiner, nous nous sommes inspirés du système METAFONT en LaTeX [42] [43] et des polices *outlines* [34] 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 3.10) 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

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.

typographié dans les règles de l'art. Cet objectif restera de toutes façons hors de portée tant que la résolution des écrans sera bien inférieure à celle de l'impression.

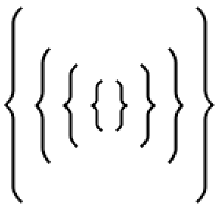


FIG. 3.10 – *Dessin des symboles mathématiques de tailles variables dans* FIGUE *: exemple de résultat obtenu pour l'affichage des accolades dans un contexte graphique comportant une grande fonte (taille 28).*

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 coupure et tenter de visualiser l'expression dans son intégralité. Certaines formules (telle les matrices) supportent mal les coupures.
- 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.

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.

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 boîtes graphiques puisqu'elle ne prend en compte que les positions des divers éléments graphiques et les actions de dessin. En utilisant une méthode similaire, lors d'un stage d'ingénieur, on a implémenté la traduction des arbres de boîtes FIGUE déjà formatées vers du SVG, le dialecte XML pour le graphique vectoriel. Une application de ce dernier travail consiste à utiliser du SVG pour visualiser les théories mathématiques développées en Coq. Un prototype qui a été développé utilise SVG pour représenter un graphe interactif contenant les différents fichiers d'une théorie avec leurs dépendances. Sélectionner un noeud du graphe permet de visualiser les définitions et théorèmes contenus dans le fichier sélectionné avec les notations mathématiques usuelles.

3.1.8 Bilan des fonctionnalités d'affichage et de manipulation

Le noyau de FIGUE offre aujourd'hui une panoplie de fonctionnalités 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 nouvelles boîtes graphiques, sous réserve d'implémenter les algorithmes de formatage et de dessin associés à ces nouvelles boîtes. En ce qui concerne l'incrémentalité et l'interaction, des méthodes par défaut sont fournies par FIGUE, mais dans certains cas ces méthodes peuvent être mal adaptées à la structure graphique ou à la sémantique d'une nouvelle boîte, et elles devront être fournies lors de l'ajout de la boîte (ce fut par exemple le cas pour la matrice lors du reformatage après modification d'un élément ou lors de la selection d'une diagonale).

3.2 Application : développement d'interfaces graphiques

Comme nous l'avons vu précédemment, FIGUE est un composant dédié 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 présentons notre approche générique pour le développement des interfaces graphiques pour les systèmes de calcul symbolique. Ensuite, nous présentons en détails l'interface graphique PCOQ qui suit cette approche générique et qui utilise FIGUE comme module d'affichage interactif.

3.2.1 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 personnalisables par l'utilisateur.

La figure 3.11 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'applique au 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 [27]). Ce protocole est utilisé pour envoyer des messages entre un environnement de programmation et des composants externes. Il permet de transférer des données

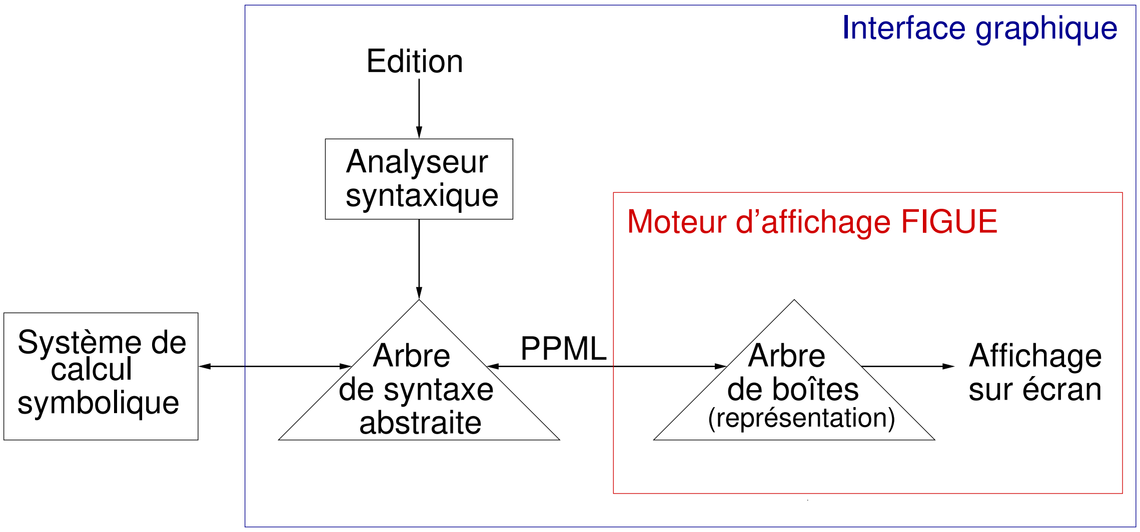


FIG. 3.11 – *Le schéma général des interfaces graphiques utilisant* FIGUE *pour les systèmes de calcul symbolique.*

structurées (texte ou arbre de syntaxe abstraite) sur lesquelles il est possible d'effectuer des opérations à distance (appel de fonctions ou émission de signaux). Un gestionnaire de messages traite les messages reçus par l'interface. Il est envisageable de transférer les données sous format XML. Au niveau de l'interface, les données peuvent être représentées en XML. Ce n'est pas forcément le cas au niveau des systèmes de calcul.

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 traduits en une structure de représentation (arbre de boîtes) afin d'être affichés par notre moteur d'affichage FIGUE. Le lien entre les deux structures (arbre de syntaxe abstraite et arbre de boîtes FIGUE) se fait grâce aux règles de traduction PPML (voir section 3.1.1). Ce lien permet un mécanisme puissant de sélection d'objets manipulés par l'interface.

L'interface graphique PCOQ respecte ce schéma général dans son environnement de travail graphique et interactif. La section suivante décrit en détails l'architecture de PCOQ, et explique comment PCOQ utilise FIGUE pour l'affichage des commandes et des formules mathématiques.

3.2.2 PCOQ : un exemple d'utilisation de FIGUE

PCOQ (c'est la version JAVA de CTCOQ [13] actuellement en cours de développement [2]) fournit un environnement de travail graphique et interactif pour le système de preuve COQ [35] 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 [14];
- un affichage paramétrable.

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 composants principaux, inspirés des concepts du système CENTAUR [17] :

- VTP (*Virtual Tree Processor*) : une machine abstraite permettant de définir un formalisme (définition par des types abstraits d'un langage), puis de spécifier et de manipuler des structures d'arbres en suivant ce formalisme.
- 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 [37]. L'utilisation principale de PPML est de générer un arbre de boîtes FIGUE à partir d'un arbre de syntaxe abstraite (VTP).
- GFXOBJ : une librairie d'objets graphiques (menus, fenêtres,...) permettant le développement d'applications interactives pour les arbres VTP.

Ainsi, AÏOLI traduit à l'aide de PPML 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 (voir section 3.1). Cet arbre de boîtes peut aussi être produit par un autre moyen (par exemple à partir d'une description XML, voir section 5.2.2).

Dans le cadre de cette thèse, PCOQ constitue une plate-forme de test pour notre moteur d'affichage FIGUE. Il nous a permis de tester les fonctionnalités de FIGUE et de regarder de près les différents besoins des utilisateurs pour la représentation des notations mathématiques des preuves.

Dans PCOQ, les données sont structurées. Le moteur d'affichage utilise les informations mises dans cette structure pour pouvoir représenter ces données. Le mécanisme d'affichage de PCOQ donne la possibilité d'ajouter et de paramétrer les notations mathématiques.

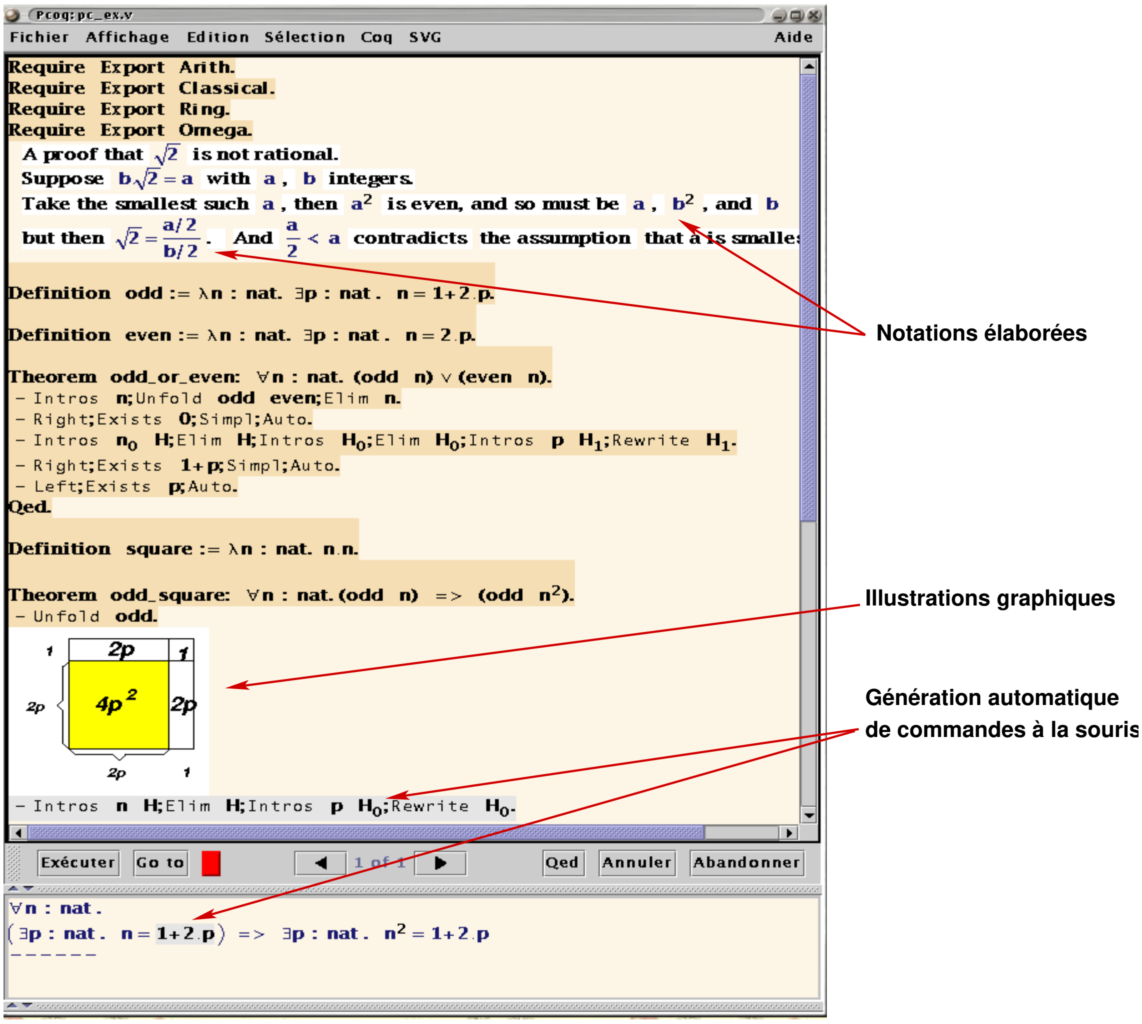


FIG. 3.12 – *Une session* PCOQ.

L'extension des notations mathématiques peut se faire à trois niveaux qui nécessitent trois différents niveaux d'expertise :

– Le moyen le plus simple est d'associer un modèle (schéma) d'affichage prédéfini à un modèle de structure (un identificateur) en utilisant une simple table de correspondance. Cette table peut être facilement éditée de façon interactive par les utilisateurs. Les utilisateurs peuvent visualiser directement l'effet des modifications de cette table. Par exemple, un utilisateur définissant un nouveau ensemble des nombres rationnels $\mathbb{Q}$ avec des fonctions d'addition et de division *Qplus*, *Qmult* et *Qdiv* aura la possibilité d'ajouter ces entrées suivantes dans la table de correspondance pour avoir ses propres notations, où la division sera affiché à l'aide d'une fraction :

Nom d'opérateur	Famille	Priorité à gauche	Priorité à droite	Nom de la fonte	Style de la fonte	Taille de la fonte	Texte
Qplus	infix	20	20	Dialog	PLAIN	12	+
Qmult	infix	25	25	Dialog	PLAIN	12	·
Qdiv	fraction	200	200	Dialog	PLAIN	12	

- Le deuxième moyen est d'utiliser le langage PPML pour décrire ses propres notations. Les notations préférées pour certaines fonctions ou relations mathématiques peuvent ne pas être décrites par une simple table de correspondance. Dans ce cas, l'utilisateur peut décrire ses choix de notations à l'aide des règles de transformation PPML où chaque règle associe une structure logique de données à une structure de boîtes décrivant le formatage correspondant. Prenons un exemple d'une règle PPML décrivant le choix d'affichage pour la fonction *iter* :

```
prettyprinter usernotations of vernac is

appc(ident "iter", formula_ne_list[*t, *f, *k, *v, **args]) ->
 [<h 0> "(" [<hv 1,0,0> [<sup> *f *k] *v (**args)] ")"];

end prettyprinter
```

  En utilisant les notations choisies dans l'exemple ci-dessous, toute fonction de cette forme
$$(iter\ t\ f\ k\ v\ a_1\ \cdots\ a_k)$$
  sera affiché de la façon suivante :
$$(f^k\ v\ a_1\ \cdots\ a_k).$$

- Le langage PPML permet uniquement d'associer à des schémas de formules des boîtes graphiques existantes. Un troisième niveau d'extensibilité est la possibilité d'ajouter de nouveaux constructeurs graphiques qui peuvent être utilisés ensuite dans notre interface graphique. Cela est possible grâce à l'extensibilité de notre moteur d'affichage FIGUE. Nous avons utilisé ce dernier moyen pour ajouter de nouveaux constructeurs mathématiques dans FIGUE pour obtenir un affichage conviviale de certaines notations mathématiques comme *table* ou *délimiteur*. Cette extension a fait l'objet du résultat de la première partie de cette thèse.

La figure 3.12 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. Grâce au mécanisme de sélection (décrit dans la section 3.1.6) l'utilisateur peut effectuer des preuves "à la souris" [14], 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* [12]) 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; e.g. on est dans un anneau).

Chapitre 4

Les algorithmes spécifiques de formatage

Contents

Dans ce chapitre, nous décrivons les algorithmes spécifiques de formatage pour chaque type de boîtes. Nous traitons uniquement le cas des boîtes implémentées dans FIGUE : paragraphe, boîte horizontale et verticale et les boîtes mathématiques (fraction, table, racine ...). Nous avons suivi les recommandations du standard MathML pour définir nos boîtes mathématiques avec les bons attributs et pour proposer les algorithmes de formatage correspondants.

4.1 Schéma général

Pour chaque type de boîte, nous décrivons une méthode spécifique lui permettant de calculer sa taille et de disposer ses boîtes filles. Cette méthode prend en entrée un vecteur de boîtes déjà formatées (boîtes filles) et elle les dispose par rapport à leur père selon le formatage désiré. Puis nous calculons la dimension de la boîte englobante. Nous décrivons le schéma de cette méthode ci-dessous :

Entrée: un vecteur de boîtes filles $[b_1 \ldots b_n]$ où chaque boîte est déjà formatée
Sortie: la boîte englobante (Width, Height, UpperLeft, BottomRight)
Algorithme: cet algorithme consiste à positionner chaque boîte fille b_i par rapport à leur père puis mettre à jour la boîte englobante. Cet algorithme détermine aussi l'alignement de la boîte englobante par rapport aux autres boîtes. Un schéma simplifié de cet algorithme est le suivant :
Soit B la boîte courante avec n boîtes filles
Soient $[b_1, \ldots, b_n]$ les boîtes filles
Pour i **allant de** 1 à n **faire**
début

- Positionner la boîte b_i par rapport à B
- Mettre à jour la taille de la boîte englobante pour qu'elle puisse contenir la boîte b_i

fin
Déterminer le point de sortie (ExitPoint)
Retourner la nouvelle boîte englobante

L'ordre de traitement des boîtes peut être différent de leur ordre logique dans la structure d'arbre (par exemple, pour le formatage d'une racine nième, l'opérande est traité après l'exposant de la racine nième). Notons que le point d'entrée (origine de la boîte) est déterminé lors du calcul de la position de la première boîte fille. Les autres boîtes filles seront positionnées par rapport à cet origine.

L'algorithme général de formatage de boîtes que nous proposons dans le chapitre 2 est récursif. Une fois toutes les boîtes filles formatées, la boîte courante calcule sa propre taille et dispose ses boîtes selon le formatage correspondant.

Le calcul de la taille des boîtes atomiques telles que les chaînes de caractères ne suit pas le même schéma que les autres types de boîtes. Les propriétés de la fonte courante (voir figure 2.4, page 28) sont utilisées pour déterminer la taille des boîtes de type *atome*. Les mesures de la fonte sont données par rapport à la ligne de référence (baseline) : ascent par exemple est la différence entre la ligne de référence et le haut de la boîte et descent est la différence entre la ligne de référence et le bas de la boîte.

A chaque étape de l'algorithme décrit ci-dessus, nous mettons à jour la dimension de la boîte englobante B pour qu'elle puisse contenir la nouvelle boîte fille b_i dans sa nouvelle position $P(x_i, y_i)$. Le calcul de cette nouvelle boîte englobante est le suivant :

Soit upper$X$:= max(UpperLeft$X[B]$,UpperLeft$X[b_i]$- x_i)
Soit upper$Y$:= max(UpperLeft$Y[B]$,UpperLeft$Y[b_i]$-y_i)
Soit bottom$X$:= max(BottomRight$X[B]$,BottomRight$X[b_i]$+x_i)
Soit bottom$Y$:= max(BottomRight$Y[B]$,BottomRight$Y[b_i]$+y_i)
UpperLeft$[B]$:= (upper$X$, upper$Y$)
BottomRight$[B]$:= (bottom$X$, bottom$Y$)

4.2 Notations et terminologies

Dans ce chapitre, nous reprenons les mêmes notations utilisées pour notre langage de boîtes vu dans le chapitre 2. Nous introduisons de nouvelles notations pour pouvoir décrire les algorithmes de formatage spécifiques à chaque type de boîte. Ces nouvelles notations sont :

(dx_i, dy_i)	la distance entre la boîte b_i et la boîte précédente b_{i-1}
ExitPoint$[b_i]$	le point de sortie de la boîte b_i
Origine$[b_i]$	l'origine de la boîte b_i
BottomRight$[b_i]$	le coin inférieur droit de la boîte b_i
UpperLeft$[b_i]$	le coin supérieur gauche de la boîte b_i
Width$[b_i]$	la largeur de la boîte b_i
Height$[b_i]$	la hauteur de la boîte b_i
Justif$[b_i]$	la justification de la boîte b_i, elle peut prendre trois valeurs RIGHT, CENTER et LEFT.
EntryType	le type d'alignement de la boîte englobante. Il peut être FIRST, LAST ou MIDDLE. FIRST signifie que la boîte englobant est alignée avec le point d'entrée de la première boîte fille, LAST signifie que la boîte englobante est alignée avec la dernière boîte fille et MIDDLE signifie qu'elle est alignée au milieu
$Ascent$	la distance entre la ligne de référence et le haut d'une boîte d'entité atomique pour une fonte métrique donnée
$Descent$	la distance entre la ligne de référence et le bas d'une boîte d'entité atomique pour une fonte métrique donnée

Au besoin nous introduirons de nouvelles notations pour définir des paramètres spécifiques à certains types de boîte.

4.3 Boîtes de base

Dans cette section, nous décrivons les algorithmes de formatage spécifiques aux boîtes de base : boîte horizontale, verticale et paragraphe.

4.3.1 Boîte Horizontale

Une boîte horizontale dispose ses éléments séparés par un espace blanc, de manière horizontale. L'algorithme décrit ci-après montre comment une boîte horizontale calcule sa taille et comment elle dispose ses boîtes filles. L'espace entre deux boîtes filles successives est donné en paramètre.

Soit H une boîte horizontale ayant n boîtes filles

Paramètres

Soit $inter$ l'espace entre deux boîtes filles successives donné en paramètre.

Algorithme

L'origine (point d'entrée) de la boîte H est alignée avec sa première boîte fille.
Origine$[b_1] = (0, 0)$

Pour i **allant de** 2 à n **faire**
Début

- Aligner le point d'entrée de la boîte fille courante b_i par rapport au point de sortie de la boîte précédente b_{i-1}
  $dx_i =$ ExitPointX$[b_{i-1}] + inter$
  $dy_i =$ ExitPointY$[b_{i-1}]$
  Origine$[b_i] =$ Origine$[b_{i-1}] + (dx_i,\ dy_i)$
- Agrandir la boîte englobante pour contenir la boîte b_i et sa nouvelle origine (voir la figure 4.1)

Fin
Déterminer le point de sortie (ExitPoint)
ExitPoint= Origine$[b_n]$+ExitPoint$[b_n]$

La figure 4.1 montre le passage de cet algorithme de l'itération $i - 1$ à l'itération i. Lors du passage à l'étape i, nous calculons la position de la boîte fille b_i par rapport aux boîtes précédentes et nous mettons à jour le calcul de la boîte englobante pour contenir la boîte b_i.

Dans notre implémentation, une boîte horizontale n'applique pas de coupures lorsqu'il ne reste pas assez de marge. Elle se contente dans ce cas de n'afficher que la partie de son contenu qui ne déborde pas et qui sera lisible.

4.3.2 Boîte Verticale

Une boîte verticale dispose ses éléments, séparés par un espace, de manière verticale. L'algorithme suivant montre comment une boîte verticale calcule sa taille et comment elle dispose ses boîtes filles. La hauteur de l'espace, l'indentation sont donnés en paramètre.

Soit V la boîte verticale ayant n boîtes filles.

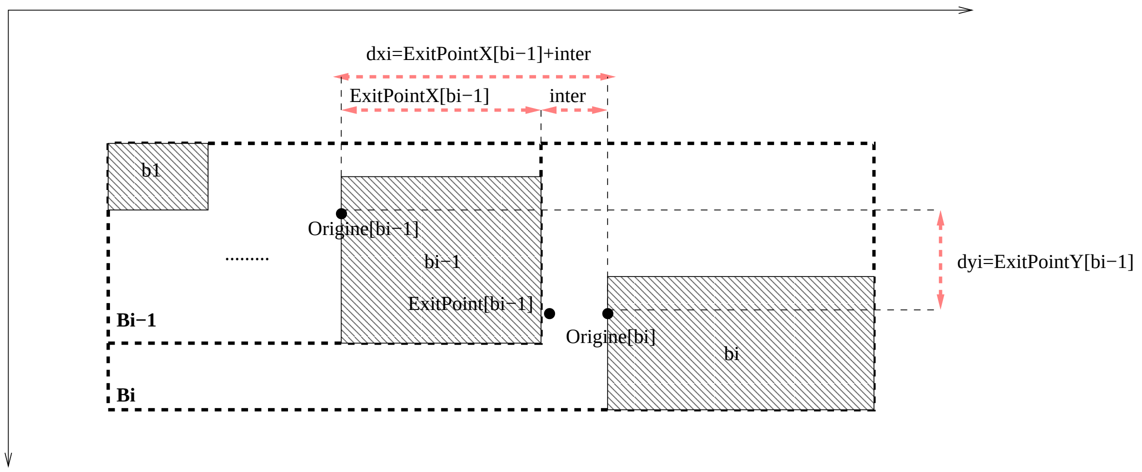


FIG. 4.1 – *Le calcul de la boîte horizontale englobante B_i à partir de la boîte précédente B_{i-1} : la boîte B_i est agrandie pour contenir la boîte b_i et sa nouvelle origine.*

Paramètres

Soit *interline* l'espace entre deux boîtes successives.
Soit *indent* l'indentation vers la droite des boîtes filles à partir de la deuxième boîte (voir la figure 4.2).

Dans le cas où le point d'entrée de la boîte V est le milieu (EntryType=Middle), nous calculons ce milieu de telle façon d'aligner V avec la boîte au milieu $b_{\frac{(n+1)}{2}}$ si n est impair et sinon elle est alignée par rapport au milieu de l'espace entre la boîte $b_{\frac{n}{2}}$ et $b_{\frac{n}{2}+1}$:

$middle = \sum_{i=1}^{2*i\leq n+1} \mathsf{Height}[b_i] + \sum_{i=2}^{2*i\leq n+1} interline - \mathsf{UpperLeftY}[b_1] + (1-(n\ mod\ 2)) * interline/2 - (n\ mod\ 2) * \mathsf{BottomRightY} b_{\frac{n+1}{2}}$

Dans le cas où le point d'entrée est aligné avec celui de la dernière boîte b_n (EntryType=LAST), nous calculons *last* comme suit :

$last = \sum_{i=1}^{i=n} \mathsf{Height}[b_i] + (n-1) * interline - \mathsf{UpperLeftY}[b_1] - \mathsf{BottomRightY}[b_n]$

Algorithme

Calculer la position de la boîte b_1 qui dépend du choix du type de point d'entrée de la boîte verticale (ici nous traitons le cas ou la boîte verticale a au minimum deux boîtes filles).

Dans le cas ou la boîte verticale a une seule boîte fille, OrigineX de cette boîte sera nul et OrigineY sera calculé comme nous montrons ci-dessous pour la boîte b_1.

La justification de la boîte b_1 se fait par rapport à la largeur de la boîte verticale.

$$\text{OrigineX}[b_1] = \begin{cases} \text{maxWidth}[b_2, \ldots, b_n] + indent - \text{Width}[b_1] & \text{si c1} \\ \frac{\text{maxWidth}[b_2, \ldots, b_n] + indent - \text{Width}[b_1]}{2} & \text{si c2} \\ 0 & \text{si c3} \end{cases}$$

c1=(JustifV = Right)
c2=(JustifV = Center)
c3=(JustifV = Left)

$$\text{OrigineY}[b_1] = \begin{cases} -middle & \text{si EntryType = Middle} \\ -last & \text{si EntryType = Last} \\ 0 & \text{si EntryType = First} \end{cases}$$

Agrandir la boîte englobante pour contenir la boîte b_1 et sa nouvelle origine.

Pour i **allant de** 2 à n **faire**
Début

- Calculer la position de la boîte b_i

  $$dx_i = \begin{cases} \text{BottomRightX}[b_{i-1}] - \text{BottomRightX}[b_i] & \text{si c1} \\ \text{UpperLeftX}[b_i] - \text{UpperLeftX}[b_{i-1}] + (\text{Width}[b_{i-1}] - \text{Width}[b_i])/2 & \text{si c2} \\ \text{UpperLeftX}[b_i] - \text{UpperLeftX}[b_{i-1}] & \text{si c3} \end{cases}$$

  c1=(JustifV = Right)
  c2=(JustifV=Center)
  c3=(JustifV=Left)

  dy_i = UpperLeftY$[b_i]$ + $interline$ + BottomRightY$[b_{i-1}]$
  Origine$[b_i]$ = Origine$[b_{i-1}]$ + (dx_i, dy_i)
  Pour la deuxième boîte ($i = 2$), il faut prendre en compte l'indentation pour calculer sa distance par rapport à la première boîte.
  Origine$[b_2]$ = Origine$[b_1]$ + $(dx_2 + indent, dy_2)$

- Agrandir la boîte englobante pour contenir la boîte b_i et sa nouvelle origine.

Fin

Déterminer le point de sortie (ExitPoint)
Nous distinguons quatre cas :

1. ExitType=First
2. ExitType=Middle
3. ExitType=Last et EntryType=First
4. ExitType=Last et EntryType=Last

Cas 1 :
ExitPoint= ExitPoint$[b_1]$

Cas 2 :

ExitPointX= ExitPointX$b_{rangMiddle}$

avec $rangMiddle = \begin{cases} \frac{n}{2} & \text{si n est pair} \\ \frac{(n+1)}{2} & \text{sinon} \end{cases}$

Si n est impair le point de sortie (ExitPoint) sera aligné par rapport à la boîte d'indice $\frac{(n+1)}{2}$ sinon il sera aligné par rapport à la boîte $\frac{n}{2}$.

ExitPointY= ExitPointY$b_{rangMiddle}$

Cas 3 :

$$\text{ExitPointX}=\begin{cases} \text{BottomRightX}[b_1] - \text{BottomRightX}[b_n] & \text{si c1} \\ \text{UpperLeftX}[b_1] - \text{UpperLeftX}[b_n] + \frac{(\text{Width}[b_1]-\text{Width}[b_n])}{2} & \text{si c2} \\ \text{UpperLeftX}[b_1] - \text{UpperLeftX}[b_n] & \text{si c3} \end{cases}$$

ExitPointX=ExitPointX+$indent$+ExitPointX$[b_n]$

ExitPointY=ExitPoint$[b_n]$+BottomRightY$[b_1]$-BottomRight$[b_n]$

Cas 4 :

ExitPoint=ExitPoint$[b_n]$

4.3.3 Le paragraphe

La boîte *Paragraphe* dispose ses boîtes filles horizontalement tant qu'il reste de l'espace à droite de la page (si la marge à droite reste positive) et si l'espace restant est insuffisant, elle passe à la ligne suivante et continue à disposer ses éléments horizontalement. Une boîte paragraphe peut être vue comme une colonne (boîte verticale) contenant des lignes (boîtes horizontales), à la différence près que la boîte paragraphe applique ses propres règles de coupure pour décider du retour à la ligne.

