Reconnaissance Structurelle de Formules Mathématiques Typographiées et Manuscrites

Stéphane Lavirotte

▶ **To cite this version:**

Stéphane Lavirotte. Reconnaissance Structurelle de Formules Mathématiques Typographiées et Manuscrites. Interface homme-machine [cs.HC]. Université Nice Sophia Antipolis, 2000. Français. NNT : . tel-00523373

HAL Id: tel-00523373

https://theses.hal.science/tel-00523373

Submitted on 5 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE NICE – SOPHIA ANTIPOLIS
École Doctorale des Sciences et Technologies de l'Information et de la Communication

Reconnaissance structurelle de formules mathématiques typographiées et manuscrites

THÈSE
de doctorat

pour obtenir le titre de

Docteur en Sciences

Discipline : Informatique

par

Stéphane LAVIROTTE

Soutenue le 14 juin 2000 à l'ESSI (Sophia-Antipolis)

Composition du jury

Président :	Jean-Marc FEDOU	Professeur à l'Université de Nice Sophia-Antipolis
Rapporteurs :	Karl TOMBRE	Professeur à l'École des Mines de Nancy
	Guy LORETTE	Professeur à l'Université de Rennes I
Examinateurs :	Loïc POTTIER	Chargé de Recherche à l'INRIA Sophia-Antipolis
	Peter SANDER	Professeur à l'Université de Nice Sophia-Antipolis
	Marc BERTHOD	Directeur de Recherche à l'INRIA Sophia-Antipolis

Université de Nice Sophia-Antipolis / Institut National de Recherche en Informatique et Automatique

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier :

- *Loïc Pottier pour ses conseils, son expérience et l'encadrement de cette thèse ;*
- *les membres du jury, Jean-Marc Fédou, Peter Sander, Marc Berthod, et plus particulièrement Karl Tombre et Guy Lorette qui ont accepté de rapporter cette thèse ;*
- *les anciens membres de l'équipe SAFIR ainsi que les membres des projets CAFE et LEMME de l'INRIA qui m'ont tous accueilli avec beaucoup de gentillesse et de bienveillance.*
- *France Limouzis et Patricia Lachaume pour leur soutient et leur aide dans les démarches administratives.*

Je veux aussi remercier toutes les personnes qui ont travaillé ponctuellement, de près ou de loin avec moi :

- *Andréas Kosmala dans le cadre d'une collaboration avec l'Université de Duisburg ;*
- *Olivier Arsac avec qui les collaborations de travail furent nombreuses et toutes plus enrichissantes les unes que les autres ;*
- *Colas Nahaboo et Jean-Michel Léon ainsi que toute l'équipe KOALA pour leurs outils de développement, leur bonne humeur et leurs conseils avisés ;*
- *José Grimm pour tous ses précieux conseils sur LaTeX.*

Enfin, je remercie collectivement tous ceux qui ont bien voulu relire ma thèse, m'apporter leur aide et plus particulièrement Frédérique, qui a su me soutenir au quotidien.

I do not fear computers.
I fear the lack of them.

Isaac Asimov

Table des matières

Table des figures **vii**

Introduction **1**

1 Nos buts initiaux 3
2 Motivations et applications possibles 3
3 Objectifs de l'étude 5
4 Résultats obtenus 5
4.1 Le composant *OFR* 5
4.2 *Irma* : une application 6
5 Plan de lecture 6
6 Conventions typographiques 7

Chapitre I Reconnaissance structurelle de formules mathématiques : état de l'art 9

1 Historique 10
2 De nombreuses applications possibles 11
2.1 Édition de formules mathématiques 12
2.1.1 Syntaxe linéaire 12
2.1.2 Palette de modèles 14
2.1.3 Édition bidimensionnelle 15
2.1.4 Des modes d'édition "coûteux" 15
2.1.5 Vers une édition manuscrite 16
2.2 Bases de formules 17
2.3 Extension des systèmes de reconnaissance de documents 18
2.4 Diverses autres applications 18
3 Définition des notations mathématiques 19
4 Difficultés par rapport à la reconnaissance de textes 20

4.1 Bruit et petits symboles 21
4.2 Segmentation 22
4.3 Reconnaissance des symboles 22
4.4 Ambiguïtés sur le rôle d'un même symbole 23
4.5 Ambiguïté sur le placement relatif des symboles 24
4.6 Ambiguïté dans la notation 25
4.7 Peu de redondance de l'information 26
5 Quelques traitements préliminaires 27
5.1 Seuil de numérisation 29
5.2 Réduction du bruit 30
5.3 Réalignement de l'image 31
5.4 Isoler une formule dans un document 33
5.5 Conclusion 34
6 Segmentation et reconnaissance des symboles 35
6.1 Caractères typographiés 35
6.2 Caractères manuscrits 37
6.3 Conclusion 39
7 Reconnaissance de la structure 39
7.1 Identification des relations spatiales et logiques entre les symboles . 40
7.2 Reconnaissance de la structure de la formule 42
8 Diversité des approches existantes 42
8.1 Diversité des méthodes pour l'analyse structurelle 42
8.1.1 Méthodes syntaxiques 43
Grammaire de coordonnées 43
Schémas de spécification de structure 45
Grammaires probabilistes 45
Grammaires de graphes 46
8.1.2 Méthodes logiques 47
8.1.3 Méthodes mixtes liant analyse géométrique et syntaxique 48
Méthodes procédurales 48
Méthode de découpage par projections 48
8.2 Difficultés pour comparer les approches 50
8.2.1 Diversité dans les données traitées 51
8.2.2 Grande diversité des approches 51
8.2.3 Diversité des notations mathématiques 51
9 Conclusion 52

Chapitre II Concepts pour la reconnaissance de formules **53**
1 Réutilisabiblité 54
1.1 Équations, matrices et notations “exotiques” 54
1.2 Manuscrit et Typographié 54
2 Évolutivité 55
2.1 Introduction de nouvelles notations 55
2.2 Adaptation aux différentes notations 56
3 Interprétation de l’arbre de syntaxe 56
4 Communication des résultats 57
5 Composants pour l’analyse de formules 58
5.1 Détail des différents composants 58
5.2 Critiques possibles de l’architecture en modules 59
6 Conclusion 59

Chapitre III Méthode et outils **61**
1 Architecture d’*OFR* 61
2 Analyse de l’image 63
2.1 Enveloppe d’un symbole 63
2.2 Résultat type attendu de l’OCR 65
2.3 Extrapolation de données 66
2.3.1 Taille relative 66
2.3.2 Ligne de base 66
3 Analyse lexicale 67
4 Analyse géométrique 69
4.1 Introduction des graphes 69
4.2 Définition d’un graphe 70
4.3 Construction du graphe initial 71
4.3.1 Analyse de proximité 71
4.3.2 Critères géométriques et graphiques 72
4.3.3 Critères divers 74
4.3.4 Rôle du type lexical 75
4.3.5 Conclusion 75
4.4 Optimisation 76
4.5 Les limites de la construction du graphe 78
5 Rappels sur les grammaires 79
5.1 Rappels sur les grammaires 80

5.1.1 Grammaires formelles 80
5.1.2 Grammaires attribuées 81
5.2 Grammaires de graphes 83
6 Analyse structurelle 86
6.1 Grammaire de graphes et formules mathématiques 86
6.2 Construction du contexte des règles 87
6.2.1 Exemple 87
6.2.2 Superpositions dans l'application des règles 88
6.2.3 Grammaire sans ambiguïté 91
6.3 Analyse structurelle 93
6.3.1 Analyse ascendante 93
6.3.2 Attributs synthétisés 94
6.3.3 Mise à jour incrémentale du graphe 95
6.4 Optimisation 95
7 Conclusion 96

Chapitre IV Applications et Validation **99**
1 Notations mathématiques typographiées 99
1.1 Un comparatif des logiciels pour la reconnaissance de symboles . . 100
1.1.1 Quelques critères 101
Formats traités 101
Numérisation et traitement de l'image 101
Analyse du document et reconnaissance des symboles . . . 102
Adaptabilité et Évolutivité 102
1.1.2 Protocole de comparaison 103
1.1.3 Tableau comparatif 103
1.1.4 Conclusion 104
1.2 Développement d'un OCR dédié et adaptatif 104
1.2.1 Numérisation du document 105
1.2.2 Extraction des composantes connexes 105
1.2.3 Extraction des caractéristiques des symboles 106
1.2.4 Apprentissage 108
1.2.5 Identification du caractère et résultat fourni 109
1.2.6 Évaluation des performances 110
2 Notations mathématiques manuscrites 111
2.1 Présentation du système NeuroGraph 111

2.2 Reconnaissance des symboles manuscrits . . . 112
3 *Irma* : Dualité du système développé . . . 114
3.1 Adaptation au format des données . . . 115
3.2 Adaptation aux notations reconnues . . . 115
3.3 Adaptation du format de sortie . . . 116
3.4 Communication à l'aide d'*OpenMath* . . . 116
4 Validation . . . 118
4.1 La grammaire de graphes . . . 118
4.2 Résultats . . . 122
4.3 Quelques exemples caractéristiques . . . 123
5 Conclusion . . . 127

Chapitre V Implémentation d'*OFR* **129**
1 Langages de script . . . 130
1.1 Bref historique des langages . . . 130
1.2 Quelques caractéristiques des langages . . . 132
1.2.1 Machine virtuelle . . . 132
1.2.2 Typage . . . 133
1.2.3 Modèle objet . . . 133
1.3 Conclusion . . . 134
2 Critères de choix pour un langage de script . . . 134
2.1 Efficacité et fiabilité . . . 134
2.2 Environnement de développement . . . 134
2.3 Extension possible . . . 135
2.4 Sauvegarde des données dans le langage . . . 135
2.5 Une interface graphique . . . 136
3 Une comparaison des langages de script . . . 136
3.1 Langages de script testés . . . 136
3.2 Conclusion . . . 138
4 Choix d'un langage . . . 139
4.1 *Klone* . . . 139
4.2 *Klm* : une extension graphique de *Klone* . . . 140
4.3 Communication avec d'autres systèmes : *OpenMath* . . . 141
5 Implémentations . . . 143
6 Conclusion . . . 143

Conclusion **145**

Annexe A Exemple de reconnaissance d'une formule avec *Irma* **149**

Annexe B Variété des symboles **153**

1 Exemples de polices et de variations 153
2 Symboles textuels les plus courants 154
2.1 Alphabet latin 154
2.2 Accentuations 154
2.3 Chiffres 155
2.4 Ponctuations 155
3 Symboles mathématiques les plus courants 155
3.1 Alphabet grec 155
3.2 Opérateurs mathématiques 156

Annexe C Un document de test **159**

Références bibliographiques **165**

Table des figures

Chapitre I **9**

1 Édition linéaire dans Maple V.5 13
2 Édition par palette de modèles 14
3 Les étapes de l'analyse d'un document papier 28
4 Exemples de différents seuils de numérisation 30
5 Méthode de projection de textes pour différentes orientations 31
6 Méthode évaluant les angles de droites entre voisins pour un texte 32
7 Méthode de projection appliqué à la formule $\sum_{i=0}^{9} x_i^2$ 32
8 Méthode évaluant les angles de droites entre voisins pour la formule $\sum_{i=0}^{9} x_i^2$ 33
9 Arbre syntaxique de l'expression $ax^2 + bx + c$ 40
10 Étiquetage statistique des liens entre symboles 41
11 Exemple de règles de la grammaire de coordonnées 43
12 Exemple de découpage par projections 49
13 Problèmes soulevés par la méthode de projections 50

Chapitre II **53**

1 Arbres de syntaxe avec attributs sémantiques 57
2 Relations entre les différents composants 58

Chapitre III **61**

1 L'architecture d'*OFR* 62
2 Construction des régions englobantes 63
3 Problème soulevé par les boîtes englobantes rectangulaires 64

4 Ligne de base 67
5 Représentation simplifiée de l'analyse syntaxique 68
6 Deux représentations possibles d'une forme par un graphe 70
7 Construction des liens de proximité 72
8 Détection des relations entre symboles, basée sur une notion d'angle 73
9 Analyse géométrique : définition des régions 74
10 Subdivision du plan en sous-régions 77
11 Optimisation du temps de construction du graphe initial 78
12 Problème de construction du graphe 79
13 Le graphe représentant la formule $A^2 + B$ 87
14 Règle pour l'addition 88
15 Règle pour l'exposant 88
16 Superposition des modèles sur un graphe 89
17 Toutes les superpositions des règles r_+ et $r_\wedge$ 90
18 Réécriture d'un sous-graphe en un nœud 94

Chapitre IV **99**
1 Filtres appliqués à l'image 107
2 Extraction du contour vectoriel 107
3 Extraction des boucles 108
4 Présentation du système pour la reconnaissance des symboles manuscrits . 112
5 Rééchatillonnage des données 113
6 Transformation des coordonnées temporelles en coordonnées spatiales . . . 114
7 Communication entre Irma et Mathematica via OpenMath 117
8 Notations simples 123
9 Notations trigonométriques 123
10 Notations de type somme, produit, intégrale 124
11 Notations de type indice, exposant 124
12 Formules plus complexes 125
13 Notations matricielles 125
14 Exemples divers de formules manuscrites 126

Chapitre V **129**
1 Styles d'architecture logicielle 131

2 Architecture d'*OpenMath* 142

Annexe A **149**

1 Irma : Symboles reconnus par l'OCR 150

2 Reconnaissance de la notation $n! = \prod_{i=1}^{n} i$ 151

3 Reconnaissance de la notation $n! = \prod_{i=1}^{n} i$ (suite) 152

Annexe C **159**

1 Page de test pour la comparaison des résultats des logiciels d'OCR 160

Introduction

Traditionnellement, la transmission et le stockage des informations ont été effectués en utilisant des documents papier. Durant les vingt dernières années, la création de documents assistée par ordinateur n'a pas cessé d'augmenter ; cette capacité de production des documents numériques, couplée à des moyens de transmission de l'information sans cesse croissants, aurait dû permettre une utilisation moindre des documents papier. Cela ne semble toutefois pas être le cas. Ils sont toujours imprimés, pour une diffusion au plus grand nombre, pour une lecture souvent plus commode lorsque la quantité d'informations est importante, ou bien encore lorsqu'il s'agit de rapporter la preuve de l'existence d'un contrat. Bien qu'une grande partie de l'information soit désormais numérique, les formidables outils que sont les ordinateurs permettent, dans le même temps, et paradoxalement, d'augmenter considérablement la production de documents papier.

En effet, les nouvelles technologies de l'information, loin de faire éclore la "société sans papier" qu'elles promettaient ont, au contraire, accru dans des proportions significatives le tonnage de papier consommé. L'objectif, du début des années 1980, d'un bureau sans papier a évolué : le traitement du flux des données numériques et des documents papier doit s'effectuer d'une manière efficace et intégrée. La solution ultime serait qu'un ordinateur soit capable de traiter avec autant d'efficacité un document papier qu'il est capable de le faire avec les autres médias numériques ; les documents papier devraient pouvoir être lus et interprétés comme un ordinateur est capable de le faire, avec un disque magnétique ou optique. Ainsi, les documents papier pourraient avantageusement être lus à la fois par l'homme et la

machine, ce qui n'est pas le cas des médias informatiques actuels.

L'objectif de l'analyse et de la reconnaissance de documents est l'identification du texte, des graphiques ou notations composant l'image, ainsi que l'extraction de l'information, à la manière d'un lecteur humain. On peut distinguer deux grands ensembles au sein de la reconnaissance de documents : l'analyse du texte et l'analyse des composants graphiques (images, schémas, notations, plans, etc.). Les différents éléments constituant le document sont identifiés et peuvent ainsi être analysés avec les meilleures techniques possibles. L'analyse textuelle s'emploie à la reconnaissance du texte. Les tâches concernées sont : la reconnaissance des caractères, la détermination d'un éventuel angle de rotation, la reconnaissance de la structure du texte en colonnes, paragraphes, lignes de textes et mots. L'analyse graphique s'occupe, quant à elle, de la reconnaissance des lignes, des schémas, des symboles constituant un diagramme, de l'identification des logos d'entreprise, etc.

Depuis une trentaine d'années, de nombreuses recherches ont été effectuées dans le cadre de l'analyse et de la reconnaissance de documents. Ces recherches concernent des domaines aussi variés que la reconnaissance de formes, de l'écriture manuscrite, de la structure de documents, de classifications, etc. Ces résultats obtenus ont conduit à la mise au point d'algorithmes et de logiciels d'analyse de documents ; il existe ainsi aujourd'hui de nombreux systèmes capables de convertir un document papier en un document électronique. Dans le domaine de la bureautique, les lecteurs optiques de caractères couplés à des logiciels comme *OmniPage* ou *TextBridge* permettent la reconnaissance de textes imprimés. D'autres systèmes plus spécialisés analysent des documents spécifiques : les services bancaires utilisent des lecteurs de chèques, ou bien encore les services postaux qui automatisent le tri du courrier par lecture du code postal.

Les systèmes actuels de traitement de documents se scindent en deux grandes classes : les applications spécifiques (lecture de plans architecturaux, lecture automatique de journaux et de revues, lecture de bibliographies, interprétation de vues aériennes, etc.), et les systèmes plus généraux. Les premières sont conçues en tenant compte du type de document traité et l'exploitent mieux grâce à une connaissance approfondie du domaine. Les secondes, plus généralistes, ont pour objectif de pouvoir traiter un maximum de documents rencontrés sans utiliser trop de connaissances spécifiques.

Même si ces derniers logiciels sont assez complets (détection des paragraphes, des tableaux, identification des zones contenant des images, des schémas, etc.), ils ignorent complètement certains types ou composants de documents. Leur analyse est laissée à d'autres sous-systèmes, plus spécialisés.

C'est le cas notamment des notations mathématiques. Les expressions ou notations mathématiques sont pourtant présentes dans la plupart des articles, revues ou livres de mathématique, physique, astronomie, économie, etc. Elles renferment une partie importante de

la connaissance et, n'étant pas analysées, le lecteur perd une grande part de l'information disponible. Une interprétation par un ordinateur des notations mathématiques, issues de documents papier, permettrait aussi leur réutilisation directe. Cela éviterait de devoir saisir manuellement la formule pour quelle soit manipulable par un système de calcul.

1 Nos buts initiaux

L'objectif principal de ce travail est l'étude et la conception d'une méthode adaptée à la reconnaissance et à l'interprétation des expressions mathématiques. L'un des principaux problèmes soulevés est la reconnaissance de la structure de la formule, en d'autres termes comment reconnaître la structure de la notation mathématique à partir de sa forme géométrique (des éléments qui la composent et de leurs positions).

Une étude des approches existantes nous a permis d'identifier les points importants d'un tel système. La méthode choisie doit tout d'abord permettre la reconnaissance la plus large possible des différentes notations mathématiques. En effet, plusieurs recherches ont été effectuées sur une sous-classe d'expressions, comme les équations ou les intégrales. Ces approches, trop restrictives, ne permettent pas la reconnaissance d'autres types de notations, comme les notations matricielles. Ensuite, la variété des notations mathématiques implique qu'il est impossible d'avoir une connaissance a priori de tous les types de notations. Il est donc nécessaire de pouvoir adapter le système à des notations jusqu'alors inconnues. Enfin, pour un même type de notations, les différences typographiques peuvent être importantes ; il est donc indispensable que le système soit également capable de s'y adapter.

Notre objectif est de concevoir un système capable d'interpréter des notations mathématiques à partir de leur représentation graphique. Ces recherches sont motivées pas les différentes applications qu'un tel système permettrait de mettre en œuvre. Nous allons présenter maintenant quelques unes des applications possibles.

2 Motivations et applications possibles

Une quantité croissante de logiciels manipulent des formules mathématiques. C'est tout particulièrement le cas des systèmes de calcul formel. Durant les vingt dernières années, le calcul formel s'est progressivement imposé comme un outil précieux pour le scientifique, qu'il soit mathématicien ou ingénieur. Des systèmes aussi généralistes que *Maple* ou *Mathematica* jouent aujourd'hui un rôle déterminant dans des domaines aussi variés que les mathématiques, la chimie, la physique, l'économie, etc. Les systèmes de calcul formel se présentent, pour la plupart, comme des programmes interactifs permettant d'effectuer des calculs sur des objets mathématiques symboliques (des variables). À l'aide de ces systèmes,

il est par exemple possible de calculer les solutions formelles de l'équation du second degré $ax^2 + bx + c = 0$ en fonction des paramètres a, b et c. Ce calcul est dit "formel" car les valeurs des paramètres a, b et c ne sont pas précisées. Depuis leur apparition dans les années soixante, et au fil des versions, les systèmes de calcul formel ont beaucoup gagné en puissance et en efficacité. La nette progression de la qualité des algorithmes employés et de leur implémentation permet d'augmenter la taille des problèmes traités, tout en réduisant le temps nécessaire à leur résolution. Cependant, dans ces différents systèmes, l'interface a peu évolué. En matière d'édition de formules, la saisie des expressions s'effectue de manière linéaire, dans une syntaxe souvent lourde et sans rapport avec le sens mathématique des objets décrits. Néanmoins, même si des recherches ont été effectuées dans le domaine de l'interface homme-machine pour la saisie d'objets mathématiques (avec utilisation du clavier et de la souris), l'interface la plus intuitive pour la saisie d'expressions mathématiques reste l'écriture. Elle est la plus naturelle. Un système de reconnaissance de la structure des formules mathématiques permettrait une saisie beaucoup plus intuitive pour l'utilisateur occasionnel.

Nombreuses sont les connaissances mathématiques potentiellement très utiles aux systèmes de calculs formels. Mais la plupart du temps, elles ne sont pas disponibles sous forme électronique, seulement sur papier. Elles sont regroupées dans des livres de mathématique, mais aussi de physique, de mécanique, etc. Des outils comme les bases déductives de formules se proposent de rendre consultables de telles grbases de connaissances pour différents types d'applications, comme un recueil électronique de formules mathématiques ou bien encore une base de données de lemmes pour un système de preuves automatiques. À l'heure actuelle, le seul moyen d'introduire des données dans ces bases est la saisie manuelle. L'automatisation de cette tâche, peu gratifiante, grâce à un outil d'extraction et de reconnaissance des formules mathématiques serait un atout.

Les systèmes actuels d'analyse et de reconnaissance de documents ne prennent pas en compte les notations mathématiques, bien qu'ils soient capables de les identifier en tant que telles. L'extension des systèmes existants permettrait d'appliquer ces outils à des articles scientifiques avec plus de succès. Une dernière application, celle-ci plus anecdotique, autoriserait de faire un copier/coller entre un logiciel permettant de visualiser du *PostScript* et un système de calcul formel par exemple, ou un éditeur de texte. Cette fonctionnalité permettrait à un utilisateur d'intégrer une expression mathématique prise dans un article à son propre document, sans avoir à effectuer une nouvelle fois la saisie.

Ces quelques exemples montrent bien l'utilité d'un système de reconnaissance structurelle des notations mathématiques. Plusieurs méthodes ont déjà été élaborées dans ce domaine, avec plus ou moins de succès.

3 Objectifs de l'étude

Les systèmes de conversion et d'analyse doivent traiter les documents papier depuis leur format d'origine, c'est à dire la feuille de papier, jusqu'à l'interprétation complète de leur contenu. Ceci conduit à concevoir des systèmes capables d'intégrer et de faire coopérer plusieurs systèmes spécialisés. Les composants intervenant sont aussi divers que :

- un scanner pour la transformation du document papier en une forme numérique exploitable par un ordinateur,
- un module de reconnaissance des symboles entrant dans la composition d'une notation mathématique,
- ou encore un module d'analyse et d'interprétation de la structure d'un objet mathématique à partir de ses composantes géométriques.

L'objectif de cette thèse est de concevoir et de mettre au point une méthode et un système qui soient capables de reconnaître la forme structurelle des notations mathématiques. Ce système doit permettre une adaptation rapide aux différents types de notations. Il doit donc être suffisamment généraliste pour remplir cette tâche, à l'opposé des systèmes figés. L'approche que nous avons choisie est basée sur les grammaires de graphes contextuelles attribuées. Ce type de grammaire fournit un formalisme très intéressant de description des manipulations structurelles multidimensionnelles, suffisamment puissant pour s'adapter aux différentes exigences d'une telle reconnaissance.