L'algorithme de formatage du paragraphe vérifie à chaque étape s'il reste de l'espace à droite pour disposer sa boîte fille courante. Dans le cas négatif, l'algorithme applique les règles de coupures pour passer à une nouvelle ligne et calculer la nouvelle position. Une nouvelle ligne commence au début du paragraphe courant et non pas au début de la page de formatage. Une conséquence directe de cette règle de coupure est l'impossibilité d'avoir un paragraphe sous la forme illustrée par la figure 4.3.a. Nous obtenons à la place le format illustré dans la figure 4.3.b. Pour résoudre ce problème, il serait judicieux de concevoir une nouvelle boîte semblable à "paragraphe" qui applique des règles de coupures différentes permettant d'obtenir le formatage désiré. Notons que dans notre contexte d'utilisation (affichage de programmes ou de scripts), ce comportement du constructeur paragraphe est adapté. Par contre il faudrait le modifier dans un contexte plus "textuel".

Le paragraphe s'aligne avec sa première et sa dernière ligne : le point d'entrée est celui de la première boîte fille et le point de sortie est celui de la dernière boîte fille.

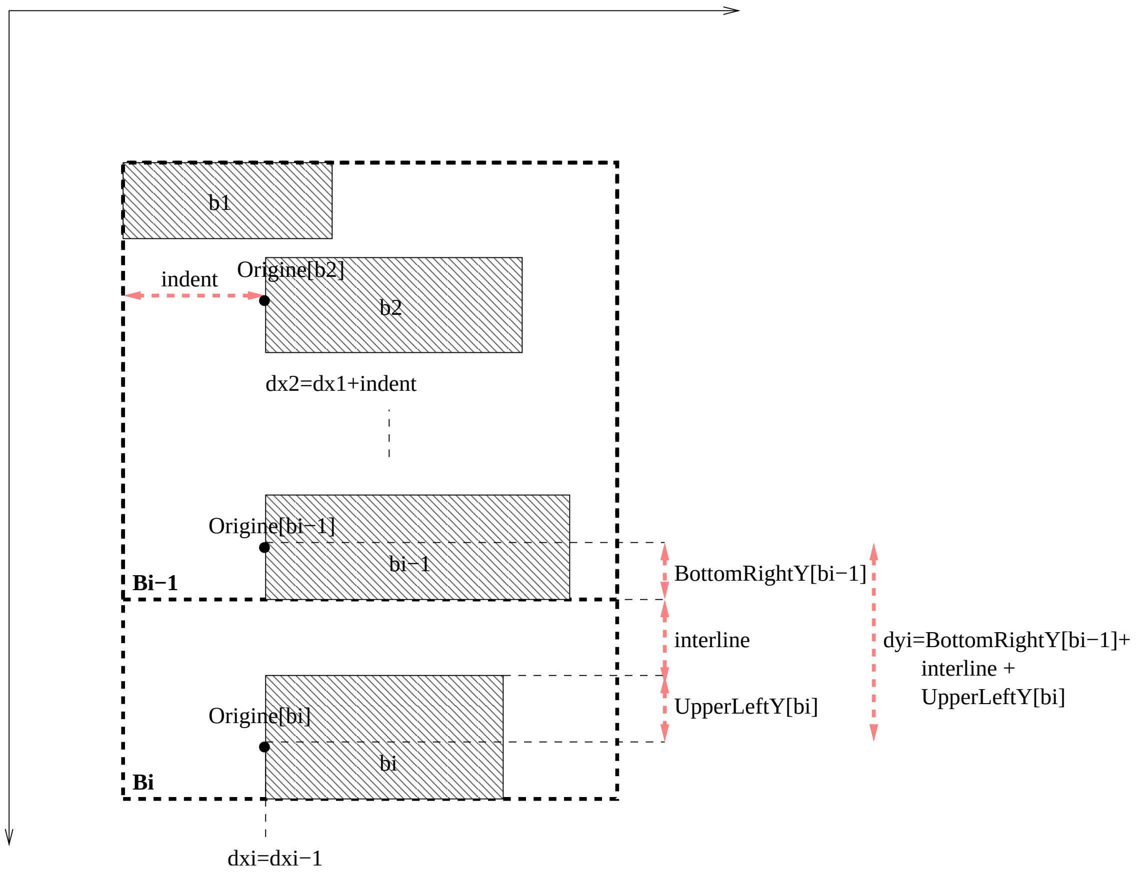


FIG. 4.2 – *Le calcul de la boîte verticale englobante B_i à partir de la boîte précédente B_{i-1} : la boîte B_i est agrandie pour contenir la boîte b_i et sa nouvelle origine. Ici la justification de la boîte verticale est à gauche Left.*

4.4 Les boîtes mathématiques

4.4.1 Fraction

La boîte *fraction* dispose le numérateur au dessus du dénominateur séparés par une barre horizontale. En général, on aligne la boite *fraction* par rapport aux autres boîtes au niveau de sa barre horizontale. L'espace entre le numérateur et le dénominateur et l'épaisseur du dessin de la barre horizontal peuvent être données comme paramètres.

Soit F la boîte *Fraction* ayant deux boîtes filles b_1 et b_2.
b_1 représente le numérateur et b_2 le dénominateur.

Paramètres

Soit *interline* l'espace entre ces deux boîtes fixé par la boîte F.

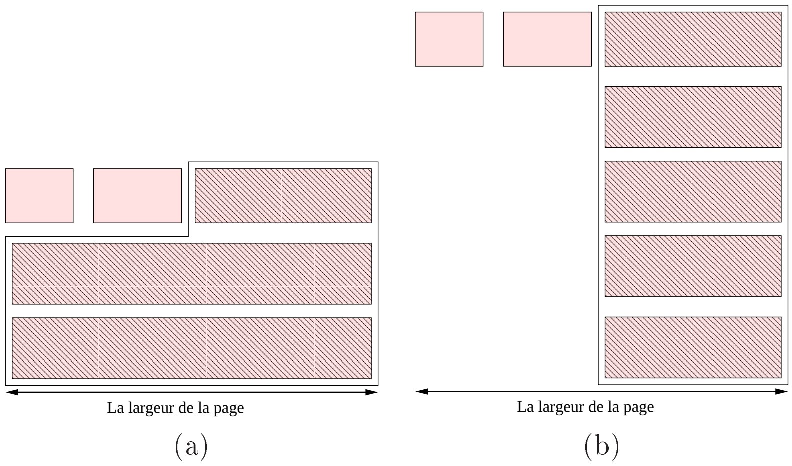


FIG. 4.3 – *(a) Un paragraphe où la nouvelle ligne commence au début de la page. (b) La nouvelle ligne commence au début du paragraphe courant.*

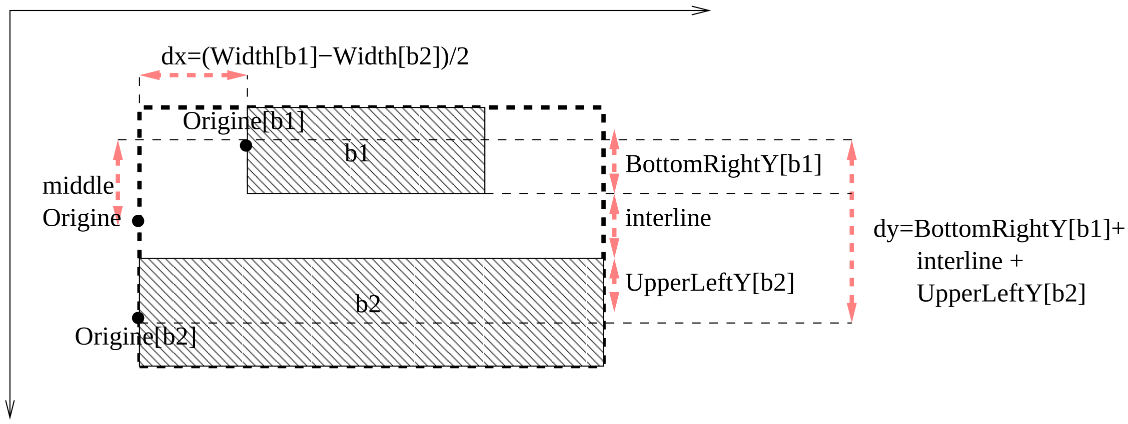


FIG. 4.4 – *Le formatage de la boîte Fraction : le numérateur et le dénominateur sont centrés et le point d'entrée est le milieu de la boîte englobante.*

Dans le cas où le point d'entrée est le milieu (EntryPoint=Middle), nous calculons ce milieu de telle façon que la barre horizontale de la fraction soit alignée avec les autres boîtes. Pour cela, nous avons besoin des propriétés de la fonte métrique utilisée ($ascent$ et $descent$).

$$middle = \mathsf{BottomRightY}[b_1] + ascent - \frac{ascent+descent}{2} + \frac{interline}{2}$$

Dans le cas ou le point d'entrée est aligné avec celui de la boîte b_2 (EntryPoint=Last), nous calculons $last$ comme suit :

$$last = \mathsf{BottomRightY}[b_1] + \mathsf{UpperLeftY}[b_2] + interline$$

Algorithme

- **Première partie**

  Calculer la position de b_1 en fonction de sa justification et le point d'entrée de la fraction.

  $$\text{OrigineX}[b_1] = \begin{cases} \text{maxWidth}(b_1, b_2) - \text{Width}[b_1] & \text{si Justif}[b_1] = \text{Right} \\ \frac{(\text{maxWidth}(b_1,b_2) - \text{Width}[b_1])}{2} & \text{si Justif}[b_1] = \text{Center} \\ 0 & \text{si Justif}[b_1] = \text{Left} \end{cases}$$

  $$\text{OrigineY}[b_1] = \begin{cases} -middle & \text{si EntryType} = \text{Middle} \\ -last & \text{si EntryType} = \text{Last} \\ 0 & \text{si EntryType} = \text{First} \end{cases}$$

  Agrandir la boîte englobante pour contenir la boîte b_1 et sa nouvelle origine.
  Calculer la distance entre l'origine de b_1 et celle de b_2.

  $$dx = \begin{cases} \text{Width}[b_1] - \text{Width}[b_2] & \text{si Justif}[b_2] = \text{Right} \\ \frac{\text{Width}[b_1] - \text{Width}[b_2]}{2} & \text{si Justif}[b_2] = \text{Center} \\ 0 & \text{si Justif}[b_2] = \text{Left} \end{cases}$$

  $dy = \text{BottomRightY}[b_1] + interline + \text{UpperLeftY}[b_2]$
  Calculer la position de b_2.
  $\text{Origine}[b_2] = \text{Origine}[b_1] + (dx, dy)$
  Agrandir la boîte englobante pour contenir la boîte b_2 et sa nouvelle origine.

- **Deuxième partie**

  Déterminer le point de sortie de la boîte englobante

  $$\text{ExitPointY} = \begin{cases} \text{OrigineY}[b_2] + \text{ExitPointY}[b_2] & \text{si condition1} \\ \text{ExitPointY}[b_2] & \text{si condition2} \\ 0 & \text{sinon} \end{cases}$$

  condition1= (EntryType=First et ExitType=Last)
  condition2 = (EntryType=Last et ExitType=Last)

  $$\text{ExitPointX} = \begin{cases} \text{OrigineX}[b_2] + \text{Width}[b_2] & \text{si Justif}[b_2] = \text{Right} \\ \text{OrigineX}[b_2] + \frac{(\text{maxWidth}(b_1,b_2) + \text{Width}[b_2])}{2} & \text{si Justif}[b_2] = \text{Center} \\ \text{OrigineX}[b_2] + \text{maxWidth}(b_1, b_2) & \text{si Justif}[b_2] = \text{Left} \end{cases}$$

4.4.2 Les délimiteurs

Les crochets, les parenthèses, les accolades, les barres verticales et d'autres types de barres sont des délimiteurs mathématiques. Les parenthèses autour d'une matrice est un exemple d'un délimiteur.

$$\begin{pmatrix} 0 & -i & 0 & 0 & -i \\ i & 0 & i & i & 0 \\ 0 & -i & -i & 0 & -i \end{pmatrix}$$

La boîte *Délimiteur* calcule sa propre taille en fonction de la taille de sa boîte fille (contenu) et en fonction de la taille des symboles de délimitation. La taille de chaque délimiteur est calculée de la façon suivante : la hauteur est égale à la hauteur de la boîte fille et la largeur est calculée en fonction de la nature du délimiteur et de sa hauteur (par exemple, la largeur d'une accolade est croissante par rapport à sa hauteur).

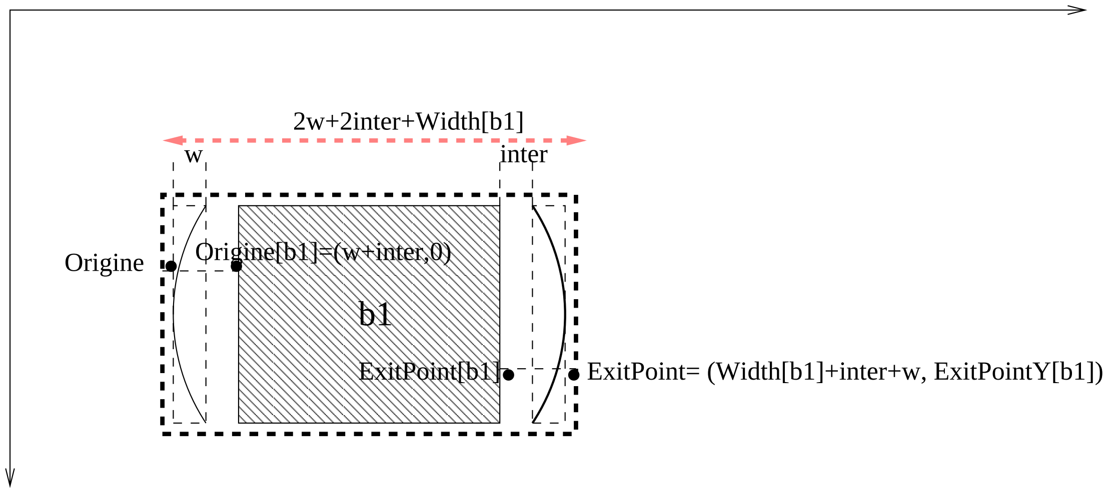


Fig. 4.5 – *Le formatage d'une boîte Délimiteur.*

Soit D la boîte *Délimiteur* ayant une boîte fille b_1.

Paramètres

Soit $inter$ l'espace entre le symbole délimiteur et la boîte délimitée b_1.
Soit h la hauteur des deux symboles du délimiteur.
Soit $w1$ la largeur du premier symbole du délimiteur (la largeur est calculée en fonction de la nature du symbole du délimiteur et sa hauteur).
Soit $w2$ la largeur du deuxième symbole du délimiteur.
Le type de symbole est donné en paramètre. Ici nous ne décrivons pas le calcul de la taille de la boîte englobant ce symbole.
Soit TypeDelimiteur le type de la disposition des symboles du délimiteur autour du contenu. Nous distinguons trois type de délimiteurs :

- Left met le délimiteur uniquement à gauche de la boîte délimitée.
- Right met le délimiteur uniquement à droite de la boîte délimitée.
- Left-Right met deux symboles du délimiteur de chaque coté de la boîte délimitée.

Algorithme

Le point d'entrée est aligné avec le point d'entrée de la boîte à délimiter.

$$\text{Origine}[b_1] = \begin{cases} (0,0) & \text{si TypeDelimiteur} = \text{Right} \\ (w1 + inter, 0) & \text{sinon (TypeDelimiteur=Left ou TypeDelimiteur=Left-Right)} \end{cases}$$

Agrandir la boîte englobante pour contenir b_1 dans sa nouvelle position.
BottomRightX= BottomRightX$+inter + w2$ si TypeDelimiteur = Right ou Left-Right

Le point de sortie (ExitPoint) est aligné avec celui de la boîte à délimiter.

$$\text{ExitPointX=} \begin{cases} \text{OrigineX}[b_1] + \text{ExitPointX}[b_1] & \text{si TypeDelimiteur = Left} \\ \text{OrigineX}[b_1] + \text{ExitPointX}[b_1] + inter + w2 & \text{sinon (Right ou Left-Right)} \end{cases}$$

ExitPointY= ExitPointY$[b_1]$

La boîte *Délimiteur* englobe la boîte *contenu* et aussi l'espace pour dessiner les symboles de délimitation. La figure 4.5 illustre le calcul de la boîte englobante d'un délimiteur de type Left-Right (met un symbole pour chaque coté) et son alignement par rapport aux autres boîtes (point d'entrée et de sortie).

4.4.3 Table

RowTable

La boîte *RowTable* représente une ligne d'une table. Elle aligne horizontalement ses éléments comme la boîte horizontale. *RowTable* propose différents type d'alignement de ses éléments. Il y a quatre principaux types d'alignements:

- *Top* signifie que l'origine (point d'entrée) de chaque élément doit être alignée avec l'origine de la boîte précédente.
- *Bottom* signifie que le point de sortie de chaque élément doit être aligné par rapport au point de sortie de la boîte précédente.
- *Center* signifie que chaque élément doit aligner le milieu de sa boîte par rapport au milieu de la boîte suivante.
- *Baseline* signifie que tous les éléments doivent être alignés par rapport à la ligne de référence.

Soit *Rtable* une boîte *RowTable* ayant n boîtes filles.

Paramètres

Soit *inter* l'espace horizontal entre deux éléments successifs de la ligne.
Soit *RowTable.TypeAlign* le type indiquant l'alignement des éléments de cette ligne (*Top, Bottom, Center* ou *Baseline*).

Algorithme

L'origine (point d'entrée) de la boîte *RowTable* est alignée avec sa première boîte fille
Origine$[b_1] = (0, 0)$

Pour i **allant de** 2 à n **faire**
Début

- Aligner la boîte courante b_i par rapport à la boîte précédente b_{i-1} en prenant en compte le type d'alignement (TypeAlign) de $Rtable$.
  dx_i = ExitPointX$[b_{i-1}]$ + $inter$

  $$dy_i = \begin{cases} 0 & \text{si c1} \\ \text{BottomRightY}[b_{i-1}] - \text{BottomRightY}[b_i] & \text{si c2} \\ \frac{\text{Heigh}[b_{i-1}]}{2} - \text{UpperLeftY}[b_{i-1}] - (\frac{\text{Heigh}[b_i]}{2} - \text{UpperLeftY}[b_i]) & \text{si c3} \\ \text{ExitPointY}[b_{i-1}] & \text{si c4} \end{cases}$$

  - c1 = (RowTable.TypeAlign=Top)
  - c2 = (RowTable.TypeAlign=Bottom)
  - c3 = (RowTable.TypeAlign=Center)
  - c4 = (RowTable.TypeAlign=Baseline)

  Origine$[b_i]$=Origine$[b_{i-1}]$ + $(dx_i,\ dy_i)$

- Agrandir la boîte englobante pour contenir la boîte b_i et sa nouvelle origine.

Fin

Déterminer le point de sortie (ExitPoint)
ExitPointX= OrigineX$[b_n]$+ExitPointX$_n$

$$\text{ExitPointY=} \begin{cases} \text{OrigineY}[b_n] & \text{si c1} \\ \text{OrigineY}[b_n] + \text{ExitPointY}[b_n] & \text{si c2 ou c4} \\ \text{OrigineY}[b_n] + \text{BottomRightY}[b_n] - \frac{\text{Height}[b_n]}{2} & \text{si c3} \end{cases}$$

Table

La boîte $Table$ représente les matrices, les tableaux et d'autres notations mathématiques. Elle dispose les lignes en forme de tableau et peut aussi placer un cadre (*frame*) autour.

Soit $Table$ la boîte Table ayant m lignes où chaque ligne contient n boîtes filles.

Les lignes sont des boîtes $RowTable$

Paramètres

Soit m le nombre de lignes de cette table.
Soit n le nombre de colonnes de cette table.
La taille de la table est $n * m$.
L_i est la ligne d'indice i.

C_j est la colonne d'indice j.
T_{ij} est la boîte placée à la case (i, j) de la table. Elle est positionnée à l'intersection de la ligne i et la colonne j.
$interColonne_j$ est l'espace horizontal entre la colonne C_j et la colonne C_{j-1}.
$interLine_i$ est l'espace vertical entre la ligne L_i et la ligne L_{i-1}.

$Table$ propose aussi différents types d'alignement par rapport aux autres boîtes du contexte. Le type d'alignement $Table.TypeAlign$ prend les valeurs suivantes :

- *Top* signifie que le point d'entrée et de sortie de la table sont alignés avec ceux de sa première ligne. Dans ce cas, la table a le même alignement que sa première ligne.
- *Bottom* signifie que le point d'entrée et de sortie de la table sont alignés avec ceux de sa dernière ligne.
- *Center* signifie que le point d'entrée et de sortie de la table sont à la moitié de la hauteur de la table.
- *Baseline* aligne la table par rapport à la ligne du milieu si le nombre des lignes est impair et sinon l'alignement sera par rapport au milieu de l'espace entre la ligne $\frac{m}{2}$ et la ligne $\frac{m}{2} + 1$.
- *RowNumber* est de type entier. La table a le même alignement que la RowNumber nième ligne si ce nombre est entre 1 et m et sinon elle s'aligne par rapport à la dernière ligne.

Pour chaque type d'alignement, nous calculons la position du point d'entrée de la table :

- $computeRowNumber = \sum_{i=1}^{i=rownumber-1} \mathsf{Height}[L_i] + \sum_{i=2}^{i=rownumber} interLigne_i - \mathsf{UpperLeftY}[L_1] + \mathsf{UpperLeftY}[L_{rownumber}]$

- $computeMiddle = (\sum_{i=1}^{i=m} \mathsf{Height}[L_i] + \sum_{i=2}^{i=m} interLigne_i)/2 - \mathsf{UpperLeftY}[L_1]$

- $computeBaseline = \sum_{i=1}^{i=m/2}(\mathsf{Height}[L_i] + interLigne_{i+1}) - \mathsf{UpperLeftY}[L_1] - interLigne_{m/2+1}/2$ avec m est pair (le nombre de ligne).

- $computeBottom = \sum_{i=1}^{i=m} \mathsf{Height}[L_i] + \sum_{i=2}^{i=m} interLigne_i - \mathsf{UpperLeftY}[L_1] - \mathsf{BottomRightY}[L_m]$

Si la table contient des bords (frames), *frameHoriz* est l'espace entre les deux bords (droite et gauche) et la première et la dernière colonnes et *frameVertic* est l'espace entre les deux bords (haut et bas) et la première et la dernière lignes.
Soit $JustifC_j$ la justification des boîtes de la colonne j. Elle prend trois valeurs : Right, Center, left.
Soit $JustifL_i$ la justification de la ligne L_i qui est de même type que la justification d'un RowTable.

Nous calculons aussi la largeur de chaque colonne et la hauteur de chaque ligne :
$\forall j \in [1...n]$ $\mathsf{Width}[C_j] = \max_{i=1}^{i=m}(\mathsf{Width}[T_{ij}])$
Le calcul de la hauteur de chaque ligne dépend de la taille et de l'alignement des boîtes filles. La hauteur est déduite du formatage de chaque boîte RowTable représentant les lignes.

Algorithme

Le traitement des boîtes se fait ligne par ligne. Le premier élément de la table est traité différemment des autres éléments car sa disposition dépend de la nature de l'alignement de la table par rapport aux autres boîtes (son point d'entrée).

Pour i **allant de** 1 à m **faire** (i est l'indice des lignes)
Pour j **allant de** 1 à n **faire** (j est l'indice des colonnes)
Début

- **Première étape**: le traitement du premier élément de la table (T_{11})

$$\text{OrigineY}[T_{11}] = \begin{cases} -computeRowNumber & \text{si c1} \\ -computeRowNumber & \text{si c2} \\ -computeBaseline & \text{si c3} \\ -computeMiddle & \text{si Table.TypeAlign=Center} \\ -computeBottom & \text{si Table.TypeAlign=Bottom} \\ 0 & \text{si Table.TypeAlign=Top} \end{cases}$$

c1 = (Table.TypeAlign=RowNumber et rownumber $\in [1..n]$)
c2 = Table.TypeAlign=Baseline avec rownumber=$\frac{n+1}{2}$ et n impaire)
c3 = (Table.TypeAlign=Baseline et n paire aligner entre la ligne [n,$\frac{n}{2}$])

La valeur de *frameHoriz* est nulle s'il n'y a pas de frame autour de la table

$$\text{OrigineX}[T_{11}] = \begin{cases} frameHoriz + \text{Width}[C_1] - \text{Width}[T_{11}] & \text{si Justif}[C_1]\text{=Right} \\ frameHoriz + (\text{Width}[C_1] - \text{Width}[T_{11}])/2 & \text{si Justif}[C_1]\text{=Center} \\ frameHoriz & \text{si Justif}[C_1]\text{=Left} \end{cases}$$

- **Deuxième étape** Traitements des autres éléments

  Si l'élément est le premier de la ligne, il faut faire un retour à la ligne. Nous traitons séparément le cas où on dispose horizontalement les éléments de la table et le cas du retour à une nouvelle ligne (les calculs dans les deux cas sont différents).

  Nous distinguons les deux cas : le premier cas est $j \neq 1$ et le deuxième cas est $j = 1$ (les éléments de la première colonne C_1).
  Si $j \neq 1$ **alors**
  Début
  dx_{ij} est la distance horizontale entre la boîte courante T_{ij} de la colonne C_j et la boîte précédente $T_{i(j-1)}$ de la colonne C_{j-1}
  $dx_{ij} = D_{i(j-1)} + interColonne_j + D_{ij}$

  avec $D_{i(j-1)}$ est la distance horizontale entre l'origine de la boîte précédente de la colonne C_{j-1} et la fin de la colonne C_{j-1} à droite.
  D_{ij} est la distance horizontale entre le début de la colonne C_j et l'origine de la boîte courante de la colonne C_j

$$D_{ij} = \begin{cases} \mathsf{Width}[C_j] - \mathsf{Width}[T_{ij}] & \text{si Justif}[C_j]\text{=Right} \\ \frac{\mathsf{Width}[C_j]-\mathsf{Width}[Tij]}{2} & \text{si Justif}[C_j]\text{=Center} \\ 0 & \text{si Justif}[C_j]\text{=Left} \end{cases}$$

$$D_{i(j-1)} = \begin{cases} \mathsf{Width}[T_{i(j-1)}] & \text{si Justif}[C_{j-1}]\text{=Right} \\ \frac{\mathsf{Width}[C_{j-1}]+\mathsf{Width}[T_{i(j-1)}]}{2} & \text{si Justif}[C_{j-1}]\text{=Center} \\ \mathsf{Width}[C_{j-1}] & \text{si Justif}[C_{j-1}]\text{=Left} \end{cases}$$

$$dy_{ij} = \begin{cases} 0 & \text{si c1'} \\ \mathsf{BottomRightY}[T_{i(j-1)}] - \mathsf{BottomRightY}[T_{ij}] & \text{si c2'} \\ \frac{\mathsf{Height}[T_{i(j-1)}]}{2} - \mathsf{UpperLeftY}[T_{i(j-1)}] - (\frac{\mathsf{Height}[T_{ij}]}{2} - \mathsf{UpperLeftY}[T_{ij}]) & \text{si c3'} \\ \mathsf{ExitPointY}[T_{i(j-1)}] & \text{si c4'} \end{cases}$$

c1' = (Justif[L_i]=Top)
c2' = (Justif[L_i]=Bottom)
c3' = (Justif[L_i]=Center)
c4' = (Justif[L_i]=Baseline)

Origine[T_{ij}] = Origine[$T_{i(j-1)}$] + (dx_{ij}, dy_{ij})

Fin

Sinon (le cas d'un élément de la première colonne $j = 1$)
Début

$$\mathsf{OrigineX}[T_{i1}] = \begin{cases} -\mathsf{OrigineX}[T_{(i-1)n}] + \mathsf{Width}[C_j] - \mathsf{Width}[T_{ij}] + frameHoriz & \text{si c1''} \\ -\mathsf{OrigineX}[T_{(i-1)n}] + \frac{(\mathsf{Width}[C_j]-\mathsf{Width}[T_{ji}])}{2} + frameHoriz & \text{si c2''} \\ -\mathsf{OrigineX}[T_{(i-1)n}] + frameHoriz & \text{si c3''} \end{cases}$$

c1" = (Justif[C_j]=Right)
c2" = (Justif[C_j]=Center)
c3" = (Justif[C_j]=Left)

OrigineY[T_{i1}] = −OrigineY[$T_{(i-1)1}$] + $interLigne_i$ + BottomRightY[L_{i-1}] + UpperLeftY[L_i]
Fin

Agrandir la boîte englobante pour contenir la boîte T$_{ij}$ et sa nouvelle origine.

fin

Déterminer le point de sortie de la table (ExitPoint)

$$\text{ExitPointX}= \begin{cases} \text{OrigineX}[T_{mn}] + \text{Width}[T_{mn}] & \text{si Justif}[C_n] = \text{Right} \\ \text{OrigineX}[T_{mn}] + (\text{Width}[C_n] - \text{Width}[T_{mn}])/2 & \text{si Justif}[C_n]\text{=Center} \\ \text{OrigineX}[T_{mn}] + \text{Width}[C_n] & \text{si Justif}[C_n]\text{=Left} \end{cases}$$

S'il y a des frames (cadres) autour de la table, la boîte englobante ajoute de l'espace en haut et en bas de la boîte

UpperLeftY=UpperLeftY+*frameVertic*
Height= Height+*frameVertic*
BottomRightY = BottomRightY + *frameVertic*

4.4.4 Row

La boîte *Row* dispose horizontalement ses boîtes filles. Elle a le même algorithme que la boîte *Horizontale* sauf qu'il est possible d'appliquer des coupures. Dans ce dernier cas, la boîte *Row* a le même comportement que le constructeur *Paragraphe*.

4.4.5 Over

La boîte Over prend en argument deux boîtes filles : la boîte qui représente la base et la boîte à mettre au dessus de cette base.

Nous distinguons deux types de boîte Over: la boîte qui se contente de mettre une boîte au dessus de l'autre et la boîte qui met un type de barre au dessus de la base et dont le dessin suit la taille de la base.

Le deuxième type de *Over* est *OverStretch* qui met un symbole de type barre horizontale ou accolade horizontale au dessus de la base. *OverStretch* a une seule boîte fille et il prend en argument le type du symbole a mettre au dessus de la base. Le calcul de la taille de ce symbole horizontal dépend de la taille de la base (largeur) et du type du symbole.

La boîte englobante est alignée par rapport à la boîte base et une fois la taille du symbole est déterminée, il suffit d'agrandir la boîte pour contenir le symbole au dessus de la base et dont l'origine sera placé au point (0,-$interbaseBracket - HeightBracket$).

Le calcul pour ce type de boîte est équivalent à celui des boîtes délimiteurs, cependant le symbole est horizontal et il est mis au dessus de la boîte.

Soit $Over$ la boîte Over ayant deux boîtes filles $base$ et $overSon$. La boîte $Over$ dispose la boîte $overSon$ au dessus de la base $base$.

Paramètres

Soit *interBaseOverSon* l'espace blanc entre les deux boîtes $base$ et $overSon$.

La boîte $Over$ s'aligne par rapport à la base.
Soit *JustifBase* la justification de la boîte $base$ et qui prend les valeurs suivantes : Right, Left et Center.
Soit *JustifOverSon* la justification de la boîte $overSon$.

Algorithme

- **Première partie**

  - Calculer la position de $overSon$ en fonction de sa justification et l'alignement de la boîte $Over$ (point d'entrée).

    $$\text{OrigineX}[overSon] = \begin{cases} (\text{maxWidth}(base, overSon) - \text{Width}[overSon])/2 & \text{si c1} \\ \text{maxWidth}(base, overSon) - \text{Width}[overSon] & \text{si c2} \\ 0 & \text{si c3} \end{cases}$$

    c1=(Justif[$overSon$]=Center)
    c2=(Justif[$overSon$]=Right)
    c3= (Justif[$overSon$]=Left)

    $$\text{OrigineY}[overSon] = -\text{UpperLeftY}[base] - interBaseOverSon - \text{BottomRightY}[overSon]$$

  - Agrandir la boîte englobante pour contenir la boîte $overSon$ et sa nouvelle origine.
  - Calculer la distance entre l'origine de $overSon$ et celui de $base$

    $$dx = \begin{cases} \text{maxWidth}(overSon, base) - \text{Width}[base] & \text{si c4} \\ \frac{(\text{maxWidth}(overSon,base)-\text{Width}[base])}{2} & \text{si c5} \\ 0 & \text{si c6} \end{cases}$$

    c4=(Justif[$base$]=Right)
    c5=(Justif[$base$]=Center)
    c6= (Justif[$base$]=Left)

    $$dy = \text{BottomRightY}[overSon] + interBaseOverSon + \text{UpperLeftY}[base]$$

  - Calculer la position de $base$

    $$\text{Origine}[base] = \text{Origine}[overSon] + (dx,dy)$$

  - Agrandir la boîte englobante pour contenir la boîte $base$ et sa nouvelle origine

- **Deuxième partie**

  Déterminer le point de sortie de la boîte englobante

  $$\text{ExitPointX}=\begin{cases} \text{OrigineX}[base] + \text{Width}[base] & \text{si c4} \\ \text{OrigineX}[base] + \frac{(\text{maxWidth}(overSon,base)+\text{Width}[base])}{2} & \text{si c5} \\ \text{OrigineX}[base] + \text{maxWidth}(overSon, base) & \text{si c6} \end{cases}$$

ExitPointY = ExitPointY[$base$]

4.4.6 Under

La boîte Under prend en argument deux boîtes filles : la boîte qui représente la base et la boîte à mettre au dessous de cette base. Nous étudions deux types de boîtes Under (comme pour Over) : *Under* et *UnderStretch*.

UnderStretch met un symbole de type barre horizontale ou accolade horizontale au dessous de la base. *UnderStretch* a une seule boîte fille et il prend en argument le type du symbole a mettre au dessous de la base. Le calcul de la taille de ce symbole horizontal dépend de la taille de la base (largeur) et du type du symbole.

La boîte englobante est alignée par rapport à la boîte base et une fois la taille du symbole est déterminée, il suffit d'agrandir la boîte pour contenir le symbole au dessous de la base et dont l'origine sera placé au point (0,$interbaseBracket$+BottomRightY[$base$]+ $HeightBracket$).

Ici, le symbole délimiteur est horizontal et il est mis sous la base.

Soit $Under$ la boîte Under ayant deux boîtes filles $base$ et $underSon$. La boîte $Under$ dispose la boîte $underSon$ au dessus de la base $base$.

Paramètres

Soit *interBaseUnderSon* l'espace blanc entre les deux boîtes $base$ et $underSon$.
La boîte $Under$ s'aligne par rapport à la base.
Soit *JustifBase* la justification de la boîte $base$ et qui prend les valeurs suivantes : Right, Left et Center.
Soit *JustifUnderSon* la justification de la boîte $underSon$.

Algorithme

- **Première partie**
  - Calculer la position de $base$ en fonction de son justification.

    $$\text{OrigineX}[base] = \begin{cases} \text{maxWidth}(base, underSon) - \text{Width}[base] & \text{si c1} \\ (\text{maxWidth}(base, underSon) - \text{Width}[base])/2 & \text{si c2} \\ 0 & \text{si c3} \end{cases}$$

    c1=(Justif[$base$]=Right)
    c2=(Justif[$base$]=Center)
    c3= (Justif[$base$]=Left)

    OrigineY[$base$] $= 0$

- Agrandir la boîte englobante pour contenir la boîte $base$ et sa nouvelle origine.
- Calculer la distance entre l'origine de $base$ et celui de $underSon$

$$dx = \begin{cases} \mathsf{maxWidth}(base, underSon) - \mathsf{Width}[underSon] & \text{si c4} \\ \frac{(\mathsf{maxWidth}(base, underSon) - \mathsf{Width}[underSon])}{2} & \text{si c5} \\ 0 & \text{si c6} \end{cases}$$

c4=(Justif[$underSon$]=Right)
c5=(Justif[$underSon$]=Center)
c6= (Justif[$underSon$]=Left)

dy = BottomRightY[$base$] + $interBaseUnderSon$ + UpperLeftY[$underSon$]

- Calculer la position de $underSon$.
  Origine[$underSon$] = Origine[$base$] + (dx,dy)
- Agrandir la boîte englobante pour contenir la boîte $underSon$ et sa nouvelle origine.

- **Deuxième partie**
  Déterminer le point de sortie de la boîte englobante.

$$\mathsf{ExitPointX} = \begin{cases} \mathsf{OrigineX}[underSon] + \mathsf{Width}[underSon] & \text{si c4} \\ \mathsf{OrigineX}[underSon] + \frac{(\mathsf{maxWidth}(base, underSon) + \mathsf{Width}[underSon])}{2} & \text{si c5} \\ \mathsf{OrigineX}[underSon] + \mathsf{maxWidth}(base, underSon) & \text{si c6} \end{cases}$$

ExitPointY = ExitPointY[$base$]

4.4.7 L'indice inférieur (Sub)

La boîte d'indice inférieur *Sub* prend en argument deux boîtes filles : la base et l'indice inférieur.

La boîte *Sub* s'occupe aussi de réduire la taille de l'indice Sub. La boîte fille *indice* se reformate pour calculer sa nouvelle taille. Toutes les boîtes avec un comportement similaire à Sub comme Sup et racine nième appliquent le même algorithme.

Soit Sub la boîte Sub ayant deux boîtes filles $base$ et $subScript$. La boîte Sub dispose la boîte $subScript$ comme indice de la base $base$.

Paramètres

Soit $interBasesubScript$ la distance entre les deux boîtes $base$ et $subSript$. Elle peut être donnée en paramètre et sinon elle est calculée en fonction de la taille des deux boîtes et les propriétés de la fonte utilisée (ascent).

$interBasesubScript$ = BottomRightY[$base$] + UpperLeftY[$subScript$] − $ascent$

La boîte Sub s'aligne par rapport à la base.

Algorithme

- **Première partie**
  - La boîte de Sub a l'alignement de la $base$. Donc, l'origine de $base$ est (0,0).
  - Agrandir la boîte englobante pour contenir la boîte $base$.
  - Calculer la distance entre l'origine de la $base$ et celui de $subScript$
    $dx = \text{Width}[base]$
    $dy = interBasesubScript$
  - Calculer la position de $subScript$
    $\text{Origine}[subScript] = \text{Origine}[base] + (dx,dy)$
  - Agrandir la boîte englobante pour contenir la boîte $subScript$ et sa nouvelle origine.
- **Deuxième partie**
  Déterminer le point de sortie de la boîte englobante.
  ExitPointX= OrigineX[$subScript$]+Width[$subScript$]
  ExitPointY= ExitPointY[$base$]

4.4.8 L'indice supérieur (Sup)

La boîte d'indice supérieur *Sup* prend en argument deux boîtes filles : la base et l'indice supérieur.

La boîte *Sup* (tout comme Sub) s'occupe aussi de réduire la taille de l'indice *Sup*. La boîte fille se reformate pour calculer sa nouvelle taille.

Soit Sup la boîte Sup ayant deux boîtes filles $base$ et $supScript$.
La boîte Sup dispose la boîte $supScript$ comme exposant de la base $base$.

Paramètres

Soit *interBasesupScript* la distance entre les deux boîtes $base$ et $supScript$. Elle peut être donnée en paramètre et sinon elle est calculée en fonction de la taille des deux boîtes et les propriétés de la fonte utilisée (ascent).

$interBasesupScript = \text{UpperLeftY}[base] + \text{BottomRightY}[supScript] - ascent$

La boîte Sup s'aligne par rapport à la base.

Algorithme

- **Première partie**
  - La boîte de Sup a l'alignement de la $base$. Donc, l'origine de $base$ est (0,0).
  - Agrandir la boîte englobante pour contenir la boîte $base$.
  - Calculer la distance entre l'origine de la $base$ et celui de $supScript$
    $dx =$ Width$[base]$
    $dy = -interBasesupScript$
  - Calculer la position de $supScript$
    Origine$[supScript]$ = Origine$[base]$ + (dx,dy)
  - Agrandir la boîte englobante pour contenir la boîte $supScript$ et sa nouvelle origine.
- **Deuxième partie**
  Déterminer le point de sortie de la boîte englobante
  ExitPointX= OrigineX$[supScript]$+Width$[supScript]$
  ExitPointY= ExitPointY$[base]$

4.4.9 Le double indice (SubSup)

La combinaison de la boîte *Sup* avec *Sub* (ou inversement) ne donne pas le résultat d'une boîte plaçant à droite de sa base un indice et un exposant (${x^i}_j$ est différent de x^i_j). Pour obtenir ce type de formatage, nous avons défini la boîte *SubSup*.

La boîte du double indice *SubSup* prend en argument trois boîtes filles : la base, l'indice supérieur et l'indice inférieur.

La boîte *SubSup* (comme Sub) s'occupe aussi de réduire la taille des indices supérieur et inférieur. Les boîtes d'indices se reformatent pour calculer leur nouvelles tailles.

Soit $SubSup$ la boîte SubSup ayant trois boîtes filles $base$, $supScript$ et $subScript$.
La boîte $SubSup$ dispose la boîte $supScript$ comme exposant et la boîte $subScript$ comme indice de la base $base$.

Paramètres

Soit $interBasesupScript$ la distance entre les deux boîtes $base$ et $supScript$. Elle peut être donnée en paramètre et sinon elle est calculée en fonction de la taille des deux boîtes et les propriétés de la fonte utilisée (ascent).

$interBasesupScript =$ UpperLeftY$[base]$ + BottomRightY$[supScript] - ascent$

Soit $interBasesubScript$ la distance entre les deux boîtes $base$ et $subScript$. Elle peut être donnée en paramètre et sinon elle est calculée en fonction de la taille des deux boîtes et les propriétés de la fonte utilisée (ascent).

$interBasesubScript$ = BottomRightY$[base]$ + UpperLeftY$[subScript] - ascent$

On regarde s'il y aura un chevauchement entre la boîte d'indice et d'exposant. Si oui, on augmente *interBasesupScript* et *interBasesubScript* afin d'éviter le chevauchement des deux boîtes.

$indiceIntersection = 2 * ascent -$ Height$[base]$

si $indiceIntersection > 0$ (chevauchement) alors
$interBasesupScript = interBasesupScript + indiceIntersection/2$
$interBasesubScript = interBasesubScrip + indiceIntersection/2$

La boîte $SubSup$ s'aligne par rapport à la base.

Algorithme

- **Première partie**
  - La boîte de $SubSup$ a l'alignement de la $base$. Donc, l'origine de $base$ est (0,0).
  - Agrandir la boîte englobante pour contenir la boîte $base$.
  - Calculer la distance entre l'origine de la $base$ et celui de $subScript$.
    $dx =$ Width$[base]$
    $dy = -interBasesubScript$
  - Calculer la position de $subScript$
    Origine$[subScript]$ = Origine$[base]$ + (dx,dy)
  - Agrandir la boîte englobante pour contenir la boîte $subScript$ et sa nouvelle origine.
  - Calculer la distance entre l'origine de $subScript$ et $supScript$
    $dx = 0$
    $dy = -interBasesupScript - interBasesubScript$
  - Calculer la position de $subScript$
    Origine$[supScript]$ = Origine$[subScript]$ + (dx,dy)
  - Agrandir la boîte englobante pour contenir la boîte $supScript$ et sa nouvelle origine.
- **Deuxième partie**
  Déterminer le point de sortie de la boîte englobante.
  ExitPointX= OrigineX$[supScript]$+maxWidth$(supScript, subScript)$
  ExitPointY= ExitPointY$[base]$

4.4.10 La racine

Nous distinguons deux type boîtes pour la fonction racine: la boîte racine $n^{\text{ième}}$ et qui met n ou toute formule à gauche de sa base et la boîte racine simple qui prend en argument la base et qui calcule l'espace pour dessiner le symbole racine. La boîte *RacineNieme* prend en argument deux boîtes filles : la boîte qui représente la base, la boîte qui représente n. La taille de la boîte fille de n sera réduite de la même manière que l'indice pour la boîte Sub. Par contre la boîte *Racine* prend en argument une seule boîte, la base.

Soit $Racine$ la boîte Racine ayant une seule boîte fille $base$.

Paramètres

La boîte $Racine$ s'aligne par rapport à la base.

Algorithme

- **Première partie**
  - La boîte $Racine$ a l'alignement de la $base$. La base sera décalé à droite pour laisser l'espace pour dessiner le symbole de la racine.
    OrigineX[$base$] $=$ Height[$base$]$/2$
    OrigineY[$base$] $= 0$
  - Agrandir la boîte englobante pour contenir la boîte $base$ et sa nouvelle origine.
- **Deuxième partie**
  - Déterminer le point de sortie de la boîte englobante.
    ExitPointX= OrigineX[$base$]+ExitPointX[$base$]
    ExitPointY= ExitPointY[$base$]
  - Ajouter deux blancs au niveau de la hauteur et la largeur de la boîte englobante pour le dessin du symbole racine.
    UpperLeft=UpperLeft+$(2,2)$

Soit $RacineNieme$ la boîte Racine ayant une boîte fille $base$ et une boîte pour la racine nième $nieme$.

Paramètres

La boîte $RacineNieme$ s'aligne par rapport à la base.

Algorithme

- **Première partie**
  - Calculer la position de la racine nième.
    OrigineX[$nieme$] $= max(0,$ Height[$base$]$/4 -$ Width[$nieme$]$)$
    OrigineY[$nieme$] $= -$BottomRightY[$nieme$]
  - Agrandir la boîte englobante pour contenir la boîte $nieme$ et sa nouvelle origine.
  - Calculer la distance entre $nieme$ et la $base$.
    $dx =$ Width[$nieme$] $+$ Height[$base$]$/4$
    $dy =$ BottomRightY[$nieme$]
  - L'origine de $base$ par rapport à celui de $nieme$.

Origine$[base]$ = Origine$[nieme]$ + (dx,dy)

- Agrandir la boîte englobante pour contenir la boîte $base$ et sa nouvelle origine.

- **Deuxième partie**
  - Déterminer le point de sortie de la boîte englobante.
    ExitPointX= OrigineX$[base]$+ExitPointX$[base]$
    ExitPointY= ExitPointY$[base]$
  - Ajouter deux blancs au niveau de la hauteur et la largeur de la boîte englobante pour le dessin du symbole racine.
    UpperLeft=UpperLeft+$(2,2)$

Chapitre 5

Environnements de preuves et MathML

Contents

5.1 Introduction

Dans le but de permettre des échanges de données mathématiques entre applications et l'intégration de nos outils de développement de preuves sur le Web, nous nous sommes intéressés au lien entre FIGUE et le standard MathML des mathématiques sur le Web [62, 61]. Nous détaillons dans ce chapitre nos développements pour nous conformer au standard MathML.

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).

La figure 5.1 montre un exemple d'une formule mathématique et sa représentation en MathML-*contenu* et MathML-*présentation*. Notons qu'il est possible de mélanger les deux types de représentation (MathML-*contenu* et MathML-*présentation*) dans la même formule (mixed markup).

$$x^2 + 1 = 0$$

(a)

```
<math>
<apply>
  <eq/>
  <apply>
    <plus/>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <cn>1</cn>
  </apply>
  <cn>0</cn>
</apply>
</math>
```

(b)

```
<math>
<mrow>
  <mrow>
    <msup>
      <mi> x </mi>
      <mn> 2 </mn>
    </msup>
    <mo> + </mo>
    <mn> 1 </mn>
  </mrow>
  <mo> = </mo>
  <mn> 0 </mn>
</mrow>
</math>
```

(c)

Fig. 5.1 – *Un exemple d'une expression MathML. (a) formule mathématique. (b)* MathML-contenu. *(c)* MathML-présentation.

Le schéma de la figure 5.2 résume les deux niveaux de collaboration entre nos outils de développement de preuves et MathML: la collaboration entre la structure d'affichage de FIGUE et les éléments de présentation MathML (partie droite de la figure 5.2) 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 5.2).

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[1], 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.

Nous avons étendu FIGUE pour être compatible avec MathML : traduire la structure de boîtes FIGUE en structure MathML et inversement manipuler du MathML directement dans FIGUE. Cette fonctionnalité nous permet de communiquer les preuves (buts et scripts) sur le Web en utilisant des navigateurs compatibles avec MathML-*présentation*. Inversement, FIGUE peut être utilisé comme moteur d'affichage MathML-*présentation* par d'autres applications. Nos programmes permettent aussi de manipuler des données présentées par MathML-*contenu*.

Dans le cadre de FIGUE, nous avons choisi de traiter d'abord les éléments de présentation de MathML. Puis nous avons intégré les éléments de description sémantique (MathML-*contenu*) dans notre système. Cette intégration était 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 qui est indépendante de FIGUE puisque le *contenu* ne concerne pas directement l'affichage. FIGUE permet l'affichage des objets mathématiques MathML et leur manipulation.

1. *Document Object Model.*

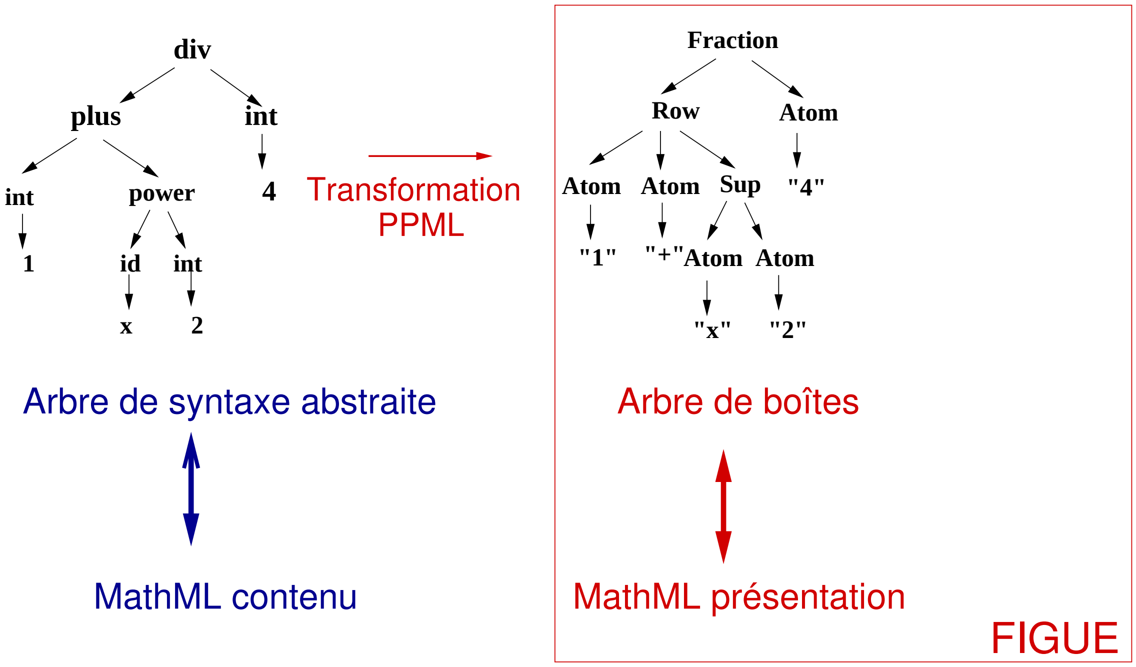


FIG. 5.2 – *Deux niveaux de collaboration entre nos outils et MathML: d'une part entre la structure de syntaxe abstraite et* MATHML-contenu *(gauche) et d'autre part entre la structure d'affichage de* FIGUE *et* MATHML-présentation *(droite).*

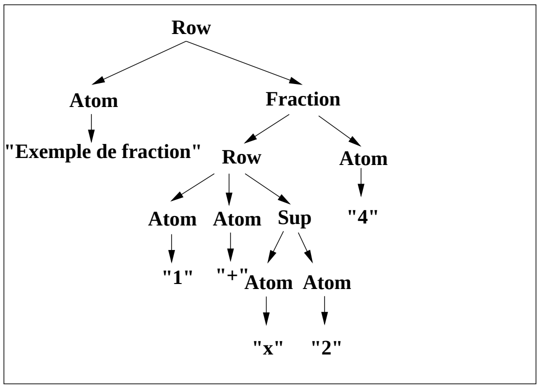
5.2 Implémentation de MathML dans FIGUE

FIGUE implémente du MATHML-*présentation* : il permet d'afficher les éléments de présentation MathML et de les manipuler. Nous avons implémenté les principaux éléments de MATHML qui recouvrent les constructeurs mathématiques de FIGUE. Néanmoins, quelques éléments et attributs n'ont pas encore été implémenté (comme mlabeledtr, mmultiscripts, maction ..). 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 boîtes filles doivent être disposés et affichés. La figure 5.3 montre un exemple de la boîte *Fraction* qui dispose le numérateur au-dessus du dénominateur et sépare les deux par une barre de fraction : la figure 5.3 montre d'abord le code source MathML de l'exemple de fraction (figure 5.3a), l'arbre de boîtes FIGUE correspondant (figure 5.3b) et son affichage par FIGUE (figure 5.3c).

```
<xml version="1.0" encoding="UTF−8" standalone="no"?>
<!DOCTYPE math SYSTEM "http://www.w3.org/TR/
MathML2/dtd/mathml2.dtd">
<math xmlns="http://www.w3.org/1998/Math/MathML">
<mrow>
  <mtext> Exemple de fraction </mtext>
  <mfrac>
    <mrow>
      <mn> 1 </mn>
      <mo> + </mo>
      <msup>
        <mi> x </mi>
        <mn> 2 </mn>
      </msup>
    </mrow>
    <mn> 4 </mn>
  </mfrac>
</mrow>
</math>
```

(a)



(b)

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

(c)

FIG. 5.3 – *Exemple de fraction: (a) encodage MathML, (b) arbre de boîtes* FIGUE*, (c) résultat du formatage.*

5.2.1 Générer du MathML-*présentation* à partir de FIGUE

A partir des objets graphiques affichés par FIGUE, il est possible de générer un document XHTML incluant du MATHML-*présentation*[2]. Les éléments HTML sont générés à partir des objets graphiques pour le formatage du texte dans FIGUE (comme *Paragraphe*, *Vertical*, *Table*, *Horizontal*) et les éléments MATHML-*présentation* sont générés à partir des objets mathématiques dans FIGUE.

Le document XHTML généré à partir de FIGUE peut être visualisé par tous les navigateurs implémentant du MathML (Amaya,Mozilla, etc.). Prenons l'exemple d'une preuve développée sous PCOQ et affichée par FIGUE (voir figure 5.4). A partir de cette preuve, FIGUE génère le XHTML correspondant. La figure 5.5 montre le résultat de l'affichage de ce document XHTML par Amaya (le code source XHTML de cet exemple est mis en http://www-sop.inria.fr/lemme/Hanane.Naciri/these/raportthese/pcoqxhtml.ml).

Pour générer du XHTML+MathML à partir des objets graphiques FIGUE, nous définissons pour chaque objet FIGUE l'élément HTML ou MathML correspondant. Cette correspondance se fait naturellement puisque la structure de formatage dans FIGUE est proche de celle du langage HTML (paragraphe, table ...) et du langage MathML pour les objets mathématiques. Nous avons essayé de respecter le plus possible les spécifications de la norme MathML (le type et les attributs des éléments MathML) pour établir cette correspondance. Quelques difficultés ont été soulevées pour faire cette correspondance: par exemple le passage des entités atomiques de FIGUE vers celles de MathML demande de déduire le type de ces entités (identificateur mi , nombre mn, opérateur mo, ...) qui n'est

2. Nous présentons ici la traduction dans ce sens Structure de boîte FIGUE → MATHML-*présentation*.

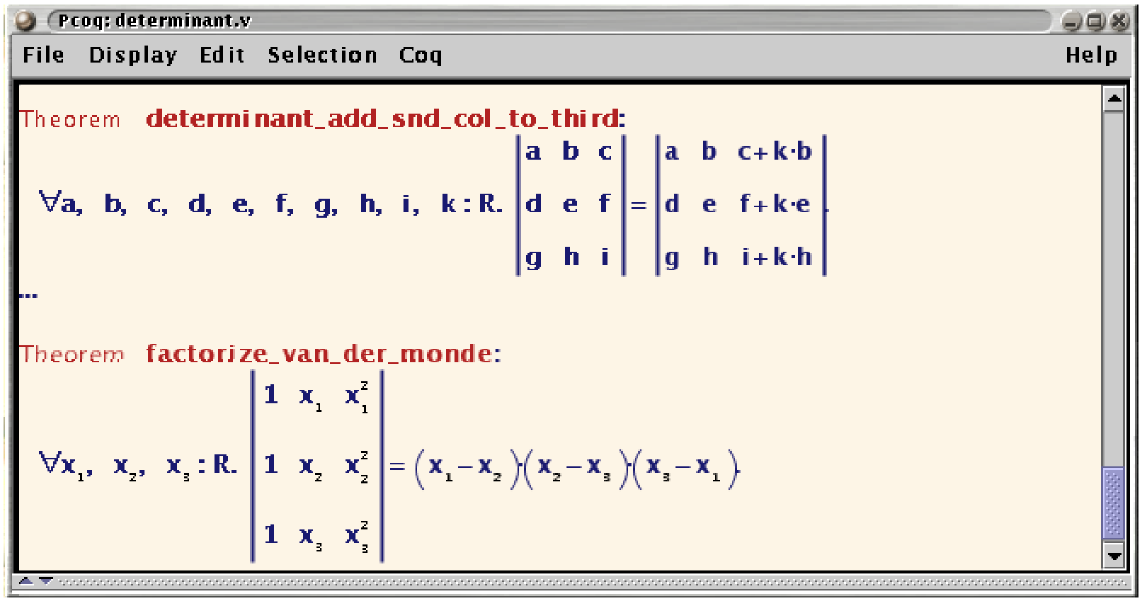


FIG. 5.4 – *Exemple d'une preuve de l'algèbre linéaire présentée par* PCOQ.

pas spécifié par FIGUE. Une table de correspondance entre les boîtes FIGUE et les éléments MathML est décrite en annexe E.