4 Résultats obtenus

Un composant logiciel paramétrable assurant la reconnaissance de formules mathématiques, baptisé *OFR* (*Optical Formula Recognition*) a été développé et expérimenté au sein d'une application : *Irma* (*Interface pour le Reconnaissance de notations MAthématiques*). C'est cette application qui nous a servi de plate-forme d'expérimentation pour la validation de nos résultats. L'architecture logicielle mise en place a permis de tester plusieurs types de données, à savoir plusieurs systèmes de reconnaissance de caractères, aussi bien typographiés que manuscrits.

4.1 Le composant OFR

La conception et la réalisation d'*OFR* constituent l'essentiel de notre travail. *OFR* est un composant logiciel paramétrable de reconnaissance structurelle des formules mathématiques. Nous avons utilisé une grammaire de graphes contextuelle attribuée pour la reconnaissance syntaxique des notations. Ce choix nous autorise une grande paramétrabilité et une adaptation rapide à la reconnaissance de nouvelles notations.

Étant donné un ensemble de symboles reconnus et leurs positions, notre système de reconnaissance structurelle fait une analyse spatiale et syntaxique de ces objets, afin de créer

l'arbre de syntaxe qui représente la notation mathématique. Cet arbre de syntaxe est ensuite transformé selon le protocole *OpenMath*, afin de pouvoir transmettre cette expression à d'autres systèmes qui pourront l'évaluer, l'éditer ou lui appliquer le traitement désiré.

4.2 Irma : une application

Pour valider *OFR*, nous avons développé une application basée sur ce composant ; elle comporte plusieurs modules. En amont du système de reconnaissance structurelle, nous avons besoin d'un logiciel capable d'extraire et d'identifier les éléments qui composent la notation. Afin de vérifier notre méthode de reconnaissance structurelle, nous avions pour objectif deux applications majeures : la reconnaissance manuscrite et la reconnaissance typographiée. Nous avons fait plusieurs recherches sur les logiciels existants dans ces deux domaines.

En ce qui concerne la reconnaissance de caractères typographiés, plusieurs logiciels existants pouvaient a priori répondre à nos besoins. Mais après une étude poussée, il s'est avéré qu'aucun d'entre eux ne répondait réellement à nos attentes. Pour la plupart d'entre eux, nous ne disposions pas du code source pour effectuer une quelconque adaptation à notre cas particulier. Ils ne considèrent pas les symboles mathématiques, et rares sont ceux qui reconnaissent l'alphabet grec usité dans les formules mathématiques. Dans le cas où nous disposions du code original, il était très difficile, voir impossible, d'extraire uniquement le moteur de reconnaissance des caractères, ou tout simplement de faire apprendre une nouvelles police de caractères. Nous avons donc développé un petit logiciel capable de répondre à nos attentes.

Concernant la reconnaissance manuscrite, nous avons collaboré avec une université allemande qui nous a fournit son logiciel de reconnaissance de symboles manuscrits. Une telle coopération nous a évité de développer un logiciel dans un domaine où nous n'étions pas experts.

Une interface sert de lien aux différents modules de reconnaissance des caractères et de la structure : c'est l'application *Irma*. Nous détaillerons plus amplement ces réalisations dans le chapitre IV, page 99.

5 Plan de lecture

Ce manuscrit se compose de cinq chapitres, d'une conclusion, d'appendices, d'une bibliographie et d'un index.

Après une présentation rapide des différentes applications possibles d'une recherche sur la reconnaissance structurelle des formules mathématiques, nous présenterons dans le chapitre I, page 9, les difficultés rencontrées par rapport aux autres domaines de la reconnaissance de documents. Nous donnerons également dans ce chapitre un aperçu des différentes approches déjà utilisées, ainsi qu'une critique objective de celles-ci. Cela nous permettra de dégager les éléments principaux d'un outil de reconnaissance structurelle d'objets mathématiques.

Le chapitre II, page 53, présente rapidement les principales caractéristiques que doit renfermer un logiciel de reconnaissance d'expressions mathématiques. Cet aperçu peut servir de base à la lecture des chapitres suivants.

L'architecture ainsi que la méthode mise en œuvre sont décrites dans le chapitre III, page 61. Ce chapitre propose, par exemple, quelques réflexions sur l'utilisation des grammaires de graphes contextuelles et attribuées.

Le chapitre IV, page 99, présente les applications réalisées ainsi que les tests de validation effectués.

Enfin, le chapitre V, page 129, présente l'implémentation du système de reconnaissance structurelle développé : *OFR*.

6 Conventions typographiques

Afin de faciliter la lecture de ce document, un certain nombre de conventions typographiques ont été utilisées :

- Les caractères gras mettent en évidence la **définition** d'un terme.
- Les caractères italiques indiquent les *noms propres*.
- les caractères obliques entre parenthèses désignent les termes anglais (*english terms*).
- Les caractères sans serif désignent du code informatique ou des identificateurs.

Chapitre I

Reconnaissance structurelle de formules mathématiques : état de l'art

L'écriture a été développée et est aujourd'hui utilisée afin de satisfaire deux besoins essentiels : la conservation de la connaissance ainsi que la communication et la diffusion du savoir au plus grand nombre. Durant plusieurs siècles, différentes techniques ont été mises au point et améliorées : l'écriture, mais aussi les schémas ou certains types de notations bidimensionnelles, comme les mathématiques. Ce type de notation a été développé dans le but de pouvoir concentrer des idées et d'avoir des représentations mathématiques visualisables.

L'imprimerie a contribué à l'accroissement du volume de la production des écrits, imposant une organisation documentaire. Plus récemment, l'apparition de l'informatique a quelque peu bouleversé notre rapport avec les documents écrits. Il est maintenant naturel de consulter des bibliothèques de documents électroniques constituées à partir de documents papier. Toutefois, cette possibilité se limite actuellement aux textes et les autres types de notations en sont exclus la plupart du temps.

Il paraît pourtant naturel et commode d'utiliser les mêmes notations dans notre communication avec les ordinateurs. Mais cela implique de devoir convertir notre méthode de représentation usuelle en données compréhensibles et manipulables par une machine. Les

systèmes informatiques actuels permettent de générer des notations mathématiques bidimensionnelles, mais aucun outil n'est actuellement capable de reconnaître une notation mathématique. La seule méthode permettant pour l'instant d'introduire ce type de données est le recours à une intervention humaine. Toutefois, la spectaculaire progression de la puissance des ordinateurs et les développements de la micro-informatique, tant sur le plan matériel que logiciel, permettent d'envisager un système capable de reconnaître et d'interpréter une notation mathématique telle que nous en utilisons couramment.

Contrairement à un texte, une notation mathématique utilise un arrangement bidimensionnel de symboles pour transmettre l'information. Être capable de comprendre et d'interpréter ce type d'expression implique deux problèmes distincts : la reconnaissance des symboles qui composent la formule et l'analyse de l'agencement de ces symboles.

Ce chapitre contient un bref aperçu historique des différentes approches utilisées jusqu'à présent pour tenter de résoudre cette difficulté. Ces réflexions sont principalement le fruit de la lecture d'ouvrages, comme [**?**][1] ou [**?**][2], aucune application implémentant les principes présentés n'ayant pu être obtenue.

1 Historique

L'analyse et la reconnaissance de documents [**?**][3] [**?**][4] [**?**][5] [**?**][6] [**?**][7] tendent à devenir des thèmes majeurs de la recherche depuis plus d'une dizaine d'années. Ces recherches ont donné lieu à des publications si nombreuses qu'il est devenu nécessaire de créer des revues (IJDAR (*International Journal on Document Analysis en Recognition*) créé en 1998) et des conférences spécialisées (comme ICDAR (*International Conference on Document Analysis and Recognition*) créée en 1991). De nombreuses applications se sont développées afin de répondre au besoin croissant de tels outils dans des domaines aussi variés que :

- le tri automatique du courrier,
- la reconnaissance automatique de formulaires,
- la rétro-conversion de références bibliographiques [**?**][8],

[1] ANDERSON R.H., Two dimensional mathematical notation (1977).

[2] BLOSTEIN D. & GRBAVEC A., Recognition of mathematical notation (1997).

[3] KUBOTA K., IWAKI O. & ARAKAWA H., Document understanding system (1984).

[4] CHENEVOY Y., *Reconnaissance structurelle de documents imprimés : études et réalisations.*, PhD Thesis (1992).

[5] CHENEVOY Y. & BELAÏD A., Low-level structural recognition of documents (1994).

[6] LORETTE G., Le traitement automatique de l'écrit et du document.état de la recherche (1996).

[7] BELAÏD A., CHENEVOY Y. & PARMENTIER F., Panorama des méthodes structurelles en analyse et reconnaissance de documents (1997).

[8] BELAÏD A. & CHENEVOY Y., Document analysis for retrospective conversion of library reference catalogues (1997).

- la reconnaissance de notations schématiques [**?**][9] comme :
  - la lecture de partitions musicales,
  - l'analyse et l'interprétation de plans architecturaux [**?**][1] [**?**][2],
- la compréhension de documents techniques ou scientifiques comme :
  - les schémas de circuits électroniques,
  - le dessin technique [**?**][3],
  - les cartes routières,
  - etc.

L'analyse et la reconnaissance de documents font intervenir plusieurs disciplines : le traitement d'images, la reconnaissance de formes (formes géométriques, caractères, symboles, etc) et la reconnaissance de la structure. Cette dernière nécessite l'utilisation de diverses méthodes comme l'analyse syntaxique, sémantique, statistique, ou bien encore des modèles hybrides, faisant intervenir deux ou plusieurs méthodes simultanément ou de manière concurrente.

L'intérêt des chercheurs envers la reconnaissance des notations mathématiques dans un document est ancien ; dès 1967 les premiers travaux sur le sujet apparaissent : R. Anderson [**?**][4], mais aussi W. A. Martin (1967), H. R. Lewis (1968) ou encore S. K. Chang [**?**][5]. L'engouement pour la reconnaissance de l'écriture manuscrite appliquée aux formules mathématiques motive ces différents travaux. Néanmoins, les performances des ordinateurs de l'époque ne permettent pas d'envisager une utilisation concrète de ces différentes méthodes. Plusieurs autres recherches en découleront dans les années suivantes ; elles seront détaillées dans les paragraphes suivants (paragraphe sur la reconnaissance structurelle des formules mathématiques 7).

Ces différentes recherches ont toutes un but similaire. Nous allons essayer de présenter les applications envisageables d'un système permettant la reconnaissance de la structure d'un objet mathématique à partir de sa représentation bidimensionnelle.

2 De nombreuses applications possibles

Plusieurs réalisations peuvent être envisagées dès lors que l'on dispose d'un outil pour la reconnaissance de la structure d'une expression mathématique. Mais une des principales ap-

[9] BLOSTREIN D., General diagram-recognition methodologies (1995).

[1] DORI D. & TOMBRE K., Paper drawings to 3d cad : A proposed agenda (1993).

[2] TOMBRE K., Analysis of engineering drawing : State of the art and challenges (1998).

[3] COLLIN S., *Interprétation de la Cotation des Dessins Techniques par Analyse Syntaxique*, PhD Thesis (1992).

[4] ANDERSON R.H., Syntax-directed recognition and hand-printed two-dimensional mathematics (1967).

[5] CHANG S.K., A method for the structural analysis of two-dimensional mathematical expressions (1970).

plications reste l'édition de formules mathématiques. Ce problème est rencontré chaque fois qu'un logiciel manipule des objets mathématiques ; c'est le cas, par exemple, des systèmes de calcul symbolique, ou bien encore des logiciels pour l'édition de documents scientifiques.

2.1 Édition de formules mathématiques

La spectaculaire progression de la puissance des ordinateurs et les développements de la micro informatique permettent à un nombre de plus en plus important d'utilisateurs de bénéficier des possibilités du calcul symbolique. Ainsi, après le monde de la recherche, ce sont entre autres, les ingénieurs, les étudiants ou les économistes qui réalisent l'intérêt de ces outils. Ces nouveaux utilisateurs sont le plus souvent occasionnels. Le faible temps qu'ils consacrent à l'emploi de ces systèmes ne leur permet pas d'investir beaucoup d'énergie dans l'apprentissage des interfaces. Elles doivent donc être le plus intuitives possibles. Pour le moment, la qualité générale des interfaces aux systèmes de calcul symbolique n'est pas satisfaisante. Les efforts ont en effet longtemps porté sur l'amélioration des moteurs de calcul ou l'enrichissement des bibliothèques de fonctions plus que sur l'interface. Aujourd'hui, il convient de reconsidérer ce point pour proposer des interfaces qui soient à la fois plus simple d'emploi et directement accessibles aux débutants [**?**][1].

Nous avons eu l'occasion de tester, au moins dans des versions de démonstration, un certain nombre de logiciels permettant l'édition de formules mathématiques. Nous ne nous sommes pas cantonnés aux interfaces spécifiques dédiées au calcul formel [**?**][2], car des solutions intéressantes sont apparues notamment dans le domaine de l'édition de documents mathématiques. Les logiciels intégrant une édition pour les formules mathématiques sont nombreux. Nous n'en avons retenus que quelques uns, les plus caractéristiques présentant des techniques différentes.

2.1.1 Syntaxe linéaire

Historiquement, c'est la première méthode utilisée lorsque l'on ne disposait pas encore de terminaux graphiques. La saisie de la formule s'effectue à l'aide d'une syntaxe linéaire basée sur des mots clés qui désignent les opérateurs. Cette approche s'apparente plutôt à un langage permettant de décrire les opérateurs constituant l'expression. La saisie est assez rapide car elle ne fait intervenir que le clavier. Mais elle nécessite de connaître la syntaxe des commandes pour les différents opérateurs. Cette méthode n'est donc pas du tout adaptée à un utilisateur occasionnel qui devra se documenter afin de pouvoir saisir une expression, même très simple. Malgré ses désavantages, cette syntaxe linéaire est toujours assez large-

[1] MARTIN G., PITTMAN J., WITTENBURG K., COHEN R. & PARISH T., Computing without keyboards (1990).

[2] KAJLER N. & SOIFFER N., A survey of user interfaces for computer algebra systems (1998).

ment utilisée.

LaTeX ([**?**][1]) est un système de formatage de texte principalement développé par Leslie Lamport. Il s'appuie sur les travaux de Donald Knuth ([**?**][2], [**?**][3]) qui, pour sa part, à mis au point TeX. LaTeX est le système le plus couramment employé dans de nombreuses disciplines scientifiques pour la publication de documents contenant des expressions mathématiques. La syntaxe utilisée est une syntaxe linéaire et il est donc nécessaire de connaître les commandes et leur syntaxe. Pour des formules plus importantes, il devient rapidement difficile d'éditer celles-ci sans erreurs, et il est quasiment impossible de les relire. Pour preuve, voici la syntaxe utilisée pour la formule suivante :

$$V = 4 \int_{x=0}^{R} \int_{y=0}^{\sqrt{R^2-x^2}} \int_{z=\frac{H}{R}\sqrt{x^2+y^2}}^{H} dzdydx = \frac{1}{3}\pi R^2 H$$

```
\[V=4\int_{x=0}^{R}{\int_{y=0}^{\sqrt{R^{2}-x^{2}}}
  {\int_{z=\frac{H}{R}\sqrt{x^{2}+y^{2}}}^{H}{dzdydx}}}
  =\frac{1}{3}\pi R^{2}H
```

Maple [**?**][4], le célèbre système de calcul formel, utilise aussi ce type de saisie, même dans la dernière version du logiciel (version V.5) que nous avons pu tester. Même si celui-ci affiche comme résultat la forme bidimensionnelle de la formule, ce qui en facilite la relecture, on peut signaler qu'en cas d'erreur dans la saisie linéaire de la formule, il est difficile de la localiser et de la corriger.

2.1.2 Palette de modèles

L'utilisation d'une palette de modèles nécessite de sélectionner, dans une barre d'outils ou un menu, un modèle de construction mathématique correspondant au souhait de l'utilisateur.

Tous les éditeurs à base de modèles sont semblables. Les opérateurs basiques comme l'addition ou la multiplication, ainsi que les lettres et chiffres, peuvent être directement saisis en sélectionnant la touche appropriée sur le clavier. Pour les opérateurs ou les symboles qui ne sont pas définis sur un clavier "classique", ou bien pour les opérateurs de nature bidimensionnelle, comme les intégrales, les racines carrées, les fractions, etc., l'utilisateur sélectionne, le plus souvent à l'aide de la souris, l'élément désiré dans une palette. Le modèle est alors inséré dans la formule en cours d'édition. Les figures suivantes présentent deux

[1] LAMPORT L., *LaTeX : a document preparation system* (1994).

[2] KNUTH D.E., *The Tex Book* (1993).

[3] KNUTH D.E., Mathematical typography (1979).

[4] CHAR B.W., GEDDES K.O., GONNET G.H., MONAGAN M.B. & WATT S., *Maple Reference Manual* (1988).

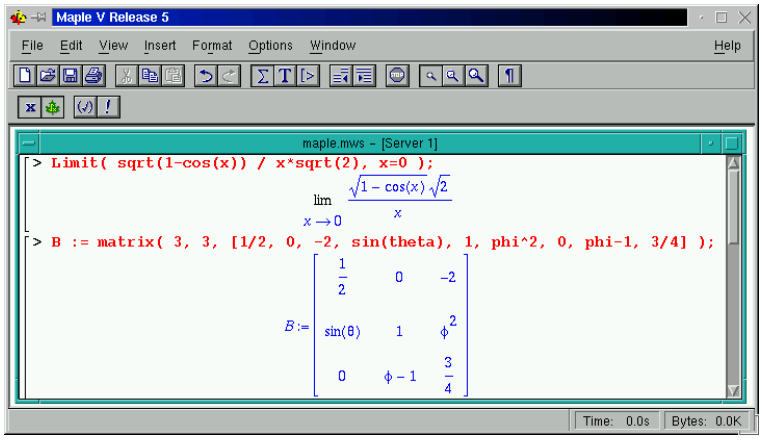
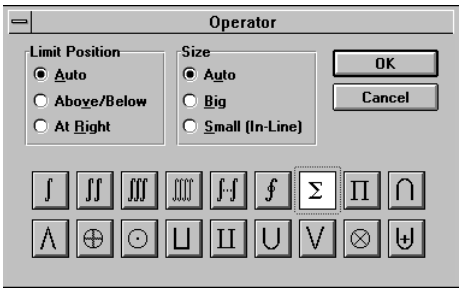
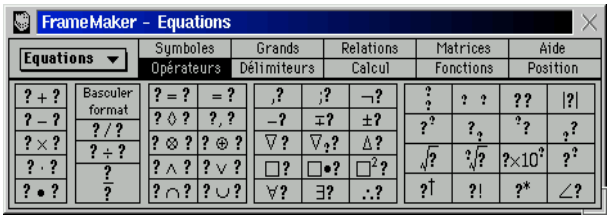


FIG. 1 – Édition linéaire dans Maple V.5

exemples de palette de modèles. Elles sont issues de *Scientific Workplace* [**?**][1], un éditeur de document scientifiques, et de *Frame Maker* de *Adobe* [**?**][2], un traitement de texte permettant l'édition de formules mathématiques.



(a) Scientific Workplace 3.0



(b) Frame Maker 5.5

FIG. 2 – Édition par palette de modèles

Cette technique, même si elle peut paraître préférable à une syntaxe linéaire, n'est cependant pas acceptable pour la saisie d'une formule par un utilisateur novice. En effet, il reste difficile de trouver l'opérateur souhaité au milieu d'une multitude de formes structurelles et de symboles spéciaux. Cette remarque, même si elle doit être nuancée pour un utilisateur averti, reste toutefois pertinente. Ce choix de l'utilisation commune de la souris et du clavier simultanément pour l'édition d'une expression n'est pas ergonomique et induit un surcoût important de temps de saisie, dû aux déplacements des mains d'un périphérique à l'autre.

[1] RESEARCH T.S., Scientific workplace (1999).

[2] *Adobe FrameMaker 5.5 Classroom in a Book* (1997).

2.1.3 Édition bidimensionnelle

D'autres recherches pour une édition bidimensionnelle des formules mathématiques sont à signaler [**?**][1]. Nous retiendrons entre autres, les travaux d'Olivier Arsac [**?**][2] [**?**][3] sur *Emath*, un serveur pour l'édition et la manipulation de formules mathématiques. Plusieurs modes d'édition sont proposés suivant les différentes formes de l'expression que l'on souhaite saisir. Si l'utilisateur veut saisir le polynôme $2x^9 + 4x^7 + 6x^5 + 8x^3$, la saisie s'effectuera de manière simplifiée : $2\ x\ 9 + 4\ x\ 7 + ...$ Cette adaptabilité permet de minimiser le nombre de touches frappées sans pour autant utiliser des commandes spécifiques. Toutefois, ces modes spéciaux d'édition ne sont pas toujours aussi bien adaptés que sur l'exemple précédent. Il faut alors avoir recours à la souris ou à des commandes plus complexes que seul un utilisateur averti pourra connaître.

2.1.4 Des modes d'édition "coûteux"

La plupart des interfaces traditionnelles nécessitent donc un apprentissage plus ou moins poussé de commandes, ce qui représente un investissement assez important en temps de formation pour l'utilisateur.

La solution de la palette de modèles qui, elle, ne nécessite aucune formation particulière, n'est pas efficace : elle contraint l'utilisateur à avoir recours à la fois à la souris et au clavier, ce qui induit un surcoût important du temps de saisie.

Pour preuve, les travaux réalisés par Stuart K. Card sur les connaissances issues de la psychologie expérimentale [**?**][4], [**?**][5]. Son but était de dégager les données utiles pour effectuer des choix de conception dans les interfaces hommes-machines. Une application intéressante de cette approche est le modèle des actions de manipulation (*keystroke-level model*) [**?**][6] qui permet d'évaluer le temps nécessaire à la réalisation de certaines tâches. Cela consiste d'abord à détailler les opérations élémentaires nécessaires à la réalisation de la tâche pour ensuite à additionner les temps nécessaires à chaque opération. Une table indique la durée de chaque opération :

---

[1] LAVIROTTE S., Interface pour le calcul formel, Technical Report (1995).

[2] ARSAC O., *Interfaces Homme Machine pour le Calcul Formel*, PhD Thesis (1997).

[3] ARSAC O., DALMAS S. & GAËTANO M., The design of a customizable component to display and edit formulas (1999).

[4] CARD S.K., ENGLISH W. & BURR B., Evaluation of mouse, rate-controlled isometric joystick, step keys and text keys for text selection on a crt (1978).

[5] CARD S.K., Human limits and vdt computer interface (1984).

[6] CARD S.K., MORAN T.P. & NEWELL A., The keystroke level model for user performance time with interactive systems (1980).

Appui d'une touche	T_K=0,30s [0,08 à 1,20]
Pointer une cible à la souris	T_P=1,10s [0,8 à 1,5]
Passer du clavier à la souris	T_H=0,40s
Tracer *n* traits de longueur totale *l* cm	$T_D(n,l)$=0,9 n + 0,10 l
Préparer mentalement une action	T_M=1,35s
Réponse du système	T_R

Cependant, le temps n'est qu'un aspect de l'interface, d'autres caractéristiques doivent être prises en compte lors de la conception : la fréquence des erreurs, la difficulté d'apprentissage et de mémorisation, la proportion des fonctionnalités réellement utilisées.

2.1.5 Vers une édition manuscrite

Tous ces facteurs laissent à penser que la meilleure interface pour l'édition d'expressions mathématiques, la plus intuitive pour l'utilisateur novice et la plus rapide et pratique pour un utilisateur expérimenté, reste une interface du type papier crayon [**?**][1]. C'est en effet la méthode qui nous est la plus naturelle.