5.2.2 Afficher et manipuler du MathML-présentation par FIGUE

Nous avons développé dans FIGUE un module permettant l'interprétation des documents structurés XML décrivant les boîtes FIGUE et incluant des expressions MathML[3]. Il s'agit premièrement d'effectuer l'analyse syntaxique[4] de l'ensemble du document XML suivant une grammaire spécifiée par sa DTD (*Document Type Definition* voir annexe F) afin d'en extraire l'arbre 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 5.6). 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 5.7) :

- traduire directement, en code Java, les objets mathématiques MathML en des objets structurés FIGUE (figure 5.7a) ;

3. Nous présentons ici la traduction dans ce sens MATHML-*présentation* → structure de boîte FIGUE.
4. Nous utilisons l'analyseur syntaxique Xerces (Xerces Java Parser 1.0.3) pour produire l'arbre DOM à partir du fichier XML.

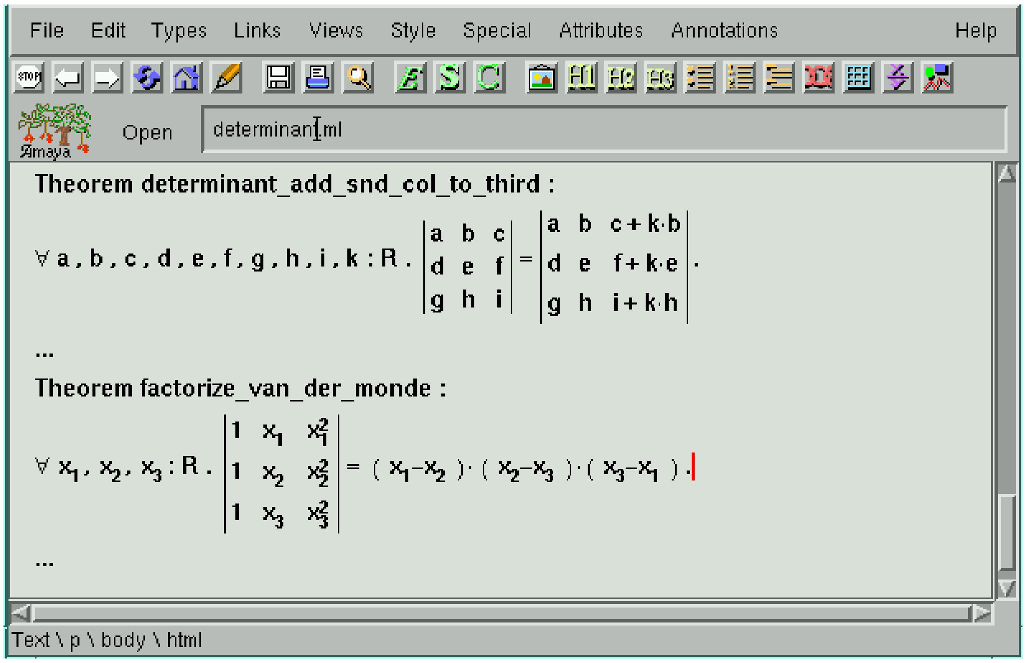


FIG. 5.5 – *L'affichage par Amaya d'un document XHTML généré à partir de* FIGUE.

– traduire à l'aide des règles de transformation paramétrées les objets mathématiques MathML en des objets structurés FIGUE (figure 5.7b).

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 5.7a). Pour chaque élément MathML, nous associons le sous-arbre de boîtes FIGUE correspondant et nous assurons, par codage, le lien entre les deux structures. Ce lien se fait à l'aide des tables de Hachages.

La deuxième implémentation applique des règles de transformation d'un protocole nommé XPPML[5](voir section 5.2.2.1) pour traduire chaque élément MathML (ou plus généralement des éléments XML) en une structure de FIGUE pour être affiché (voir figure 5.7b). Notons que les règles XPPML permettent de générer du FigueML (forme XML pour décrire les boîtes FIGUE), du HTML, du xsl:fo ou d'autres formes XML pour le formatage des données. Nous avons mis en place aussi des règles XPPML pour traduire du MathML en FigueML dans le but d'avoir un exemple d'utilisation de XPPML pour traduire une structure XML en une autre structure XML.

En conclusion, 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.

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

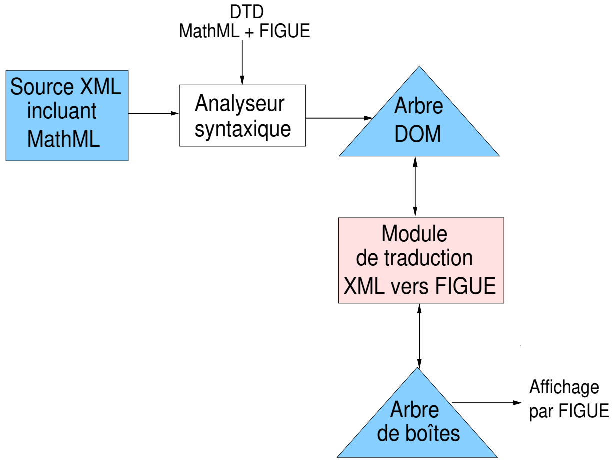


FIG. 5.6 – *Module d'interprétation de MathML vers* FIGUE *et vice versa.*

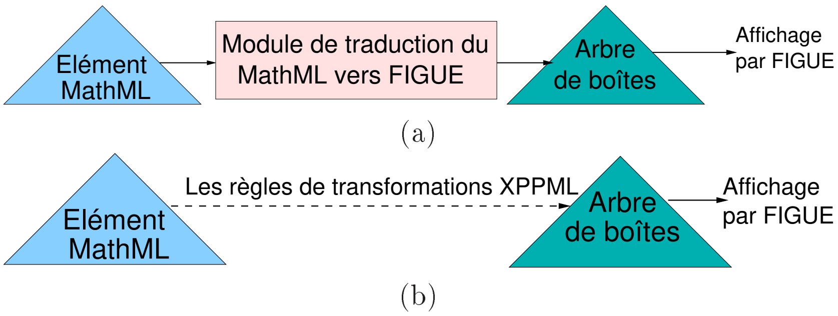


FIG. 5.7 – *Deux implémentations possibles pour interpréter du MathML dans* FIGUE. *(a) Schéma de traduction directe d'un élément MathML en un objet structuré* FIGUE. *(b) Schéma de traduction par utilisation de règles XPPML sur un élément MathML pour obtenir un objet structuré* FIGUE.

Notons que dans FIGUE il est possible de produire du SVG à partir de l'arbre de boîtes déjà formaté (voir section 3.1.7) et d'afficher du MathML. En combinant ces deux fonctionnalités de FIGUE, nous pouvons générer du SVG à partir de la structure MathML. Cela pourra être un autre moyen pour nous de manipuler les formules mathématiques sur le Web en utilisant SVG. Les travaux [78, 18] se sont intéressés à la génération du SVG à partir du MathML.

Dans la section suivante, nous présentons le langage XPPML [66] développé au sein de notre équipe LEMME, que nous avons utilisé pour traduire du MathML vers les objets FIGUE.

5.2.2.1 Description du langage XPPML

XPPML est une extension XML du langage PPML (voir section 3.1.1). Les règles XPPML décrivent au moyen d'un document XML les règles de conversion d'un arbre source en un autre arbre résultat.

Une règle XPPML associe à un pattern avec template une structure de formatage décrite en XML. Le compilateur XPPML traverse l'arbre source et pour chaque noeud cherche (match) la règle correspondante et une fois trouvée, cette template est instanciée pour produire le sous-arbre résultat.

Le compilateur des règles XPPML maintient le lien entre la structure de l'arbre source et la structure de l'arbre résultat (le même mécanisme est utilisé dans PPML, voir section 3.1.2). Il est possible d'obtenir le sous-élément dans la structure de départ (source) correspondant à un élément sélectionné dans la structure d'arrivée (résultat). Il est aussi possible d'organiser les spécifications XPPML sous plusieurs modules : un fichier XPPML a la possibilité d'importer un autre fichier XPPML.

Un fichier XPPML a la structure suivante:

```
<prettyPrinter ppName="..." langName="...">
<extension name="...">
<import package="..."/>
<rule>
...
</rule>
</prettyPrinter>
```

Une spécification XPPML est identifiée par le nom du langage de formatage "Pretty Printer" (ppName) et aussi par le nom du langage sur lequel il va être appliqué (langName). L'élément *extension* de XPPML est utilisé pour inclure une à plusieurs spécifications dans le même fichier XPPML. L'élément *import* permet d'importer un ensemble de méthodes externes Java pour les utiliser par la suite dans les règles XPPML. Dans le cas ou nous utilisons XPPML pour transformer une structure MathML-*présentation* vers une structure de boîtes FIGUE, le nom du langage de formatage est *toFigue* (ppName="toFigue") et le nom du langage source est MathML-*présentation* (langName="mathml-presentation").

Définition d'une règle XPPML

La figure 5.8 montre quelques règles XPPML utilisées pour traduire l'exemple de fraction codée en MathML (voir figure 5.3) 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.

Une règle XPPML est composée de deux parties: la première partie *pattern* (l'équivalent de la partie gauche d'une règle PPML) et la deuxième partie est le format correspondant *format* (l'équivalent de la partie droite d'une règle PPML). L'équivalent de la règle dans la figure 5.8.b en PPML est :

$$\mathrm{mfrac}(*num, *den) \quad \rightarrow \quad [<\mathrm{Fraction}> \quad *num \; *den]$$

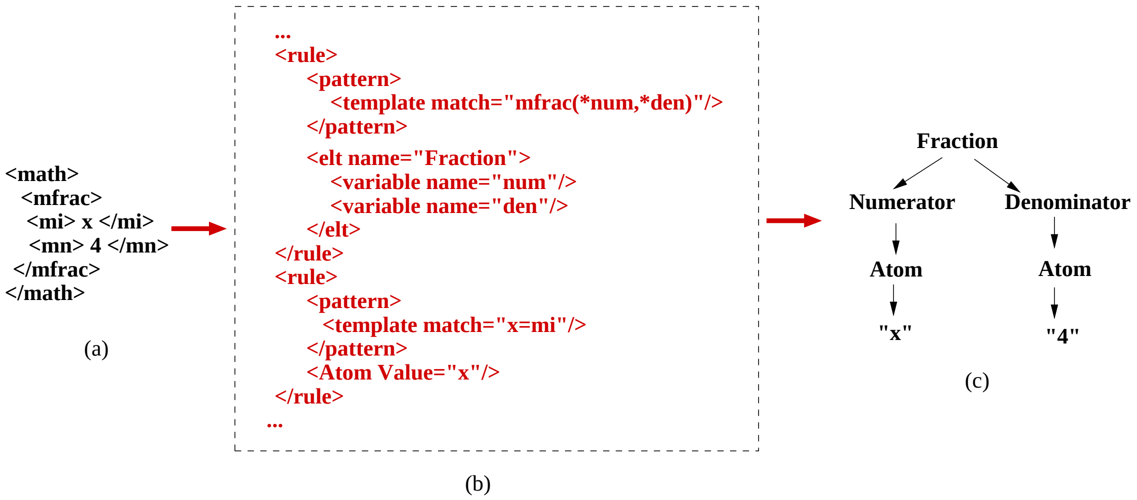


FIG. 5.8 – *Un exemple de traduction d'un élément MathML en un objet structuré* FIGUE *par application de règles XPPML. (a) élément MathML. (b) règles XPPML. (c) objet structuré* FIGUE.

Pattern

Le *pattern* d'une règle est défini par l'élément *template*. Cet élément a un attribut *match* qui identifie le noeud source pour lequel la règle s'applique. Le tableau suivant montre la syntaxe de quelques termes de pattern matching de syntaxe.

$*$	sélectionne tous les éléments.
$*x$	sélectionne tous les éléments et garde le noeud sélectionné dans la variable x.
$**y$	sélectionne une liste d'éléments et garde cette liste dans la variable y.
list($*$first,$**$other)	sélectionne toute liste d'éléments avec deux fils, le premier fils est mis dans la variable *first* et la liste des autres éléments est gardée dans la variable *other*.

Il est possible d'ajouter à chaque règle un ensemble de contraintes. La règle ne sera sélectionnée et exécutée que si ces contraintes sont satisfaites. Nous distinguons quatre types de contraintes: les contraintes sur la profondeur des arbres à traiter, les contraintes définies par des fonctions externes de type booléen et spécifiées par l'attribut *test* de l'élément *constraint* (par exemple <constraint test="myFun(∗value)"/>), les contraintes sur les annotations attachées aux noeuds sélectionnés (l'attribut *annotation* de l'élément *pattern* contient le nom de l'annotation) et les contraintes utilisant les informations sur le contexte, qui permettent de traiter différemment le même élément selon le traitement de ses précédents éléments.

Format

La deuxième partie d'une règle XPPML correspond à la structure de formatage en sortie et qui peut-être par exemple du HTML, du xsl:fo ou notre structure FIGUE.

Les éléments utilisés pour décrire cette partie de la règle sont :

- *variable* représente le résultat de l'appel récursif sur une variable définie dans la partie pattern de la règle courante.
- *extFun* représente l'appel à une fonction externe qui s'occupe de la création du sous-arbre résultat.
- *iter* représente un appel récursif sur tous les éléments d'une variable de type liste.
- *if* représente une structure conditionnelle qui dépend du résultat d'une fonction booléenne externe. Les sous-éléments *then* et *else* définissent le format résultat si la fonction retourne *true* ou *false* respectivement.
- *case* est une autre structure de contrôle qui permet de mettre des conditions sur le formatage en fonction de la valeur numérique de retour d'une fonction externe. Les sous-éléments *select* et *default* définissent l'affichage à appliquer.

Bilan

L'utilisation des règles XPPML pour traduire du MathML vers la structure FIGUE représente un bon exemple pour tester le compilateur XPPML développé au sein de l'équipe LEMME. Lors de cette traduction, nous avons rencontré des problèmes pour le traitement des attributs des éléments de la structure d'arrivée. Au départ, XPPML permet uniquement de mettre des attributs avec des valeurs connues. Nous avons étendu ce langage pour pouvoir affecter aux attributs des valeurs obtenues par des appels de fonction externe sur les variables.

Prenons un exemple de règles XPPML montrant le besoin de récupérer la valeur des attributs par un appel de fonction :

Une règle appliquée dans le cas d'un vecteur horizontal *mrow* ayant trois fils dont le premier et le dernier fils sont de types *mo*.

```
<rule>
<pattern>
  <template match="mrow(*first,*y,*last)"/>
</pattern>
<if>
<funcall name="isMoElementpp">
<arg value="first" type="var"/>
<arg value="last" type="var"/>
</funcall>
<then>
```

Dans ce cas, cette règle produit une boîte *délimiteur* de FIGUE. Elle spécifie ses attributs par un appel d'une fonction externe *typeBracketpp* qui détermine le type des symboles délimitant.

```
 <then>
<elt name="delimitor">
<att name="type" value="LEFT_RIGHT"/>
<att name="leftbracket">
<extFun name="typeBracketpp">
 <arg value="first" type="var"/>
</extFun>
</att>
<att name="rightbracket">
<extFun name="typeBracketpp" >
 <arg value="last" type="var"/>
</extFun>
</att>
<variable name="y"/>
</elt>
</then>
 </rule>
```

Un autre choix possible serait d'utiliser le langage XSLT pour traduire du MathML vers notre structure de boîtes. Toutefois l'architecture de FIGUE se prête bien à l'utilisation du langage XSLT comme alternative à XPPML. Une étude regardant comment utiliser XSLT comme alternative à PPML et à XPPML a été effectuée dans notre équipe. Cette étude montre comment traduire les règles PPML en XSLT tout en proposant pour chaque concept son équivalent en XSLT s'il existe ou sinon une autre alternative.

5.2.3 Manipuler du MathML-*contenu* dans FIGUE

Pour manipuler du MATHML-*contenu* dans FIGUE, nous utilisons les règles XPPML (comme pour la deuxième technique utilisée pour manipuler du MATHML-*présentation* et vu dans la section 5.2.2). Les règles XPPML permettent de transformer les éléments du MATHML-*contenu* en une structure d'affichage FIGUE. Pour chaque élément MATHML-*contenu*, une règle XPPML par défaut est spécifiée. L'utilisateur peut aussi spécifier son propre style et ses propres notations pour afficher des éléments MATHML-*contenu* en ajoutant de nouvelles règles XPPML. Prenons comme exemple une règle XPPML spécifiant comment afficher une formule décrite en MATHML-*contenu* et contenant à la fois une relation logique et un intervalle représenté par l'élément MathML *apply* et l'élément *interval* (voir figure 5.9).

Nous associons à l'élément MathML *apply* dont son premier fils est un opérateur de relation <in/> (l'opérateur $\in$) et ses deux autres fils sont $*x$ et $*y$, la boîte graphique *Row* de FIGUE avec trois fils. Le premier et le dernier fils de la boîte *Row* sont les résultats d'un appel récursif des fonctions d'affichages sur $*x$ et $*y$ respectivement; tandis que le

```
<rule>
  <pattern>
    <template match=”apply(op=in,*x,*y)”/>
  </pattern>
  <elt name=”Row”>
    <variable name=”x”/>
    <extFun name=”getUnicodepp”>
      <arg value=”op” type=”var”/>
    </extFun>
    <variable name=”y”/>
  </elt>
</rule>
```

(a)

```
<rule>
  <pattern>
    <template match=”z=interval(*x,*y)”/>
  </pattern>
  <if>
    <funcall name=”isOpenIntervalpp”>
      <arg value=”z” type=”var”/>
    </funcall>
    <then>
      <elt name=”row”>
        <elt name=”atom”>
          <att name=”value” value=”]”/>
        </elt>
        <elt name=”row”>
          <variable name=”x”/>
          <elt name=”atom”>
            <att name=”value” value=”,”/>
          </elt>
          <variable name=”y”/>
        </elt>
        <elt name=”atom”>
          <att name=”value” value=”[”/>
        </elt>
      </elt>
    </then>
  </if>
</rule>
```

(b)

Fig. 5.9 – *(a) Les règles de transformation XPPML d'une relation logique. (b) Les règles XPPML d'un intervalle ouvert.*

deuxième fils est une boîte *Atom* ou sa valeur est le résultat d'un appel de la fonction externe JAVA *getUnicodepp* qui retourne le caractère Unicode correspondant à l'opérateur de relation *op* (si la valeur de *op* est *in*, cette fonction retourne le caractère Unicode correspondant \u2208). Pour des questions de lisibilité, dans la suite nous décrivons ces règles de transformations par le langage de plus haut niveau PPML. En syntaxe PPML, les règles correspondantes à la transformation du précédent exemple (Figure 5.9) sont écrites comme suit :

```
apply(op=in, *x, *y)     →   [<Row>  *x getUnicodepp(op) *y]
*z as interval(*x, *y)   →   if isOpenIntervalpp(*z)
                              then
                              [<Row>  “]” [<Row> *x “,” *y] “[”]
                              end
```

La figure 5.10 montre la représentation de la formule $x \in]a,b[$ en MathML-*contenu* et sa structure de boîte FIGUE après l'application des règles de transformation XPPML décrites dans la figure 5.9.

Les règles de transformation XPPML donnent la possibilité à l'utilisateur de spécifier sa propre notation. Certains objets mathématiques changent leur notation selon leur contexte d'utilisation (par exemple selon les pays ou la théorie mathématique). Dans le précédent exemple, au lieu d'écrire l'intervalle ouvert $]a,b[$, on peut vouloir l'écrire sous la forme

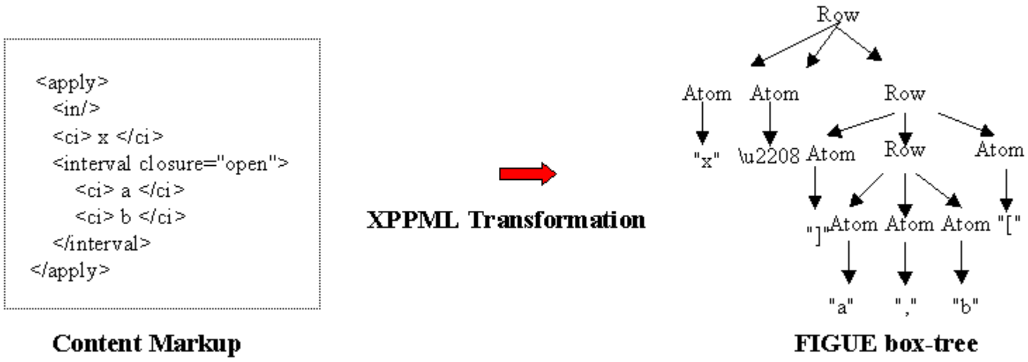


FIG. 5.10 – *La transformation de la formule mathématique $x \in]a, b[$ à partir de la structure* MATHML-contenu *vers la structure de boîtes* FIGUE.

(a, b). Cela peut être fait à l'aide de la règle suivante :

z as interval(x, *y) $\rightarrow$ if isOpenIntervalpp(*z)
then
[<Row> "(" [<Row> *x "," *y] ")"]
end

Le choix de ses propres notations est souvent nécessaire pour bien comprendre les documents mathématiques. Les notations mathématiques changent selon le contexte de leur utilisation et aussi selon les cultures et l'époque, par exemple le coefficient binomial C_m^n en notation française est équivalent à C_n^m en notation russe et à $\binom{n}{m}$ en notation américaine.

Ce module d'interprétation peut être utilisé pour déterminer les notations de chaque élément contenu dans différentes configurations. Certains éléments ont une large variété de notation qui dépend de leur contexte. Par exemple, $tanx$ au lieu de tgx. Parfois il est aussi utile d'utiliser les règles de transformation pour produire un affichage non ambigu de l'affichage des formules mathématiques, par exemple afficher $(a + b)!$ au lieu de $a + b!$, ou afficher $a - b + c$ au lieu de $a + -b + c$.

5.3 Arbre de Syntaxe Abstraite et MathML-*contenu*

Les objets manipulés dans nos environnements de preuves et contenant des formules mathématiques sont des objets structurés représentés par des arbres de syntaxe abstraite (AST). Cette structure permet de faire des manipulations puissantes sur les objets symboliques. L'utilisation des arbres comme structure de données facilite la traduction de nos documents de preuves vers MATHML-*contenu*. Toutes les formules logiques apparaissant dans des preuves peuvent être représentées directement avec MATHML-*contenu* tout en gardant la même sémantique (contenu). Par contre, les autres objets abstraits comme par exemple *theorem* ou *definition* n'ont pas d'éléments correspondants dans le langage MathML. Pour résoudre ce problème, nous avons adopté notre propre langage XML représentant ces objets abstraits avec des tags XML. Nous utilisons uniquement les éléments MathML-*contenu* pour représenter les formules mathématiques incluses dans les termes de preuves.

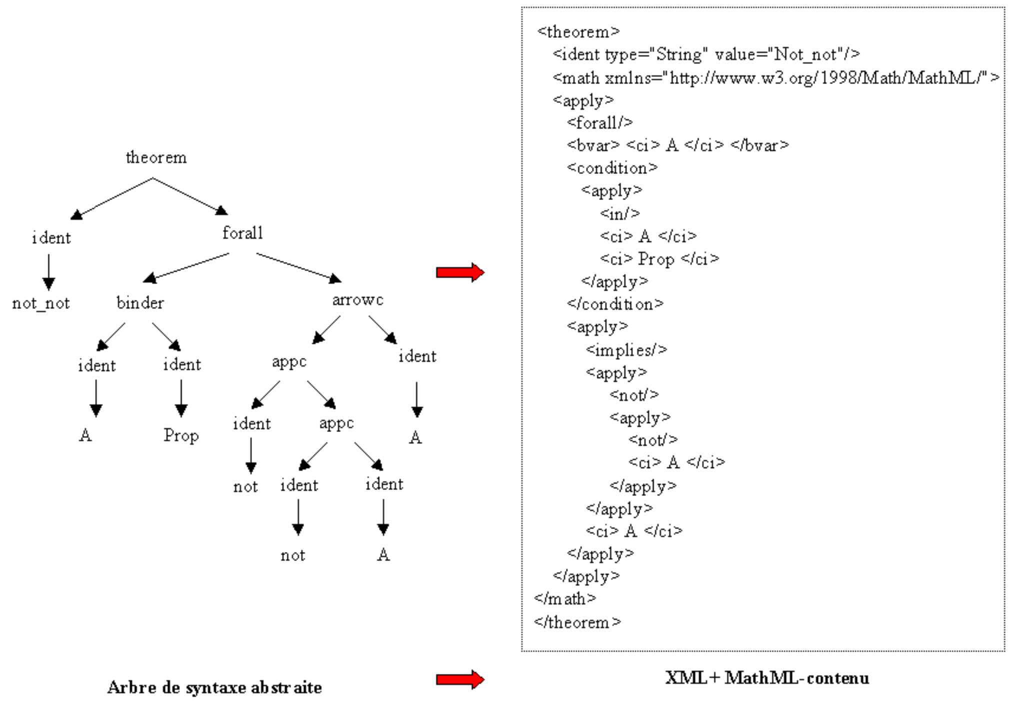


FIG. 5.11 – *Theorem Not_not* : $\forall A \in Prop\ \neg(\neg A) \Rightarrow A$. *traduit depuis l'AST vers XML+*MathML.

Pour représenter une preuve entière uniquement avec un langage standard comme MathML-*contenu*, nous avons besoin d'avoir la possibilité de créer de nouveaux objets mathématiques dans ce langage. Pour le moment, nous traduisons les formules des sous parties de preuves en MathML-*contenu*. Cette traduction nous permet de collaborer avec d'autres systèmes de calcul symboliques utilisant MathML comme protocole de communication (par exemple, demander un calcul à un système de calcul formel).

Lors de la traduction des formules d'une preuve en MathML-*contenu*, parfois nous perdons la notion de typage forte en preuve. MathML prend en compte que quelque types de base comme le type entier, réel et complexe. Par exemple, présenter en MathML x de type *nat* ($x : nat$), on perd le type de la variable x.

La figure 5.11 montre un exemple de traduction d'un arbre de syntaxe abstraite correspondant à cette déclaration de théorème: $Theorem\quad not\_not:\quad \forall A \in Prop\quad \neg(\neg A) \Rightarrow$

A. en code XML. Le corps de ce théorème est représenté par MathML-*contenu* et la déclaration du type de la variable A est représentée par l'élément *condition* en supposant que A a le bon type (nous nous sommes pas sûr que c'est la bonne façon d'exprimer le typage). Les noeuds de l'arbre de syntaxe abstraite qui ne sont pas des formules sont encodés par des tags XML, comme <theorem> et <ident/>.

Prenons le même exemple de la figure 5.11 pour montrer les principales étapes de la traduction d'une preuve sous PCOQ vers MathML-*contenu*. Cette figure 5.12 montre une session PCOQ où nous avons défini le théorème Not_not.

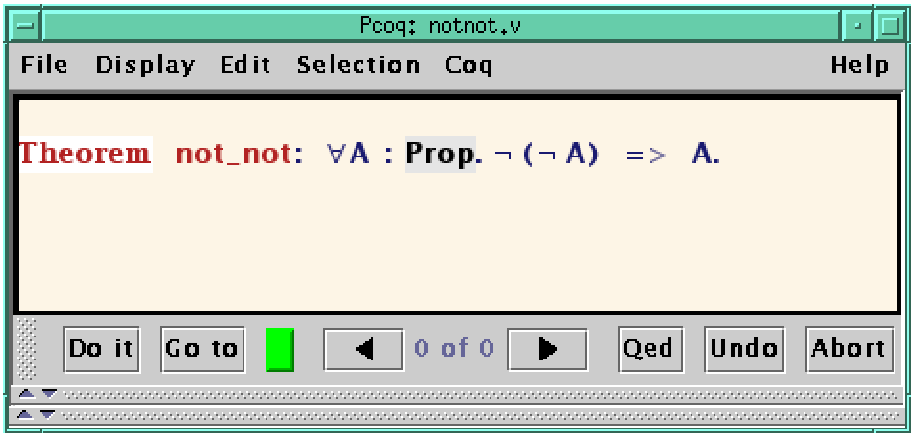


Fig. 5.12 – *La définition du théorème not_not dans* PCOQ.