Un exemple d'utilisation de ce type d'interface pour la saisie d'expressions mathématiques est le système développé par Richard H. Littin [**?**][2]. Il est possible d'avoir recours soit à la souris soit à une tablette graphique et à un stylo. Les caractères sont reconnus au moment du tracé. La méthode utilisée pour la reconnaissance structurelle est basée sur une grammaire LR. Le résultat produit par le système est une expression sous forme linéaire utilisant une notation préfixée, à la manière de Lisp. Mais le système impose plusieurs contraintes à l'utilisateur : l'ordre dans lequel les symboles doivent être saisis, l'édition où la modification de symboles est limitée à la correction du dernier caractère. Toutes ces contraintes impliquent qu'une utilisation concrète de ce système s'avère presque impossible.

Un autre système se servant d'une saisie manuscrite pour les formules mathématiques est celui de R. Fukuda [**?**][3]. Lui aussi effectue une reconnaissance immédiate des symboles ainsi que de leur position. L'utilisateur doit tout d'abord faire une pause après l'écriture de chaque symbole et vérifier que la reconnaissance est bien conforme à ce qu'il souhaitait saisir. Dans le cas contraire, il doit corriger l'erreur avant la saisie de la suite de la notation. Cette approche, pour l'avoir testée, semble trop pénalisante d'un point de vue ergonomique pour être concrètement utilisable dans le cadre d'une application.

---

[1] NAKAGAWA M., KATO N., MACHII K. & SOUYA T., Principles of pen interface design for creative work (1993).

[2] LITTIN R.H., *Mathematical Expression Recognition : Parsing Pen/Tablet Input in Real-Time Using LR Techniques*, Master's thesis (1995).

[3] FUKUDA R., SOU I., TAMARI F., MING X. & SUZUKI M., A technique of mathematical expression structure analysis for the handwriting input system (1999).

Comparé à d'autres méthodes pour la saisie de formules mathématiques, les systèmes basés sur l'écriture présentent l'avantage, s'ils sont bien utilisés, d'être plus naturels et plus intuitifs à l'utilisation. C'est ce qui ressort d'une étude faite par S. Smithies [**?**][1]. Le désavantage de ce type de système est qu'il doit être capable de s'adapter aux différents styles d'écriture, et aux imperfections de l'écriture manuscrite (alignement des symboles approximatif, taille des symboles moins significative que dans la cas typographié). Le système doit aussi permettre la saisie la plus naturelle possible pour l'utilisateur, afin d'en faciliter l'utilisation par un large public. Mais plusieurs approches étudiées imposent un ordre d'écriture des symboles afin de faciliter la phase d'analyse structurelle. C'est par exemple le cas des systèmes présentés par S. Smithies [**?**][1] ou A. Kosmala [**?**][2].

Des systèmes comme celui de S. Smithies permettent une saisie des symboles à l'aide de la souris, mais l'écriture et donc les résultats sont de moins bonne qualité qu'avec une tablette à digitaliser. Un des obstacles au développement de telles interfaces pourrait donc être la nécessité de disposer d'une tablette graphique. Néanmoins, le faible coût de ces périphériques permet d'envisager une démocratisation de ce type d'interface pour la saisie des expressions mathématiques. Un outil pour la reconnaissance structurelle d'une formule à partir de sa représentation rendrait possible un tel mode de saisie, plus ergonomique et habituel pour toutes les catégories d'utilisateurs. Nous détaillerons dans le chapitre II les différents éléments que doit intégrer un tel système pour être concrètement utilisable.

2.2 Bases de formules

D'autres applications sont envisageables dans un tout autre domaine que celui de la saisie interactive. Il s'agit d'exploiter la possibilité de réutiliser la masse d'informations mathématiques répartie dans une littérature scientifique abondante. Des outils comme les bases déductives de formules [**?**][3] ou les tables d'intégrales [**?**][4] se proposent de mettre leur connaissance à disposition des systèmes de calcul symboliques ou des systèmes de preuves automatiques.

Le problème majeur est la saisie des expressions qui composent ces bases. Le seul moyen à l'heure actuelle est la saisie manuelle. L'automatisation de cette tâche fastidieuse serait un atout et contribuerait au développement de tels systèmes. Une application possible à la reconnaissance d'expressions mathématiques est donc l'extraction et la compréhension des formules qui composent les recueils scientifiques édités depuis de nombreuses années, et qui ne sont pas disponibles sous forme électronique.

[1] SMITHIES S., *Freehand Formula Entry System*, Master's thesis (1999).

[2] KOSMALA A. & RIGOLL G., Recognition of on-line handwritten formulas (1998).

[3] DALMAS S., GAËTANO M. & HUCHET C., A deductive database for mathematical formulas (1996).

[4] EINWOHNER T.H. & FATEMAN R.J., Searching techniques for integral tables (1995).

2.3 Extension des systèmes de reconnaissance de documents

Les logiciels actuels d'analyse et de reconnaissance de documents ne traitent pas le problème des notations mathématiques. Ils utilisent les résultats de plusieurs recherches sur la segmentation et l'étiquetage des éléments composant un document [**?**][1], [**?**][2], [**?**][3], [**?**][4] et sont donc capables de les identifier. Mais aucun des logiciels ne traite ces données pour en extraire la représentation syntaxique de l'expression. Dans le cas de l'analyse et de la reconnaissance d'un article scientifique, c'est la partie principale du document qui n'est par traitée, perdant ainsi l'avantage d'utiliser ce type de logiciels. Une extension de ces systèmes, issus de la recherche ou commerciaux, afin qu'ils prennent en compte les notations mathématiques, serait tout à fait souhaitable dans le cadre de la reconnaissance d'articles scientifiques.

2.4 Diverses autres applications

Avec l'explosion de la communication numérique, grâce en particulier à Internet, la quantité de données consultables dans un domaine scientifique est sans cesse croissante. Beaucoup d'outils pour l'indexation automatique de textes, mais aussi d'images, de sons ou bien encore de vidéos, sont apparus. Toutefois, la possibilité d'effectuer ce type de recherches n'existe pas dans le cadre des formules mathématiques. Des standards comme MathML permettent la représentation des mathématiques en HTML. Pour rendre disponible et consultable à distance l'abondance de données scientifiques, un système d'indexation de documents est nécessaire, ainsi que pour pouvoir bénéficier des connaissances fournies sous la forme d'expressions mathématiques. Un tel système découlerait directement de l'extension des systèmes pour la reconnaissance de documents scientifiques.

Quelques applications plus anecdotiques peuvent être mentionnées comme la possibilité de faire un copier/coller entre un visualisateur de document et un éditeur (pour les cas où l'on ne possède pas le document source) ou un système de calcul. Ce type d'application permettrait dans le même temps une conversion de plusieurs sources en un format unique.

Après avoir passé en revue différents types d'applications réalisables avec un système de reconnaissance structurel des formules mathématiques, nous allons analyser plus en détail dans les paragraphes suivants, les diverses méthodes permettant de résoudre ce problème. Nous examinerons les principales difficultés liées à la reconnaissance de notations bidimen-

[1] WAHL F.M., WONG K.Y. & CASEY R.G., Block segmentation and text extraction in mixed text/image documents (1982).

[2] KRISHNAMOORTHY M., NAGY G., SETH S. & VISWANATHAN M., Syntactic segmentation and labeling of digitized pages from technical journals (1993).

[3] JAIN A.K. & JU B., Page segmentation using document model (1997).

[4] FATEMAN R.J., How to find mathematics on a scanned page (1997).

sionnelles, comme pour les expressions mathématiques, par rapport à une “simple” reconnaissance de texte. Mais commençons par définir plus précisément ce qu’est une notation mathématique.

3 Définition des notations mathématiques

Avant de s’intéresser au problème de la reconnaissance proprement dite, abordons avant tout la définition d’une notation mathématique. Comment définir la syntaxe et la sémantique de ce langage bidimensionnel ? Les notations mathématiques n’ont malheureusement pas été définies formellement et n’ont été qu’à demi standardisées [**?**][1], autorisant par là-même des variantes dans les notations et la manière de les représenter.

G. Martin suggère dans [**?**][2] que l’étape préalable à l’automatisation du traitement des notations mathématiques est l’étude de ces notations. Il présente une brève liste des conventions trouvées dans les publications scientifiques. Voici un aperçu des sources d’informations possibles sur les notations mathématiques.

- Descriptions de formatage : plusieurs ouvrages ont été publiés décrivant les notations mathématiques.
- Descriptions introduites dans les systèmes de formatage de formules : les générateurs de notations mathématiques contiennent des descriptions qui peuvent être utilisables. Dans [**?**][3], D. Knuth décrit les règles d’écriture définies dans TEX pour le formatage des expressions mathématiques.
- Expertise humaine : la communication avec des experts en notations mathématiques peut aider à la définition de conventions appropriées et adaptées aux systèmes de reconnaissance.

Afin de simplifier quelque peu la reconnaissance, il est possible de restreindre le problème en limitant le système à un certain style de formatage. Ce principe a été utilisé par P. Chou [**?**][4] pour la mise au point de son application. Il semble évident que cette restriction contribue à une simplification du problème si l’on connaît la technologie ayant permis de mettre en forme la notation. Il faut toutefois bien penser à conserver la possibilité d’adapter le principe à des notations produites par d’autres systèmes.

Nous pouvons cependant déterminer un ensemble de critères regroupant les opérateurs mathématiques. Cela nous conduit à des classes d’opérateurs qui ont des propriétés communes sur la manière de les représenter, de les reconnaître et donc de les interpréter.

---

[1] HIGHAM N.J., *Handbook of Writing for the Mathematical Sciences* (1993).

[2] MARTIN W.A., Computer input/output of mathematical expressions (1971).

[3] KNUTH D.E., Mathematical typography (1979).

[4] CHOU P., Recognition of equations using a two-dimensional stochastic context-free grammar (1989).

- **Arrangement spatial** : l'arrangement spatial d'un opérateur définit la localisation possible des opérandes. L'agencement des opérandes autour de l'opérateur qui les relie, relève plus des conventions d'usage que de réelles normes. On peut donner comme exemple le cas d'une intégrale ou les indices définissant l'intervalle d'intégration peuvent être au dessus et en-dessous du symbole, ou bien en haut à droite et en bas à droite.
- **Priorité des opérateurs** : la priorité des opérateurs définit un ordre pour l'évaluation de l'expression.
- **Identification** : l'identification de symboles particuliers restreint les possibilités de regroupement avec les symboles environnants. Dans la plupart des cas, l'identification d'un symbole permet de savoir si ce symbole est un opérateur ou une opérande, et la connaissance de son arrangement spatial permet de localiser les zones où se situent les opérandes.
- **Placement relatif des symboles** : le placement relatif des symboles est crucial dans le cas d'opérateurs implicites. Si un opérateur explicite regroupe plusieurs opérandes ($+, \Sigma, \int$), la liberté pour positionner les opérandes est plus grande. Dans le cas d'une notation implicite ($x \times y$ que l'on note xy), le placement relatif des symboles est primordial pour déduire le rôle de chacun.
- **Taille et style relatif des symboles** : la taille relative des symboles donne des indices pour résoudre les ambiguïtés dans les relations spatiales, ce qui est particulièrement important dans le cas de notations implicites. Prenons l'exemple d'une diminution de la taille des caractères, qui signifie de toute évidence que les symboles considérés sont en position d'indice ou d'exposant. Pourtant, ce constat n'est pas une contrainte forte ; ceci est particulièrement flagrant dans le cas de notations manuscrites, où les libertés que prennent les rédacteurs sont grandes.

Malgré quelques indices qui peuvent aiguiller la reconnaissance d'expressions mathématiques, le problème reste difficile tant les particularités sont nombreuses. Nous allons détailler dans le paragraphe suivant quelques unes des difficultés spécifiques à la reconnaissance de notations mathématiques.

4 Difficultés par rapport à la reconnaissance de textes

Beaucoup de recherches sont effectuées dans le domaine de la reconnaissance optique de caractères (*OCR : Optical Character Recognition*). Celles-ci ont conduit à des produits commerciaux comme *OmniPage*, *TextBridge*, etc. Nous allons nous attacher à identifier les différences fondamentales entre la reconnaissance de mots et la reconnaissance de caractères isolés afin de mettre en exergue les difficultés liées à la reconnaissance des symboles composant une expression mathématique.

À des fins de recherche, il est possible de s'abstraire de cette étape pour se concentrer sur l'analyse de l'agencement des symboles afin d'en déduire la structure de la formule. Ainsi, des jeux de données peuvent être obtenus en corrigeant manuellement les erreurs, ou en simulant l'étape de reconnaissance des symboles. Cette approche à été utilisée, entre autre, dans [**?**][1].

Nous commençons par nous intéresser aux problèmes qui interviennent lors de la reconnaissance des symboles composant une expression mathématique. Il est en effet important de comparer les problèmes rencontrés avec ceux qui se produisent dans d'autres domaines de l'analyse de documents. Quelques critères ont été retenus qui nous permettront de déduire les principes majeurs des difficultés rencontrées lors de la reconnaissance des symboles. Les remarques faites s'appliquent à divers types de notations bidimensionnelles, et en particulier au cas des expressions mathématiques.

4.1 Bruit et petits symboles

Plusieurs opérations préparatoires, appliquées à l'image, sont courantes dans l'analyse de documents. Elles servent par exemple à rétablir l'orientation de l'image, ou bien encore à supprimer les petits défauts dûs à l'étape de numérisation du document. C'est le cas, par exemple, de poussières sur la vitre du scanner qui s'ajoutent aux données du document original. Plusieurs types d'algorithmes ont été développés pour réaliser cette opération de nettoyage de l'image. Malheureusement, la plupart d'entre eux ne peuvent pas s'appliquer au cas de la reconnaissance d'expressions mathématiques. En effet, ces notations sont constituées de petits symboles comme le point (.), la virgule (,), l'apostrophe ($'$), le tilde ($\tilde{}$), l'accent circonflexe ($\hat{}$), pour ne citer qu'eux.

La formule suivante représente le noyau de la $k$ème application dans la résolution de Taylor de $m_1, ..., m_n$. Cette formule est contituée pour plus de 40% (47 symboles sur 116) de petits symboles (points, virgules, petits accent circonflexes).

$$\sum_{I=(i_1,...,i_k)} P_I \sum_{j=1}^{k} (-1)^j \frac{ppcm(m_{i_1}, ..., m_{i_k})}{ppcm(m_{i_1}, ..., \hat{m}_{i_j}, ..., m_{i_k})} \delta^{k-1}_{m_1,...,m_n}(e_{i_1} \wedge ... \wedge \hat{e}_{i_j} \wedge ... \wedge e_{i_k}) = 0$$

Nous sommes aussi obligés de constater que ces petits symboles sont souvent décisifs, en tout cas très importants, pour la reconnaissance et pour la compréhension de l'expression dans sa globalité. On peut toutefois nuancer cette argumentation par l'augmentation des performances matérielles et logicielles de la micro-informatique et des photocopieurs. Les impuretés ajoutées à l'image lors de l'étape de numérisation du document tendent à se réduire. La qualité des capteurs et du pré-traitement directement inclus lors de la numérisa-

[1] OKAMOTO M. & TWAAKYONDO H.M., Structure analysis and recognition of mathematical expressions (1995).

tion réduisent considérablement les problèmes. De plus, les erreurs n'étaient pas uniquement dues aux poussières diverses, mais aussi à la faible résolution des capteurs. Ce problème n'est plus réellement d'actualité, les appareils actuels bon marché atteignant des résolutions de 1200 dpi optique (300 dpi pour les télécopieurs).

4.2 Segmentation

La séparation ou segmentation des différents symboles est relativement simplifiée dans le cas des expressions mathématiques typographiées ; il y a en effet très peu de symboles qui se superposent dans une notation. Ce n'est pas le cas dans d'autres applications comme la reconnaissance des partitions musicales, où il y a superposition des notes sur les lignes, ou bien encore dans la reconnaissance d'un texte. Dans un mot, il peut arriver que certaines lettres soient rapprochées au point de se toucher ; c'est par exemple le cas pour ff, fi, ffi, etc. Dans ces différents cas, les lettres sont ligaturées à des fins esthétiques, mais aussi par tradition historique. Cette pratique est en effet un reliquat des origines de l'imprimerie.

Dans le cas de notations manuscrites, le travail est beaucoup moins aisé. De nombreux symboles peuvent être suffisamment proches au point d'avoir une superposition partielle des caractères. Le problème de la segmentation peut alors devenir très difficile à résoudre dans le cas où l'on ne dispose que du visuel de ce qui a été écrit. Quand la formule a été écrite sur une tablette, des données supplémentaires sont utilisables et des techniques mathématiques statistiques permettent d'effectuer la séparation des symboles. C'est le cas, par exemple, des chaînes de Markov (*HMM : Hidden Markov Models*) [**?**][1].

4.3 Reconnaissance des symboles

La reconnaissance des symboles dans les notations mathématiques est beaucoup plus difficile que dans la plupart des autres applications d'analyse de documents. Voici les différents points qui nous amènent à cette conclusion.

- L'ensemble des symboles utilisés : alors que dans un grand nombre d'applications l'ensemble des symboles utilisés est assez réduit (environ une trentaine de symboles pour l'alphabet plus une éventuelle ponctuation), les notations mathématiques disposent de deux alphabets (l'alphabet romain et grec), de chiffres, de symboles de ponctuations utilisés comme notation (point, apostrophe, tilde, etc) et de symboles mathématiques spécifiques (plus, moins, racine, etc). Une évaluation sommaire nous laisse à penser qu'environ 150 à 250 symboles sont assez couramment employés (voir annexe B).
- La variété des styles : les différents styles sont mis à contribution. L'utilisation dans une même formule de caractères en fonte normale, grasse ou italique est courante.

[1] KOSMALA A. & RIGOLL G., Recognition of on-line handwritten formulas (1998).

Ce choix permet de mieux identifier certains groupes de symboles. C'est le cas par exemple pour le nom des fonctions trigonométriques, souvent en italique.

– Les très grandes différences dans la taille des symboles : un simple exemple illustre très bien ce problème, qui est sûrement la caractéristique principale d'une notation mathématique. La taille des symboles s'étend du plus petit, le point par exemple, au plus grand comme une parenthèse ou un signe intégrale, qui regroupe un ensemble de symboles. Cette caractéristique peut même être poussée à l'extrême dans le cas d'un point, qui a la signification d'une dérivée quand il est situé au dessus d'une lettre. Il est alors dans une fonte plus petite que le caractère auquel il se rapporte. Ou bien, dans le cas extrême contraire, l'accolade qui groupe un ensemble de formules, et dont la taille peut alors atteindre celle de la page elle-même.

Certaines de ces caractéristiques ne sont pas exclusivement réservées aux notations mathématiques, mais c'est en tout cas l'une des seules notations regroupant autant de particularités.

4.4 Ambiguïtés sur le rôle d'un même symbole

Un même symbole peut avoir une signification différente suivant le contexte dans lequel il est utilisé. Comparons, par exemple, le sens que peut avoir une ligne dans le cadre d'une analyse de carte routière ; elle peut représenter une route, une rivière, une frontière, la délimitation du centre ville, entre autres choses. Certains attributs visuels donnent un indice pour distinguer ces différents cas, comme la couleur, l'épaisseur du trait, etc. Dans le cas d'un dessin technique une ligne peut représenter le contour d'un objet, une cotation, une hachure, etc.

Il en va de même pour les expressions mathématiques, où un symbole peut avoir différentes significations. Afin d'illustrer ces propos, prenons quelques exemples de symboles pouvant avoir une signification différente suivant leur position dans une formule :

– le point (.) peut représenter un opérateur de multiplication, une notation décimale dans le système anglo-saxon, une opération de dérivation ($\dot{x}$), être une partie d'un symbole plus complexe (: ; !) ou bien encore n'être qu'un parasite obtenu après numérisation du document ;
– la virgule ou apostrophe (,) est employée pour séparer une liste d'arguments, ou peut signifier la notation prime pour une variable, ou bien être un composant d'un symbole (;) ;
– le trait horizontal ($-$) peut matérialiser l'opérateur moins (unaire ou binaire), un trait de fraction, le conjugué d'une variable ($\overline{x}$), ou simplement n'être qu'un composant d'un symbole plus complexe comme le signe égal($=$), inférieur ou égal ($\leq$), inclusion ($\subseteq$), etc.

La seule méthode pour déterminer la signification est donc de tenir compte des informations contextuelles. Résoudre l'ambiguïté est alors un problème qui doit être pris en compte lors de l'analyse de l'agencement des symboles.

$$1 - \frac{x}{\overline{y}} = 0$$

Dans l'exemple ci-dessus, presque tous les traits horizontaux ont une signification différente. On comprend bien alors les difficultés soulevées par un même symbole, ayant un rôle différent suivant son contexte. La seule manière de faire une analyse syntaxique correcte est de tenir compte des autres objets environnant le symbole considéré, donc de faire une analyse en fonction du contexte.

4.5 Ambiguïté sur le placement relatif des symboles

Nous venons de montrer que le contexte était très important pour déterminer la signification d'un symbole. On peut y ajouter le fait que l'agencement des symboles est en grande partie responsable du sens de la notation. Malheureusement, les mathématiciens n'ont pas réellement standardisé leurs notations. De nombreuses libertés peuvent être prises dans le placement relatif des symboles, ce qui devient critique dans le cas d'opérateurs implicites. Seul le positionnement "précis" des symboles est alors capable d'indiquer le sens de la notation, contrairement au cas où l'on a un opérateur qui unifie ses opérandes. Les notations du type indice, exposant, multiplication implicite ou encore les notations matricielles sont des notations sans réel symbole groupant, donc dites implicites, alors que l'addition, la soustraction, la division utilisent des symboles pour unifier leurs opérandes. Prenons un exemple pour illustrer ce problème lié au placement des symboles.

$$2_{\mathrm{x}} \quad 2_{\mathrm{x}} \quad 2_{\mathrm{x}} \quad 2_{\mathrm{x}} \quad 2\mathrm{x} \quad 2^{\mathrm{x}} \quad 2^{\mathrm{x}} \quad 2^{\mathrm{x}} \quad 2^{\mathrm{x}}$$

La difficulté provient du fait que les positions ne sont pas réparties sur un espace discret, comme cela peut être le cas pour un texte, mais dans un espace continu. Il se pose donc un problème de cas limite pour déterminer si un symbole est en position d'indice, s'il est aligné ou s'il est en exposant. Cette confusion est encore plus sensible dans le cas d'une reconnaissance de notations manuscrites.

L'exemple précédent démontre bien qu'une grande partie de l'essence de la notation est donnée par le placement des symboles les uns par rapport aux autres. Il peut être difficile de déterminer quelles sont les relations exactes qui existent entre deux symboles, cette structure pouvant être remise en cause par d'autres symboles proches. Pour illustrer notre propos, considérons la configuration ${}^{\mathrm{x}}\!_{\mathrm{i}}$. Celle-ci peut signifier trois choses :

- i est en position d'indice par rapport à x ($x_i y_j$)
- i peut être sur la ligne principale de référence de l'expression ($a^x i$),
- x est en position exposant précédent la variable i comme pour la notation transposée.

D'autres exemples de ce type, mettant en évidence les ambiguïtés liées au placement des symboles et du contexte par rapport auxquels on évalue les relations, sont exposés par G. Martin [**?**][1] et M. Okamoto [**?**][2]. Ce problème de l'ambiguïté des relations spatiales entre les éléments est encore plus critique pour les notations manuscrites, le rédacteur prenant souvent des libertés avec le placement et l'alignement des symboles.

4.6 Ambiguïté dans la notation

Zhao, Sakurai, Sugiura et Torii [**?**][1] se proposent de définir deux types de conventions tacites dans les notations mathématiques : identifiables et non identifiables. Les conventions non identifiables incluent les exemples ambigus décrits par Martin dans [**?**][2]. Voici une des illustrations de ce type d'ambiguïté :

$$\text{Est-ce que } \sum_{i=5}^{10} i + Y \text{ signifie } \sum_{i=5}^{10} (i + Y) \text{ ou bien } \left(\sum_{i=5}^{10} i \right) + Y \text{ ?}$$

L'interprétation de cette expression dépend de différentes conventions, du contexte, du champ d'application de l'expression ou bien encore des habitudes personnelles. Elle nécessite donc la connaissance et l'expérience du lecteur, qui permettraient de déterminer l'ordre de priorité.

Les conventions décidables correspondent aux règles implicites de précédence des opérateurs mathématiques . Une personne qui lit la formule $a + b \times c$ connaîssant les règles de priorité standard (la multiplication prime sur l'addition), comprend le sens de la formule. Cela signifie qu'il l'interprétera en $a + (b \times c)$ et non $(a + b) \times c$.

Les auteurs présentent différents types de formalismes qui peuvent être utilisés pour la saisie d'expressions mathématiques : un formalisme fort, faible et libre. Plus le formalisme est fort, moins les conventions tacites doivent être encodées dans la grammaire effectuant l'analyse de l'expression, la rendant plus complexe. Il est donc plus facile d'écrire une grammaire se basant sur un formalisme fort.

Formalisme fort : chaque structure mathématique employée dans un système utilisant un formalisme fort doit être placée dans une "boîte" par l'utilisateur. Ainsi, toutes les conventions implicites sont explicitées par la personne qui effectue la saisie. Aucune information

[1] MARTIN W.A., Computer input/output of mathematical expressions (1971).

[2] OKAMOTO M. & TWAAKYONDO H.M., Structure analysis and recognition of mathematical expressions (1995).

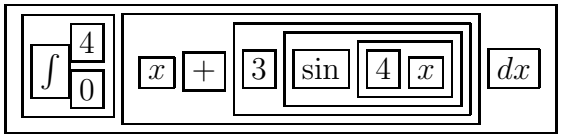
[1] ZHAO Y., SAKURAI T., SUGIURA H. & TORII T., A methodology of parsing mathematical notation for mathematical computation (1996).

[2] MARTIN W.A., Computer input/output of mathematical expressions (1971).

sur la priorité des opérateurs n'est nécessaire dans la grammaire car toutes la connaissance est fournie par la structure. Prenons la formule suivante pour illustrer nos propos :

$$\int_0^4 x + 3 \sin 4x \; dx$$

Elle est encodée de la manière suivante :

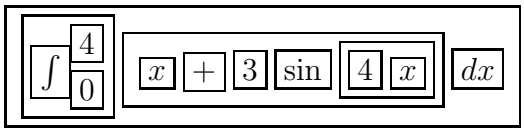


Ceci peut être interprété comme si l'utilisateur avait explicitement encodé la précédence de tous les opérateurs, ainsi que les arguments de chacun des opérateurs.

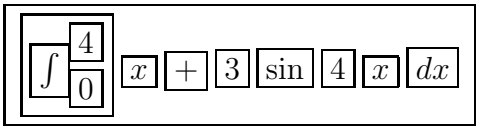
Formalisme faible : l'utilisateur fournit moins d'information dans le formalisme faible ; la grammaire a connaissance de la précédence des opérateurs. Les "boîtes" qu'il n'est plus nécessaire d'expliciter, sont celles qui peuvent donc être déduites de la précédence implicite des opérateurs. La grammaire résultante utilise des catégories grammaticales comme les termes, les facteurs, les atomes, etc. Les priorités sont spécifiées en utilisant ces catégories. Sur l'exemple ci-dessus, la précédence de la multiplication implicite sur l'application de fonction se traduit par une règle grammaticale du type :

<sin op> ← sin <terme>
<terme> ← <terme> <facteur>

Le niveau de précision des "boîtes" structurelles devient donc :



Formalisme libre : Les seules boîtes structurelles spécifiées sont celles qui permettent d'effectuer la mise en forme de l'expression.



Quand on utilise un formalisme libre, la grammaire permettant l'analyse de la notation doit être étendue afin de déterminer le début et la fin d'un groupe de symboles. Dans l'exemple cité, la grammaire devra déterminer quels sont les éléments qui constituent l'argument de l'intégrale.

4.7 Peu de redondance de l'information

Dans la plupart des notations utilisées, la redondance de l'information est souvent présente. C'est le cas des partitions musicales. L'information concernant le bémol (♭) ou le dièse (♯) pour une note est présente en début de portée, à la clé. Mais elle est aussi souvent répétée juste devant la note pour rappeler la modification.

Il est aussi possible, dans certaines applications, que le domaine des combinaisons de symboles soit restreint, le cas typique étant les caractères composant un mot. Il est souvent inutile de reconnaître l'ensemble des symboles constituant un mot pour pouvoir identifier celui-ci. Cette propriété à conduit à l'introduction de trois techniques majeures dans les logiciels de reconnaissance de texte :

- l'utilisation de dictionnaires : ils ont fait leur apparition dès 1959 pour reconnaître un mot en ne possédant qu'une partie des lettres qui le composent. Une erreur courante des systèmes de reconnaissance de caractères est la confusion entre *m* et *rn*. L'utilisation d'un dictionnaire adapté au texte que l'on cherche à identifier permet en effet de modifier "come" en "corne". Mais cette technique peut toutefois avoir ses limites comme dans le cas de "darne" et "dame" qui existent tous les deux dans la langue française ;
- l'utilisation de statistiques sur l'apparition des lettres dans une langue donnée : cette méthode se base sur la fréquence d'apparition des lettres ou des combinaisons de lettres dans une langue. Par exemple, la lettre la plus courante en anglais est le "e" (13%), et la paire de lettres la plus couramment utilisée est "he". À l'aide d'une prospection encore plus poussée, on peut constater que la lettre "q" est plus utilisée au début des mots qu'au milieu ou en fin de mot ;
- l'analyse syntaxique de la phrase : cette technique permet d'identifier des mots en fonction de leur rôle. Reprenons un exemple de mauvaise reconnaissance de *m* et *rn* : "chamelle" et "charnelle". L'analyse syntaxique de la phrase pourra permettre de lever une ambiguïté potentielle entre ces deux mots étant donné que l'un est un nom et l'autre un adjectif.

Ces techniques permettent d'augmenter considérablement les performances des logiciels pour la reconnaissance de textes.

Nous n'irons pas plus avant dans les explications de ces méthodes, ceci s'éloignant de notre sujet d'étude. Le lecteur pourra se reporter à [**?**][1] et [**?**][2] pour de plus amples informations. Mais aucune des techniques citées n'est applicable directement aux notations mathématiques, qui ne contiennent que très peu de redondances de l'information. Ceci exclut, par là-même, l'aide à la résolution des ambiguïtés lors de la reconnaissance des symboles.

[1] NAGY G., Optical character recognition - theory and practice (1982).

[2] DENGEL A., HOCH R., HÖNES F., JÄGER T., MALBURG M. & WEIGEL A., Techniques for improving ocr results (1997).

5 Quelques traitements préliminaires

Les notations mathématiques utilisent un arrangement bidimensionnel des symboles pour transmettre l'information souhaitée. Comprendre et interpréter une expression implique deux étapes, pouvant être éventuellement concurrentes : la reconnaissance des symboles et l'analyse de l'agencement de ces symboles. La première étape convertit une image en un ensemble de symboles avec leurs caractéristiques ; la seconde analyse l'agencement de cet ensemble de symboles dans le plan (relativement aux conventions de notations de ce langage bidimensionnel), afin de retrouver l'information contenue dans cette notation mathématique.

Ces deux étapes majeures peuvent elles-mêmes se subdiviser en plusieurs sous catégories. Nous allons étudier séparément ces composantes en gardant toujours à l'esprit que chacun des traitements doit avoir une approche globale du problème qu'est la reconnaissance des notations mathématiques, afin de tenir compte des spécificités.

La première étape dans la reconnaissance structurelle de formules mathématiques est l'acquisition de données. Elle peut être de deux ordres :

- en ligne : c'est le cas lorsque l'on saisit une notation à l'aide d'une tablette graphique (ou tablette à numériser). Des données temporelles sont alors disponibles et d'une grande utilité pour la séparation des caractères ainsi que leur reconnaissance.
- hors ligne : c'est le cas pour un document papier (typographié ou manuscrit) que l'on numérise à l'aide d'un scanner.

Dans ce dernier cas de figure, les mécanismes mis en jeu peuvent introduire plusieurs types d'erreurs ou de défauts lors de la transformation vers une image numérique. Nous allons étudier leur nature et évoquer les techniques utilisées pour les corriger.

La figure 3 illustre les différentes étapes nécessaires au passage d'un document papier à un document sous une forme compacte et utilisable par un ordinateur.

La capture des données, ou **numérisation**, consiste à transformer les informations dont on dispose sous forme papier, à l'aide d'un scanner optique. Le résultat est un ensemble très important de points, appelés pixels, caractérisant le document original. Pour mieux comprendre le traitement effectué, celui-ci équivaut à voir le document original au travers d'une grille très fine, et une information est déterminée pour chaque case de la grille. Elle peut être de plusieurs types : vrai (0) ou fausse (1), typique d'une image noir et blanc, ayant une valeur comprise entre 0 et 255 (image en niveau de gris), ou décomposée en couleurs "primaires" (rouge, vert et bleu) pour une image couleur. La taille des cases de la grille est déterminée en fonction de la résolution de numérisation que l'on utilise. Communément, une résolution de 300 dpi est utilisée ce qui représente, pour une page A4, 2600x4200 pixels soit 1 méga-octets pour une image noir et blanc, 10 méga-octets pour une image en niveau de gris et 30 méga-octets pour une image couleur.

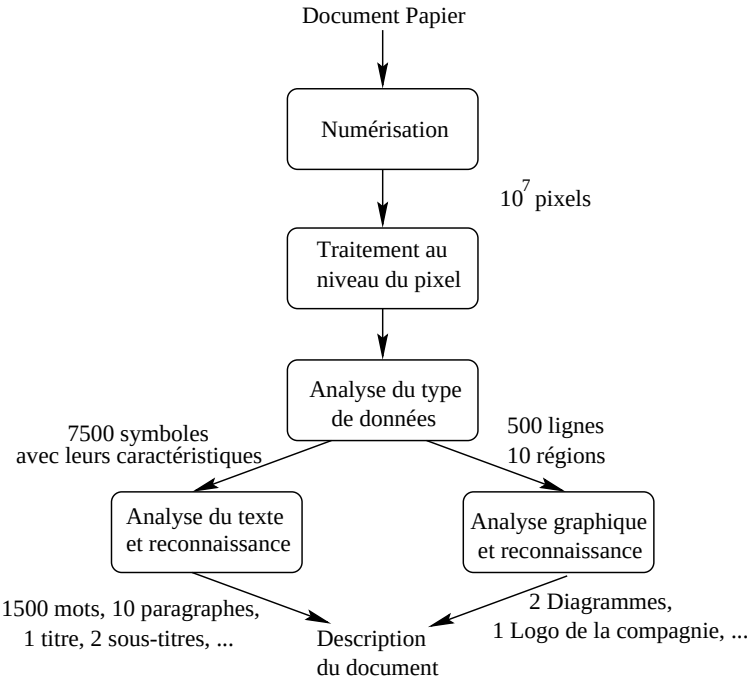


FIG. 3 – Les étapes de l'analyse d'un document papier

Il est important de comprendre que l'image d'un document contient un ensemble de données qu'il est nécessaire d'analyser plus en profondeur pour avoir une information plus pertinente. Entre l'image qui représente un caractère et le fait de reconnaître que cette image correspond à un caractère précis, plusieurs étapes préliminaires sont nécessaires. Nous allons présenter les plus importantes.

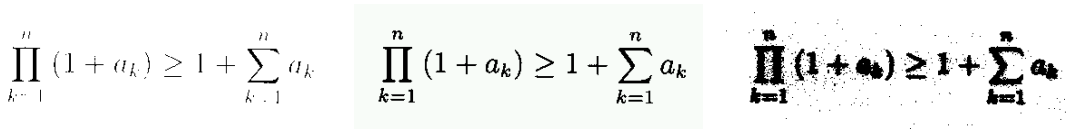
5.1 Seuil de numérisation

Une des premières difficultés pouvant survenir lors de la transformation d'un document en une image numérique est le choix du **seuil de numérisation** (*thresholding*). Afin de mieux la percevoir, prenons un exemple. Considérons la formule suivante :

$$\prod_{k=1}^{n} (1 + a_k) \geq 1 + \sum_{k=1}^{n} a_k$$

Suivant les paramètres sélectionnés lors de la numérisation, à savoir le contraste et la luminosité principalement, l'image obtenue peut être de plus ou moins bonne qualité. Si la numérisation a un seuil trop bas, l'image comportera des anomalies comme le manque de points dans des zones sombres, ou le manque de pixels pour certains caractères. Dans le cas contraire d'une numérisation avec un seuil trop élevé, l'image contiendra beaucoup de points parasites, ou bruit, introduisant de grandes difficultés lors de la reconnaissance des

caractères. La figure 4 illustre ces différents cas.



(a) Seuil trop bas (b) Bonne numérisation (c) Seuil trop élevé

FIG. 4 – Exemples de différents seuils de numérisation

Plusieurs techniques ont été envisagées afin de résoudre ce problème [**?**][1] : une méthode globale de calcul du seuil optimal pour l'ensemble du document [**?**][2], ou une méthode adaptative (ou locale) qui dépend de l'intensité des composants du document. Nous noterons aussi, même s'il n'est pas primordial de déterminer automatiquement ce seuil de numérisation automatiquement, que la qualité du traitement de numérisation influe fortement sur les performances de la reconnaissance des caractères.

Nous n'entrerons pas dans le détail de ces différentes méthodes, notre étude ne portant pas sur ce problème spécifique de traitement de l'image. D'autant plus qu'il ne présente pas de particularités dans le cas spécifique des formules mathématiques.

5.2 Réduction du bruit

Après la numérisation, l'image du document obtenue doit être filtrée afin de réduire le "bruit" (*noise reduction*) dû à cette transformation. Ces imperfections sont souvent la marque des documents numérisés avec du matériel de faible qualité, comme les télécopies, ou les mauvaises photocopies d'originaux. Ce "bruit" prend la forme de pixels noirs souvent isolés, ou bien regroupés en petites taches noires au milieu d'une zone blanche. Ce traitement de réduction du bruit est primordial. En effet, ce bruit introduit des perturbations dans les données et donc une incertitude dans la reconnaissance. Cette étape s'avère essentielle dans le cas de documents de mauvaise qualité. Nous ne détaillerons pas ici les différentes techniques mises en jeu pour effectuer ce traitement. Pour de plus amples informations, le lecteur intéressé pourra se reporter à l'ouvrage de L. O'Gorman et R. Kasturi [**?**][1] pages 16 et suivantes. Un comparatif détaillé sur les différentes méthodes de réduction du bruit pourra être trouvé dans [**?**][2].

---

[1] SAHOO P.K., SOLTANI S., WONG A.K.C. & CHEN Y.C., A survey of threhloding techniques (1988).

[2] LEE S.U., CHUNG S.Y. & PARK R.H., A comparative performance study of several global thresholding techniques for segmentation (1990).

[1] O'GORMAN L. & KASTURI R., *Executive Briefing : Document Image Analysis* (1997).

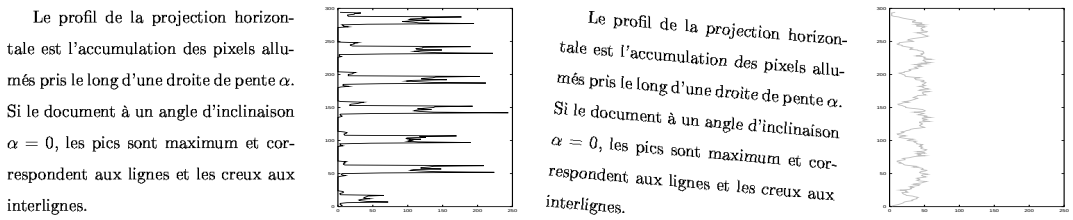
[2] WU W.Y., WANG M.J.J.J. & LIU C.M., Performance evaluation of some noise reduction methods (1992).

Mais le problème capital, dans le cas des notations mathématiques, réside dans le fait que beaucoup de symboles majeurs sont d'une taille très réduite et peuvent éventuellement être confondus avec du bruit. Aucune technique n'a été spécialement développée pour le cas qui nous intéresse. Toutefois, on peut citer le travail de P. Chou [**?**][3] qui présente une méthode, basée sur une grammaire probabiliste, très peu sensible au bruit.

5.3 Réalignement de l'image

Lors de la numérisation, il arrive que le document papier soit mal positionné, et que la feuille soit "de travers". De ce fait, l'image numérique créée est "penchée". Les lignes de texte, et donc les caractères qui les composent, sont plus ou moins légèrement penchées. Il est alors nécessaire de rétablir l'orientation de l'image afin de ne pas perturber le processus de reconnaissance des caractères. Deux techniques principales sont utilisées pour déterminer l'angle de rotation (*skew orientation*) qu'a subi le document, et pour pouvoir ainsi automatiquement corriger cette erreur.

La première méthode illustrée par la figure 5, la plus populaire, consiste à faire une succession de projections horizontales et/ou verticales (*projection profile method*), en faisant varier faiblement l'angle de projection, afin de déterminer un histogramme des pixels noirs qui composent l'image, et ainsi chercher à maximiser les pics et les creux de cette courbe.



(a) Texte bien orienté (b) Projection (c) Texte penché de -5° (d) Projection

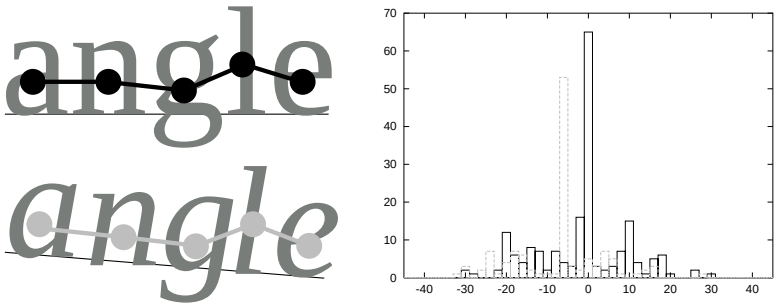
FIG. 5 – Méthode de projection de textes pour différentes orientations

Dans la deuxième approche, le principe consiste à trouver les voisins les plus proches (*nearest-neighbor method*), au sens de la distance euclidienne, et à déterminer la pente de la droite reliant le centre de ces couples de caractères. Pour obtenir le résultat final, qui est l'angle de correction permettant d'avoir un alignement correct du document, il suffit de construire l'histogramme des pentes des droites obtenues.

La figure 6 illustre cette méthode pour le mot angle. L'histogramme I.6(b) a été construit pour le texte choisi dans l'exemple de calcul des projections : la partie foncée correspond au texte orienté correctement (angle=0 deg), la partie claire, quant à elle, est relative au texte

[3] CHOU P., Recognition of equations using a two-dimensional stochastic context-free grammar (1989).

penché d'un angle de -5 deg. On constate bien que dans ces deux cas, on peut avoir une lecture directe de la rotation qu'a subi le document.



(a) Exemple de texte (b) Histogramme

FIG. 6 – Méthode évaluant les angles de droites entre voisins pour un texte

Toutefois, ces deux méthodes de réorientation de documents textuels, s'appliquent plus difficilement dans le cas de formules mathématiques isolées. Ceci est dû à la structure des notations mathématiques, qui n'est pas aussi linéaire qu'un paragraphe de texte. Pour mettre en évidence cette affirmation, prenons l'exemple de la formule suivante :

$$\sum_{i=0}^{9} x_i^2$$

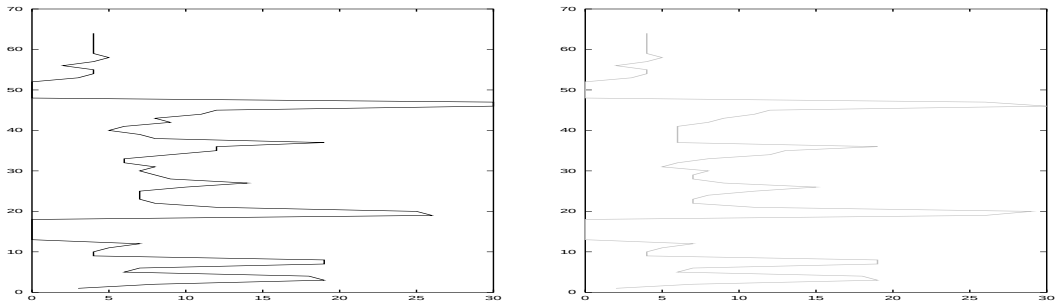
Si nous appliquons les deux méthodes exposées précédemment, nous pouvons constater que ces stratégies ne donnent pas de résultats aussi satisfaisants que dans le cas d'un paragraphe de texte. Pour le cas des projections, les différences entre une formule inclinée ou correctement orientée sont très peu sensibles (cf figure 7).

Quant à l'histogramme construit à partir des angles de droites formés par les plus proches voisins (cf figure 8), il est impossible d'en déduire des données fiables.

Cependant, ces remarques ne s'appliquent qu'à la détection de l'orientation d'une formule isolée. Si celle-ci est placée dans un environnement textuel, il est alors possible d'appliquer la méthode sur le reste du document afin d'obtenir le résultat.

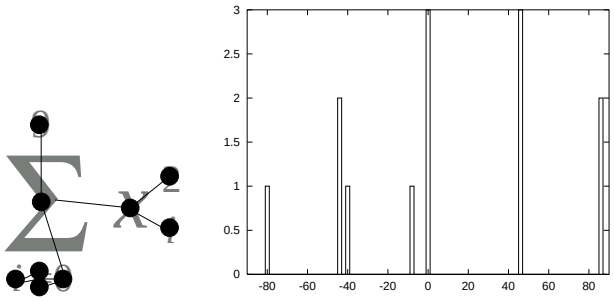
5.4 Isoler une formule dans un document

Les expressions mathématiques sont la plupart du temps incluses dans un document textuel, soit en tant que formule mise en valeur, donc isolée du reste du texte, soit directement au fil du texte. La première opération à réaliser après le traitement de l'image, qui est plus général, est donc l'identification et la séparation de la notation mathématique du reste du



(a) Projection de l'image bien orientée (b) Projection l'image penchée de -5°

FIG. 7 – Méthode de projection appliqué à la formule $\sum_{i=0}^{9} x_i^2$



(a) Formule (b) Histogramme

FIG. 8 – Méthode évaluant les angles de droites entre voisins pour la formule $\sum_{i=0}^{9} x_i^2$

document.

En règle générale, les systèmes de reconnaissance supposent que l'expression a été isolée. On peut toutefois citer les travaux de A. Jain et B. Ju [**?**][1] sur la reconnaissance des différents éléments composant un document (zones de textes, images, tableaux, formules, etc.) ainsi que ceux de H. J. Lee et J. S. Wang [**?**][2], qui ont plus particulièrement étudié le cas de la localisation des expressions mathématiques dans un document. Dans cet article, les auteurs expliquent qu'une formule "isolée" ne pose pas de problème, car celle-ci étant bien séparée du texte, l'espacement vertical avant et après l'expression ainsi que les symboles caractéristiques la composant sont des facteurs déterminants pour sa localisation. Cette zone est alors étiquetée comme ayant un contenu mathématique. Les lignes de texte restantes sont composées de texte et d'expressions mathématiques embarquées. Ces lignes sont converties en un flux de mots dont certains sont reconnus comme faisant parti des notations mathématiques. Aucun détail supplémentaire n'est fourni.

Une autre approche très intéressante, basée sur l'utilisation de logique floue, est présentée par A. Kacem, A. Belaïd et M. Hamed [**?**][1]. Extrafor est un système permettant d'isoler les notations mathématiques présentes dans un document scientifique. Il permet à la fois d'extraire les notations isolées du texte (formules mises en évidence) mais aussi les notations ou symboles inclus dans un paragraphe. Les résultats obtenus sont très intéressants : près de 100% des notations isolées et plus de 90% des symboles ou expressions incluses dans un paragraphe sont extraits correctement du document. Elles peuvent donc être fournies à un système de reconnaissance structurelle des notations mathématiques indépendant du système pour la reconnaissance de texte, autorisant ainsi l'utilisation du système le plus adapté à chaque type de données.