La description de l'arbre de syntaxe abstraite correspondant à l'exemple de la figure 5.12 est la suivante :

```
command_list[
theorem_goal(
   thm Theorem,
   ident not_not,
   prodc(
      binder(id_opt_ne_list[ident A],sortc Prop),
      arrowc(appc(ident not,
                  formula_ne_list[appc(ident not,
                                       formula_ne_list[ident A])]),
             ident A)))]
```

Les arbres de syntaxe abstraite suivent un langage décrit par un formalisme nommé *vernac*. Une partie de la description de ce langage correspondant aux commandes utilisées dans notre exemple est la suivante :

```
formalism of vernac is

command_list  -> COMMAND + ... ;
theorem_goal  -> DEFN_OR_THM ID FORMULA ;
```

```
COMMAND  ::= command_list theorem_goal ....
defn  -> implemented as STRING ;
DEFN  ::= defn ;
thm  -> implemented as STRING ;
THM  ::= thm ;
DEFN_OR_THM  ::= DEFN THM ;
ident  -> implemented as STRING ;
ID  ::= ident ;
FORMULA  ::= appc prodc arrowc ...

appc  -> FORMULA FORMULA_NE_LIST ;
prodc  -> BINDER FORMULA ;
arrowc  -> FORMULA FORMULA ;

formula_ne_list  -> FORMULA + ... ;
FORMULA_NE_LIST  ::= formula_ne_list ;

binder  -> ID_OPT_NE_LIST FORMULA ;
BINDER  ::= binder ;

id_opt_ne_list  -> ID_OPT + ... ;
ID_OPT_NE_LIST  ::= id_opt_ne_list ;

ID_OPT  ::= ID NONE ;

none  ->  ;
NONE  ::= none ;
```

Cet arbre de syntaxe suivant ce formalisme va être transformé en XML + MathML-*contenu* à l'aide des règles de transformation décrites par le langage PPML. Nous avons effectué quelques tests pour vérifier la validité du XML généré par rapport à sa DTD et à celle de MathML-*contenu*.

Une partie des règles PPML utilisée pour traduire l'arbre de syntaxe abstraite de notre exemple en un document XML +MathML-*contenu* est :

```
prettyprinter mathml of vernac is
auxillary package "lang.vernac.xxmathml";
command_list[**vernac_list] -> [<v 0,0>  (**vernac_list)];
```

Cette règle associe au terme *Theorem*, l'élément XML "<theorem>" où le premier fils est le résultat de l'application d'une règle sur la variable x dans un contexte *proof* (proof:*x) et le deuxième fils est l'élément <math> de MathML englobant le résultat de l'application d'une nouvelle règle sur la variable y.

```
theorem_goal(thm "Theorem", *x, *y) -> [<v 1,0> "<theorem>" proof:*x
"<math xmlns=\"http://www.w3.org/1998/Math/MathML/\">" *y "</math>"
"</theorem>"];
proof:ident *x -> [<hv 0,2,0> "<ident type=\"String\" value=" stringpp(*x)"/>"];
```

Ici, on associe à la fonction $forall$, la description en MathML de l'application de l'opérateur $forall$ avec des conditions de type sur les variables.

```
prodc (*bind, *form) -> [<v 0,0> "<apply>" "<forall/>" forall:*bind  *form
"</apply>"];
forall:binder(*id_ne_list, *formula) -> [<v> bvar:*id_ne_list "<condition>"
"<apply>" "<in/>" cond:*id_ne_list cond:*formula "</condition>"];

bvar:id_opt_ne_list[**id_list] ->   [<v> (bvar1:**id_list)];
cond:id_opt_ne_list[*id]  ->  [<h 0> "<ci>" identitypp(*id) "</ci>"];
cond:id_opt_ne_list[**id_list] -> [<v> "<list>" (cond1:**id_list) "</list>"];
cond:ident *x -> [<h 0> "<ci type=\"Set\">" identitypp(*x) "</ci>"];
cond:sortc *x  -> [<h 0> "<ci type=\"Set\">" identitypp(*x) "</ci>"];
bvar1:none() -> [<h 0> ];
bvar1:ident *x -> [<h 0> "<bvar> <ci>" identitypp(*x) "</ci> </bvar>"];
cond1:none() -> [<h 0> ];
cond1:ident *x -> [<h 0> "<ci>" identitypp(*x) "</ci>"];
```

Les règles suivantes associent à un identificateur *ident*, l'élément *ci* de MathML, à la fonction *arrowc* l'application de l'opérateur *implies* décrit en MathML et à l'identificateur *not* son équivalent en MathML.

```
ident *x ->  [<h 0> "<ci>" identitypp(*x) "</ci>"];
arrowc(*f1, *f2) ->  [<v 0,0> "<apply>" "<implies/>" *f1 *f2 "</apply>"];
appc(ident "not", formula_ne_list[*x]) -> [<v 0,0> "<apply>" "<not/>" *x
"</apply>"];
```

Pour chaque noeud de l'arbre de syntaxe abstraite, on associe un formatage (soit en horizontal ou en vertical) d'un texte décrivant les éléments XML correspondants. Les boîtes de formatage (h ou v) servent uniquement pour formater le document XML résultat. Par exemple, on associe au noeud *ident*, l'élément MathML *ci* pour représenter un identificateur (ident *x → [<h 0> "<ci>" identitypp(*x) "</ci>"];).

La figure 5.13 montre le document XML généré à partir de l'exemple de la figure 5.12. Le résultat est affiché dans une fenêtre PCOQ. Le code XML généré est valide par rapport à la DTD décrite dans :

http://www-sop.inria.fr/lemme/Hanane.Naciri/these/raportthese/vernac.dtd.

5.4 Conclusion

La collaboration entre nos outils de développement de preuves et MathML existe concrètement dans PCOQ sous la forme suivante :

– Nous suivons les recommandations du standard MATHML pour définir les bons constructeurs de bases avec les bons attributs dans notre moteur d'affichage.

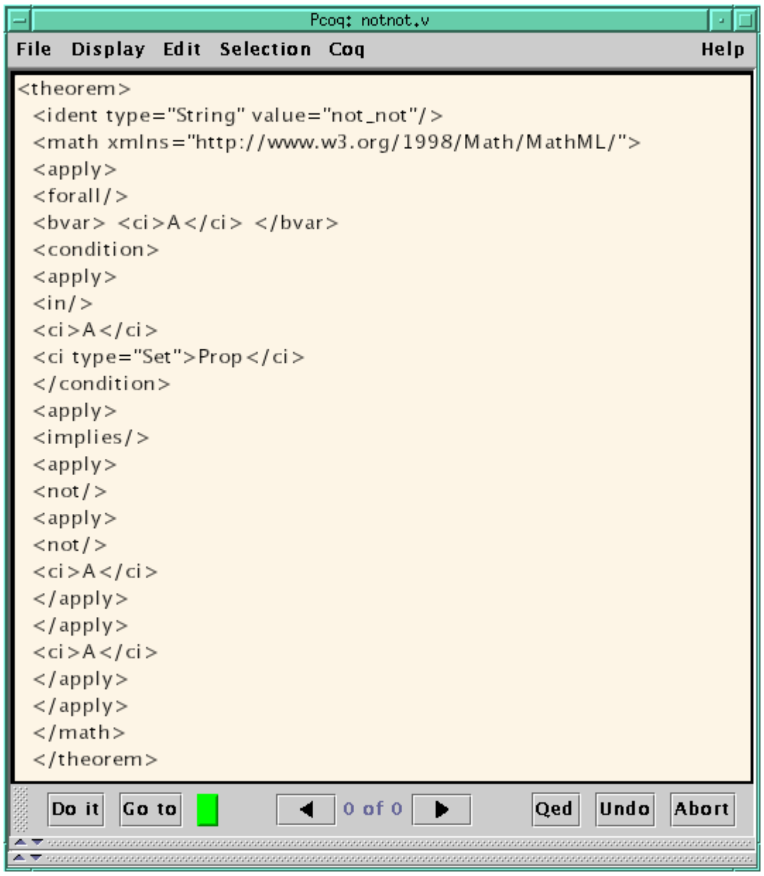


FIG. 5.13 – *La description en XML+*MATHML *de la définition du théorème not_not généré par* PCOQ.

- La correspondance naturelle entre les objets MATHML-*présentation* et les constructeurs FIGUE facilite la traduction entre les deux formalismes, ce qui permet la présentation des preuves sur le Web.
- La représentation de nos données sous forme d'arbre de syntaxe abstraite (structure d'arbres) rend la traduction vers MATHML-*contenu* facile, ce qui nous permet de collaborer avec d'autres systèmes de calcul symboliques en utilisant MATHML comme protocole de communication.

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.

L'ajout du support MathML dans FIGUE permet de l'utiliser comme composant 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 le Web.

Chapitre 6

Affichage bidirectionnel

Contents

Nous décrivons d'abord les problèmes que pose l'affichage de texte arabe (de droite à gauche) dans un contexte prévu pour l'affichage de texte indo-européen (de gauche à droite), puis l'affichage bidirectionnel mélangeant les deux modes (par exemple du texte arabe incluant du texte indo-européen ou des formules mathématiques). Nous présentons ensuite une extension de la librairie FIGUE que nous avons développée pour afficher des textes de preuves mathématiques en langue arabe. Nous montrons aussi comment cette expérience peut nous servir à diffuser ces textes mathématiques sur le Web, grâce à l'usage de MathML dans un contexte bidirectionnel.

6.1 Introduction

Dans ce chapitre, nous décrivons comment nous avons étendu FIGUE [30, 59], notre moteur d'affichage interactif d'objets structurés, pour l'adapter au formatage de droite à gauche du texte en arabe, hébreu ou perse et étudier comment mélanger les directions dans le cas du texte comportant des formules mathématiques (texte de droite à gauche, incluant des formules écrites de gauche à droite). Notre but est de fournir un composant d'affichage bidirectionnel permettant de manipuler et de mélanger les formules mathématiques et le texte. Comme nous l'avons déjà vu précédemment le problème se pose de façon particulière pour les formules mathématiques qui ont leurs propres règles d'affichage bidirectionnel différentes de celles appliquées pour du texte simple (linéaire). Cette différence est due à la nature bidimensionnelle des formules mathématiques et à l'importance des relations spatiales dans les notations mathématiques. Nous distinguons deux familles de formules mathématiques en arabe : les formules à l'égyptienne écrites de droite à gauche conformément au sens de déroulement de l'écriture arabe et les formules à la marocaine écrites de gauche à droite suivant le modèle européen mais qui peuvent contenir du texte ou des symboles arabes.

FIGUE est utilisé actuellement pour le développement du système PCOQ (voir section 3.2.2). Il est possible dans PCOQ d'avoir des explications des preuves mathématiques en langue naturelle [25, 2], en français ou en anglais. Nous avons expérimenté l'implémentation des explications en arabe, ce qui permet de tester notre algorithme d'affichage bidirectionnel dans FIGUE pour écrire de droite à gauche ou même mélanger l'affichage droite-gauche et l'affichage gauche-droite (texte contenant des formules mathématiques).

Les implémentations actuelles de MathML permettent uniquement l'affichage de formules mathématiques de gauche à droite. Nous collaborons avec le professeur Stephen Watt, invité expert du groupe de travail W3C autour de MathML, dans le but de définir l'usage de la norme MathML pour l'affichage de droite à gauche et l'affichage bidirectionnel des formules mathématiques. Nous utilisons FIGUE comme support d'implémentation de ces propositions de recommandations.

6.2 Affichage bidirectionnel d'objets structurés

Il existe déjà des algorithmes ou des outils permettant un affichage bidirectionnel (voir section 1.4, page 22). L'algorithme bidirectionnel d'Unicode [15] prend uniquement en compte le changement de direction du texte ordinaire sans objets complexes telles les formules mathématiques. Nous nous reposons sur cet algorithme bidirectionnel d'Unicode et sur l'expérience de TEX [46] concernant le mélange de texte écrit de droite à gauche avec du texte écrit de gauche à droite pour manipuler une grande diversité d'objets structurés dans un contexte bidirectionnel (par exemple du texte arabe incluant des formules mathématiques indo-européenes).

La manipulation des objets d'un document purement en Arabe ou purement en Hébreu (où la direction est uniquement de droite à gauche) ne pose pas beaucoup de difficultés. Il s'agit dans ce cas de produire une image miroir de la structure usuelle en plaçant les objets de la droite vers la gauche. L'adaptation des algorithmes de formatage pour un affichage purement de droite à gauche est uniquement une adaptation pour l'inversion de la disposition des objets graphiques.

Par contre, la complexité augmente lorsqu'on veut manipuler à la fois des objets orientés de droite à gauche (par exemple suivant le modèle Arabe) et des objets orientés de gauche à droite (par exemple suivant le modèle Latin) dans le même document. Citons les exemples d'un dictionnaire Arabe/Français, ou d'une revue scientifique en Arabe référant des noms occidentaux et incluant des formules mathématiques de style indo-européen.

Terminologies et conventions

Nous gardons la même notation utilisée dans le chapitre 2 pour décrire le langage de boîtes d'affichage. Nous introduisons de nouvelles notations pour définir la direction d'affichage de ces boîtes.

- Notons respectivement LR et RL la direction de gauche à droite (*left-to-right*) et la direction droite à gauche (*right-to-left*).
- Chaque objet structuré correspond à une boîte graphique. Par exemple les chaînes de caractères sont représentées par la boîte *Atom*.
- Notons *LR-boîte* la boîte orientée de gauche à droite et *RL-boîte* la boîte orientée de droite à gauche.
- Notons *d-boîte* la boîte de direction inconnue qui peut être déduite implicitement.
- L'ordre logique des objets est l'ordre de sauvegarde dans la structure d'arbre sous-jacente et l'ordre d'affichage est l'ordre de la représentation des objets selon une direction donnée. Dans le cas des documents bidirectionnels, l'ordre logique des objets peut être différent de leur ordre d'affichage.

Par la suite, nos exemples seront un mélange de français et d'arabe. Le texte arabe sera écrit de droite à gauche inversement au texte français. Nous ajoutons des flèches dans nos exemples indiquant le sens de lecture pour faciliter la compréhension de nos algorithmes.

6.2.1 Formatage des objets structurés dans un contexte bidirectionnel

Chaque boîte graphique a son propre algorithme de formatage adapté au contexte bidirectionnel. Les formules mathématiques ont des règles d'affichage bidirectionnel différentes de celles appliquées pour du texte simple. Dans la suite, nous étudions séparément les deux cas : textes linéaires et formules mathématiques.

6.2.1.1 Texte linéaire

Entité atomique ou chaîne de caractères

Les caractères Unicode sont divisés en plusieurs familles (Grec, Hébreu, Arabe, symboles mathématiques alphanumériques, opérateurs mathématiques ...). Chaque caractère appartient à une unique famille Unicode et chaque famille possède des propriétés de direction.

Certains symboles mathématiques tels que les signes, les opérateurs mathématiques, les identificateurs ou les nombres, et certains signes de ponctuation tels que la virgule ou le signe d'interrogation peuvent être représentés en miroir (remplacé par leur symbole symétrique) dans un contexte où la direction est de droite à gauche (dans un paragraphe arabe par exemple).

Le sens d'écriture d'une entité atomique de type chaîne de caractères peut être donné explicitement ou déduit de la nature de ses caractères. Le premier caractère rencontré et appartenant à une famille d'Unicode qui n'est pas neutre par rapport à la direction (comme la famille Arabe ou Grecque) détermine la direction d'écriture de cette entité atomique. Par exemple la chaîne de caractère جملة où la première lettre ج est une lettre arabe a une direction d'écriture RL (droite-à-gauche).

Formatage horizontal

L'algorithme de formatage horizontal des boîtes dépend fortement de la direction de ses éléments. La figure 6.1 montre un exemple de formatage horizontal du texte bidirectionnel mélangeant du français et de l'arabe. La table de la figure 6.1.a montre l'ordre logique des éléments de la boîte *Horizontale*. En appliquant l'algorithme bidirectionnel dans le cas où la direction principale est de droite à gauche (le cas des documents arabes incluant du texte français), nous obtenons le formatage montré dans la figure 6.1.b et dans le cas où la direction principale est de gauche à droite (le cas du texte français incluant de l'arabe), nous obtenons le formatage correspondant à la figure 6.1.c.

(a)

Ordre logique	Contenu
0	مثال
1	توضيب
2	النص
3	Exemple
4	de
5	formatage
6	باستعمال
7	فيڭ
8	utilisant
9	FIGUE

(b) direction du texte (de droite à gauche)

utilisant FIGUE باستعمال فيڭ Exemple de formatage مثال توضيب النص

(c) direction du texte (de gauche à droite)

مثال توضيب النص Exemple de formatage باستعمال فيڭ utilisant FIGUE

FIG. 6.1 – *(a) L'ordre logique des éléments de Horizontal (b) le formatage dans le cas où la direction principale est de droite à gauche (c) de gauche à droite*

Nous décrivons deux approches d'implémentation de l'algorithme de formatage horizontal des boîtes qui prennent en compte la direction de ces boîtes :

– La première approche se base sur le formatage horizontal du texte bidirectionnel dans TEX [46]. Cet algorithme réunit des blocs de boîtes partageant la même direction. Lorsqu'il s'agit d'un bloc de boîtes ayant la direction principale (la direction de la boite englobante), l'algorithme procède de la même façon que dans le cas normal. Par contre dans le cas d'un bloc de boîtes avec une direction opposée à la direction principale, seul l'ordre pour disposer les boîtes est inversé. La taille de ce bloc horizontal reste la même que dans le cas du mode normal et elle est indépendante de la direction des boîtes filles. Cet algorithme de formatage est décrit comme suit :

Soit d la direction de la boîte *Horizontale*.
Soit d_i la direction de la boîte fille d'ordre logique i.
Soit n le nombre de boîtes filles de *Horizontale*.
Traiter les boîtes filles par leur ordre logique.
Tant que $i < n$ **faire**
Début
Si $d_i = d$ (la boîte i a la même direction que la boîte *Horizontale*) **faire**
    Début
    Calculer la position (h_i, v_i) de la boîte i (le même calcul qu'en mode normal).
    $i := i + 1$
    Fin
Sinon $(d_i \neq d)$
    Début
    Prenons l'indice k t.q $k = i$.
    Tant que $(d_k \neq d)$ **faire**
        Début
        Calculer et mémoriser la position de la boîte courante $p_k(h_k, v_k)$ comme dans le cas normal sans affecter l'attribut de la position de cette boîte.
        $k := k + 1$
        Fin
    La boîte d'indice k a la même direction que la boîte *Horizontale*.
    Les boîtes précédentes (de l'indice i à l'indice $k-1$) seront disposées en mode *miroir-réfléchi*.
    miroir-réfléchi(b_i ... b_{k-1}, $p_i(h_i, v_i)$, $p_k(h_{k-1}, v_{k-1})$)
    Fin
Fin

Le mode *miroir-réfléchi* consiste à positionner les boîtes entre la position $p_i(h_i, v_i)$ et la position $p_{k-1}(h_{k-1}, v_{k-1})$ en miroir par rapport à la position en mode normal (voir figure 6.2) : la boîte d'indice j à mettre en position (h_j, v_j) en mode normal sera placée à la position (h'_j, v'_j) où h'_j vérifie l'équation suivante $h_k - h'_j = h_j - h_i$. La figure 6.2 montre le passage du mode normal au mode miroir en appliquant une symétrie verticale. La boîte d'indice j placée entre (h, v) et $(h + w_j, v)$ en mode normal sera placée entre $(h' - w_j, v)$ et (h', v) en mode miroir ou h' vérifie l'équation suivante $h_k - h' = h - h_i$

– La deuxième approche consiste à déterminer d'abord l'ordre d'affichage des éléments

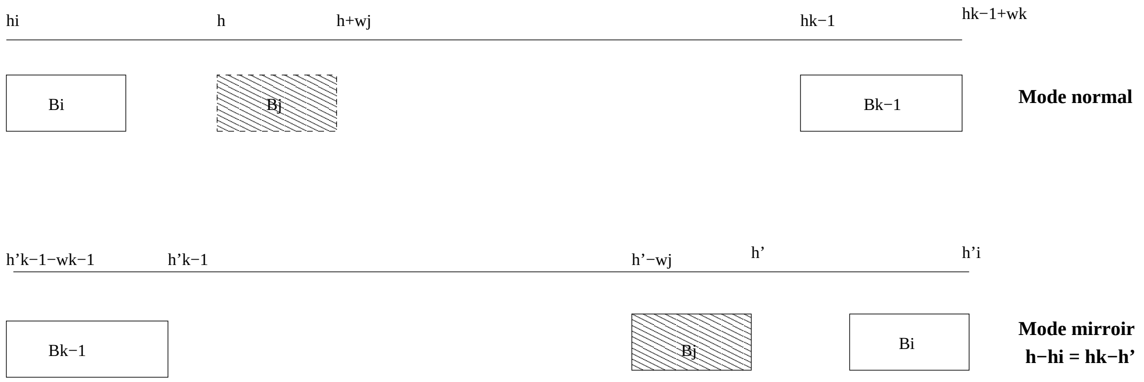


FIG. 6.2 – *Application d'une symétrie verticale pour déduire la position de chaque boîte en mode miroir à partir de sa position en mode normal. Le calcul de la dimension d'un bloc horizontal est la même pour les deux modes.*

en fonction de leur ordre logique et de leur direction et ensuite à disposer ces éléments suivant cet ordre d'affichage. Le formatage horizontal se fait en deux étapes :

1. déterminer l'ordre d'affichage des éléments en fonction de leur direction;
2. disposer les éléments suivant l'ordre d'affichage déjà établi.

L'algorithme suivant détermine l'ordre d'affichage des éléments de la boîte *Horizontale* en fonction de leur direction :

Soit d la direction de la boîte *Horizontale*.
Soit d_i la direction de sa boîte filles i.
Soit n le nombre de boîtes filles de *Horizontale*.
Tant que $i < n$ **faire**
Début
Si $d_i = d$ (la direction de la boîte est la même que la direction principale) **alors**

> **Début**
> ordreAffichage[i]= i
> $i := i + 1$
> **Fin**

Sinon $(d_i \neq d)$

> **Début**
> Prenons l'indice k t.q $k = i$.
> **Tant que** $(d_k \neq d)$ **faire** $k := k + 1$
> **Si** $k = i + 1$ (la direction de la boîte suivante $i + 1$ est différente de celle de la boîte i) **alors** ordreAffichage[i]= i.
> **Sinon** $(k > i + 1$ et $\forall j \in [i, k] d_j \neq d)$
> k est l'indice de la dernière boîte qui a une direction différente de la direction principale. $\forall j \in [i, k]$ ordreAffichage[j] = k-j+i-1
> Pour toutes les boîtes à partir de l'indice i jusqu'à l'indice k, l'ordre d'affichage est inversé.

$i := k$
Fin
Fin

Lors de l'extension de FIGUE, nous avons choisi plutôt de suivre l'algorithme de la deuxième approche pour passer d'un formatage horizontal de gauche à droite vers un formatage bidirectionnel. Une fois l'ordre d'affichage des éléments est calculé, l'algorithme de formatage reste le même tout en traitant les éléments par leur ordre d'affichage au lieu de leur ordre logique. Cette solution ne demandait pas beaucoup de changements.

Paragraphe et coupures

L'approche utilisée pour le formatage horizontal dans un contexte bidirectionnel ne peut pas être appliquée dans le cas d'un paragraphe en plusieurs lignes. Un paragraphe est composé de plusieurs lignes après l'application des règles de coupure. Chaque ligne sera ensuite formatée suivant l'algorithme de formatage horizontal.

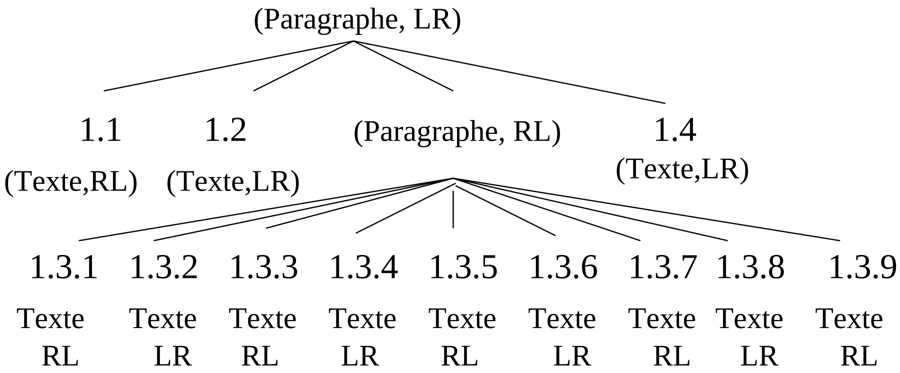
Citons les plus célèbres oeuvres de Averroes: « تهافت التهافت و فصل المقال، الكشف عن مناهج الأدلة، القسم الرابع من وراء الطبيعة » Averroes (nom arabe: Ibn Roshd) est un médecin et philosophe arabe. Ses commentaires des oeuvres d'Aristote figurent parmi les plus fidèles; ils eurent une grande influence sur la pensée chrétienne et philosophique dans l'Europe médiévale.

Citons les plus célèbres oeuvres de Averroes: « تهافت التهافت و فصل المقال، الكشف عن مناهج الأدلة، القسم الرابع من وراء الطبيعة » Averroes (nom arabe: Ibn Roshd) est un médecin et philosophe arabe. Ses commentaires des oeuvres d'Aristote figurent parmi les plus fidèles; ils eurent une grande influence sur la pensée chrétienne et philosophique dans l'Europe médiévale.

Les deux exemples ci-dessus représentent deux paragraphes dont la direction est de gauche à droite (LR-paragraphe). Ces deux paragraphes contiennent du texte orienté de droite à gauche (RL-texte). Les règles de coupures appliquées dans les deux exemples prennent en compte la largeur du paragraphe. Un bloc de texte orienté de droite à gauche (RL-texte) est placé le plus à gauche possible de la nouvelle ligne après le retour à ligne.

La profondeur de l'imbrication des objets dans un paragraphe augmente la complexité du mélange des deux directions LR et RL dans le même document. L'héritage des directions dans un paragraphe à plusieurs niveaux d'imbrication est ambiguë. La figure 6.3 montre la structure d'un paragraphe orienté de gauche à droite qui contient un autre paragraphe orienté de droite à gauche. L'affichage de ce paragraphe est le suivant (c'est le cas d'un paragraphe français contenant un autre paragraphe arabe qui a son tour inclut du texte français) :

«تعتقدون أن الانجليزية هي said حنان
لكن بأعتقادي: “English written backwards”
English الانجليزية تكتب بالوراء.
أني متأكدة أن Knuth و Mackay
And she was right. معا متفقان معي»



1.1: حنان
1.2: said
1.4: And she was right
1.3.1: «تعتقدون أن الانجليزية هي
1.3.2: “English written backwards”
1.3.3: لكن بأعتقادي
1.3.4: English
1.3.5: الانجليزية تكتب بالوراء. أني متأكدة أن
1.3.6: Knuth
1.3.7: و
1.3.8: Mackay
1.3.9: معا متفقان معي »

FIG. 6.3 – *Arbre représentant la structure d'un paragraphe orienté de gauche à droite et contenant un autre paragraphe orienté de droite à gauche.*

La figure 6.4 représente la squelette de ce paragraphe et montre la direction des différents blocs de ce paragraphe.

L'imbrication des objets augmente si les formules mathématiques sont incluses dans le

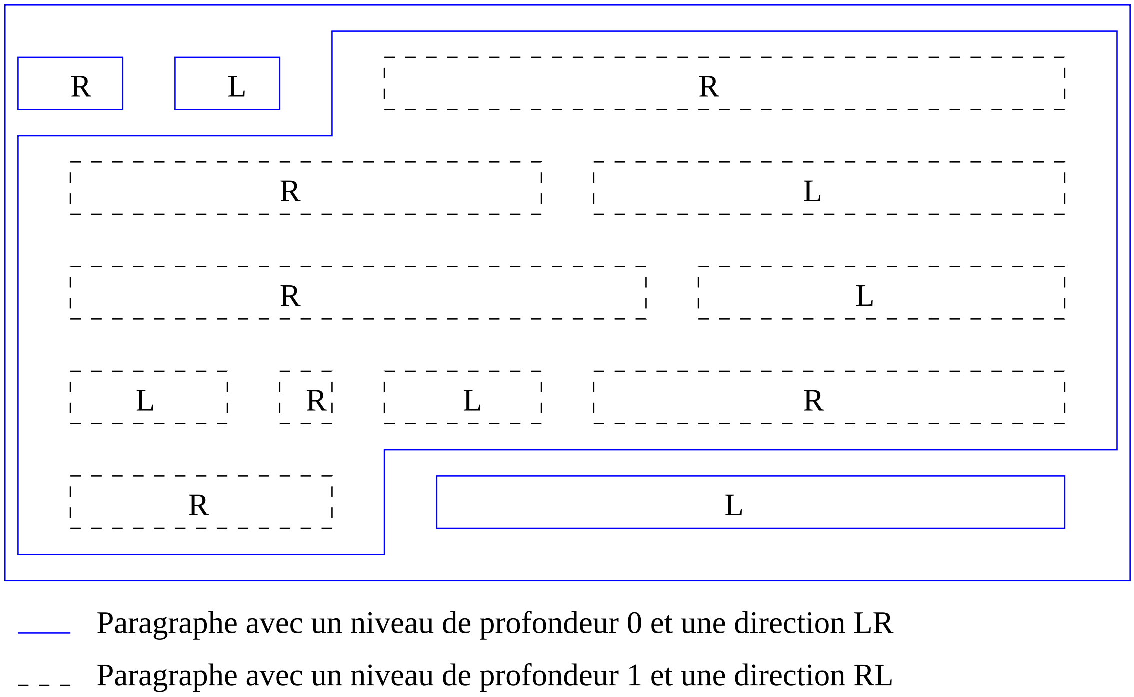


FIG. 6.4 – *La squelette d'un paragraphe orienté de gauche à droite mais qui contient un autre paragraphe orienté de droite à gauche.*

document (par exemple un RL-document incluant des formules mathématiques en mode LR-langage). Dans [46], seules les formules formatées de gauche à droite incluses dans du RL-texte sont traitées; c'est un bon exemple montrant comment mélanger les deux directions avec de grandes profondeurs d'imbrication.

6.2.1.2 Formules mathématiques

La manipulation des formules mathématiques dans un contexte bidirectionnel a un niveau de complexité plus élevé que la manipulation du texte simple (linéaire). Cela est dû à la nature bidimensionnelle de formules mathématiques et à l'importance des relations spatiales dans les notations mathématiques. La profondeur d'imbrication des objets dans une formule augmente la difficulté de mélanger les deux directions LR et RL au sein de la même formule. Dans notre étude, nous distinguons deux cas de formules mathématiques :

- les formules écrites suivant le sens de déroulement de l'écriture du texte qui les contient (par exemple, les formules à l'égyptienne écrites de droite à gauche incluses dans le texte arabe)
- les formules écrites dans le sens inverse du texte qui les contient (par exemple, les formules à la marocaine écrites de gauche à droite à l'inverse du texte arabe).

La figure 6.5 montre un exemple de texte mathématique définissant la valeur absolue écrit en français (gauche à droite figure 6.5.a); puis en arabe suivant deux modèles : le modèle marocain(figure 6.5.b) et le modèle égyptien (figure 6.5.c).

(a) Définition de la valeur absolue: $|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$

(b) تعريف القيمة المطلقة : $|x| = \begin{cases} x & x \geq 0 \text{ إذا كان} \\ -x & x < 0 \text{ إذا كان} \end{cases}$

(c) تعريف القيمة المطلقة : د(س) = $\left.\begin{matrix} \text{س} & \text{إذا كان س} \geq 0 \\ \text{س}- & \text{إذا كان س} < 0 \end{matrix}\right\}$

FIG. 6.5 – *Une formule mathématique écrite suivant : (a) le modèle français où le texte et les formules sont orientés de gauche à droite (b) le modèle marocain où les formules mathématiques sont orientées de gauche à droite contrairement au texte (mélange des deux directions) (c) le modèle égyptien où le texte et les formules sont orientés de droite à gauche.*

Notre but est de proposer une approche générale pour manipuler les formules mathématiques dans un contexte bidirectionnel couvrant toutes les familles de formules mathématiques (mode normal, mode bidirectionnel et mode miroir). Dans la suite de cette section, nous étudions séparément deux cas de formules mathématiques : formules égyptiennes (écrites de droite à gauche) et formules marocaines (écrites de gauche à droite et qui peuvent contenir du texte ou des symboles arabes).

Les formules mathématiques égyptiennes (mode miroir)

La manipulation des formules mathématiques écrites de droite à gauche dans un document purement Arabe ou Hébreu (où la direction est uniquement de droite à gauche) ne pose pas beaucoup de difficultés. Il s'agit d'inverser la disposition des termes de chaque formule mathématique et produire une image miroir de la formule désirée (par exemple, les termes d'une matrice sont orientés de droite à gauche). Il faut aussi réfléchir certains symboles mathématiques (par exemple les symboles $<$, $\in$ deviennent respectivement les symboles $>$,$\ni$ et vice-versa).

Nous décrivons comment à partir d'un affichage de formules mathématiques en mode normal (formules et texte sont écrits de gauche à droite), nous passons sans beaucoup de difficultés en mode miroir (formules et texte sont écrits de droite à gauche). Nous classifions les boîtes mathématiques en quatre catégories pour leur comportement en mode miroir :

- **Les symboles** sont les entités de base comme les signes, les opérateurs mathématiques, les identificateurs ou les nombres. Certains symboles ne changent pas (=، ∗ ، − ، + ، / ، % ، ـ) tandis que d'autres possèdent, au sein de la même fonte, leur symbole symétrique. Pour ces derniers symboles, nous pouvons intervertir l'Unicode des deux symboles (par exemple, le symbole $<$ devient le symbole $>$ et vice-versa). Dans [48] l'auteur classifie les familles de symboles pour lesquels une transformation en mode miroir est possible.

- **Les boîtes horizontales** disposent horizontalement les termes de l'expression comme par exemple un vecteur regroupant horizontalement ses sous-expressions, les lignes d'une table. Ces boîtes inversent la disposition de leurs termes suivant les mêmes principes d'inversion vus dans le cas du texte écrit de droite à gauche.
- **Les boîtes verticales** disposent verticalement leurs éléments comme par exemple une fraction ou une matrice. Les boîtes verticales ne changent pas leurs algorithmes de formatage par rapport à la direction horizontale (gauche à droite et droite à gauche) et ils héritent de la direction de la boîte qui les englobe. Par exemple la fraction dispose le numérateur ci-dessus du dénominateur séparés par une barre horizontale indépendamment de la direction (voir figure 6.6).
- **Les boîtes exposant, indice, racine** ont une disposition spatiale (bidimensionnelle) importante et elle dépend fortement de la direction principale, par exemple l'indice ou l'exposant sont disposés à gauche de la base dans le cas de la direction droite à gauche (voir figure 6.6). La racine nième inverse la disposition de ses termes et le dessin du symbole racine est réfléchi.

Formules en mode latin	Formules en mode arabe (égyptien)
$\in, <$	$> \text{،} \ni$
$1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$	$\frac{(1+\text{ن}2)(1+\text{ن})\text{ن}}{6} =^2 \text{ن} + \cdots +^2 2 +^2 1$
$M = \begin{pmatrix} 4b & 0 \\ 5 & 4 \end{pmatrix}$	$\begin{pmatrix} 0 & \text{ب}4 \\ 4 & 5 \end{pmatrix} = \text{م}$

FIG. 6.6 – *Quelques exemples de formules mathématiques écrites en mode latin et leurs correspondances en mode arabe-égyptien.*

Les formules mathématiques marocaines (mode bidirectionnel)

La tendance actuelle dans le monde arabe est d'écrire les formules mathématiques de gauche à droite contrairement à la direction du texte arabe qui les contient. Cela facilite la traduction des documents mathématiques en arabe depuis les langues latines et permet d'adopter des notations standard pour les mathématiques. Ces formules peuvent à leur tour contenir du texte arabe, ce qui implique plus de difficultés pour gérer le mélange de directions dans les documents mathématiques arabes.

Les règles d'affichage sont alors différentes de celles utilisées pour les formules mathématiques égyptiennes. Les symboles mathématiques ne sont pas réfléchis et la disposition des termes des expressions n'est pas inversée. Le problème se pose uniquement dans le cas des boîtes mathématiques horizontales où il faut gérer le mélange de directions (si certains termes sont en arabe). Les autres types de boîtes ne changent pas (fraction, exposant, matrice, ...). Nous distinguons deux types de règles pour déterminer la direction de tous les objets du document dans le cas du mode bidirectionnel : les règles générales d'héritage et les règles spécifiques aux objets mathématiques.

Les règles générales d'héritage s'appliquent sur l'ensemble des objets du document; elles sont décrites ci-dessous :

- La direction de l'objet racine du document est donnée.
- Tous les noeuds non terminaux qui ne sont pas des objets mathématiques (par exemple les objets paragraphes ou les boîtes horizontales) prennent la direction de l'objet racine (ils héritent de la direction de leur noeud père).
- Les noeuds terminaux (les atomes) qui sont des éléments textuels prennent la direction Unicode déduite de la nature des caractères qui les composent (par exemple un élément textuel arabe prend la direction droite à gauche).
- Les noeuds terminaux comme les opérateurs, les identificateurs ou les chiffres gardent une direction neutre.

Les règles spécifiques pour déterminer la direction des objets mathématiques sont :

- Par défaut, une formule mathématique est orientée de gauche à droite
- Seul les éléments textuels peuvent changer la direction d'un vecteur horizontal (Row) dans une formule. Si le vecteur horizontal contient un élément textuel avec une direction opposée, le vecteur changera sa direction. La figure 6.7 montre un exemple de formule arabe écrite suivant le modèle marocain. Le vecteur horizontal prend la direction de son premier élément textuel qui est la direction de droite à gauche (application de cette règle courante).
- Si un des éléments d'un objet vertical a une direction opposée, cet objet vertical change sa direction. L'exemple suivant montre un formule contenant un vecteur vertical dont le premier fils est un vecteur horizontal ayant une direction opposée (une direction de droite à gauche).

لتكن g الدالة العددية المعرفة ب : $\left.\begin{array}{r} \overleftarrow{x \neq -1 \quad \text{إذا كان} \quad g(x) = ln(x)} \\ g(-1) = -2 \end{array}\right\}$

Une conséquence directe de l'application de ces règles est l'importance de l'imbrication des éléments horizontaux en présence des éléments textuels pour déterminer l'affichage des formules. La figure 6.8 illustre comment l'affichage d'une formule dans le mode bidirectionnel peut être influencé par les imbrications des éléments horizontaux. Cet exemple montre que si les objets ne sont pas bien regroupés dans une formule (au niveau de la structure), le résultat obtenu de l'affichage peut être incorrect.

Dans la structure d'une formule, il est important de faire la distinction entre les identificateurs, les opérateurs, les nombres et les éléments textuels. Seul les éléments textuels peuvent influencer la direction de la formule. Les autres éléments terminaux sont de nature neutre et ils suivent plutôt la direction de la formule qui les englobe. Dans la formule $P = 2 * \pi *$ شعاع la dernière variable شعاع est un identificateur et non pas un élément textuel. Cela a pour conséquence que cette formule reste orientée de gauche à droite.

En appliquant ces règles sur l'arbre représentant la formule, nous déterminons la direction d'affichage dans chaque noeud de l'arbre. Ensuite, chaque objet applique son propre

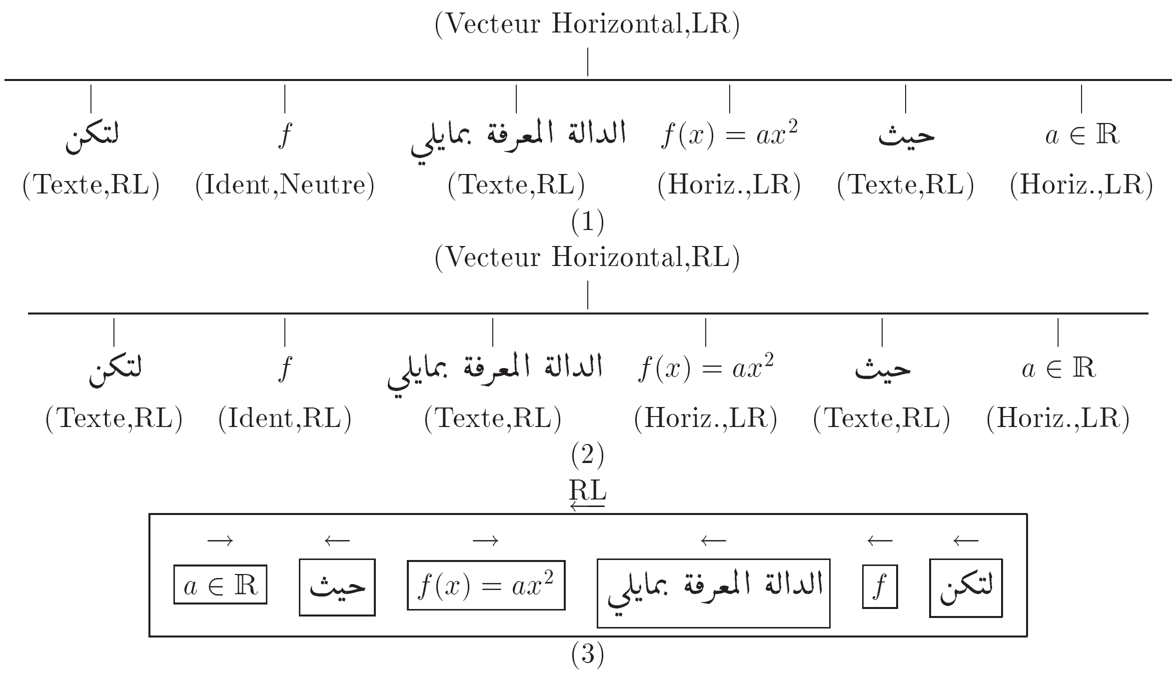
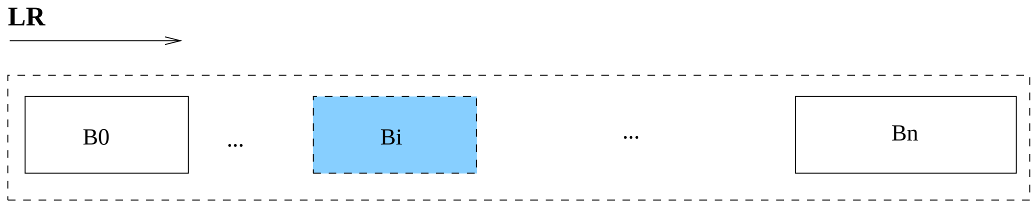


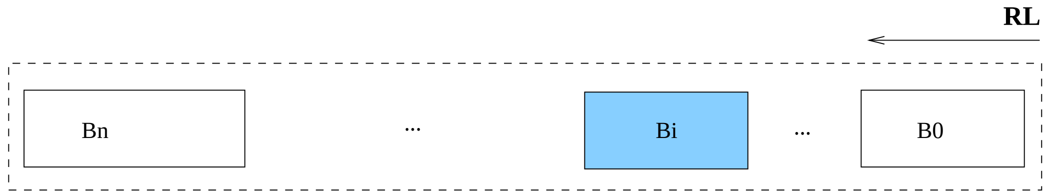
FIG. 6.7 – *(1) Règles d'héritages : seul les éléments non terminaux héritent de la direction et du mode d'affichage du père, les éléments textuels gardent leur direction et l'identificateur reste avec une direction neutre. (2) Règles de l'affichage bidirectionnel de formules mathématiques : le vecteur horizontal prend la direction de son premier fils textuel et ensuite l'identificateur hérite cette direction du père (RL). Ensuite, les éléments neutres prennent cette nouvelle direction. (3) Application de l'algorithme bidirectionnel sur l'arbre final pour disposer correctement les éléments graphiques*

algorithme de formatage. Le formatage des éléments textuels et les éléments terminaux utilise l'algorithme bidirectionnel d'Unicode. L'algorithme de formatage des éléments horizontaux prend en compte la direction de ses éléments et applique l'algorithme suivant :

– Si la direction de l'objet est de gauche à droite, l'ordre d'affichage de ses éléments est normal (c.-à-d l'ordre d'affichage des éléments est égal à leur ordre logique).



– Si la direction de l'objet est de droite à gauche, l'ordre d'affichage de ses éléments est inversé.



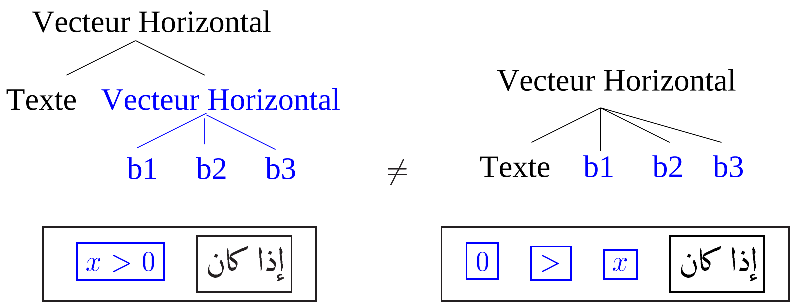


FIG. 6.8 – *L'affichage de la formule à droite est différent de celui de la formule à gauche à cause de la suppression d'un bloc de vecteur horizontal en présence d'un élément textuel.*

L'algorithme de formatage des autres objets comme *Table, Fraction, Over* ne changent pas par rapport à la direction. La figure 6.9 montre un autre exemple de formule arabe écrite suivant le modèle marocain. L'ordre d'affichage des éléments de ce vecteur horizontal est inversé car sa direction est de droite à gauche.

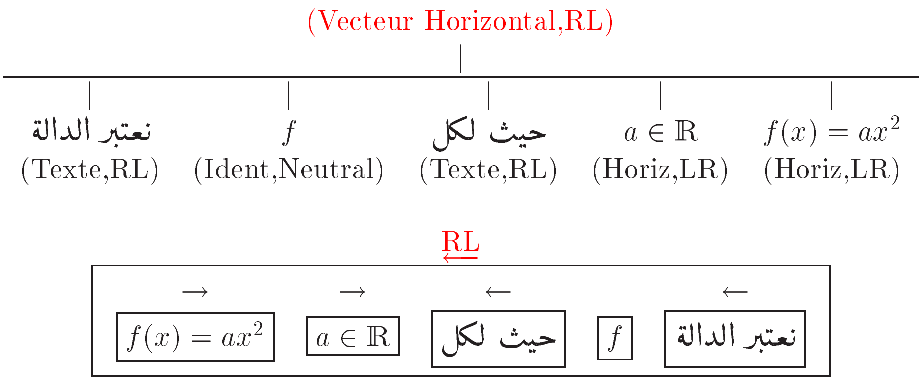


FIG. 6.9 – *Un exemple d'un vecteur horizontal suivant le modèle marocain : le vecteur est orienté de droite à gauche et il inverse l'affichage de ses éléments.*

Traitement des objets structurés du même document

Dans un document, chaque objet structuré possède un attribut indiquant sa direction d'affichage et un autre indiquant son mode d'affichage: le mode normal de gauche à droite , le mode miroir de droite à gauche ou le mode bidirectionnel mélangeant les deux directions. La direction seule sans le mode ne suffit pas pour assurer un affichage correct dans un contexte bidirectionnel. Des règles d'héritage sont appliquées sur l'arbre du document pour déterminer la direction de tous les objets du document. Ces règles d'héritage sont des règles générales applicables sur tous les objets. Pour le mode bidirectionnel, d'autres règles spécifiques aux formules mathématiques s'appliquent. Ensuite, chaque objet applique son propre algorithme de formatage en fonction de son mode d'affichage et de sa direction et de celle de ses boîtes filles. La direction principale et le mode d'affichage du document sont donnés explicitement. Un seul mode d'affichage est utilisé dans un document. Mélanger deux modes d'affichage dans un même document augmenterait rapidement l'ambiguïté de lecture de ce document.

6.2.2 Extension de Figue pour l'affichage bidirectionnel

FIGUE manipule une grande diversité d'objets structurés comme du texte simple, des formules mathématiques, des schémas, etc. Notre extension de FIGUE supporte de plus un affichage bidirectionnel du document : d'abord un affichage de droite à gauche (par exemple l'affichage du texte arabe) et ensuite un affichage bidirectionnel mélangeant à la fois la direction de gauche à droite et la direction de droite à gauche (par exemple du texte arabe incluant du texte indo-européen). Nous nous sommes basés sur les règles d'affichage d'objets structurés vu en section 6.2.1.2 pour étendre les algorithmes d'affichage dans FIGUE.

FIGUE se base sur la notion de boîtes [45] pour représenter tous les objets structurés du document (voir le chapitre 2). Dans un contexte bidirectionnel, chaque boîte a un indicateur de direction. La direction peut prendre comme valeur *LR* (*left-to-right* de gauche à droite) ou *RL* (*right-to-left* de droite à gauche) ou *Neutral* (neutre). Pour distinguer les différents mode d'affichage, chaque objet dans FIGUE a aussi un indicateur *mode* identifiant son mode d'affichage. Nous distinguons trois modes d'affichage dans FIGUE :

– le mode normal où la direction est purement de gauche à droite
– le mode miroir où la direction est purement de droite à gauche
– le mode bidirectionnel où les deux directions LR et RL sont mélangées

FIGUE applique des règles d'héritage pour propager la direction suivant le mode d'affichage choisi sur l'ensemble de l'arbre du document. Dans le cas du mode bidirectionnel, des règles d'affichage spécifiques aux objets mathématiques sont appliquées pour déterminer leur direction : par exemple, les objets horizontaux dans une formule mathématique appliquent des règles vues dans la section 6.2.1.2. Il est nécessaire de spécifier dans FIGUE la direction principale et le mode d'affichage du document.

Chaque boîte dans FIGUE applique son propre algorithme de formatage. Cet algorithme dépend du mode d'affichage choisi et de l'indicateur de direction d'affichage. Dans la figure 6.10, nous classifions les boîtes FIGUE suivant le comportement de leur algorithme de formatage vis à vis du mode d'affichage.

Nous distinguons deux choix d'implémentation pour appliquer les règles déterminant la direction des objets pour chaque mode d'affichage : la première implémentation consiste à appliquer des règles de transformation sur les objets structurés pour déterminer leur direction et leur mode d'affichage et une deuxième implémentation consiste à appliquer directement ces règles lors du formatage des objets. Dans la première implémentation, le moteur d'affichage se contente d'appliquer son algorithme d'affichage selon le mode d'affichage choisi. Dans la deuxième implémentation le moteur d'affichage applique à la fois ces règles et aussi implémente les algorithmes de formatage. Dans FIGUE nous avons choisi la deuxième implémentation pour pouvoir afficher directement les objets dans un contexte bidirectionnel sans que l'utilisateur se soucie des règles d'affichage bidirectionnel.

Catégories de boîtes	Mode miroir	Mode bidirectionnel
Verticaux (Vertical, Fraction, Over, Under, Table ...)	neutre	neutre
Horizontaux (Horizontal, TableRow, Row ...	inverser la disposition des éléments par rapport au mode normal	appliquer l'algorithme bidirectionnel d'Unicode pour les objets non mathématiques. Les objets horizontaux de type mathématiques suivent l'algorithme décrit dans la section 6.2.1.2, page 107
Paragraphe	inverser la disposition des éléments par rapport au mode normal et appliquer des règles de coupures différentes	appliquer l'algorithme bidirectionnel d'Unicode
Exposant, indice, racine Sup, Sub, NRoot	changer la disposition spatiale et le dessin du symbole racine	neutre
Atome	quelques symboles sont réfléchis	neutre

FIG. 6.10 – *Classification des boîtes dans* FIGUE *suivant le comportement de leur algorithme de formatage par rapport au mode d'affichage*

Prenons un exemple d'une boîte horizontale dans FIGUE pour décrire le comportement de son algorithme de formatage en fonction de son mode d'affichage. Il s'agit d'un vecteur Horizontal (*Row*) appartenant à la catégorie des boites horizontales (la deuxième implémentation est appliquée dans ce cas). Dans la suite, nous montrons les principales étapes de l'algorithme de formatage de cette boîte pour déterminer la direction et l'ordre d'affichage de ses boîtes filles.

Étape 1
Cette étape est appliquée uniquement dans le cas d'un affichage bidirectionnel (mode = **bidi**).

La direction du vecteur horizontal est déterminée en fonction du type et de la direction de ses boîtes filles.

Début
Si une boîte fille est de type texte avec une direction opposée à la direction courante **alors**
La direction du vecteur horizontal sera modifiée

direction= directionOpposee
Ensuite, les éléments neutres (s'ils existent comme par exemple les identificateurs, les opérateurs) héritent de la direction courante.
Fin

Étape 2

Déterminer l'ordre d'affichage des éléments en fonction de la direction courante et du mode d'affichage.

Si (mode = **normal**) (où la direction est uniquement de gauche à droite) **faire**
l'ordre des traitements des éléments est le même que leur ordre logique
(ordre d'affichage est égale à l'ordre logique)

Si (mode = **miroir**) (la direction est uniquement de droite à gauche) **faire**
L'ordre des traitements des éléments est l'inverse de leur ordre logique

Si (mode = **bidi**) **faire**
Déterminer l'ordre d'affichage des éléments en fonction de leur direction.

- **Si** (direction=LR) **alors** l'ordre des traitements des éléments est le même que leur ordre logique (comme en mode normal).
- **Si** (direction=RL) **alors** l'ordre des traitements des éléments est l'inverse de leur ordre logique (comme en mode miroir).

6.2.3 MathML et l'affichage bidirectionnel

Les implémentations actuelles de MathML permettent uniquement l'affichage des formules mathématiques de gauche à droite suivant le modèle latin. Elles ne traitent pas le cas des formules écrites de droite à gauche, ou des formules écrites de gauche à droite incluant du texte orienté de droite à gauche. Nous nous basons sur notre étude générale sur le comportement des algorithmes d'affichage des formules mathématiques dans un contexte bidirectionnel pour définir l'usage de la norme MathML dans ce contexte. Le support de cette extension de MathML par des navigateurs Web, permettrait une large diffusion de documents scientifiques sur le Web en différentes langues.

Pour étendre MathML pour un affichage bidirectionnel de formules mathématiques, il est nécessaire de définir un nouvel attribut de direction *dir* pour les éléments MathML. Cet attribut définit la direction d'écriture de chaque terme d'une formule et il peut prendre trois valeurs : *LR* (gauche à droite), *RL* (droite à gauche) ou *neutral* (direction neutre). La direction principale d'une formule peut être définie par l'élément racine *math* ou déduite de la direction d'écriture du document entier.

MathML doit définir aussi un nouvel attribut *mode* spécifiant le mode d'affichage choisi. Cet attribut peut prendre trois valeurs : *normal* (le mode normal), *mirror* (le mode miroir) ou *bidi* (le mode bidirectionnel). Le mode par défaut est le mode normal. Pour chaque formule mathématique, un seul mode est accepté pour éviter les ambiguïtés au niveau de la lecture de ces formules. Par exemple, la formule A-B peut avoir deux significations différentes selon la direction d'affichage (LR et RL). Dans ce cas, la précision du mode de direction d'affichage avant la visualisation de cette formule est nécessaire. L'élément *math* doit avoir un attribut *mode* qui sera propagé (hérité) sur ses sous éléments.

La direction des formules ne peut pas être déduite de la langue du document qui peut être définie par une source externe à MathML (par exemple la langue utilisée dans un document HTML). Dans le cas d'un document en arabe, la direction des formules peut être opposé au texte arabe (le cas des formules suivant le modèle marocain).

Il est recommandé d'appliquer des règles pour déterminer la valeur correcte de l'attribut *dir* pour tous les éléments d'une formule MathML et pour propager le mode d'affichage sur l'ensemble des formules. Les règles à appliquer (basées sur notre étude générale décrite dans la section 6.2.1) sont décrites ci-dessous :

- Pour chaque formule, un seul mode d'affichage est utilisé (pour éviter l'ambiguïté lors de la lecture des formules). Le mode d'affichage est défini par l'élément *math* et il est propagé de haut en bas (chaque élément hérite de l'attribut *mode* de son élément père).
- Les éléments non terminaux prennent la direction de l'élément père. La direction est propagée à partir de l'élément racine *math*.
- Seul les éléments terminaux textuels *mtext* n'héritent pas de l'attribut de direction. La direction est définie selon la nature des caractères composant leur texte.
- Les éléments terminaux *mi, mn, mo* (identificateur, nombre et opérateur) ont une direction neutre dans le cas du mode d'affichage bidirectionnel sinon ils héritent de la direction de l'élément MathML qui les englobe.

Dans le cas d'un mode d'affichage bidirectionnel, des règles spécifiques s'appliquent pour déterminer la direction correcte de chaque élément de la formule :

- Les éléments textuels *mtext* détermine la direction des éléments horizontaux *mrow* et *mtr* qui les englobent. Le premier élément *mtext* par ordre logique ayant une direction opposée détermine la direction de l'élément père horizontal.
- Les éléments neutres prennent la direction de leur noeud père.

Ces attributs de direction et de mode d'affichage seront ensuite interprétés par le moteur d'affichage MathML pour ajuster le formatage des formules. A travers les propriétés des éléments et leurs attributs de direction et de mode d'affichage, MathML pourrait offrir le moyen de contrôler le changement de direction du texte et des formules mathématiques, et pourrait contenir un algorithme bidirectionnel pour les formules mathématiques.