Mais, comme nous avons pu le constater dans les paragraphes précédents, l'arrangement bidimensionnel des symboles est caractéristique des formules mathématiques. Cette propriété peut donc être utilisée afin d'isoler les données mathématiques du reste des informations constituant le document.

5.5 Conclusion

Nous avons présenté les différentes étapes préliminaires nécessaires à la reconnaissance des symboles :

– seuil de numérisation,

[1] JAIN A.K. & JU B., Page segmentation using document model (1997).

[2] LEE H.J. & WANG J.S., Design of a mathematical expression recognition system (1995).

[1] KACEM A., BELAÏD A. & AHMED M., Extrafor : Automatic extraction of mathematical formulas (1999).

– réduction du bruit,
– réalignement de l'image.

Ces techniques ont toutes été introduites dans le cadre de la reconnaissance de documents textuels et sont donc toutes particulièrement adaptées à ceux-ci.

Bien que la détection automatique du seuil de numérisation du document ne soit pas un problème spécifique aux documents constitués de notations mathématiques, la réduction du bruit et la réorientation de l'image sont des étapes qui doivent prendre en compte les spécificités du domaine. Ces techniques ne sont donc pas transposables directement au cas de la reconnaissance de notations mathématiques. Elles devraient faire l'objet d'études spécifiques et tout particulièrement dans le cadre de notations isolées (exemple donné de la réorientation d'une formule isolée non résolue par les techniques classiques).

Parmi ces étapes préliminaires, on notera toutefois les travaux très récents sur l'extraction, des notations mathématiques, du reste du document, permettant ainsi d'effectuer un traitement spécifique, tant au niveau :

– des étapes de traitement de l'image préliminaires à la reconnaissance,
– que de l'identification des symboles dans les formules, présentant des particularités par rapport au texte.

6 Segmentation et reconnaissance des symboles

Dans le paragraphe précédent, nous avons présenté, à l'aide de plusieurs exemples, les étapes préliminaires à la reconnaissance, nombreuses et variées. La plupart d'entre elles ne sont pas toujours adaptées au cas particulier des notations mathématiques, comme nous l'avons mis en évidence dans le cas de la réorientation d'image ou de la réduction du bruit. Les traitements présentés ne s'appliquent, bien sûr, que dans le cas d'un document numérisé, et non dans celui de notations manuscrites saisies à l'aide d'une tablette graphique.

En ce qui concerne la reconnaissance proprement dite du symbole, deux cas de figures sont à considérer : le cas où les caractères sont typographiés et le cas où ils sont manuscrits. Une première étape reste cependant commune, même si les techniques mises en œuvre pour la réaliser ne sont pas similaires : la segmentation, c'est à dire l'opération qui consiste à isoler chaque symbole pour en effectuer la reconnaissance. Comme c'est le cas dans la plupart des applications dédiées à la vision par ordinateur, l'efficacité des traitements préliminaires est déterminante pour la qualité du résultat final. Nous évoquerons les méthodes mises en jeu pour résoudre ce problème dans chacun des deux cas considérés, ainsi que les théories employées pour la reconnaissance du motif.

6.1 Caractères typographiés

Classiquement, la reconnaissance de caractères typographiés inclut les étapes suivantes : l'image correspondant à la page est segmentée pour en trouver les composantes (titre, paragraphes, etc), puis chacun des composants subit à son tour cette opération, pour extraire ces caractères propres. Avant d'en revenir à la reconnaissance du symbole en lui-même, étudions tout d'abord la segmentation dans le cas de caractères typographiés.

L'étape initiale de segmentation des caractères est primordiale pour une bonne reconnaissance du symbole. Elle permet l'identification individuelle des caractères composant les mots ou les phrases. L'approche couramment utilisée pour réaliser cette segmentation est d'effectuer une projection verticale de chacune des lignes. Le profil d'une projection horizontale a déjà été présenté dans le cadre du réalignement de l'image. Cette étape permet, outre de détecter l'angle de rotation éventuel qu'a pu subir le document, de mettre en évidence les lignes qui composent le texte. En effectuant une projection verticale pour chacune de ces lignes, le profil met en évidence les zones de pixels allumés composant chacun des caractères. Cependant, cette technique est limitée dans le cas d'un document très bruité, ou plus simplement dans le cas d'un mot en italique, la séparation entre chaque lettre apparaissant moins clairement.

Une zone contenant un caractère est donc maintenant identifiée par la segmentation. Reste à identifier celui-ci. Pour cela, la technique consiste à étudier un ensemble de caractéristiques pour chaque symbole. Le choix de ces caractéristiques est primordial et doit présenter les propriétés suivantes : la variance intra-classe doit être minimale alors que la variance inter-classe doit être maximale ; le rapport largeur hauteur, la convexité, la connexité, l'asymétrie, etc, sont autant de caractéristiques discriminantes couramment utilisées.

Historiquement, la technique utilisée pour effectuer cette identification de motifs est basée sur une analyse syntaxique ([**?**][1], [**?**][2]), introduite au début des années soixante. Ces techniques sont toujours d'actualité. L'approche structurelle ([**?**][3]) a également été exploré. Toutefois, étant donnée la complexité et la variété des formes à reconnaître, les tendances actuelles ne sont plus à l'utilisation d'une méthode unique. L'emploi de méthodes mixtes ou hybrides tend à se généraliser. :

- modèle mêlant analyse syntaxique et relaxation [**?**][4],
- modèle combinant description syntaxique et statistique des formes [**?**][5], [**?**][6], afin

[1] GONZALEZ R.C. & THOMASON M.G., *Syntactic Pattern Recognition : an Introduction* (1978).

[2] BARTSCH-SPÖRL B., Grammatical inference of graph grammars for syntactic pattern recognition (1982).

[3] NAGY G., Optical character recognition - theory and practice (1982).

[4] DON H.S. & FU K.S., A syntactic method for image segmentation and object recognition (1985).

[5] TSAI W.H., *Syntactic and Structural Pattern Recognition : Theory and Applications* (1990).

[6] FU K.S., A step towards unification of syntactic and statistical pattern recognition (1983).

d'améliorer la représentation des formes avec erreurs. Ces modèles conduisent soit à la définition d'une grammaire stochastique associée à un analyseur correcteur d'erreur, soit à une définition de vecteurs caractéristiques, exploités par une technique de classification [**?**][7].

– mélange des approches syntaxiques et des approches à réseaux de neurones ([**?**][8] appliqué à la reconnaissance de caractères numériques manuscrits).

Plusieurs article de H. Bunke sont à citer sur l'utilisation de ces méthodes hybrides : [**?**][9], [**?**][10], [**?**][11]. Ces modèles hybrides constituent actuellement un domaine de recherche des plus intéressant pour la reconnaissance de motifs.

La reconnaissance des caractères imprimés, ou typographiés, est l'application principale des systèmes de reconnaissance optique de caractères (*OCR*). Les systèmes les plus anciens étaient limités à la reconnaissance de quelques fontes dans une taille unique. Chacun des caractères isolés était reconnu en utilisant des modèles de caractères. Pour que ces systèmes soient les plus performants possibles, il était nécessaire que le document n'ait subi aucune rotation, aucun changement d'échelle, aucune distorsion. Mais il est vite apparu qu'il était nécessaire de développer de nouvelles applications capables de reconnaître un nombre plus significatif de fontes et de tailles de caractères.

Ces systèmes, dits omnifontes, ont été, et sont encore, un défi pour les concepteurs de ces applications. Pas moins de 300 fontes différentes peuvent être reconnues, avec des tailles de caractères allant de 6 à plus de 26 points, avec des variations très nombreuses comme l'italique, le gras, les ligatures, etc.

La plupart génèrent tout de même trop d'erreurs, dues à une mauvaise segmentation, à une confusion de fonte, ou bien encore aux nombreux points communs des caractères, difficiles alors à différencier. Pour ne citer que quelques exemples, on peut classer dans cette dernière catégorie : 2 et Z, la lettre O et la chiffre 0, S et 5, I, 1 et l, ou bien encore m et rn, etc. Il devient alors nécessaire d'avoir recours au contexte pour pouvoir prendre la bonne décision. L'une des techniques les plus évidentes pour améliorer les résultats de la reconnaissance est l'utilisation d'un dictionnaire, afin de mettre fin à l'ambiguïté d'un des symboles en fonction de ses voisins, en se basant sur un contexte linguistique (probabilité d'apparition des lettres dans une langue, règles grammaticales [**?**][1]), ou bien encore sur un contexte spatial (faire prévaloir le choix des symboles dans la police de caractères que l'on a pu identifier comme étant la police par défaut du document).

[7] TSAI W.H. & FU K.S., Attributed grammar : A tool for combining syntactic and statistical approaches to pattern recognition (1980).

[8] BAPTISTA G. & KULKARNI K.M., A high accuracy algorithm for recognition of handwritten numerals (1988).

[9] BUNKE H., Hybrid approaches (1986).

[10] BUNKE H., Hybrid pattern recognition methods (1990).

[11] BUNKE H., Structural and syntactic pattern recognition (1993).

[1] NAGY G., Teaching a computer to read (1992).

Malheureusement, toutes ces méthodes heuristiques, utilisées couramment pour augmenter les performances de la reconnaissance de textes, ne sont pas transposables à la reconnaissance des symboles composant une notation mathématique. Toutefois, les méthodes classiques de reconnaissance de motifs que nous avons présentées sont, quant à elles, employées, même si quelques adaptations sont nécessaires. Nous détaillerons ce dernier point dans le chapitre "réalisations".

6.2 Caractères manuscrits

Même si quelques méthodes adaptées à la reconnaissance de l'écriture manuscrites sont similaires à celles employées dans le cas de la reconnaissance de symboles typographiés, plusieurs différences sont à souligner.

- La méthode utilisée pour l'acquisition des données :
  - les données capturées en direct à l'aide d'une tablette graphique sont disponibles sous la forme d'une séquence temporelle de position du stylet sur la tablette, préservant ainsi la manière dont les caractères ont été dessinés (vitesse, accélération, etc). A noter que sur certains modèles de tablette, l'information de pression du stylet est disponible.
  - l'acquisition des données s'effectue comme dans le cas d'un document typographié, c'est à dire en ayant recours à un scanner.
- Le processus de segmentation utilisé pour isoler chaque caractère individuellement est un des composants majeurs de la reconnaissance de l'écriture cursive.
- Les modes d'écriture d'un texte sont potentiellement illimités : un même texte écrit par un même utilisateur à deux moments différents est très rarement identique en tout point.

Dans le cas de l'acquisition des données en temps réel, un des éléments prépondérant est l'information temporelle disponible (l'ordre dans lequel les gestes ont été effectués, ainsi que la vélocité relative des différents mouvements composant ce geste). Ce type d'information facilite quelque peu la segmentation.

Plusieurs approches ont été utilisées pour effectuer la reconnaissance des caractères manuscrits ; les plus anciennes utilisaient des méthodes syntaxiques [**?**][1], alors que d'autres études ont été développées spécialement pour la reconnaissance d'une classe particulière d'éléments comme les chiffres [**?**][2]. Les travaux les plus récents concernant la reconnaissance de l'écriture manuscrite sont basés sur la théorie des chaînes de Markov cachées.

---

[1] BERTHOD M., *Une Méthode Syntaxique de Reconnaissance des Caractères Manuscrits en Temps Réel avec un Apprentissage Continu,* PhD Thesis (1975).

[2] BAPTISTA G. & KULKARNI K.M., A high accuracy algorithm for recognition of handwritten numerals (1988).

La théorie des chaînes de Markov (*HMM : Hidden Markov Model*) a été introduite à la fin des années 1960 par Baum. La reconnaissance est obtenue par calcul de la probabilité a posteriori de la classe de la forme. Ce calcul fait intervenir plusieurs termes qui, suivant certaines hypothèses de dépendances liées à l'application traitée, peuvent se décomposer en probabilités conditionnelles élémentaires. Peu de temps après son introduction, cette méthode a été utilisée dans le cadre de la reconnaissance de la parole, chez IBM par exemple, et plus récemment, dans des domaines comme la reconnaissance de formes, la poursuite de cibles, l'identification de signaux de sonars, etc. L'engouement pour cette méthode s'explique tout simplement par sa grande capacité d'intégration du contexte et d'absorption du bruit. Elle s'est imposée dans de nombreux domaines et a donc été tout naturellement utilisée dans les applications de reconnaissance de l'écriture manuscrite. Beaucoup de travaux sur ce sujets sont à noter ([**?**][3], [**?**][4], [**?**][5], [**?**][6], [**?**][7]), [**?**][8] et plusieurs d'entres eux concernent exclusivement la segmentation et la reconnaissance de symboles mathématiques ([**?**][9]). Cette méthode à même été utilisée dans le cadre de la reconnaissance de caractères typographiés [**?**][10].

6.3 Conclusion

La reconnaissance de symboles proprement dite n'est pas spécifique à l'identification des notations mathématiques. C'est un domaine de recherche à part entière pour lequel nous ne prétendons pas donner un aperçu exhaustif en deux paragraphes. Nous n'avons présenté que les principales caractéristiques et les principales approches existantes pour les cas typographiés et manuscrits.

La diversité des symboles utilisés dans les notations mathématiques implique quelques spécificités ; l'importance des petits symboles, ainsi que la grande variété de taille des caractères doivent être prises en compte dans le processus de classification et d'identification des symboles. Dans le chapitre IV, nous reviendrons plus amplement sur les conséquences de cette diversité et les répercussions sur les méthodes et algorithmes pour un logiciel de reconnaissance des caractères, plus spécifiquement dédié aux caractères entrant dans la com-

[3] BELAÏD A. & SAON G., Utilisation des processus markoviens en reconnaissance de l'écriture (1997).

[4] GUYON I., SCHENKEL M. & DENKER J., Overview and synthesis of on-line cursive handwritten recognition techniques (1997).

[5] KUNDU A., Handwritten word recognition using hidden markov model (1997).

[6] ANQUETIL E., G.LORETTE & GENTRIC P., Reconnaissance en ligne d'écriture cursive par chaînes de markov cachées (1995).

[7] WINKLER H.J., Hmm-based handwritten symbol recognition using on-line and off-line features (1996).

[8] *Pattern Recognition* (1999).

[9] LEHMBERG S., WINKLER H.J. & LANG M., A soft-decision approach for symbol segmentation within handwritten mathematical expressions (1996).

[10] ELMS A.J., *The Representation and Recognition of Text Using Hidden Markov Models*, PhD Thesis (1996).

position des formules mathématiques. Pour ne citer ici qu'un exemple, le système à base de chaînes de Markov pour la reconnaissance des caractères manuscrits doit tenir compte de la diversité de taille des caractères, en introduisant des modèles pour les différents types de caractères :

- modèle à 12 états pour les grands symboles (grandes parenthèses, Σ, Π, etc) et les lettres majuscules,
- modèle à 8 états pour les minuscules,
- modèle à 3 états pour les petits symboles (point, virgule, etc).

Une réflexion a été faite sur la possibilité d'améliorer la reconnaissance des caractères dans le contexte spécifique des formules. Nous n'avons toutefois pas trouvé de solution concrète, l'absence de redondance de l'information dans une notation étant un obstacle majeur à l'utilisation des heuristiques du type de celles mises en œuvre dans le cas de documents textuels (utilisation de dictionnaires ou de statistiques par exemple). En conséquence, nous avons opté pour une architecture logicielle constituée de modules que nous présenterons plus en détail dans le chapitre II. Mais présentons tout d'abord le dernier aspect du processus de reconnaissance : la reconnaissance structurelle.

7 *Reconnaissance de la structure*

La reconnaissance des symboles ayant été effectuée, nous disposons alors d'un ensemble de caractères avec leurs propriétés respectives, à savoir principalement la localisation mais aussi éventuellement la taille, le style, etc. Le problème devient alors : comment construire l'arbre de syntaxe abstraite à partir des données géométriques caractérisant la notation que l'on souhaite reconnaître. La figure 9 illustre la transition nécessaire entre la représentation géométrique de la notation et son arbre de syntaxe.

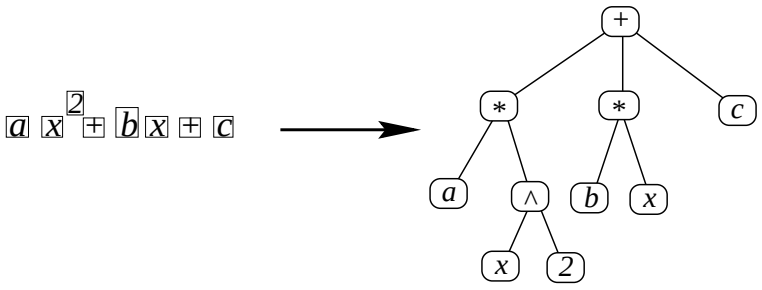


FIG. 9 – Arbre syntaxique de l'expression $ax^2 + bx + c$

Pour effectuer ce traitement, plusieurs phases sont indispensables, comme l'identification des relations spatiales et logiques entre les symboles, ainsi que l'analyse de ces différents liens, pour en déduire l'arbre de syntaxe de la formule.

7.1 Identification des relations spatiales et logiques entre les symboles

L'identification des relations indice, exposant, ou de toute autre notation utilisant un regroupement implicite (sans opérateur explicite), est complexe et problématique. Les exemples d'erreurs cités dans [**?**][1] sont principalement dus à une mauvaise reconnaissance des indices et exposants.

Z. X. Wang et C. Faure [**?**][2] proposent une méthode basée sur l'étiquetage statistique des liens entre les symboles, dans le but de déterminer les relations d'indices et d'exposants. Cet article met en évidence la difficulté de la tâche et propose une approche intéressante du problème. Étant donnée une séquence de boîtes englobantes pour les symboles reconnus, le but est d'étiqueter les relations entre paires d'éléments en indiquant en position "exposant" (E), sur la même ligne (L), ou en position indice (S) : voir figure 10.

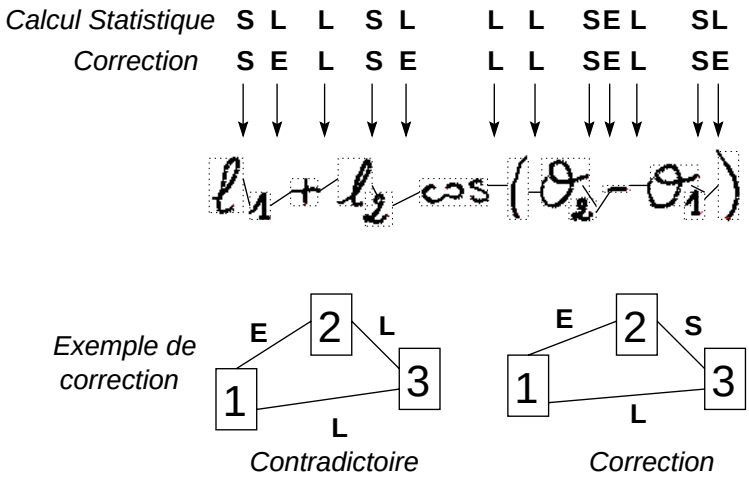


FIG. 10 – Étiquetage statistique des liens entre symboles

Un ensemble de données composé de 35 expressions produites par 7 utilisateurs est utilisé pour entraîner le système. Chaque caractéristique est mesurée pour chacune des paires de boîtes englobantes : le ratio pour la hauteur et le décalage vertical. Toutes les liaisons entre symboles doivent être étiquetées, que ceux-ci soient adjacents ou non. Pour illustrer l'utilité de cette contrainte, prenons l'exemple suivant pour lequel l'étiquetage est identique (E suivit de S) : x^{a_i} et $x^a i$. Pour distinguer ces deux expressions, la relations x-i doit être connue (E dans le premier cas, L dans le deuxième). Le processus ne tient compte que de la position des symboles (leur boîte englobante), mais en aucun cas de l'identité du symbole, ou encore de la ligne de référence du caractère, d'où de possibles ambiguïtés ; c'est le cas lorsque les boîtes englobantes ont des configurations similaires, mais que les relations spatiales entre

[1] OKAMOTO M. & TWAAKYONDO H.M., Structure analysis and recognition of mathematical expressions (1995).

[2] WANG Z.X. & FAURE C., Structural analysis of handwritten mathematical expressions (1988).

les symboles sont différentes : $\mathcal{y}c$ et bc. Ce système d'étiquetage entre pour partie dans la reconnaissance globale d'expressions, système exposé dans [**?**][3].

Une approche différente est mentionnée dans [**?**][1] et [**?**][2]. La reconnaissance est là aussi basée sur les coordonnées des objets, mais également sur un contexte local. Des configurations d'éléments sont reconnues comme légales ou illégales ; a^x et x^n sont des notations légales alors que A_*, $B_.$ ou $x^!$ ne le sont pas.

Une fois la reconnaissance des relations spatiales et/ou logiques réalisée entre les différents symboles composant une expression mathématique, il est nécessaire d'effectuer une reconnaissance de la structure de la formule. Prenons la formule suivante pour illustrer la différence d'interprétation qui peut être faite pour la notation :

$$x + y$$

Deux types d'interprétations sont possibles :
- une description de la représentation : x est sur la même ligne et à gauche d'un $+$, et y sur la même ligne et à droite de ce symbole $+$,
- et avoir reconnu la somme de x et de y qui révèle qu'une analyse plus fine de la représentation a été réalisée ; le sens de la notation a été extrait.

Une étape d'analyse structurelle est nécessaire pour passer du premier niveau de reconnaissance d'une formule au deuxième, où la formule est identifiée en tant que telle.

7.2 Reconnaissance de la structure de la formule

Le regroupement de symboles en sous-expressions est le problème central de la reconnaissance des expressions mathématiques. Un des points les moins étudiés, est la reconnaissance des notations matricielles. Deux approches sont toutefois à citer. Dans [**?**][1], l'analyse consiste à repérer une paire de délimiteurs de la même taille et du même type. Une projection horizontale est effectuée pour la zone ainsi délimitée ; si la projection met en évidence plusieurs lignes, une projection verticale est réalisée pour séparer les différentes colonnes. Chaque élément composant la matrice peut ainsi être identifié séparément. Dans [**?**][2], le point de départ est aussi la recherche de symboles délimitants, mais les symboles présents

[3] WANG Z.X. & FAURE C., Automatic perception of the structure of handwritten mathematical expressions (1990).

[1] LEE H.J. & WANG J.S., Design of a mathematical expression recognition system (1995).

[2] OKAMOTO M. & TWAAKYONDO H.M., Structure analysis and recognition of mathematical expressions (1995).

[1] OKAMOTO M. & TWAAKYONDO H.M., Structure analysis and recognition of mathematical expressions (1995).

[2] LEE H.J. & WANG J.S., Design of a mathematical expression recognition system (1995).

dans cette zone sont groupés suivant leur proximité ; on peut noter qu'aucune information n'est donnée sur les propriétés métriques utilisées.

Outre le cas de notations particulières, comme la notation vectorielle ou matricielle, plusieurs méthodes ont été étudiées dans le but de reconnaître des expressions mathématiques. Les méthodes syntaxiques se fient à l'analyse lexicale, dans le but de déterminer le regroupement correct en sous-expressions. Le découpage par projections exploite l'existence de l'espacement entre les symboles pour extraire, assez efficacement, la structure d'une expression. Nous allons étudier les différents cheminements qui ont été utilisés pour tenter de résoudre ce problème.

8 Diversité des approches existantes

Les critères détaillés dans le paragraphe 4 montrent bien que la reconnaissance de formules mathématiques est un cas très particulier. Plusieurs recherches ont été effectuées afin de tenter de résoudre le problème de la reconnaissance de la structure d'une expression mathématique en ne disposant au départ que de l'image de cette formule. Nous allons essayer de les répertorier en classant les approches utilisées par thèmes, en citant les avantages ou inconvénients de chacunes d'elles, ainsi que les principaux problèmes rencontrés.

8.1 Diversité des méthodes pour l'analyse structurelle

Comme nous avons pu le voir, le regroupement de symboles en sous-expressions est un des problèmes centraux de la reconnaissance des expressions mathématiques. Nous avons classé les différentes approches utilisées en trois catégories :

- les méthodes syntaxiques à base de grammaires de coordonnées, de grammaires probabilistes ou de grammaires de graphes,
- les méthodes logiques,
- les méthodes ne dissociant pas l'analyse géométrique de l'analyse syntaxique.

8.1.1 Méthodes syntaxiques

Une stratégie commune à la plupart des méthodes de reconnaissance des expressions mathématiques est l'utilisation, sous une forme ou sous une autre, d'une grammaire bidimensionnelle. L'analyse syntaxique est adaptée aux notations mathématiques car celles-ci subdivisent en primitives (les symboles), ont une structure récursive et une syntaxe plutôt bien définie comparée aux autres notations schématiques. Les règles de grammaires sont utilisées pour effectuer le regroupement correct des symboles mathématiques et pour définir la signification du résultat de ce regroupement.

Grammaire de coordonnées

L'un des premiers travaux sur la reconnaissance structurelle de notations mathématiques est celui de R. Anderson à la fin des années 1960. Il présente, dans sa thèse de doctorat [**?**][1], une méthode basée sur une grammaire de coordonnées. Ce travail fournit une excellente illustration des méthodes de reconnaissance syntaxique, tant au niveau théorique que pratique ; il y présente en effet, de manière très précise, les règles de sa grammaire. La figure 11 présente six règles extraites de [**?**][1]. Chaque règle s'articule autour d'un opérateur. Les régions grisées sont les zones où la règle recherche les opérandes.

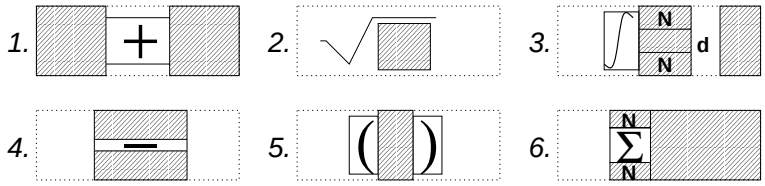


FIG. 11 – Exemple de règles de la grammaire de coordonnées

Une grammaire de coordonnées opère sur un ensemble de symboles, contrairement aux grammaires textuelles traditionnelles qui travaillent sur une chaîne de caractères. Les terminaux et les non-terminaux de la grammaire ont des attributs, par exemple les coordonnées de la boîte englobante, le centre de l'objet, ainsi qu'une chaîne de caractères m représentant le sens donné à l'objet. L'interprétation finale de la formule est contenue dans cette chaîne m. L'analyse syntaxique est descendante, chacune des règles de production de la grammaire spécifiant comment subdiviser un ensemble de symboles en sous-ensembles. Dans le cas d'une règle analysant la division (figure 11, règle 4), celle-ci spécifie la subdivision en sous-éléments : une barre horizontale (de type termediv), un ensemble de symboles au dessus et un autre au dessous de la barre de fraction (de type expression). Si cette partition échoue, l'analyseur essaye une autre alternative, c'est à dire la règle suivante de production. En cas de réussite, l'analyseur a désormais deux nouveaux sous-buts syntaxiques à satisfaire.

Cette méthode d'analyse descendante a toutefois des inconvénients, bien qu'elle fournisse une approche claire et bien structurée. La vitesse de l'analyse syntaxique est un désavantage, particulièrement dans le cas de la reconnaissance de notations implicites. En effet, ces règles n'ont pas de symbole terminal permettant de les identifier rapidement, et de nombreux essais d'application de règles de partition doivent être effectuées avant de trouver la bonne. Autre problème, lié à ces méthodes d'analyse à l'aide de grammaires : le rajout d'un mécanisme de récupération des erreurs.

[1] ANDERSON R.H., *Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics*, PhD Thesis (1968).

Plus récemment, A. Belaïd [**?**][1] et J. P. Haton [**?**][2] ont défini une extension aux grammaires de coordonnées en proposant un système avec une analyse à la fois descendante et ascendante. Dans le cas d'une incertitude lors de la reconnaissance d'un caractère, une liste de candidats potentiels est proposée avec leurs probabilités respectives. L'utilisation de l'information contextuelle durant l'application des règles permet de déterminer le bon candidat. L'analyseur est, par exemple, capable de choisir entre une parenthèse "(" et un "C" majuscule. Toutefois, certains cas restent non résolus, notamment lorsque les différentes alternatives ont un rôle syntaxique similaire. L'intervention de l'utilisateur est alors nécessaire.

Y. Dimitriadis, J. Coronado et C. de la Maza [**?**][3] proposent aussi une extension aux grammaires de coordonnées pour y ajouter la détection et la correction d'erreurs. Le système est basé sur une grammaire de coordonnées attribuée, avec une analyse ascendante. Malheureusement, cet article ne fait pas état de résultats concrets mais de spéculations sur la méthode. De plus, chaque symbole est représenté par une série de candidats potentiels, avec pour chacun une probabilité correspondant à la confiance accordée à la reconnaissance. L'analyseur prend les candidats avec les meilleurs probabilités, et si le résultat n'est pas satisfaisant il s'intéressera au candidat suivant. Il n'est en aucun cas fait état de l'explosion combinatoire d'une telle méthode. Chou [**?**][4] propose aussi une solution au problème de détection et de correction d'erreurs sur la reconnaissance d'un symbole. La méthode employée est basée sur une grammaire probabiliste (voir page 45). La complexité de l'analyseur syntaxique est $O(n^3)$. Cette approche a toutefois l'inconvénient d'être très sensible à l'angle de rotation qu'a pu subir le document lors de la numérisation, et elle est aussi très gourmande en temps de calcul. Ceci est principalement dû à la complexité de l'algorithme mis en œuvre.

Schémas de spécification de structure

La méthode de Chang est basée sur des schémas de spécification de structure. Cette technique est en fait une restriction de l'approche syntaxique classique, et a été mise au point pour rechercher efficacement la structure d'une expression. La complexité de l'algorithme mis en place est en $O(n^2)$, n étant le nombre de symboles. Les symboles sont considérés comme étant correctement reconnus.

La spécification de la structure par schémas est basée sur l'existence d'opérateurs qui

[1] BELAÏD A., *Reconnaissance Structurelle de Caractères Manuscrits et de Formules Mathématiques*, PhD Thesis (1979).

[2] BELAÏD A. & HATON J.P., A syntactic approach for handwritten mathematical formula recognition (1984).

[3] DIMITRIADIS Y.A., CORONADO J.L. & DE LA MAZA C., A new interactive mathematical editor, using on-line handwritten symbol recognition, and error detection-correction with an attribute grammar (1991).

[4] CHOU P., Recognition of equations using a two-dimensional stochastic context-free grammar (1989).

subdivisent l'image en plusieurs sous-images. Le schéma est défini comme un ensemble de règles de subdivisions, une par opérateur, qui indique les relations spatiales entre les différents composants. La précédence des opérateurs est utilisée pour déterminer l'ordonnancement des opérateurs les uns par rapport aux autres.

L'inconvénient de cette technique est qu'elle repose uniquement sur l'identification d'opérateurs spécifiques, donc explicites. Elle ne permet pas l'analyse d'opérateurs implicites comme les multiplications implicites, les exposants, indices, matrices, etc. Elle est donc très limitée au regard de l'approche syntaxique classique.

Grammaires probabilistes

Les grammaires probabilistes sont reconnues comme ayant de bons résultats dans le domaine de l'analyse de notations typographiées, et ont aussi été utilisées dans celui des notations manuscrites. Une approche probabiliste peut être ajoutée à tous les types de grammaires ; deux exemples de systèmes utilisant cette approche sont ceux développés par P. Chou [**?**][1] et, Miller et Viola [**?**][2].

Dans une grammaire probabiliste, on associe à chaque règle une probabilité que la production soit appliquée. Ainsi, pour chaque séquence de production dans une analyse donnée, une probabilité globale peut être calculée. L'analyse correcte d'un ensemble de symboles est l'analyse qui obtient la plus forte probabilité.

L'utilisation de cette approche nécessite donc l'association d'une probabilité à chacune des règles de la grammaire. Plusieurs types d'algorithmes ont été développés pour réaliser cette association. P. Chou [**?**][1] décrit l'adaptation de l'algorithme "inclus/exclus", originellement dédié à une syntaxe linéaire, au cas d'une grammaire bidimensionnelle utilisant la concaténation d'opérateurs verticaux et horizontaux. Cet algorithme effectue plusieurs passes sur la grammaire et un ensemble d'exemples issus de ce langage, déterminant ainsi une probabilité pour chacune des règles de la grammaire.

On notera aussi que ces grammaires probabilistes sont intéressantes d'utilisation dans le cadre de l'analyse géométrique, associant une probabilité à une relation entre deux symboles ou sous-expressions. Cette approche vient contrebalancer l'approche classique qui juge si un type de lien est valide ou non. C'est l'approche qui a été utilisée par Miller et Viola ; la probabilité que deux éléments soient en relation l'un avec l'autre est définie par une distribution de type Gaussienne bidimensionnelle, autour de la position supposée de la deuxième expression. Un autre avantage de cette méthode est l'utilisation d'une alternative dans la

[1] CHOU P., Recognition of equations using a two-dimensional stochastic context-free grammar (1989).

[2] MILLER E.G. & VIOLA P.A., Ambiguity and constraint in mathematical expression recognition (1998).

forme reconnue, associant à chacune des formes un indice de confiance. Ainsi, l'analyseur peut lui-même choisir entre les différentes alternatives possibles pour un symbole donné dans les cas ambigus.

L'utilisation des grammaires probabilistes semble donc être l'une des deux meilleures approches utilisées dans la reconnaissance de notations typographiées ou manuscrites. L'ajout de l'information probabiliste peut être utilisée tant au niveau de l'analyse géométrique que de l'analyse syntaxique, rendant le processus plus homogène et plus sûr.

Toutefois, des limites à cette méthode sont à signaler, en particulier pour le positionnement des symboles, qui est primordial : la ligne de référence ne doit pas varier de plus de quelques pixels, la fonte et la taille de la fonte doivent être connues. L'inclusion de symboles, comme dans le cas d'une racine carrée, n'est pas prise en compte ; seules les concaténations verticales et horizontales le sont.

Grammaires de graphes

Les méthodes à base de grammaires de graphes ont été introduites vers la fin des années 1960, afin de résoudre des problèmes pratiques de traitement d'images, comme la reconnaissance de symboles manuscrits, de schémas de circuits électroniques... Cette méthode consiste à réécrire un graphe ou un sous-graphe et à itérer ce processus. Les règles de la grammaire permettent de spécifier le type de sous-graphe recherché et la manière dont celui-ci sera remplacé par le processus de réécriture. Le graphe est construit à partir des éléments reconnus qui composent la formule.

Les travaux utilisant cette approche dans le cadre de l'analyse de formules mathématiques sont peu nombreux et récents. On citera les travaux précurseurs de L. Pottier [**?**][1]. D. Blostein et A. Grabvec [**?**][2] se sont aussi intéressées au problème de la reconnaissance de la structure des expressions mathématiques, en adaptant leur méthode testée sur l'analyse de partitions musicales. Nous n'avons toutefois pas pu obtenir d'autres informations sur la méthode ou ses applications que celles fournies dans l'article cité.

C'est une méthode à base de grammaires de graphes que nous avons étudiée avec L. Pottier, afin de l'enrichir et d'évaluer son potentiel pour la reconnaissance de notations mathématiques, dans le cadre typographié et manuscrit. Nous ne fournissons pas d'informations exhaustives dans cette partie de notre exposé. Le lecteur intéressé par cette méthode pourra se reporter aux chapitres suivants, qui décrivent dans le détail la mise en pratique ainsi que les avancées auxquelles nous avons contribué.

[1] POTTIER L., Reconnaissance de formules planes, Technical Report (1994).

[2] BLOSTEIN D. & GRABVEC A., Mathematics recognition using graph rewriting (1995).

8.1.2 Méthodes logiques

Z. X. Wang et C. Faure présentent un système à base de règles de modélisation des connaissances [**?**][1]. Elles sont organisées en plusieurs catégories :

- connaissances mathématiques : syntaxe des expressions mathématiques,
- connaissances lexicales : relatives aux entités lexicales et à la forme des symboles,
- connaissances sur le processus de l'écriture : ordre des mouvements conduisant à un motif,
- connaissances procédurales : par exemple, les procédures qui réalisent la segmentation des données selon des critères de morphologie,
- connaissances empiriques : qui sont fournies par les concepteurs du système.

L'interprétation d'une expression mathématique est le résultat d'un processus complexe mettant en œuvre le traitement de différents types d'informations. Le processus complet de résolution peut être subdivisé en plusieurs sous-buts. L'application est composée de plusieurs modules : l'acquisition des données, la segmentation de données, la segmentation suivant les connaissances et la labellisation des données. La simulation d'un processus naturel de reconnaissance par un processus artificiel requiert une connaissance des différentes informations à traiter et la manière dont elles doivent l'être. La description structurelle d'une expression est un des sous-buts à atteindre pour son interprétation. Il est donné priorité à la reconnaissance structurelle de l'expression par rapport à la reconnaissance des symboles la constituant.

Le principal inconvénient d'un système à base de règles est la faible structuration des connaissances que l'on a. En revanche, c'est un moyen simple et naturel de traduire une grande quantité de connaissances. Mais on représente toujours la connaissance d'un domaine très précis, et on ne peut se permettre des assertions floues. Les systèmes experts, dont une partie au moins de la base de connaissances est représentée sous cette forme, sont construits pour fonctionner sur une application bien ciblée. Même si l'on peut penser, au premier abord, qu'un système de règles de production est très modulaire, cette idée se révèle fausse. Il est en effet difficile de maintenir la cohérence de la base de règles. La priorité et l'ordonnancement des règles déterminent l'efficacité de la résolution du problème, ainsi que la réduction de conflits entre les diverses connaissances. Cette réorganisation est, dans certains cas, un obstacle à l'utilisation d'une base de règles.

[1] WANG Z.X. & FAURE C., Automatic perception of the structure of handwritten mathematical expressions (1990).

8.1.3 Méthodes mixtes liant analyse géométrique et syntaxique

Méthodes procédurales

Le système d'interprétation des notations mathématiques de H. J. Lee [**?**][1], [**?**][2], [**?**][3] a été développé en introduisant l'ensemble des connaissances dans des procédures. On parle alors de représentation procédurale de la connaissance. L'avantage de cette méthode est sa rapidité, mais au détriment de la réutilisabilité ou de l'extension vers d'autres notations.

Le système décrit par M. Okamoto combine le découpage par projection (cf. paragraphe suivant) et la méthode procédurale pour éliminer les erreurs engendrées par la première étape de découpage. L'article propose une illustration de la difficulté rencontrée à réaliser un système pour la reconnaissance des expressions mathématiques, étant donné le nombre très élevé des configurations des symboles. Les auteurs justifient ce choix par la facilité à mettre en œuvre une définition implicite de la syntaxe en utilisant la méthode procédurale, plutôt qu'une définition syntaxique explicite à base de règles de grammaire.

Mais, inévitablement, ces procédures sont très fortement liées à l'application traitée, et le processus de reconnaissance a entièrement perdu sa généralité. On évoque dans ce cas, et à juste titre, les expressions ad hoc, ou codage en dur de l'application. De ces quelques points émergent de façon évidente les problèmes posés par cette représentation : connaissance morcelée et éparpillée dans un ensemble de procédures, absence de modularité et de généralité du modèle. L'approche que l'on a du problème est complètement figée, et les mises à jour s'avèrent très souvent laborieuses. Cette approche va à l'encontre des aspects de généricité, d'adaptabilité et de réutilisatbilité, ce qui s'avère un défaut majeur à son emploi.

Méthode de découpage par projections

Le découpage par projections est une méthode qui est largement utilisée dans le domaine de la reconnaissance de documents. C'est le cas, par exemple, pour la reconnaissance de partitions musicales, ou bien encore pour l'analyse de la structure de documents afin de déterminer une mise en page en colonnes, ou bien encore les paragraphes qui composent le document.

Afin d'analyser la structure d'une expression mathématique, M. Okamoto ([**?**][1], [**?**][2]

[1] LEE H.J. & LEE M.C., Understanding mathematical expressions in a printed document (1993).

[2] LEE H.J. & LEE M.C., Understanding mathematical expressions using procedure-oriented transformation (1994).

[3] LEE H.J. & WANG J.S., Design of a mathematical expression recognition system (1995).

[1] OKAMOTO M. & MIAO B., Recognition of mathematical expressions by using the layout structure of symbols (1991).

[2] OKAMOTO M. & MIYAZAWA A., An experimental implementation of a document recognition system

et [**?**][3]) propose d'appliquer cette méthode récursivement, en alternant successivement projections verticales et horizontales, jusqu'à ce que plus aucune subdivision ne soit possible. Cette méthode décompose l'arrangement structurel des symboles constituant l'expression, et ceci avec une analyse très peu coûteuse (cf figure 12). Le résultat obtenu est un arbre des relations spatiales entre les éléments. Des traitements spécifiques sont utilisés afin de reconnaître certains types de notations que la méthode ne permet pas d'analyser. C'est le cas de la racine carrée, des notations de type indice, exposants, etc.

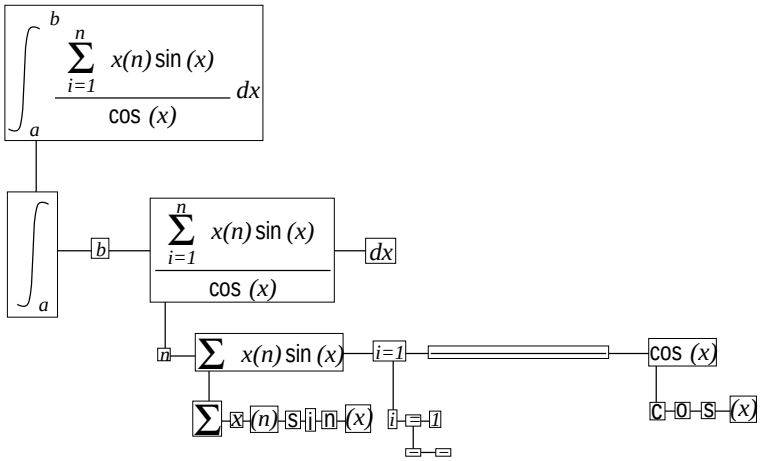


FIG. 12 – Exemple de découpage par projections

La même méthode a aussi été utilisée par Z. X. Wang et C. Faure [**?**][4] dans leur système de reconnaissance des expressions manuscrites. Ils décrivent aussi les limites de cette technique et les cas d'échecs rencontrés [**?**][5]. Nous citerons deux exemples révélateurs des deux principaux problèmes :

- l'analyse d'une relation d'inclusion (comme pour l'expression $\sqrt{x^2 + y^2}$) fait l'objet d'un cas particulier, ce type de relation n'étant pas pris en compte par le processus normal,
- la superposition de symboles, courante dans le cas de l'écriture manuscrite, empêche l'application du processus et donc la reconnaissance de l'expression. Cette superposition peut être visible (figure I.13(a)) ou n'être révélée qu'au moment de la projection (figure I.13(b)).

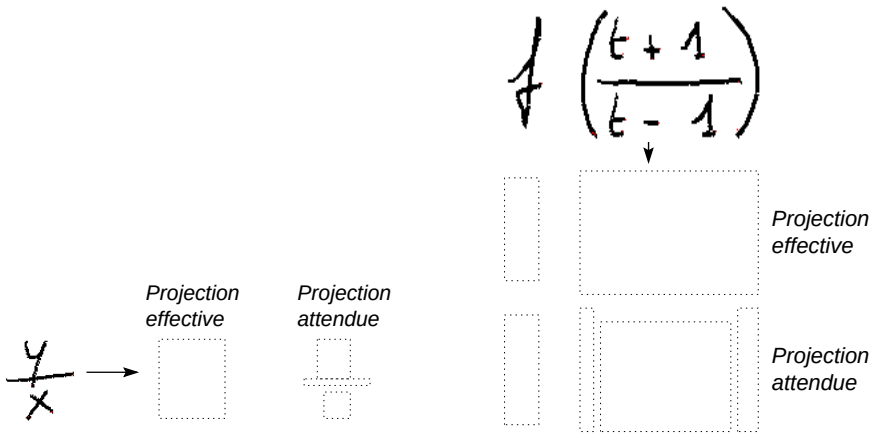
J. Ha, R. Haralick et I. Phillips proposent un autre système basé sur une décomposi-

for papers containing mathematical expressions (1992).

[3] OKAMOTO M. & TWAAKYONDO H.M., Structure analysis and recognition of mathematical expressions (1995).

[4] WANG Z.X. & FAURE C., Structural analysis of handwritten mathematical expressions (1988).

[5] WANG Z.X. & FAURE C., Automatic perception of the structure of handwritten mathematical expressions (1990).



(a) Superposition “visible” (b) Superposition “invisible”

FIG. 13 – Problèmes soulevés par la méthode de projections

tion récursive en X-Y [**?**][1]. Fondé sur les attributs des primitives, l’arbre de syntaxe d’une expression initiale est construit en utilisant une approche descendante. Puis une approche ascendante permet de vérifier les erreurs de syntaxe.

Un système complet basé sur le découpage par projections entraînerait l’ajout massif de cas particuliers, afin de prendre en compte les notations qui ne peuvent pas être reconnues directement à l’aide de cette méthode ; c’est particulièrement le cas d’une notation comme la racine carrée (inclusion). Enfin, l’utilisation de cette méthode dans le cas manuscrit montre rapidement ses limites : le processus étant basé sur l’alignement vertical et horizontal des éléments, tout défaut de positionnement inhérent à l’écriture manuscrite entraîne une impossibilité d’analyser correctement la notation. Cette méthode simple est donc efficace et rapide pour l’analyse d’une catégorie restreinte de notations mathématiques, mais ne peut être employée dans le cadre d’une utilisation sur des cas concrets.

8.2 Difficultés pour comparer les approches

La reconnaissance des notations mathématiques dans un document est un problème significatif dans la reconnaissance de documents composites (documents composés de textes, graphiques, photographies, schémas, notations mathématiques, etc.). Durant les trente dernières années, des études ont été effectuées sur le sujet avec des approches diverses. Mais la reconnaissance et l’interprétation de ces notations est complexe, il est difficile de comparer les différentes méthodes ou techniques développées pour résoudre le problème. Il est

[1] HA J., HARALICK R.M. & PHILLIPS I.T., Understanding mathematical expressions from document images (1995).

aussi à signaler qu'aucune implémentation de ces méthodes n'a pu être obtenue auprès des auteurs, ce qui ne facilite pas cette tâche de comparaison. Nous avons dû nous contenter des publications comme seule et unique source d'informations. Mais d'autres facteurs sont aussi à prendre en compte si l'on parle de difficultés à comparer les différentes approches.

8.2.1 Diversité dans les données traitées

Comme nous avons pu le voir dans les paragraphes précédents, la première difficulté provient de la diversité et de la variété des données à traiter. Les travaux effectués concernent aussi bien la reconnaissance de notations manuscrites capturées en temps réel, le plus souvent à l'aide d'une tablette à digitaliser, que des notations typographiées, en ayant recours à la numérisation. Certains systèmes se contentent de récupérer une liste de symboles déjà reconnus et leurs positions, ou bien intègrent la reconnaissance des symboles, avec une partie effectuant un traitement de l'image plus ou moins poussé (suppression du bruit, réorientation, etc.). Enfin, la dernière caractéristique des applications est la nécessité ou non de devoir isoler la formule du reste du document, dans le cas où celle-ci serait incluse dans un texte par exemple. Cette grande diversité dans la définition du problème ne permet pas d'effectuer une comparaison sérieuse sur l'efficacité de ces systèmes hétéroclites.