En annexe C, nous proposons une classification des éléments MathML suivant le comportement de leur algorithme de formatage dans un contexte bidirectionnel. Nous décrivons aussi le comportement des algorithmes de formatage pour chaque catégorie d'éléments MathML dans le cas du mode bidirectionnel et aussi dans le cas miroir.

6.3 Application : les explications de preuves en arabe

Nous avons appliqué ce travail (et l'extension FIGUE correspondante) aux explications de preuves en langue arabe. La production d'un arbre d'explications de preuve en arabe peut se faire de deux façons : la première consiste à produire un arbre textuel arabe à partir de l'objet preuve et la deuxième façon consiste à reprendre l'arbre textuel anglais (produit par un autre module) pour le traduire en un arbre textuel arabe. Dans les deux cas, une fois l'arbre textuel arabe produit, nous appliquons des règles de transformations sur cet arbre pour produire l'arbre de boîtes FIGUE (décrivant le formatage correspondant). Le schéma 6.11 décrit les méthodes de production d'explications de preuve en arabe dans PCOQ.

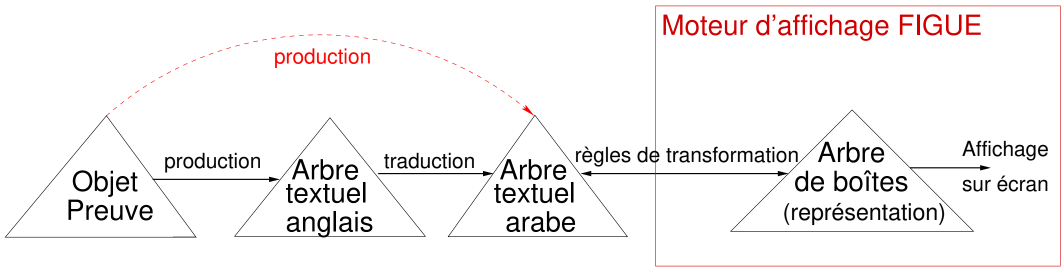


FIG. 6.11 – *Les étapes de production des explications de preuves en arabe dans* PCOQ *: produire un arbre textuelle en anglais à partir de l'objet preuve et le traduire par la suite en arabe. L'arbre textuel arabe produit sera affiché par* FIGUE

Nous nous contentons dans un premier temps de traduire l'arbre textuel anglais en un autre arbre textuel arabe en appliquant des règles simples de traduction (de l'anglais vers l'arabe, voir schéma 6.11). La figure 6.12 montre un exemple de traduction d'un texte anglais décrivant une preuve en un texte arabe. Cette traduction mot à mot n'est pas toujours adaptée pour bien exprimer des preuves en langue arabe mais nous suffit comme plateforme de test de FIGUE bidirectionnel. Nous étudions comment produire un arbre textuel en arabe directement à partir de l'objet représentant la preuve (figure 6.11) mais ceci est indépendant des problèmes d'affichages.

La figure 6.13 montre un exemple de session PCOQ où les explications de preuves sont en arabe. Cet exemple est la preuve en COQ d'une propriété de l'addition dans l'ensemble des entiers de Peano. Cet ensemble est défini par récurrence et il est constitué d'une constante o et de ses successeurs obtenus par la fonction s. Dans cet exemple, nous utilisons la tactique *Induction* qui lance une preuve par récurrence sur son argument, qui doit être quantifié universellement dans le but.

La première fenêtre de cette session PCOQ représente les commandes et le script de la preuve et la deuxième fenêtre affiche l'explication en arabe générée automatiquement à partir de la preuve (voir figure 6.12 pour la traduction du texte de cette preuve en anglais). Dans cet exemple, différentes notations sont manipulées : texte, formules mathématiques. Le texte d'explication de cette preuve en arabe s'écrit de droite à gauche et les formules mathématiques de gauche à droite (modèle marocain). FIGUE permet de manipuler ces

(a)		(b)
Let us prove $\forall n : N0 + n = n$ Let n be an element of N. By induction on n: -Case 0: Imagine a proof of $0 + 0 = 0$ -Case (s): Imagine a proof of $\forall n : N0 + n = n \Rightarrow 0 + s(n) = s(n)$.	$\xrightarrow{Traduction}$	لنبرهن المبرهنة $\forall n : N0 + n = n$ ليكن n عنصرا من N . باستعمال البرهان بالترجع على n : الحالة 0 : لنتخيل المبرهنة التالية $0 + 0 = 0$ الحالة (s) : لنتخيل المبرهنة التالية $\forall n : N0 + n = n \Rightarrow 0 + s(n) = s(n)$.

FIG. 6.12 – *Exemple du texte d'explications de preuve par récurrence (a) en anglais et (b) sa traduction en arabe produite à l'aide de règles simples de traduction.*

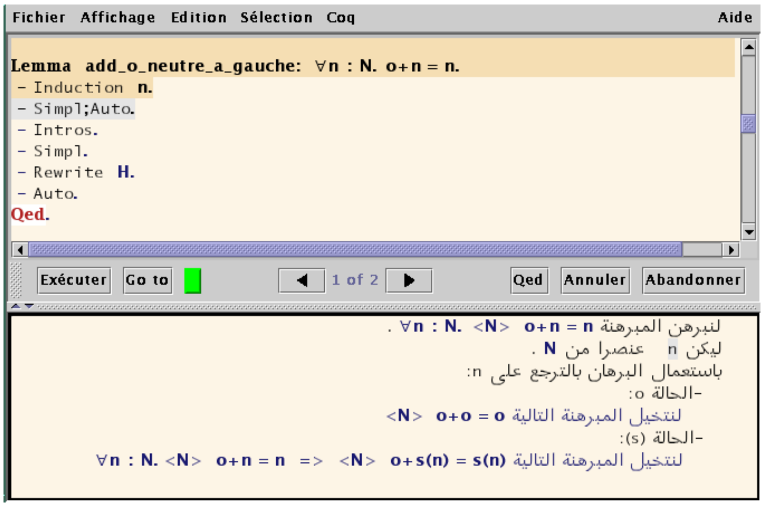


FIG. 6.13 – *Un exemple de session* PCOQ *où les explications de preuves sont en arabe.*

objets structurés et de gérer le changement de direction de leur affichage (affichage bidirectionnel). Il est possible de changer la langue utilisée pour les explications de preuves durant la même session PCOQ (arabe, anglais, français).

Conclusion

Nous avons présenté dans ce mémoire notre travail concernant le développement d'outils pour l'interaction homme machine dans les environnements de démonstrations mathématiques. Comme nous l'avons déjà dit dans l'introduction, afficher du texte mathématique et manipuler les formules mathématiques de manière interactive sont des atouts primordiaux pour les outils dédiés aux mathématiques. Aussi, dans la perspective de produire une bibliothèque d'affichage pour le développement d'outils interactifs (WYSIWYG) comme le développement des interfaces graphiques pour les systèmes de calcul symbolique ou d'éditeur de documents scientifiques, nous nous sommes fixés comme objectifs de respecter les fonctionnalités suivantes :

- un affichage performant de formules mathématiques (bidimensionnel, incrémental, facilement personnalisable);
- une manipulation simple et naturelle des formules (interactivité);
- des moyens de transfert et de communication des données mathématiques vers le Web et vers d'autres applications;
- un affichage de formules mathématiques dans un contexte multilingue, que ce soit sur le Web ou dans une interface graphique.

Pour atteindre ces objectifs, nous avons étendu le système FIGUE, dont le noyau est un module de formatage et d'affichage bidimensionnel interactif. Nous lui avons ajouté un affichage de qualité des formules mathématiques. Nous avons proposé un langage de boîtes permettant le formatage des objets structurés et offrant un moyen puissant et naturel pour interagir et manipuler ces objets du document. Cette version de FIGUE est utilisée dans l'interface graphique PCOQ, ce qui nous a permis d'améliorer les fonctionnalités de FIGUE suivant les besoins des utilisateurs. Dans le but de tester FIGUE sur d'autres applications, nous avons aussi expérimenté son utilisation pour le développement d'une interface graphique commune pour deux systèmes de calcul symbolique BERNINA et SHASTA. Cette expérience nous a permis de tester FIGUE dans un autre contexte et d'améliorer la présentation des notations mathématiques. Cette interface est au stade de prototype et nous envisageons continuer ce travail afin de produire une version plus complète de cette interface.

D'autre part, nous nous sommes intéressés à la collaboration entre nos outils de développement de preuves et le standard des mathématiques sur le Web, MathML, afin de permettre le partage de nos données mathématiques avec d'autres applications et leur diffusion sur le Web. Cette collaboration existe concrètement dans nos outils de développement sous la forme suivante :

- La correspondance naturelle entre les objets MATHML-*présentation* et les boîtes de formatage FIGUE facilite la traduction entre les deux formalismes, ce qui permet la présentation des preuves sur le Web.
- La représentation de nos données sous forme d'arbre de syntaxe abstraite rend la traduction vers MATHML-*contenu* facile, ce qui nous permet de collaborer avec des systèmes de calcul symboliques en utilisant MATHML comme protocole de communication.

FIGUE offre donc un support de MathML: il affiche et manipule des éléments de présentation MathML et génère du MathML à partir des objets mathématiques affichés par FIGUE. L'ajout du support MathML dans FIGUE permet de l'utiliser comme composant 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 le Web.

Cette migration de nos outils vers le Web devrait, à terme, élargir la visibilité de nos travaux, et nous permettre d'atteindre une plus grande communauté d'utilisateurs afin de valider nos outils. Dans le cadre de ce travail, nous participons au projet européen MoWGLI(Mathematics on the Web: Get It by Logic and Interfaces) [6] dont le sujet concerne les mathématiques formelles sur le Web. Le but de ce projet est d'étudier et de développer une infrastructure pour la création et la manipulation des connaissances mathématiques sur le Web en se basant sur une description sémantique de ces informations. Nous envisageons d'intégrer FIGUE, comme moteur d'affichage MathML, dans le prototype de MoWGLI et de le tester dans ce nouveau contexte.

Enfin, dans l'objectif de pouvoir afficher et manipuler des documents mathématiques dans différentes langues et systèmes d'écriture, en particulier l'arabe, nous avons étudié le comportement des formules mathématiques dans un contexte multilingue. Nous avons obtenu les résultats suivants :

- une approche générale pour manipuler les formules mathématiques dans ce contexte bidirectionnel (mélange des directions d'écritures).
- une extension de FIGUE pour qu'il serve comme base d'expérimentation de cette étude. FIGUE est aussi utilisé dans l'interface PCOQ pour présenter des explications en langue arabe de preuves mathématiques.
- utilisation du résultat de ce travail pour aider à définir de nouvelles recommandations de la norme MathML pour l'affichage bidirectionnel de formules mathématiques.

Notons que ce travail a démontré la puissance et la souplesse de l'architecture de FIGUE, qui, conçu pour de l'affichage gauche à droite, a pu être adapté au bidirectionnel. Jusqu'à présent, nous avons étudié uniquement l'affichage bidirectionnel des formules mathématiques dans le cas des langues écrites de droite à gauche comme l'arabe et l'hébreu ou des langues écrites de gauche à droite comme le français et l'anglais. Une perspective intéressante consiste à généraliser cette étude pour le cas des langues écrites de haut en bas comme le japonais ou le coréen, étudier comment mélanger les différentes directions (gauche à droite, droite à gauche, haut en bas) dans le même texte et établir de nouvelles règles d'affichage des formules mathématiques écrites de haut en bas.

Bibliographie

[1] Amaya, http://w3c1.inria.fr/amaya/.

[2] A. Amerkad, Y. Bertot, L. Pottier, and L. Rideau. Mathematics and Proof Presentation in Pcoq. In *Proceedings of Workshop Proof Transformation and Presentation and Proof Complexities in connection with IJCAR 2001*, Siena, Italy, June 2001.

[3] J. André and I. Vatton. Contextual typesetting of mathematical symbols taking care of optical scaling. Technical Report RR-1972, INRIA, 1993.

[4] O. Arsac. *Interfaces homme machine pour le calcul formel.* Thèse en Informatique, Université de Nice Sophia Antipolis, Juillet 1997.

[5] O. Arsac, S. Dalmas, and M. Gaëtano. The Design of a Customizable Component to Display and Edit Formulas. In S. Dooley, editor, *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation (ISSAC-99)*, pages 283–290, New York, 1999. ACM Press.

[6] A. Asperti and M. Kohlhase. MathML in the MOWGLI project. In *MathML International Conference: Hickory Ridge Conference Center, Chicago, IL, USA, June 28–30*, 2002. Extended abstract in http://www.mathmlconference.org/2002/presentations/asperti/.

[7] A. Asperti, L. Padovani, C. Coen, and I. Schena. Formal Mathematics in MathML. In *MathML International Conference 2000: University of Illinois, Urbana-Champaign, USA, October 20-21*, 2000. Extended abstract in http://www.mathmlconference.org/2000/Talks/asperti/.

[8] D. Aspinall. Proof General: A Generic Tool for Proof Development. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *LNCS*. Springer, 2000.

[9] P. Attar. Prendre en compte les notations mathématiques dans les documents électroniques. *Document numérique*, 1(2):147–159, 1997.

[10] J. D. Becker. Multilingual Word Processing. *Scientific American*, 251(1):96–107, July 1984.

[11] J. D. Becker. Arabic word processing. *Communications of the ACM*, 30(7):600–610, July 1987.

[12] Y. Bertot. Direct manipulation of Algebraic Formulae in Interactive Proof Systems. In *Workshop on User-Interfaces for Theorem Provers*, Sophia Antipolis, sep 1997.

[13] Y. Bertot. The CtCoq System: Design and Architecture. *Formal aspects of Computing*, 11:225–243, 1999.

[14] Y. Bertot, G. Kahn, and L. Théry. Proof by Pointing. volume 789 of *Lecture Notes in Computer Science*, pages 141–160. Springer, 1994.

[15] The Bidirectional Algorithm, http://www.unicode.org/unicode/reports/tr9/bidi.

[16] R. Bornat and B. Sufrin. Animating Formal Proof at the Surface: The Jape proof calculator. *The Computer Journal*, 42(3):177–192, 1999.

[17] P. Borras, D. Clement, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: The system. Research Report RR-0777, INRIA, 1987.

[18] D. Broeglin, S. Lavirotte, and P. Sander. Displaying Mathematics on the Semantic Web: MathML Content to SVG. In *MathML International Conference: Hickory Ridge Conference Center, Chicago, IL, USA, June 28–30*, 2002. Extended abstract in http://www.mathmlconference.org/2002/presentations/broeglin/.

[19] B. Buchberger. *Mathematica:* A System for Doing Mathematics by Computer. Technical Report 93-50, RISC-Linz, Johannes Kepler University, Linz, Austria, 1993.

[20] D. Carlisle. xmltex: A non validating and not 100 in TEX. *TUGboat*, 21(3):193–199, 2000.

[21] D. Carlisle. MathML on the Web, Using XSLT to enable cross platform support for XHTML and MathML in current Browsers. In *MathML International Conference: Hickory Ridge Conference Center, Chicago, IL, USA, June 28–30*, 2002. Extended abstract in http://www.mathmlconference.org/2002/presentations/carlisle/.

[22] J. Chlebíková. The Euromath System - The Structured Editor for Mathematicians. *Proceedings of the Tenth European TeX Conference*, pages 82–93, 1998.

[23] L. Constable, S. Allen, H. Bromely, W. Cleveland, et al. *Implementing Mathematics with the Nuprl Development System.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

[24] J. Cooke and E. Sobel. MathWriter: mathematical typesetting with the Macintosh. In *Ithaca*, New York, 1986. Cooke Publications.

[25] Y. Coscoy. A Natural Language Explanation for Formal Proofs. In *Proceedings of the International Conference on logical Aspects of Computational Linguistics (LACL)*, volume 1328 of *LNCS/LNAI*, Nancy, September 1996. Springer.

[26] S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 implementation. In Wolfgang, editor, *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, pages 241–248, New York, NY 10036, USA, July 1997. ACM Press.

[27] A. Dery and L. Rideau. Distributed programming environments: an example of a message protocol. Technical Report RT-0165, INRIA, 1994.

[28] Design Science. *MathType User Manual*, 1987.

[29] L. Dirat. *Outils pour des mathématiques interactives et distribuées.* Thèse en Informatique, Université de Nice Sophia Antipolis, Janvier 2001.

[30] Figue, http://www-sop.inria.fr/croap/figue.

[31] Frame technology Corporation. *Using Framemaker 4*, 1993.

[32] M. Gordon. Introduction to the HOL system. In M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley, editors, *Proceedings of the International Workshop on the HOL Theorem Proving System and its Applications*, pages 2–3, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[33] HELM project, http://www.cs.unibo.it/˜asperti/helm/home.html.

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

[35] G. Huet, G. Kahn, and C. Paulin-Mohring. The Coq Proof Assistant : A tutorial : Version 6.1. Technical Report RT-0204, INRIA, 1997.

[36] Internet Accessible Mathematical Computation (IAMC) Framework Project, http://icm.mcs.kent.edu/research/iamcproject.html.

[37] I. Jacobs and L. Rideau-Gallot. The PPML Manual. Technical report, INRIA, 1994.

[38] N. Kajler. CAS/PI:A Portable and Extensible Interface for Computer Algebra Systems. In *International Symposium on Symbolic and Algebraic Computation (ISSAC'92)*, pages 376–386. ACM, 1992.

[39] N. Kajler. User interfaces for Symbolic Computation: a Case Study. In *Proceedings of the 6th Annual Symposium on User Interface Software and Technology*, pages 1–10, New York, NY, USA, Nov. 1993. ACM Press.

[40] N. Kajler and N. Soiffer. A Survey of User Interfaces for Computer Algebra Systems. *Journal of Symbolic Computation*, 25(2):127–160, 1998.

[41] E. Kaltofen and S. Watt, editors. *Computers and Mathematics*, New-York, 1989. Springer-Verlag.

[42] D. Knuth. *Computers and Typesetting.* Addison-Wesley, Reading, 1986.

[43] D. Knuth. Typesetting *Concrete Mathematics*. *TUGboat*, 10(1):31–36, Apr. 1989.

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

[45] D. Knuth. *The TeXbook*, volume A of *Computers and typesetting.* Addison-Wesley, Reading, MA, USA, 23rd printing, with corrections. edition, 1993.

[46] D. Knuth and P. MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat*, 8(1):14–25, April 1987.

[47] K. Lagally. ArabTEX — typesetting Arabic with vowels and ligatures. In J. Zlatuška, editor, *EuroTeX 92: Proceedings of the 7th European TeX Conference*, pages 153–172, Brno, Czechoslovakia, Sept. 1992. Masarykova Universita.

[48] A. Lazrek. Aspects de la problématique de la confection d'une fonte pour les mathématiques arabes. *Cahiers GUTenberg*, Mai 2001. Numéros 39-40.

[49] A. Lazrek. A package for typesetting arabic mathematical formulas. In *DANTE, Deutschsprachige Anwendervereinigung TeX e.V*, 2001. http://www.dante.de/dante2001/handouts/lazrek-arabmath/sysla.ps.

[50] J. Li and G. Lett. Using MathML to Describe Numerical Computations. In *MathML International Conference 2000: University of Illinois, Urbana-Champaign, USA, October 20-21*, 2000. Extended abstract in http://www.mathmlconference.org/2000/Talks/li/.

[51] Maple, http://www.maplesoft.com.

[52] MathCad, http://www.mathsoft.com.

[53] Mathematica, http://www.wolfram.com.

[54] MathML support in Figue, http://www-sop.inria.fr/lemme/hanane.naciri/these/mathml/main.html.

[55] Design Science MathPlayer, http://www.dessci.com/webmath/mathplayer.

[56] Design Science MathType, http://www.dessci.com/company/press/releases/dec01.stm.

[57] R. Miner and P. Topping. Math on the Web: A Status Report, January 2002. Focus: Authoring Tools.

[58] Mozilla 1.0, http://www.mozilla.org/roadmap/mozilla-1.0.html.

[59] H. Naciri and L. Rideau. Affichage interactif, bidimensionnel et incrémental de formules mathématiques. In *Proceedings of the fifth African conference on research in computer science*, Antananarivo, Madagascar, Octobre 2000.

[60] H. Naciri and L. Rideau. Affichage et manipulation interactive de formules mathématiques dans les documents structurés. Rapport de recherche RR-4140, INRIA, Mars 2001.

[61] H. Naciri and L. Rideau. FIGUE: Mathematical Formula Layout with Interaction and MathML Support. In *Proceedings of the Fifth Asian Symposium on Computer Mathematics ASCM'2001*, Matsuyama, Japan, September 2001. Ehime University.

[62] H. Naciri and L. Rideau. The Mariage of MathML and Theorem Proving. In *Proceedings of the IAMC'2001 Workshop on Internet Accessible Mathematical Computation in connection with ISSAC'2001*, Western Ontario Canada., July 2001. http://icm.mcs.kent.edu/research/iamc.html.

[63] H. Naciri and L. Rideau. Formal Mathematical Proof Explanations in Natural Language Using MathML: An Application to Proofs in Arabic. In *MathML International Conference: Hickory Ridge Conference Center, Chicago, IL, USA, June 28–30*, 2002. Extended abstract in http://www.mathmlconference.org/2002/presentations/naciri/.

[64] Omega, http://omega.cse.unsw.edu.au:8080/index.html.

[65] M. A. Ozols, A. Cant, and K. A. Eastaughffe. XIsabelle: A system description. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 400–403, Berlin, 1997. Springer.

[66] C. Pasquier and L. Théry. A distributed editing environment for XML documents. In *first ECOOP Workshop dedicated to XML and Object Technologies*, Cannes, June 2000.

[67] L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[68] Quelques exemples de preuves développées sous Pcoq, http://www-sop.inria.fr/lemme/hanane.naciri/pcoq-exemples/.

[69] O. Plaice and Y. Haralambous. Generating MathML and Other ...ML from Omega. In *MathML International Conference 2000: University of Illinois, Urbana-Champaign, USA, October 20-21*, 2000. Extended abstract in http://www.mathmlconference.org/2000/Talks/plaice/.

[70] *The Publisher User Manual.* Ann Arbor, MI, USA, 1988.

[71] V. Quint. An Interactive System for Mathematical Text Processing. *Technology and Science of Informatics*, 2(3):169–179, 1983.

[72] V. Quint. Interactive Editing of Mathematics. In *Proceedings of the First International Conference on Text Processing Systems*, pages 55–68, Dublin, Ireland, October 1984. Boole Press.

[73] V. Quint, I. Vatton, and H. Bedor. Le système Grif. *T.S.I. Téchniques et sciences informatique*, 5(4):337–441, 1986.

[74] B. Ritchie. *The Design and Implementation of an Interactive Proof Editor.* PhD thesis, University of Edinburgh, Departement of Computer Science, 1988.

[75] P. Rosier. Développement d'un environnement de programmation Centaur pour Maple. Rapport de stage de DEA Calcul et Déduction, Université de Nice - Sophia Antipolis, 1995.

[76] G. L. Schaps. Compiler construction with Antlr and Java. *Dr. Dobb's Journal of Software Tools*, 24(3):84, 86–89, Mar. 1999.

[77] ANTLR Complete Language Translation Solutions, http://www.antlr.org/.

[78] SmarTools, http://www-sop.inria.fr/oasis/smarttools/.

[79] C. J. Smith and N. Soiffer. MathScribe: a user interface for computer algebra systems. In B. W. Char, editor, *Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation: Symsac '86*, pages 7–12, Waterloo, Ontario, 1986. ACM Press.

[80] N. Soiffer. The Design of a User Interface for Computer Algebra Systems. Technical Report CSD-91-626, University of California, Berkeley, 1995.

[81] A. Solomon and C. A. Struble. The Javamath API, an architecture for internet accessible mathematical services, 2001.

[82] J. Srouji and D. Berry. Arabic formatting with `ditroff/ffortid`. *EPODD, Electronic Publishing - Origination, Dissemination and Design*, 5(4):163–208, 1992.

[83] D. Syme. A New Interface for HOL - Ideas, Issues and Implementations. In E.T. Schubert, P.J. Windley, and J. Alves-Foss, editors, *8th International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume 971 of *Lecture Notes in Computer Science*, pages 324–339, Aspen Grove, Utah, USA, 1995. Springer-Verlag.

[84] T. Talbot. *Equation Builder User Manual*, 1992.

[85] Techexplorer, http://www-4.ibm.com/software/network/techexplorer/.

[86] T. Teitelbaum and T. Reps. The Cornell Program Synthesizer: a syntax-directed Programming Environment. *Communications of the ACM*, 24(9):152–168, 1981.

[87] TeX4ht, http://www.cis.ohio-state.edu/ gurari/tex4ht/mn.html.

[88] TeXmacs, http://www.texmacs.org/.

[89] L. Théry, Y. Bertot, and G. Kahn. Real Theorem Provers Deserve Real User-Interfaces. In S. Dooley, editor, *Proceedings of the Fifth ACM Symposium on Software Development Environments (SDE5)*, pages 283–290, Washington D. C, dec 1992.

[90] P. Topping. Using MathType to create TEX and MathML equations. *TUGboat*, 20(3):184–188, 1999.

[91] S. van Egmond, F. Heeman, and J. van Vliet. INFORM: an interactive syntax-directed formulae editor. *The Journal of Systems and Software*, 9(3):169–182, 1989.

[92] I. Vatton. W3C's Amaya 4.0 Editor/Browser. Technical report, W3C, 2000. http://w3c1.inria.fr/Amaya/.

[93] D. Venable. MacEqn (a Macintosh computer program). *Software for Recognition Technologies*, (9):169–182, 1985.

[94] W3C Math Home, http://www.w3.org/math/.

[95] P. Wang, S. Gray, N. Kajler, D. Lin, W. Liao, and X. Zou. IAMC architecture and prototyping: a progress report. In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, pages 337–344, New York, NY 10036, USA, 2001. ACM Press.

[96] WebEQ, http://www.webeq.com/.

[97] Wolfram Research, 100 Trade Center Drive, Champaign, IL, USA. *Mathlink Reference Guide, Version 2.2*, 1993.

[98] Word User's Guide, 1993.

[99] Universal Math Stylesheet, http://www.w3.org/math/.

[100] The Extensible Stylesheet Language (XSL), http://www.w3.org/style/xsl/.

[101] D. A. Young and P. S. Wang. GI/S: a graphical user interface for symbolic computation systems. *Journal of Symbolic Computation*, 4(3):365–380, 1987.

Annexe A

Algorithme bidirectionnel d'Unicode

Quand le texte se présente sous forme de lignes horizontales, la plupart des systèmes d'écritures affichent les caractères de gauche à droite tandis que d'autres (comme l'arabe et l'hébreu) affichent de droite à gauche. Si le texte possède une seule direction horizontale, l'ordre d'affichage du texte est sans ambiguïté. Par contre, en présence de texte bidirectionnel (un mélange de texte écrit de gauche à droite et de droite à gauche), déterminer l'ordre d'affichage des caractères peut parfois être ambigu.

Dans la plupart des cas, il n'est pas nécessaire d'ajouter au texte des informations supplémentaires pour obtenir l'ordre d'affichage correct mais il existe des textes où l'ordre bidirectionnel implicite ne suffit pas pour produire un texte compréhensible. Afin de résoudre ces cas, Unicode prévoit des codes de formatage directionnel permettant de contrôler de manière précise l'ordre d'affichage et assurant de la sorte un échange lisible. On utilise deux types de codes de formatage bidirectionnel explicites qui permettent de modifier l'algorithme bidirectionnel implicite d'Unicode : les codes d'enchâssement (*embedding*) et de forçage (*Overrides*).

EDAG (RLE)	Enchâssement de droite à gauche	Considérer le texte qui suit comme enchâssé et de droite à gauche.
EGAD (LRE)	Enchâssement de gauche à droite	Considérer le texte qui suit comme enchâssé et de gauche à droite.
FDAG (RLO)	Forçage droite à gauche	Forcer la directionalité des caractères qui suivent : forte directionalité droite à gauche.

FGAD (LRO)	Forçage gauche à droite	Forcer la directionalité des caractères qui suivent : forte directionalité gauche à droite.
DFD (PDF)	Dépilement de formatage directionnel	Rétablir l'état bidirectionnel à ce qu'il était avant le dernier FDAG, FGAD, EDAG ou EGAD.
MDAG (RLM)	Marque droite à gauche	Caractère droite à gauche de chasse nulle.
MGAD (LRM)	Marque gauche à droite	Caractère gauche à droite de chasse nulle.