8.2.2 Grande diversité des approches

De nombreuses méthodes ont été étudiées pour la reconnaissance des expressions mathématiques. L'analyse syntaxique met en œuvre une grammaire de type bidimensionnelle, comme c'est le cas dans les grammaires de coordonnées. La reconnaissance des symboles est un pré-requis à l'application de cette méthode. Elle fournit une représentation claire et explicite des notations mathématiques reconnues, même si l'analyseur a une complexité élevée et nécessite donc des temps de calculs parfois importants. Le découpage par projections, horizontales et verticales, détermine la structure de l'expression en subdivisant récursivement l'image en sous-régions. La reconnaissance des symboles peut être effectuée a posteriori. La stratégie procédurale n'utilise pas de méthodologie bien définie et sans une organisation stricte des règles, le programme devient rapidement confus et difficile à étendre à d'autres notations.

Autant d'approches différentes, qu'il est difficile de comparer à la fois quantitativement et qualitativement.

8.2.3 Diversité des notations mathématiques

Les systèmes de reconnaissance ne considèrent pas tous les mêmes classes de notations mathématiques. La plupart d'entre eux se sont concentrés sur la reconnaissance des notations arithmétiques, ou même ont spécialisé leurs recherches sur une classe très particulière

de notations. C'est le cas de R. Fateman [**?**][1], [**?**][2] qui étudie les solutions pour la reconnaissance des intégrales, dans le but de développer un logiciel capable d'analyser des tables d'intégrales déjà imprimées dans un document. Assez peu de recherches ont été effectuées sur les notations matricielles et très peu de systèmes sont suffisamment généralistes, dans la méthodologie employée, pour être capables de reconnaître une nouvelle classe de notations mathématiques. C'est tout particulièrement le cas de la méthode procédurale, où il serait nécessaire de réécrire presque complètement l'analyseur.

Une recherche est nécessaire pour clarifier la définition des notations et des "dialectes" mathématiques. Une telle définition permettrait de disposer d'une base solide de comparaison des différents systèmes de reconnaissance de notations mathématiques.

9 Conclusion

Une étude approfondie de la bibliographie concernant la reconnaissance structurelle des formules mathématiques met en évidence l'engouement récent pour ce domaine de recherche. En effet, plus de 75% des références datent de moins de 10 ans et plus de 50% de moins de 5 ans ; c'est tout dire quant à l'intérêt porté au sujet. Le nombre important de ces travaux peut s'expliquer par deux facteurs. Tout d'abord, la capacité de traitement des ordinateurs rend possible une application directe des résultats obtenus ; le faible coût des périphériques nécessaires (scanner ou tablette à digitaliser) favorise l'utilisation par le plus grand nombre. D'autre part, les applications multiples que nous avons décrites comme une interface plus intuitive pour l'utilisateur ou la possibilité de reconnaître automatiquement les documents scientifiques, ouvrent des perspectives nouvelles dans plusieurs domaines. C'est particulièrement le cas d'une indexation automatique des documents scientifiques, avec la possibilité de faire des recherches sur une base de données de formules mathématiques.

Dans ce chapitre, nous avons tenté de décrire les difficultés inhérentes à la reconnaissance des notations mathématiques. Nous avons répertorié les différentes approches utilisées jusqu'à présent pour effectuer la reconnaissance structurelle de formules mathématiques à partir de leur représentation. Sur la base de ce constat, nous allons dégager les aspects les plus importants que doit intégrer un système dédié à la reconnaissance d'expressions mathématiques. Ces points seront développés dans les chapitres suivants en mettant l'accent sur les solutions retenues.

[1] FATEMAN R.J. & TOKUYASU T., Progress in recognizing typeset mathematics (1996).

[2] FATEMAN R.J., TOKUYASU T., BERMAN B.P. & MITCHELL N., Optical character recognition and parsing of typeset mathematics (1996).

Chapitre II

Concepts pour la reconnaissance de formules

Ce chapitre présente de façon assez générale les idées mises en pratique durant la réalisation d'*OFR*. Sa lecture doit permettre une compréhension globale des problèmes que nous avons abordés et donc fournir un cadre pour les chapitres suivants. La méthode utilisée pour la reconnaissance structurelle proprement dite sera développée dans le chapitre suivant. Les thèmes abordés ici sont approfondis dans les chapitres consacrés à la méthode choisie et à l'implémentation.

Les nombreuses applications possibles que nous avons présentées dans le chapitre I, paragraphe 2, montrent bien la nécessité d'un tel système de reconnaissance. Dans ce chapitre, nous avançons les éléments qui nous semblent essentiels à un système de reconnaissance structurelle des notations mathématiques. Peu de justifications et de conclusions sont données, ces développements étant réservés aux chapitres suivants.

Les trois caractéristiques essentielles sont la réutilisabilité, l'évolutivité, et la communication avec d'autres systèmes indépendants. Nous allons détailler pour chacune d'elles les points qui nous semblent actuellement avoir été oublié dans les diverses solutions proposées.

1 Réutilisabiblité

Le premier point que nous tenons à mettre en évidence est la nécessité d'avoir un système qui soit suffisamment paramétrable pour être réutilisé dans le cadre des diverses applications que nous avons déjà pu présenter. Parmi ces applications, nous focaliserons nos efforts sur deux d'entre elles :

– l'utilisation du système en tant qu'interface à un système manipulant des formules mathématiques,
– la possibilité de récupérer les expressions mathématiques présentes dans un certain nombre de documents imprimés.

Cette double optique de travail, qui ouvre de larges champs d'application, nécessite la mise en place d'un système suffisamment adaptable à ces différentes réalisations.

Un bon système de reconnaissance structurelle requiert un effort important de conception et d'implémentation. Il serait donc profitable de pouvoir bénéficier d'un composant facilement intégrable et paramétrable, qui fournirait les fonctions de base de l'analyse géométrique et syntaxique de l'expression à reconnaître.

1.1 Équations, matrices et notations "exotiques"

Les diverses approches utilisées, pour accomplir une reconnaissance structurelle à partir de la représentation géométrique d'une formule, ont été présentées dans le chapitre I, paragraphe 8. Il nous semble important de souligner une nouvelle fois que la majorité d'entre elles était dédié à un type de notation bien particulier. Les travaux de R. Fateman, concernant ce sujet, sont presque tous axés sur la reconnaissance des intégrales [**?**][1]. Il serait à coup sûr assez difficile d'adapter les méthodes qu'il a employées à des notations réellement bidimensionnelles, comme les notations matricielles. C'est d'ailleurs le cas de la plupart des articles qui mentionnent bien le fait que la méthode doit être modifiée pour les notations matricielles. Mais la solution retenue permet rarement cette adaptation.

Le procédé utilisé pour effectuer l'analyse structurelle doit donc prendre en compte le fait que beaucoup de notations mathématiques ne sont pas basées sur une structure semi-linéaire, comme pour les intégrales ou les équations, mais sur une notation bidimensionnelle.

1.2 Manuscrit et Typographié

La réutilisabilité du système peut être étendue à l'utilisation de la même méthode pour la reconnaissance structurelle des notations typographiées et manuscrites. Si cette hypothèse de travail n'est pas primordiale dans nos recherches, les méthodes spécialisées étant souvent

[1] FATEMAN R.J. & BERMAN B.P., Optical character recognition for typeset mathematics (1994).

plus efficaces, une étude plus approfondie pourrait tout de même être intéressante, à titre de comparaison.

Une bonne architecture logicielle doit permettre d'effectuer des tests sur les cas manuscrits et typographiés. Il serait donc profitable de pouvoir bénéficier d'un composant facilement intégrable et paramétrable, qui fournirait les fonctions de base de l'analyse géométrique et de l'analyse structurelle des notations mathématiques.

OFR a été conçu pour remplir ce rôle. Nous détaillerons, page 61, les choix effectués dans l'architecture logicielle mise en place afin de remplir cette condition de test des différentes applications possibles. Il convient toutefois de souligner que ce composant est conçu pour être employé avec plusieurs types d'applications lui fournissant les symboles, ainsi que leur position, des éléments composant l'expression. Il est donc important de pouvoir adapter ce composant aux différents logiciels que l'on pourra employer, pour fournir les informations dont l'analyse dépend.

2 Évolutivité

Un des principaux atouts dont doit bénéficier le système est la flexibilité. Il doit être conçu en permettant la modification des comportements, afin d'assurer l'évolution possible vers de nouvelles notations. Cela est indispensable pour assurer l'adaptabilité du composant aux besoins spécifiques liés à une classe particulière d'éléments à traiter.

2.1 Introduction de nouvelles notations

Les systèmes que nous avons pu étudier, la plupart du temps au travers d'articles car nous ne pouvions pas disposer d'une implémentation de la méthode présentée, sont assez, voire complètement, fermés. Nous entendons par là qu'il est impossible de modifier ou bien d'ajouter une nouvelle notation au système de reconnaissance.

Il est bien entendu indispensable que le système soit ouvert et qu'il soit possible d'ajouter la reconnaissance d'un nouveau type de notation. En effet, les notations mathématiques n'étant que semi-standardisées, ils n'est pas rare d'en voir apparaître de nouvelles. De plus, les mathématiciens, les physiciens ou encore les économistes ont des besoins différents de notations, chacun en introduisant de nouvelles selon ses besoins propres. Étant donné qu'il n'est pas concevable d'être suffisamment exhaustif lors de la conception du logiciel, il est indispensable que la méthode retenue permette une adaptation du système à moindre frais.

2.2 Adaptation aux différentes notations

Sans aller jusqu'à l'introduction de nouvelles notations, quoique celà soit parfois indispensable, un autre point est à considérer : les notations mathématiques ne sont que semi-standardisées. De la même manière qu'il est impossible d'introduire toutes les notations mathématiques imaginables, il n'est pas concevable de reconnaître d'emblée toutes les variations possibles des notations. Prenons le cas d'une intégrale. La position des bornes de l'intégrale est différente selon le cas. En LaTeX, par exemple, les bornes d'une intégrale ou d'une somme peuvent être au dessus du symbole ou bien en position d'exposant par rapport au symbole, selon le contexte :

– $\sum_{i=5}^{10} i + Y$,

– $\sum_{i=5}^{10} i + Y$.

Le système adopté devra donc être suffisamment souple pour permettre l'adaptation du logiciel à toutes les variations des notations que l'on souhaite reconnaître.

3 Interprétation de l'arbre de syntaxe

La reconnaissance des symboles ayant été effectuée, nous disposons alors d'un ensemble de caractères avec leurs propriétés. La reconnaissance structurelle consiste donc en la construction de l'arbre de syntaxe abstraite à partir des données géométriques caractérisant la notation. La figure 9, page 40, illustre la transition entre la représentation géométrique de la notation et son arbre de syntaxe. Pour effectuer ce traitement, plusieurs phases sont nécessaires, comme l'identification des relations spatiales et logiques entre les symboles, ainsi que l'analyse de ces différents liens, pour en déduire l'arbre de syntaxe de la formule.

Toutefois, l'arbre de syntaxe abstraite obtenu n'est pas suffisant pour exploiter le résultat. R. Fateman, dans [**?**][1], donne un exemple de la nécessité d'avoir des informations sur la sémantique des éléments. Si nous prenons les exemples suivants : $(a + b)(c + d)$ et $(f + g)(x + y)$, un lecteur humain interprétera sûrement de manière différente ces deux formules, même si l'arbre de syntaxe abstraite est identique.

Cette interprétation peut dépendre de plusieurs paramètres :

– d'un contexte extérieur à la formule elle-même,

– de conventions standards,

– de conventions tacites pour un groupe d'utilisateurs.

Pour notre exemple, cette interprétation de l'arbre dépendra de la signification des symboles f, g, x, y d'une part et de a, b, c, d, d'autre part. On peut penser que, par convention, f et g sont des symboles fonctionnels, x et y des variables symboliques, et a, b, c, d des

[1] FATEMAN R.J., More versatile scientific documents extended abstract (1997).

variables. Cette connaissance du type sémantique des symboles composant la formule permettra de synthétiser l'information, afin d'interpréter correctement chacun des opérateurs (en fonction du type de leur opérande).

La formule $(f+g)(x+y)$ devra donc être interprétée comme l'application des fonctions f et g à $(x+y)$ alors que $(a+b)(c+d)$ comme la multiplication implicite de $(a+b)$ et $(c+d)$.

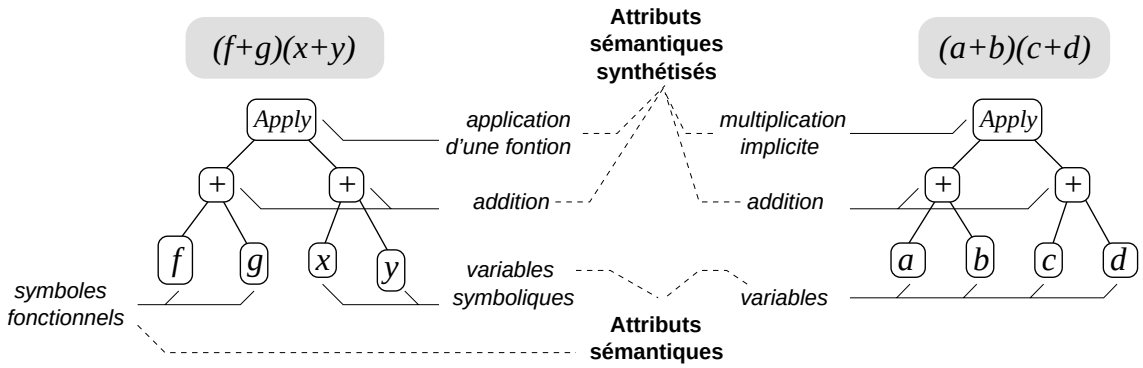


FIG. 1 – Arbres de syntaxe avec attributs sémantiques

Cette interprétation ne pourra être effectuée que si l'on connaît les conventions utilisées ou encore le contexte d'application de la formule considéré. Si la reconnaissance s'effectue dans un contexte d'analyse de document, on peut imaginer que le contexte textuel d'où a été extraite la formule, permettrait de fournir ce type d'informations. Dans le cas d'une expression isolée de son contexte, l'utilisateur devra spécifier la sémantique des symboles.

4 Communication des résultats

De la même manière que *OFR* doit permettre une utilisation avec plusieurs outils pour la reconnaissance de symboles, manuscrits ou typographiés, il doit aussi être capable de communiquer le résultat de la reconnaissance à d'autres applications.

La communication d'objets mathématiques est, en lui-même, un sujet de recherche. Afin d'éviter de disperser nos recherches et afin de pouvoir faire communiquer *OFR* avec le plus grand nombre possible d'applications, le choix du développement d'un protocole personnel n'était pas souhaitable. Nous n'avons donc pas orienté nos recherches vers l'élaboration d'un nouveau protocole de communication, même si, au début de nos recherches, aucun standard n'existait. Depuis, plusieurs travaux ont abouti à des standards de représentation et de communication d'objets mathématiques, en conservant la sémantique, comme *OpenMath*. L'intégration d'un tel protocole pour la communication des résultats est souhaitable, afin de tirer le meilleur parti des résultats obtenus et d'intégrer *OFR* à d'autres systèmes.

5 Composants pour l'analyse de formules

Pour résumer les différents points que nous venons d'aborder, nous allons présenter les composants qui nous semblent nécessaires à la réalisation de la plate-forme de tests pour la reconnaissance structurelle de formules mathématiques. La figure 2 ci-dessous, illustre les interactions entre ces modules.

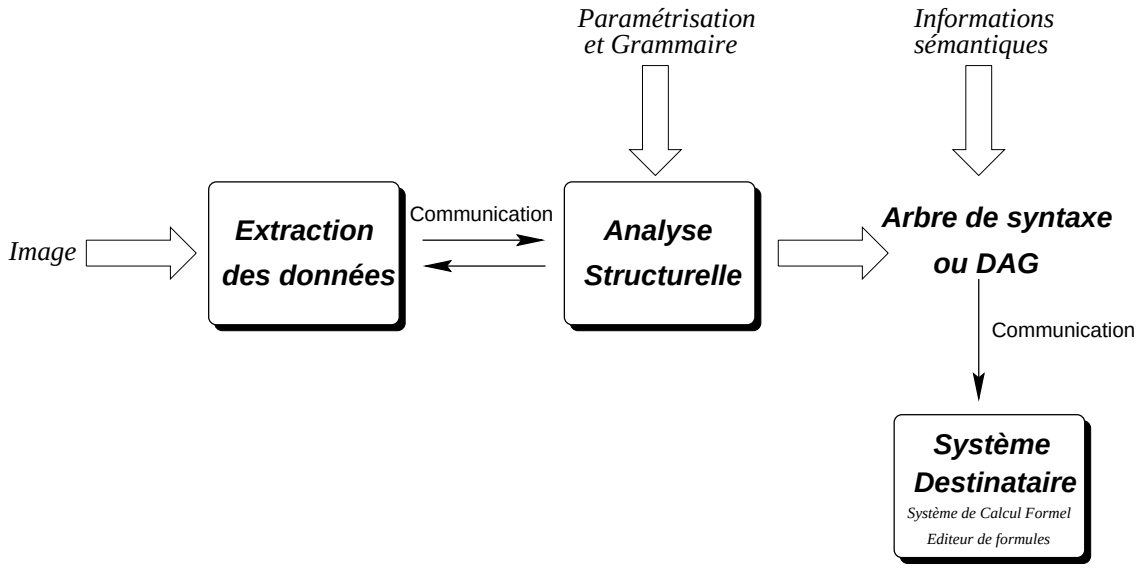


FIG. 2 – Relations entre les différents composants

5.1 Détail des différents composants

Dans le cadre de recherches sur la reconnaissance structurelle de notations mathématiques, il est souhaitable de ne pas consacrer un temps trop important aux sujets connexes, comme la reconnaissance des caractères et symboles composants la formule. Nous nous sommes donc orientés sur l'utilisation de composants pré-existants. Même si cela n'a pas toujours été possible, nous avons tenté, le plus souvent possible, d'obtenir des logiciels effectuant le travail d'extraction et de reconnaissance des symboles, aussi bien dans le cas typographié que manuscrit.

Nous avons donc opté pour un système en deux parties distinctes : la partie concernant la reconnaissance des symboles (que nous avons, autant que possible, essayé de récupérer), et la reconnaissance structurelle sur laquelle nous avons plus particulièrement travaillé.

Pour pouvoir communiquer les résultats obtenus vers d'autres applications, comme un système de calcul formel, il est nécessaire de disposer d'informations sémantiques sur les

éléments composant la formule. Cette acquisition peut être envisagée de deux manières différentes :

- l'utilisation du système dans le cadre d'une interface de saisie de formules, par une spécification de la part de l'utilisateur ;
- l'utilisation du système pour une analyse de document imprimé, comme un article scientifique, d'extraire automatiquement l'information du texte environnant la notation.

Toutefois, cette dernière solution, qui permettrait une automatisation complète du processus, nécessite une recherche plus approfondie.

5.2 Critiques possibles de l'architecture en modules

Cette architecture logicielle répartie en différents modules s'est imposée comme étant la plus adaptée à notre étude spécifique. Elle offre la possibilité de réutiliser des modules existants, pour la reconnaissance de l'écriture manuscrite par exemple, évitant ainsi une perte de temps conséquente dans le développement d'un tel module. La séparation bien marquée entre la reconnaissance des symboles, l'analyse géométrique et l'analyse syntaxique permet aussi une paramétrisation plus fine de chacun des modules, et donc de l'adapter au mieux aux besoins (cas typographié, cas manuscrit).

Toutefois, cette architecture, même si elle présente de nombreux avantages, implique aussi des limites. Parmi les méthodes présentées dans le chapitre précédent, la méthode probabiliste développée par P. Chou mène de front la segmentation et l'analyse structurelle. Ceci permet en particulier de rendre la reconnaissance des symboles et la reconnaissance structurelle très peu dépendantes du bruit que peut avoir subi l'image de la notation. Cette possibilité de réaliser de manière concurrente la reconnaissance des symboles et l'analyse structurelle n'est pas envisageable avec notre architecture en modules. Une possibilité de correction d'éventuelles erreurs sur l'identification de symboles a toutefois été intégrée dans le module d'analyse structurelle, dans la mesure où le logiciel fournissant les symboles offre des alternatives possibles à un caractère reconnu. D'autres avantages et inconvénients de ce type d'architecture sont donnés dans le chapitre sur l'implémentation (chapitre V).

6 Conclusion

Ces quelques pages ont donné un bref aperçu des différents thèmes abordés dans la suite du document. Le but principal est de fournir un cadre général, afin de faciliter la lecture des chapitres suivants. Chacune des idées exposées précédemment est reprise et illustrée dans les chapitres suivants.

Chapitre III

Méthode et outils

Après avoir présenté les recherches effectuées jusqu'à présent et avoir mis en évidence, dans le chapitre précédent, les points essentiels pour accomplir la reconnaissance structurelle des notations mathématiques, nous allons maintenant exposer les choix et les réalisations effectués.

Dans ce chapitre, nous mettons en évidence les aspects les plus intéressants de l'architecture et de la méthode mise au point pour la réalisation d'*OFR*. L'architecture pour une plus grande adaptabilité aux différentes expérimentations envisagées, l'analyse géométrique et l'analyse structurelle à l'aide de grammaires de graphes, sont les principaux thèmes abordés.

1 Architecture d'OFR

Comme nous avons pu l'exposer dans le chapitre précédent, paragraphe 5, des opérations préliminaires sont nécessaires pour disposer des informations à analyser. L'analyse de l'image et la reconnaissance des symboles est un sujet d'étude à part entière. Plusieurs solutions ont été utilisées dans les recherches précédentes : elles vont du système complètement intégré qui effectue à la fois l'identification des symboles et la reconnaissance structurelle, à la simulation de la reconnaissance des symboles par un utilisateur comme dans [**?**][1]. Cette

[1] BLOSTEIN D. & GRABVEC A., Mathematics recognition using graph rewriting (1995).

dernière méthode, si elle a le mérite d'être très simple à mettre en œuvre, n'en est pas pour autant préférable : elle masque en effet de manière trop importante les problèmes inhérents à la reconnaissance des symboles. La méthode consistant à intégrer l'analyse de l'image au processus de reconnaissance structurelle permet des résultats plus probants. Il est en effet possible d'effectuer la reconnaissance du symbole dans un contexte syntaxique.

Nous avons opté pour une solution intermédiaire, privilégiant l'utilisation d'un système déjà existant, effectuant l'analyse de l'image et l'identification des symboles. Si cette solution ne permet pas une intégration élevée de l'analyse de l'image dans un contexte syntaxique, elle a pour avantage d'être flexible et de permettre l'évaluation de la méthode dans différents contextes. Ce choix a pour conséquence de devoir mettre en place un système permettant d'unifier le type de résultats fournis au processus de reconnaissance structurelle, d'où l'introduction d'un filtre dans le schéma 1 présentant l'architecture d'*OFR*. Les données attendues sont, pour l'essentiel, les symboles identifiés et leurs coordonnées. Toutefois, lorsqu'il le permet, le système peut fournir une information supplémentaire : une liste des alternatives possibles à la reconnaissance d'un symbole, avec une probabilité associée. Ces informations seront utilisées dans le processus d'analyse syntaxique.