Le standard Unicode prescrit un ordre que les caractères doivent respecter en mémoire, appelé *ordre logique*. Quand on traite du texte bidirectionnel, on continue d'interpréter les caractères dans l'ordre logique et seul l'affichage est affecté. L'ordre d'affichage du texte bidirectionnel dépend des propriétés directionnelles des caractères qui composent le texte.

l'algorithme bidirectionnel exécute trois phases principales :

- Découpe du texte en paragraphe. Le reste de l'algorithme s'applique consécutivement à chaque paragraphe.
- Résolution des niveaux d'enchâssement du texte.
- Après découpe du texte en lignes, réordonnancement du texte pour l'afficher ligne à ligne en utilisant les niveaux d'enchâssement résolus.

Résolution des niveaux d'enchâssement l'algorithme bidirectionnel fait appel aux types bidirectionnels des caractères (valeur attribuée chaque caractère Unicode) et aux codes de formatage explicites pour produire une liste de niveaux d'enchâssement résolus. Cette résolution comprend cinq étapes :

1. établissement du niveau d'enchâssement du paragraphe qui précise l'orientation bidirectionnelle implicite du paragraphe en question,
2. établissement des niveaux d'enchâssement et des directions explicites,
3. résolution des types faibles,
4. résolution de types neutres et
5. résolution des niveaux d'enchâssement implicites.

1 - Niveau de paragraphe

Pour chaque paragraphe, trouver le premier caractère à forte directionalité (G, DA, D). Si le caractère obtenu est de type DA ou D alors initialiser le niveau d'enchâssement à 1, sinon à 0. Lorsque un protocole de niveau supérieur précise le niveau de paragraphe, il n'est pas nécessaire de le calculer.

2 - Niveaux d'enchâssement et des directions explicites

On détermine les niveaux d'enchâssement explicites à l'aide des codes d'enchâssement et de forçage et par application des règles suivantes :

- initialiser le niveau d'enchâssement courant *nec* au niveau d'enchâssement du paragraphe et l'état de forçage bidirectionnel *efc* à neutre. Traiter chaque caractère de manière itérative en appliquant les règles ci-dessous.
- pour chaque code *EDAG*, calculer le plus petit niveau d'enchâssement n impair supérieur au niveau courant. Empiler le niveau d'enchâssement et l'état de forçage courants. Réinitialiser le niveau courant à ce nouveau niveau n et l'état de forçage à *neutre* ($nec \leftarrow n$ et $efc \leftarrow neutre$).
- pour chaque code *EGAD*, calculer le plus petit niveau d'enchâssement n pair supérieur au niveau courant. Empiler le niveau d'enchâssement et l'état de forçage courants. Réinitialiser le niveau courant à ce nouveau niveau n et l'état de forçage à *neutre* ($nec \leftarrow n$ et $efc \leftarrow neutre$).
- pour chaque code de forçage *FDAG*, calculer le plus petit niveau d'enchâssement n impair supérieur au niveau courant. Empiler le niveau d'enchâssement et l'état de forçage courants. Affecter le niveau courant à ce nouveau niveau n et l'état de forçage à *droite-à-gauche* ($nec \leftarrow n$ et $efc \leftarrow$ droite-à-gauche).
- pour chaque code de forçage *FGAD*, calculer le plus petit niveau d'enchâssement n pair supérieur au niveau courant. Empiler le niveau d'enchâssement et l'état de forçage courants. Affecter le niveau courant à ce nouveau niveau n et l'état de forçage à *gauche-à-droite* ($nec \leftarrow n$ et $efc \leftarrow$ gauche-à-droite).
- pour tous les types sauf *EDAG*, *EGAD*, *FDAG*, *FGAD*, et *DFD*, ajuster le niveau du caractère courant au niveau d'enchâssement courant et si l'état de forçage directionnel n'est pas neutre, réinitialiser le type bidirectionnel du caractère courant à l'état de forçage directionnel courant. Si l'état de forçage est neutre alors les caractères conservent leurs types habituels : les caractères arabes restent DA, les caractères latin demeurent L, les neutres N et ainsi de suite. Si l'état de forçage directionnel est égal à D alors les caractères deviennent D. Si l'état de forçage est égal à G, les caractères passent alors à G.
- pour chaque code *DFD*, trouver le code d'enchâssement ou de forçage correspondant. Si un code correspondant valable existe, rétablir (dépiler) le dernier niveau d'enchâssement et l'état de forçage directionnel mémorisés (empilés).
- éliminer tous les codes EDAG, EGAD, FDAG, FGAD, DFD et SED.
- On applique les règles restantes à chaque passage de caractères situé au même niveau, appelé palier. Pour chaque passage, établir la directionalité du début-de-palier (*dp*) et de la fin-de-palier (*fp*), G ou D. Cette valeur dépend du plus haut des deux niveaux de chaque côté de la limite (au début ou à la fin du paragraphe le niveau de l'autre passage est le niveau d'enchâssement de base). Si le niveau supérieur est impair, la directionalité est D, sinon elle est G.

3 - Résolution des types faibles

On peut maintenant résoudre les types faibles, palier par palier. Aux limites de palier, quand il faut connaître le type du caractère situé de l'autre côté de la limite, on utilise la directionalité affectée au *fp* ou au *dp*.

- Vérifier chaque signe à chasse nulle (SACN) du palier et modifier leur type de SACN à celui du caractère précédent. Si le SACN se situe en début de palier, il prend la directionalité du *dp*.
- Remonter en amont de chaque occurrence d'un nombre européen jusqu'à trouver le premier type fort (D,G,DA ou *dp*). Si on trouve un DA, changer le type de nombres européens à nombres arabes.
- Changer tous les DA en D.
- Un séparateur européen entre deux nombres européens se change en un nombre européen. Un séparateur commun entre deux chiffres de même type, prend ce type.
- Une suite de terminateurs européens contiguë à des nombres européens devient une série de nombres européens.
- Sinon, les séparateurs et terminateurs se changent en autres neutres.
- Remontrer en amont de chaque occurrence d'un nombre européen jusqu'à trouver le premier type fort (D,G ou *dp*). Si on trouve un G, changer le types de chiffres européens à G.

4 - Résolution des types neutres

On peut maintenant résoudre les types neutres, palier par palier. Aux limites de palier, quand il faut connaître le type du caractère situé de l'autre coté de la limite, on utilise la directionalité affectée à *fp* ou à *dp*.

La phase suivante détermine la direction des neutres. À la fin de cette phase, tous les neutres sont devenus G ou D. En règle générale, les neutres prennent la direction du texte qui les jouxte. En cas de conflit, ils adoptent la direction de l'enchâssement.

- Une suite de neutres adopte la direction du texte fort contigu si le texte des deux cotés a la même direction. Les chiffres européens et arabes sont considérés comme du texte D. On utilise un début-de-palier (*dp*) et une fin-de-palier (*fp*) pour délimiter les paliers.
- tous les neutres restants adoptent la direction de l'enchâssement.

5 - Résolution des niveaux d'enchâssement implicites

Dans la dernière phase, il se peut que le niveau d'enchâssement du texte soit augmenté, suite à la directionalité déterminée. Un texte droite-à-gauche aura toujours en fin de compte un niveau impair et les textes gauche à droite ou composés de nombres aboutiront toujours à un niveau pair.

- Pour tous les caractères ayant une direction d'enchâssement paire (gauche-à-droite), ceux de type D montent d'un niveau, ceux de type NA ou NE montent de deux niveaux.

– Pour tous les caractères ayant une direction d'enchâssement impaire (droite-à-gauche), ceux de type G, NA ou NE montent d'un niveau.

Annexe B

Règles XPPML pour traduire du MathML vers la structure figue

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE prettyPrinter SYSTEM "../ppml.dtd">

<prettyPrinter ppName="std" langName="mymathml"
 xmlns="http://www-sop.inria.fr/lemme/xppml/1.0"
 xmlns:f="http://www-sop.inria.fr/lemme/figue/1.0">

  <import package="lang.mymathml.ppaux"/>
  <rule>
    <pattern>
      <node  alias="z" match="math">
        <sons>
          <var name="x" type="list"/>
        </sons>
      </node>
    </pattern>
    <f:v>
       <iter>
        <variable name="x"/>
      </iter>
    </f:v>
  </rule>
  <rule>
    <pattern>
      <node alias="z" match="root">
        <sons>
          <var name="x" type="list"/>
        </sons>
      </node>
```

```
    </pattern>
    <f:hv>
      <iter>
         <variable name="x"/>
      </iter>
    </f:hv>
  </rule>
   <rule>
    <pattern>
      <node alias="z" match="mrow">
        <sons>
          <var name="x" type="list"/>
        </sons>
      </node>
    </pattern>
    <f:row >
          <iter>
      <variable name="x"/>
      </iter>
      </f:row>
    </rule>
    <rule>
    <pattern>
      <template match="z=mi"/>
    </pattern>
     <extFun name="miTestAttrpp">
      <arg value="z" type="var"/>
    </extFun>
  </rule>
  <rule>
    <pattern>
      <template match="z=mn"/>
    </pattern>
    <extFun name="miTestAttrpp">
      <arg value="z" type="var"/>
    </extFun>
  </rule>

 <rule>
    <pattern>
      <template match="z=mo"/>
    </pattern>
    <extFun name="miTestAttrpp">
      <arg value="z" type="var"/>
    </extFun>
  </rule>
```

```
<rule>
   <pattern>
     <template match="z=mtext"/>
   </pattern>
   <extFun name="mtextAttrpp">
     <arg value="z" type="var"/>
   </extFun>
 </rule>

<rule>
   <pattern>
     <template match="z=ms"/>
   </pattern>
   <extFun name="msAttrpp">
     <arg value="z" type="var"/>
   </extFun>
 </rule>
 <rule>
   <pattern>
     <template match="x=mchar"/>
   </pattern>
   <extFun name="mcharAttrpp">
     <arg value="x"  type="var"/>
   </extFun>
 </rule>
 <rule>
   <pattern>
     <template match="z=msqrt(*base)"/>
   </pattern>
   <f:sqrt>
     <variable name="base"/>
   </f:sqrt>
 </rule>

   <rule>
   <pattern>
     <template match="z=mfrac(*x,*y)"/>
   </pattern>
   <f:fraction>
     <att name="test">
     <getAtt element="z" nameatt="linethickness"/>
     </att>
     <att name="linethickness">
      <extFun name="mfracAttrpp" >
        <arg value="z" type="var"/>
     </extFun>
   </att>
```

```
      <att name="line">
       <extFun name="mfracAttrpp" >
         <arg value="z" type="var"/>
      </extFun>
    </att>
      <variable name="x"/>
      <variable name="y"/>
    </f:fraction>
  </rule>

</prettyPrinter>
```

Annexe C

MathML et l'affichage bidirectionnel

Nous proposons une classification des éléments MathML suivant le comportement de leur algorithme de formatage dans un contexte bidirectionnel. Nous décrivons aussi le comportement des algorithmes de formatage pour chaque catégorie d'élément MathML dans le cas du mode d'affichage bidirectionnel et aussi dans le cas du mode miroir.

Éléments entités (Token Elements)

mi	identificateur (identifier)
mn	nombre
mo	opérateur, délimiteur, ou séparateur
mtext	texte
mspace	espace
ms	chaîne littéral
mglyph	ajoutant un nouveau caractère (glyph) dans MathML

Pour les éléments *Token* qui contiennent du texte (*mtext, mo, mi, mn* et *ms*), l'algorithme bidirectionnel implicite d'Unicode [15] sera appliqué pour afficher leurs contenu (i.e caractères sont réordonnées en se basant sur leurs propriétés). La base de direction restera toujours *left-to-right*.

Par contre dans le cas des entités références, il faut regarder quel est le symbole correspondant suivant le mode d'affichage. Par exemple, pour l'entité référence `∈`, elle est présentée en $\in$ dans le cas du mode d'affichage normal et en $\ni$ dans le cas du mode miroir. Pour les entités références, il serait préférable de proposer leurs codes pour le mode normal et aussi pour le mode miroir. L'application des règles modifiant le style de ces entités dans le cas miroir peut être une autre solution envisageable.

Dans le cas des chiffres en arabe, il faut préciser le style (fonte) à utiliser: les chiffres arabes du Maghreb en style usuel (0,1,2 ..) ou les chiffres arabes du Machrek (٠, ١, ٢). L'attribut **mathvariant** peut être utilisé dans ce cas pour préciser le style d'affichage des chiffres.

L'élément *mspace* représente un espace blanc entre deux éléments successifs ayant la même direction d'affichage (la définition de l'espace dans le mode normal et le mode miroir). Par contre dans le mode bidirectionnel, *mspace* représente l'espace blanc entre deux blocs d'éléments regroupés par le critère de direction. Par exemple, prenons un *mrow* avec n éléments ou les éléments $A_1..A_i$ sont orientés de gauche à droite et les les éléments $A_{i+2}..A_n$ sont orientés de droite à gauche. Les deux blocs de direction différente sont séparés par un *mspace* (A_{i+1} est un mspace élément). L'affichage de ce *mrow* de direction gauche à droite aura la forme suivante :

$$\boxed{A_1A_2..A_i}\ |----|\ \boxed{A_n..A_{i+2}}$$

L'espace blanc ne sépare pas forcement les deux éléments successif par ordre logique dans le mode bidirectionnel. L'élément *mspace* est alors un élément neutre qui prend la direction de son élément englobant. Dans le cas bidirectionnel *mspace* met l'espace blanc entre deux blocs successifs sans influencer l'algorithme d'affichage : le traitement du premier bloc, le positionnement ensuite du blanc et le traitement enfin du second bloc.

L'élément *ms* représente du texte pour les environnements de programmation ou pour les systèmes de calcul formel. L'algorithme de formatage de la chaîne latérale applique aussi l'algorithme bidirectionnel d'Unicode. L'élément *ms* a une direction neutre qui n'influence pas les autres éléments.

L'élément *mglyph* permet d'ajouter des caractères non standard à MathML. Leur direction est aussi neutre et leur présentation reste la même pour tous les modes d'affichage et pour toutes les directions (équivalent à une image). Dans le cas du mode miroir, le glyph ne sera pas réfléchi automatiquement. Pour pouvoir le réfléchir il faut le préciser en envisageant par exemple un attribut *mirror*. Si la fonte proposée ne contient pas la symétrie de ce glyph, un message d'erreur sera affiché.

Schémas de disposition générale (General Layout Schemata)

mrow	regroupe horizontalement tout nombre de sous expressions
mfrac	forme une fraction à partir de deux sous expressions
msqrt	forme une racine carrée (racine sans indice)
mroot	forme une racine avec indice
mstyle	change le style
merror	stocke le message d'erreur d'un préprocesseur
mpadded	ajoute de l'espace autour du contenu
mphantom	rend le contenu invisible mais garde sa taille
mfenced	entoure le contenu avec deux délimiteurs
menclose	englobe le contenu par un symbole qui s'étire comme un long signe de division

La direction de l'élément *mrow* est importante dans le cas du mode bidirectionnel. Dans ce cas, si la direction est de gauche à droite l'algorithme de formatage ne change pas (comme en mode normal) et sinon il inverse l'ordre d'affichage de ses éléments (comme en mode miroir). Les règles de coupure de l'élément *mrow* dépendent aussi de la direction d'affichage : en mode miroir, le retour à ligne se fait de droite à gauche. En mode bidirectionnel, la ligne de retour a la même direction que l'élément *mrow* et elle suit le même algorithme de formatage bidirectionnel.

Une conséquence directe des règles d'affichage bidirectionnel est l'importance de regrouper correctement les éléments par bloc horizontal. Par exemple, une formule marocaine composée d'un élément *mtext* avec du texte arabe (direction de droite a gauche) et d'une expression mathématique prend la direction RL. La présentation de cette formule change selon si l'expression de cette formule est représenté par un mrow ou non. Le mauvais regroupement peut donner un affichage non correcte. Pour cela, il faut définir des règles pour un bon regroupement des éléments: en présence d'un mtext de direction RL, tous les *mi, mo, mn* (éléments neutres) successifs doivent être regroupés par un nested mrow. Lors de la génération du MathML à partir de la formule origine ou lors de la spécification des notations, il faut faire attention à l'importance de la structure pour un affichage bidirectionnel correcte.

La direction de l'élément *mfrac* ne change pas son algorithme de formatage. Cet élément appartient à la famille des constructeurs verticaux qui ont un formatage indépendant de la direction d'affichage. Par contre le formatage des éléments *msqrt* et *mroot* dépend de leur direction d'affichage. En mode bidirectionnel, le comportement de leur algorithme de formatage est le même en mode normal et en mode miroir, le dessin du symbole racine est réfléchi et l'élément *mroot* inverse la disposition de ses éléments fils.

Les éléments de décoration comme *mstyle*, *merror*, *mpadded* ou *mphantom* ne changent pas. Ils gardent le même comportement qu'en mode normal. Le schéma général de l'élément *mfenced* est le suivant :

```
<mfenced open="opening-fence"
         close="closing-fence"
         separators="sep#1 sep#2 ... sep#(n-1)">
         arg#1
         ...
         arg#n
</mfenced>
```

L'élément *mfenced* inverse la disposition de ses arguments si sa direction est de droite à gauche, il dispose à droite le premier délimiteur (*open*) et à gauche le deuxième délimiteur et il inverse aussi l'ordre de disposition des séparateurs. L'élément *menclose* ne change pas son formatage car il s'agit d'un élément qui impose un style indépendamment de la direction.

Script et schémas de limite (Script and Limit Schemata)

msub	associe un indice inférieur à une base
msup	associe un indice supérieur à une base
msubsup	associe un couple d'indices inférieur et supérieur à une base
munder	associe un underscript à une base
mover	associe un overscript à une base
munderover	associe un couple de underscript et de overscript à une base
mmultiscripts	associe plusieurs indices à une base

Les éléments *msub*, *msup*,*msubsup* et *mmultiscripts* dépendent de la direction d'affichage. Ils disposent à gauche de la base l'indice ou l'exposant dans le cas de la direction droite à gauche. Le formatage des éléments *munder*, *mover* et *munderover* de type vertical ne dépend pas de la direction. Dans ce dernier cas, les algorithmes de formatage de changent pas.

Tables et matrices

mtable	table ou matrice
mlabeledtr	ligne de table avec un label ou un numéro d'équation
mtr	une ligne de table ou de matrice
mtd	une élément d'une table ou d'une matrice
maligngroup et malignmark	marqueurs d'alignement

Seuls les éléments horizontaux de cette famille changent leur formatage en fonction de la direction : les éléments *mlabeledtr* et *mtr*. Les lignes de tables sont inversés si la direction est de droite à gauche. Les autres éléments de ce groupe ne changent pas leur algorithme de formatage.

Expression d'action

maction	attache des actions à une sous expression

Le comportement de l'élément *maction* ne dépend pas de la direction d'affichage.

Annexe D

Définition de la syntaxe du langage ppml

```
program:   "prettyprinter" id "of" id
             ( "is" | "extends" id ("," id)* "with")
             ("auxillary" "package" string ";")?
             ( rule ";")*
           "end" "prettyprinter"
           ;

rule:      cpattern "->" box;

cpattern:  (id ":")? pattern  ("!" int)? (annot)?
            ("when" fun ("and" fun)*)?
           ;
pattern:     var ("as" pattern)?
           | id "(" (pattern ("," pattern)*)? ")"
           | id "[" (pattern ("," pattern)*)? "]"
           | id var           ;
           | id string           ;

fun:         fname "(" (farg ("," farg)*)? ")"
           ;

farg:        string
           | int
           | var
           | annot
           ;
```

```
box:          string
            | (id ":")? var  ("!" sint)?
            | (id ":")? annot  ("!" sint)?
            | "in" "class" "=" id ":" box
            | fname "(" var ")"
            | "(" (box)+  ")"
            | "[" combinator  (box)+ "]"
            | "if" fun ("then" box )? ("else" box )? "end"
            | "case" fun  (int ":" box)* ("default" ":" box)? "end"
            ;

combinator: "<" id (int ("," int)*)? ">"
            ;
var:         star | doublestar;

star :       "*" (id)?;
doublestar: "**" (id)?;
string:      "\"" (~\"")* "\"";
sint:        ("-"|"+") int
int:         "0"|("1".."9")("0".."9")*
fname:         id ("." id)*
id:          ("a".."z"|"A".."Z") ("a".."z"|"A".."Z"|"_"|"0".."9")*
annot:       "^"  id
```

Annexe E

Table de correspondance entre les boîtes FIGUE et les éléments MathML-*présentation*

Le code source java du module qui traduit la structure MathML en une structure de boîtes FIGUE est dans

http://www-sop.inria.fr/lemme/Hanane.Naciri/these/raportthese/Utility.java.

La table de correspondance entre les boîtes FIGUE et les éléments MathML-*présentation* est :

```
Schémas de disposition générale

<mrow> f0 f1 ..fn <\mrow>              => Row
<mfrac> numerator denominator <\mfrac> => Fraction
<msqrt>   base <\msqrt>                => NRoot
<mroot>   base index <\mroot>          => NRoot
Seule l'attribut color et l'attribut background sont pris en compte lors de
cette traduction. Les autres attributs de mstyle sont ignorés par FIGUE.

<mstyle> child </mstyle>               => ChangeGraphicalContextAnonymous
L'erreur est affiché en couleur rouge
<merror> ... </merror>                 => Atom "Syntax Error Announced Here"
<mpadded> ... </mpadded> => ignorer par FIGUE
La couleur du style d'affichage de mpantom est blanche.
Alors les éléments sont transparents
<mphantom> ... </mphantom>             => ChangeGraphicalContextAnonymous
```

La traduction suivante ajoute des séparateurs entre les éléments du vecteur Row

```
<mfenced> ... </mfenced>               => Delimitor ‘‘(‘‘Row ‘‘)’’
```

Script et schémas de limite (Script and Limit Schemata)

```
<msubsup> base subscript superscript <\msubsup>           => SubSup
<msup>    base subscript <\msup>                          => Sup
<msub>    base subscript <\msub>                          => Sub
<munder>  base underscript <\munder>                      => Under
<mover>   base overscript <\mover>                        => Over
<munderover>  base underscript overscript <\munder>       => UnderOver
<munder accentunder=’’true’’> base underscript <\munder>  => UnderStretche
<munder accentunder=’’false’’> base underscript <\munder> => Under
<mover accent=’’true’’> base overscript <\mover>          => OverStretche
<mover accent=’’false’’> base underscript <\mover>        => Over
```

Tables et matrices

```
<mtable> ... </mtable>    => Table
<mtr> ... </mtr>          => TableRow
<mtd> child </mtd>        => Glyph
```

Expression d’action

```
<maction> ... </maction>  => ignorer par FIGUE
```

Eléments entités

```
<mi> contenu </mi>        => Atom
<mn> contenu </mn>        => Atom
<mo> contenu </mo>        => Atom
<ms> contenu </ms>        => Atom
<mtext> contenu </mtext>  => Atom
```

Annexe F

DTD de FIGUE

La description de la DTD incluant à la fois MathML et la description des boîtes FIGUE est la suivante :

```
<!ENTITY % math    SYSTEM "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<!ENTITY % figue   SYSTEM "http://www-sop.inria.fr/lemme/Hanane.Naciri/
these/mathml/classes/figue.dtd">
%math;
%figue;
```

Ici, la DTD décrivant les boîtes FIGUE.

```
<!-- ********************************************************** -->
<!-- DTD for Figue                                              -->
<!--                                                            -->
<!-- Author: Claude Pasquier <Claude.Pasquier@sophia.inria.fr>     -->
<!--                                                            -->
<!-- ********************************************************** -->
<!-- All combinators can thus be easily represented with a 'Comb'  -->
<!-- element.                                                   -->
<!-- For examples, the two declarations below are identical :      -->
<!--     1. <V Indent="30" Interline="10">                       -->
<!--     2. <Comb Name="V" Args="30,10">                         -->
<!--                                                            -->
<!-- The way to specify a list of arguments with the 'Args'        -->
<!-- attribute is also possible for commonly used elements.        -->
<!-- The following declaration is also equivalent to the two       -->
<!-- previous ones :                                            -->
<!--     3. <V Args="30,10">                                     -->

<!-- ********************************************************** -->
```

```
<!-- *                 ENTITY DECLARATIONS                       * -->
<!-- ********************************************************** -->

<!-- general attribute definitions                                -->

<!ENTITY % name       "Name       NMTOKEN                   #REQUIRED">
<!ENTITY % foreground "Foreground CDATA                     #IMPLIED" >
<!ENTITY % background "Background CDATA                     #IMPLIED" >
<!ENTITY % unit       "Unit       (Pixel|Blank10e|Line10e) #IMPLIED" >
<!ENTITY % interword  "Interword  NMTOKEN                   #IMPLIED" >
<!ENTITY % indent     "Indent     NMTOKEN                   #IMPLIED" >
<!ENTITY % interline  "Interline  NMTOKEN                   #IMPLIED" >
<!ENTITY % justify    "Justif     (Left|Center|Right)       #IMPLIED" >
<!ENTITY % args       "Args       CDATA                     #IMPLIED" >

<!-- glyph entity definition                                      -->

<!ENTITY % glyph "Atom|H|V|P|Context|Insets|Hyperlink|Comb|Object|
                  Above|Below|Over|NRoot|Ind|IndExp|Exp|math"       >

<!-- ********************************************************** -->
<!-- *            ELEMENT AND ATTRIBUTE DECLARATIONS           * -->
<!-- ********************************************************** -->

<!ELEMENT Facade        (Resources?, Selections?, Root)             >
<!ATTLIST Facade
          Width         NMTOKEN                           #IMPLIED >

<!-- Resources declarations                                       -->

<!ELEMENT Resources     (Fonts?, Colors?, Styles?)                  >
<!ELEMENT Fonts         (Font+)                                     >
<!ELEMENT Colors        (Color+)                                    >
<!ELEMENT Styles        (Style+)                                    >
<!ELEMENT Font          EMPTY                                       >
<!ATTLIST Font
          %name;
          Style         (Plain|Italic|Bold|BoldItalic)      #REQUIRED
          Size          CDATA                             #REQUIRED
          Family        (HELVETICA   | COURIER |
                         TIMES_ROMAN | DIALOG  |
                         DIALOG_INPUT| SYMBOL  |
                         DEFAULT)                         #REQUIRED
          Underlined    (true|false)                      #REQUIRED>
<!ELEMENT Color         EMPTY                                       >
<!ATTLIST Color
          %name;
```

```
          Blue          NMTOKEN                               #REQUIRED
          Red           NMTOKEN                               #REQUIRED
          Green         NMTOKEN                               #REQUIRED>
<!ELEMENT Style         EMPTY                                          >
<!ATTLIST Style
          %name;
          Font          CDATA                                 #IMPLIED
          %foreground;
          %background;                                                 >
<!ELEMENT Selections    (Selection+)                                   >
<!ELEMENT Selection     EMPTY                                          >
<!ATTLIST Selection
          %name;
          Priority      (Min|Low|Medium|High|Max)             #REQUIRED
          %foreground;
          %background;
          Visible       (true|false)                          #REQUIRED>


<!-- Structure declarations                                          -->

<!ELEMENT Root          (%glyph;)                                      >

<!ELEMENT Atom          EMPTY                                          >
<!ATTLIST Atom
          Value         CDATA                                 #IMPLIED
          Style         CDATA                                 #IMPLIED >

<!ELEMENT H             (%glyph;)*                                     >
<!ATTLIST H
          %interword;
          %unit;
          %args;                                                       >


<!ELEMENT V             (%glyph;)*                                     >
<!ATTLIST V
          %indent;
          %interline;
          %unit;
          %justify;
          %args;                                                       >

<!ELEMENT P             (%glyph;)*                                     >
<!ATTLIST P
          %interword;
          %indent;
```

```
        %interline;
        %unit;
        %args;                                                  >

<!ELEMENT Context       (%glyph;)                               >
<!ATTLIST Context
        Style           CDATA                         #IMPLIED
        Font            CDATA                         #IMPLIED
        %foreground;
        %background;                                            >


<!ELEMENT Insets        (%glyph;)                               >
<!ATTLIST Insets
        SpaceBefore     CDATA                          #IMPLIED
        SpaceAfter      CDATA                          #IMPLIED
        SpaceToLeft     CDATA                          #IMPLIED
        SpaceToRight    CDATA                          #IMPLIED
        %unit;                                                  >

<!ELEMENT Hyperlink     (%glyph;)                               >
<!ATTLIST Hyperlink
        Color           CDATA                          #IMPLIED
        VisitedColor    CDATA                          #IMPLIED
        SelectedColor   CDATA                          #IMPLIED
        Font            CDATA                          #IMPLIED >
<!-- The 'Comb' element is used to represent new combinators.
     The name of the combinator and its arguments are specified
     in the 'Name' and 'Args' attributes.
     'Args' contains a list of arguments separated by commas      -->

<!ELEMENT Comb          (%glyph;)*                              >
<!ATTLIST Comb
        %name;
        %args;>

<!-- The 'object' element is used to represent an instance of the
     'LwComb' class                                               -->

<!ELEMENT Object        (Atom)*                                 >
<!ATTLIST Object
        Class            CDATA                          #REQUIRED
        Count            CDATA                          #REQUIRED>
```