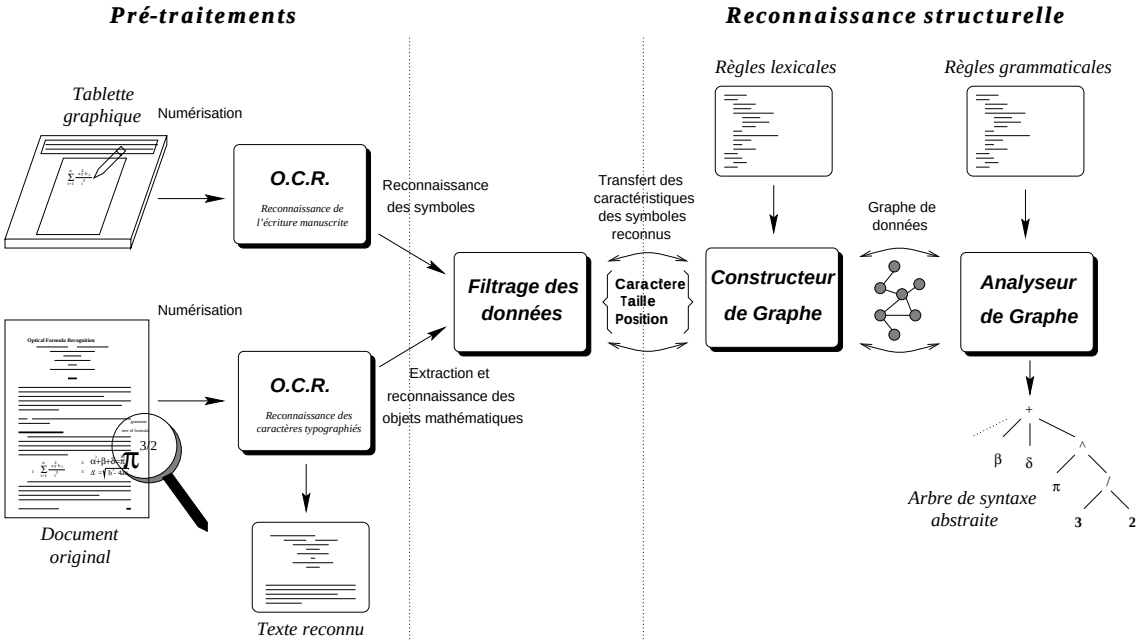


FIG. 1 – L'architecture d'*OFR*

La reconnaissance structurelle est subdivisée en deux modules : l'un effectuant l'analyse géométrique, l'autre l'analyse syntaxique. Ces deux éléments sont étroitement liés et permettent pour l'un la construction d'un graphe, pour l'autre son analyse et la production de l'arbre de syntaxe abstraite correspondant à la notation mathématique identifiée. Ces modules sont tous deux paramétrables à l'aide des règles lexicales et grammaticales qui leurs

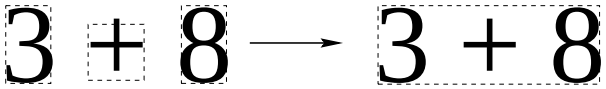
sont fournies. Nous reviendrons tout au long de ce chapitre sur ces différents points en les détaillant. Mais revenons en premier lieu sur l'analyse de l'image.

2 Analyse de l'image

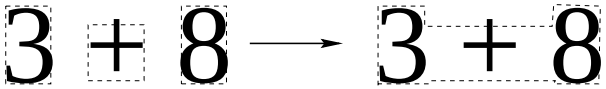
Dans ce paragraphe, nous allons détailler les étapes correspondant à l'identification des propriétés des symboles composant la notation que l'on souhaite identifier. Toutefois, nous ne décrirons pas les mécanismes mis en œuvre pour la reconnaissance du symbole lui-même, cette opération ne faisant pas partie intégrante de nos recherches. Nous nous contenterons de mettre en lumière un ensemble de présupposés nécessaires à la réalisation de l'étape suivante : l'analyse géométrique.

2.1 Enveloppe d'un symbole

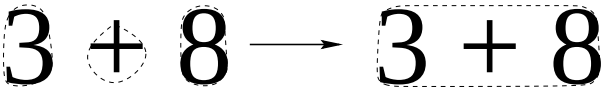
Les informations sur la zone où s'inscrit un symbole ou un groupe de symboles représentent une donnée importante. Plusieurs méthodes peuvent être utilisées pour la gestion des zones englobant les symboles. La figure 2 présente les principaux types de systèmes utilisés.



(a) Régions rectangulaires



(b) Régions rectangulaires individuelles



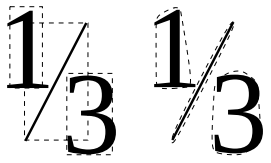
(c) Enveloppes convexes

FIG. 2 – Construction des régions englobantes

– La première approche, figure III.2(a), est celle que nous avons utilisée. Les boîtes englobantes sont les plus petits rectangles qui enserrent le symbole. La combinaison de plusieurs régions est la plus petite boîte englobant toutes les autres.
– Une variante de cette première approche, illustrée par la figure III.2(b) consiste à faire une combinaison au plus près des différentes boîtes englobantes. Cette solution, peu utilisée, n'apporte guère d'éléments supplémentaires par rapport à la première, et complique donc le problème inutilement.

– La figure III.2(c) illustre l'utilisation des enveloppes convexes. Cette méthode a été utilisée par E. Miller et P. Viola [**?**][1]. La nouvelle enveloppe convexe créée lors de la combinaison de symboles est la plus petite englobant tous les symboles.

L'un des désavantages d'utiliser les rectangles englobant par rapport aux enveloppes convexes se manifeste dans l'exemple présenté sur la figure 3.



(a) Régions rectangulaires (b) Enveloppes convexes

FIG. 3 – Problème soulevé par les boîtes englobantes rectangulaires

D'un point de vue de l'utilisateur, les caractères ne se superposent pas. Mais un système basé sur l'utilisation de rectangles englobants pourrait mal interpréter les relations géométriques qui lient ces symboles. Ce type de problème conduit, la plupart du temps, à une mauvaise interprétation de la formule ou à un échec lors de la reconnaissance syntaxique. Dans le paragraphe 4 nous présenterons les solutions mises en place pour éviter ce type de difficultés.

Toutefois, l'étude de E. Miller et P. Viola [**?**][1] montre que l'utilisation d'une solution à base d'enveloppes convexes est préférable. La création des enveloppes convexes a une complexité en $O(n \log n)$ avec n le nombre de pixels composant l'image. L'union de deux enveloppes convexes peut être évaluée en $O(l + m)$ avec l et m le nombre de nœuds des enveloppes convexes. Cette approche permet donc d'apporter une solution plus homogène que les régions rectangulaires.

Dans notre optique de recherche, nous n'avons pas pu bénéficier des avantages de l'utilisation des enveloppes convexes afin de décrire les contours du symbole. Ce choix a principalement été motivé par la volonté de ne pas investir de temps dans un logiciel de reconnaissance optique des symboles, et donc de s'adapter à la forme la plus standard possible.

[1] MILLER E.G. & VIOLA P.A., Ambiguity and constraint in mathematical expression recognition (1998).

2.2 Résultat type attendu de l'OCR

Les informations attendues de la part du logiciel de reconnaissance de caractères ont donc une forme générique du type boîte englobante et symbole. Par exemple, pour la formule suivante,

$$\sum_{n=0}^{\infty} \frac{z^n}{n!}$$

le type d'informations fournies par le logiciel sera :

```
31 0 86 25 infty
5 50 114 166 Sigma
150 36 184 74 z
192 17 227 42 n
146 106 234 110 hbar
155 149 199 187 n
210 127 219 169 exclam
210 177 219 186 dot
0 201 35 226 n
42 203 83 206 hbar
42 217 83 220 hbar
92 188 117 227 0
```

Pour pouvoir adapter la méthode à la reconnaissance de notations manuscrites et typographiées, nous ne ferons aucun présupposé sur un ordre quelconque de saisie des symboles. L'ordre d'apparition de ces informations n'est donc pas pris en compte. Seules comptent les informations géométriques.

Cette information est l'information minimale attendue. Elle pourra être enrichie d'alternatives possibles pour chacun des symboles reconnus, avec un indice de confiance de la reconnaissance associé à chacun des symboles. Dans l'exemple ci-dessous, se référant toujours à la même formule, des alternatives sont fournies pour certains symboles. On retrouve tout d'abord les coordonnées de la boîte englobante correspondant au symbole, suivies du symbole et d'un indice d'erreur sur la reconnaissance. Plus cet indice est proche de zéro, plus le système est assuré de fournir la bonne solution.

```
31 0 86 25 infty
5 50 114 166 Sigma
150 36 184 74 z 0.1 Z 0.7
192 17 227 42 n 0.15 u 0.9
146 106 234 110 hbar
155 149 199 187 n 0.1 Uparrow 1.2
210 127 219 169 exclam
210 177 219 186 dot 0.1 comma 0.4
```

```
0 201 35 226 n 0.15 u 0.9
42 203 83 206 hbar 0.08 bar 0.2
42 217 83 220 hbar 0.08 bar 0.2
92 188 117 227 0 0.2 O 0.3
```

2.3 Extrapolation de données

Étant donnés les symboles reconnus fournis par le système de reconnaissance, nous souhaitons disposer également d'informations plus générales permettant d'affiner l'analyse géométrique. Cette déduction de paramètres complémentaires est nécessaire si le système de reconnaissance des caractères ne les fournit pas.

2.3.1 Taille relative

Afin de différencier les relations entre les éléments composant une formule, il est nécessaire d'avoir une information sur la taille des éléments. Cette donnée n'est pas nécessairement utilisée, mais elle permet de lever certaines ambiguïtés lors de l'analyse géométrique. Le calcul de la taille d'un caractère ne dépend évidemment pas uniquement de la taille de la boîte englobante, mais aussi du caractère reconnu. Comparons par exemple les caractères "e" et "g". Si nous ne considérons que la taille de la boîte englobante, nous conclurons que "e" est plus petit que "g". Il y a donc nécessité d'introduire une pondération en fonction de l'élément considéré. Ceci aboutit à l'identification de classes de symboles, ordonnés selon leur taille.

L'algorithme mis en jeu pour effectuer cette tâche est du type recherche de sous-groupes dans un groupe d'éléments. On recherche en effet quels sont les éléments qui ont des tailles du même ordre, parmi l'ensemble des symboles reconnus. L'application de cet algorithme à la formule ci-dessus donne le résultat suivant :

- éléments de petite taille : trois symboles n, 0, ∞,
- éléments de taille moyenne : z, n, !,
- éléments de grande taille : $\sum$.

Cette notion de petite, moyenne, et grande taille est quantifiée par une valeur et est intégrée en tant qu'attribut du symbole reconnu.

2.3.2 Ligne de base

L'information concernant la ligne de base par rapport à laquelle l'objet graphique est positionné est importante. Elle permet de connaître l'alignement des symboles. La figure 4 illustre cette notion de ligne de référence pour les caractères b et q.

La déduction de cette donnée nous sera donc soit fournie par le système de reconnaissance de caractères, soit calculée de manière approximative grâce à des tables extraites

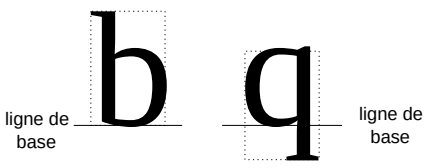


FIG. 4 – Ligne de base

d'informations sur la fonte utilisée, ou de données déduites pour l'écriture manuscrite d'une personne. Nous avons développé cette dernière solution de manière suffisamment générique pour s'adapter aux différentes études de cas. Mais la meilleure solution reste l'utilisation d'un système de reconnaissance des symboles qui fournisse cette information.

L'ensemble de ces informations métriques et géométriques va permettre d'effectuer une reconnaissance structurelle des notations mathématiques. La première étape de cette analyse structurelle est l'analyse lexicale, que nous allons maintenant présenter.

3 Analyse lexicale

De nombreuses raisons justifient de découper l'analyse en une analyse lexicale et une analyse syntaxique.

1. Une conception plus simple est peut-être le critère le plus important. La séparation de l'analyse lexicale et de l'analyse syntaxique permet souvent de simplifier l'une ou l'autre des phases. Dans le cas plus classique de la conception d'un nouveau langage, la séparation des conventions lexicales et syntaxiques peut amener à une conception globale plus soignée.
2. L'efficacité de l'analyse syntaxique est améliorée. Un analyseur lexical séparé permet de construire un processeur spécialisé et potentiellement plus efficace pour cette tâche.
3. La portabilité est accrue. Les particularités de l'alphabet d'entrée et d'autres anomalies spécifiques au matériel peuvent être confinées à l'analyseur lexical.

Tous ces points nous ont conduit à introduire une étape d'analyse lexicale dans le processus d'analyse de la structure de la formule.

Un analyseur lexical lit et convertit un flot d'entrée en un flot d'unités lexicales qui sera traité par l'analyseur syntaxique. Une suite de caractères d'entrée qui constitue une seule unité lexicale est appelée lexème. Un analyseur lexical permet de cacher la représentation physique des unités lexicales à l'analyseur syntaxique. Dans un modèle de compilateur par exemple [**?**][1], l'analyseur syntaxique obtient une chaîne d'unités lexicales de l'analyseur

[1] AHO A.V., SETHI R. & ULLMAN J.D., *Compilateurs : Principes, Techniques et Outils* (1991).

lexical, comme illustré par la figure, 5 et vérifie que la chaîne peut être engendrée par la grammaire du langage source.

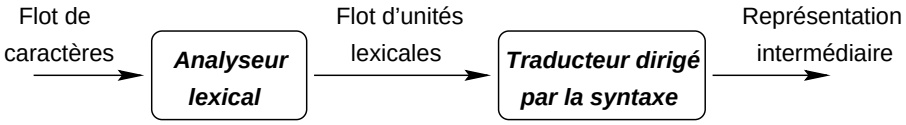


FIG. 5 – Représentation simplifiée de l'analyse syntaxique

Toutefois, cette étape reste assez sommaire, du fait de la nature bidimensionnelle des données traitées. En effet, il n'est pas possible, lors de l'analyse lexicale, de reconstituer les identifiants et les nombres. Leur construction dépend du contexte et de paramètres géométriques que nous ne pouvons pas introduire durant cette phase de l'analyse. Le traitement effectué se limite donc à abstraire les données reçues du logiciel se chargeant de réaliser la reconnaissance des symboles. Outre une simplification de la grammaire, il permet ainsi de rendre la grammaire indépendante des identifiants utilisés pour les symboles mathématiques.

Unité lexicale	Lexèmes	Description informelle des modèles
oprel	$< \leq = > \geq \neq$ ...	Opérateur d'égalité et inégalité
pargauche	$($ $[$ $\{$	Marqueur gauche de bloc
accent	$\hat{}$ $\tilde{}$ $\rightarrow$	Accentuation d'un identifiant
digit	0 1 2 ...	Chiffre
letter	$[a-z]$ $[A-Z]$	Caractère
...	...	...

Quand plus d'un modèle reconnaît un lexème, l'analyseur lexical doit fournir aux phases suivantes du compilateur des informations additionnelles sur le lexème reconnu. L'analyseur lexical réunit les informations sur les unités lexicales dans des attributs qui lui sont associés. Les unités lexicales influent sur les décisions d'analyse syntaxique ; les attributs influent sur la traduction des unités lexicales.

C'est sur ce schéma classique, utilisé en compilation, que nous avons construit l'analyseur lexical. Les attributs d'un lexème sont :

- la valeur du symbole reconnu,
- les coordonnées graphiques de ce symbole (coordonnées de la boîte englobante du symbole, ...),
- la ligne de référence,
- le taille relative du symbole.

L'ensemble des informations collectées va servir de base à l'analyse géométrique et syntaxique.

4 Analyse géométrique

L'analyse géométrique permet de hiérarchiser un ensemble de données a priori difficiles à utiliser en l'état. À partir des données fournies par le système sous-jacent, nous allons construire un graphe. Ce type de structure permet d'abstraire les informations et de les traiter efficacement. Nous allons introduire le type de graphe que nous utilisons, et nous présenterons les différentes étapes qui conduisent à sa construction.

4.1 Introduction des graphes

Dans les problèmes informatiques ou mathématiques, en ingénierie et dans de nombreux autres domaines, il est très fréquent d'avoir à représenter des relations arbitraires entre des objets considérés comme des données. Les graphes sont les modèles naturels de telles relations.

Deux classes de graphes sont à distinguer : orientés et non orientés.

- Un **graphe orienté** G est représenté par un couple (N, A), où N est un ensemble fini et A est une relation binaire sur N. L'ensemble N est appelé ensemble des **nœuds**, ou **sommets**, de G. On dit que l'ensemble A est l'ensemble des **arcs** de G.
- Dans un **graphe non orienté** $G = (N, A)$, l'ensemble des **arêtes** A n'est pas constitué de couples, mais de paires de nœuds non ordonnées.

Beaucoup de définitions sont identiques pour les graphes orientés et non orientés, mais certains termes ont des significations différentes, selon qu'ils sont employés dans l'un ou l'autre contexte.

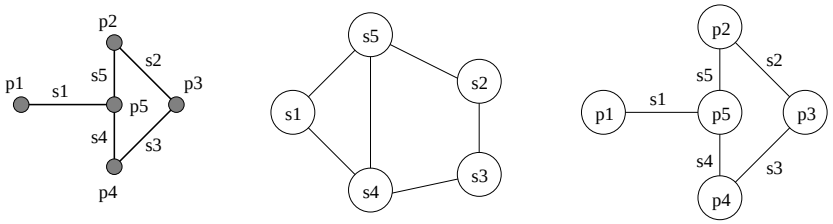
On peut distinguer deux utilisations fréquentes des graphes en reconnaissance de formes :

- les nœuds du graphe représentent les symboles ou formes reconnues et les arcs décrivent les relations entre ces éléments.
- les arcs du graphe sont les éléments, et les nœuds symbolisent des coïncidences de points de ces formes.

La figure 6 illustre ces deux types d'utilisation de graphes. La forme d'origine est constituée de cinq segments (s1 à s5) et de cinq nœuds (p1 à p5).

Cette structure de graphe est bien adaptée pour décrire ou imposer des relations sur n'importe quel couple de formes. Elle constitue un modèle puissant de représentation de la structure des formes, mais aussi de leur contexte (grâce aux arcs reliant toute forme avec les formes voisines). De plus, les nœuds et les arcs peuvent être augmentés d'attributs réutilisables par une grammaire attribuée.

Toutes les propriétés de cette structure de données nous ont conduit à la préférer. Nous allons maintenant donner des définitions pour la modélisation du graphe. Ces notions seront nécessaires à la poursuite de la lecture de ce chapitre.



Forme d'origine | Les noeuds sont des formes | Les arcs sont des formes

FIG. 6 – Deux représentations possibles d'une forme par un graphe

4.2 Définition d'un graphe

Étant donné la nature de notre problème, la reconnaissance structurelle de formules mathématiques, nous avons opté pour une modélisation des données utilisant la première représentation. Les nœuds du graphe représentent les symboles reconnus de l'expression mathématique et les arcs du graphe, les relations entre ces différents éléments. Les relations entre les objets n'étant pas symétriques, nous nous sommes dirigés vers l'utilisation de graphes orientés.

En ce qui concerne la modélisation même des objets composant le graphe, les nœuds et les arcs, nous avons choisi une modélisation un peu particulière, mais qui nous a permis de démontrer des propriétés sur les grammaires de graphes contextuelles. Définissons tout d'abord notre modélisation d'un graphe.

Chaque objet composant un graphe est représenté par un terme ou un ensemble fini de termes. Comme il est courant, l'ensemble de termes $T(F, V)$ est défini par un ensemble F de symboles fonctionnels et un ensemble V de variables. Les variables sont des termes et si $t_1, \ldots, t_n$ sont des termes et f est un symbole fonctionnel n-aire de F, alors $f(t_1, \ldots, t_n)$ est un terme. Notons $Var(t)$ l'ensemble de variables apparaissant dans un terme t (ou un ensemble fini de termes). Des définitions plus formelles peuvent être trouvées dans [**?**][1].

Nous utiliserons des lettres majuscules pour les symboles fonctionnels et des lettres minuscules pour les variables. Les nœuds, arcs et graphes sont ainsi définis :

Définition 1 : Noeud d'un graphe

Un nœud est un terme $N(t, i, v)$ pour lequel :

- *t est le type lexical ("Opérateur", "Variable", "Chiffre", ...),*
- *i est un identifiant pour repérer les différentes occurrences d'une même expression mathématique,*

[1] RUSINOWITCH M., *Démonstration Automatique par des Techniques de Réécriture*, PhD Thesis (1987).

– *v est la valeur, typiquement l'expression mathématique sous la forme d'un terme.*

■

Définition 2 : Arc d'un graphe

Un arc est un terme $A(t, n_1, n_2)$ avec :

– *t est le type de l'arc (son étiquette) : une direction dans le plan ("Gauche", "Droite", "Au dessus", ...),*

– *n_1 et n_2 sont des nœuds.* ■

Définition 3 : Graphe

Un graphe est un ensemble fini d'arcs : $\{A(t_1, n_{11}, n_{21}), \ldots, A(t_n, n_{1n}, n_{2n})\}$. L'ensemble $\{n_{ij}\}$ étant l'ensemble des nœuds du graphe. ■

Pour simplifier, nous supposerons que les graphes sont connexes et qu'ils contiennent au moins un arc. Ainsi, chaque nœud apparaît au moins une fois dans un arc. Ceci n'est pas une restriction : nous pouvons toujours rajouter un nœud générique, connecté à tous les autres nœuds avec un arc générique.

4.3 Construction du graphe initial

L'étape de construction du graphe contenant les données n'est cependant pas évidente. Il est pourtant très important de bien séparer la partie d'analyse géométrique de la reconnaissance syntaxique.

Cette construction du graphe initial utilise les informations fournies par les étapes précédemment présentées dans les paragraphes sur l'analyse de l'image et l'analyse lexicale.

Une structure de graphe est donc utilisée afin de modéliser les différentes relations entre les entités constituant la notation que l'on souhaite reconnaître. Un nœud du graphe est créé et associé à chacun des symboles composant la formule. Des attributs sont attachés à chacun des nœuds, comme le type lexical du symbole.

Les arcs, qui relient les nœuds du graphe, représentent les relations qui existent ou peuvent exister entre les entités de la formule. La construction de ces arcs est le résultat de l'analyse géométrique et lexicale. Nous allons détailler chacune des ces étapes.

4.3.1 Analyse de proximité

En règle générale, les notations mathématiques utilisent toutes une notion fondamentale : la proximité des symboles est révélatrice de liens entre les éléments concernés. Une étude de plusieurs ouvrages traitant de la typographie des expressions mathématiques comme [**?**][1], [**?**][1] a permis de mettre en évidence cette notion. C'est en effet le principe fondamental

[1] HIGHAM N.J., *Handbook of Writing for the Mathematical Sciences* (1993).

[1] KNUTH D.E., Mathematical typography (1979).

utilisé en écriture ; les symboles composant un mot sont rassemblés et un espace, donc une notion de distance plus grande entre des groupes de symboles, permet de faire des séparations entre les entités de base.

Un arc étiqueté est ainsi ajouté au graphe afin de représenter les relations spatiales entre les deux nœuds du graphe qu'il relie. Une première limitation intervient alors : il ne faut pas relier un objet avec tous les objets dans une direction donnée, mais ne relier un élément qu'avec l'élément le plus proche dans cette direction. Seuls les arcs ayant un sens géométrique intéressant sont ainsi pris en compte. La figure 7 illustre cette restriction sur la création des liens.

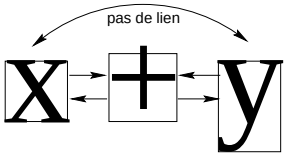


FIG. 7 – Construction des liens de proximité

Un système d'évaluation permet de déterminer si l'on ajoute le lien ou pas en fonction des différents autres paramètres qui entrent en jeu : les critères géométriques (position, distances, ...), les critères lexicaux.

4.3.2 Critères géométriques et graphiques

Les critères géométriques sont utilisés afin de déterminer si les symboles ou sous-expressions, représentés par un nœud, ont des liens possibles entre eux.

Plusieurs méthodes ont été testées pour effectuer la subdivision des données entourant un symbole donné en plusieurs zones. Une méthode simple consiste à analyser la valeur de l'angle formé avec l'horizontale par la droite reliant les centres des boîtes englobantes. Par exemple, un objet est placé à la droite d'un autre si l'angle est compris entre -22,5° et +22,5°. Cette approche convient bien aux symboles isolés, mais conduit à des échecs dans le cas d'une boîte englobant une sous-formule. Soit B "en haut à droite" de A. Si la taille de la sous-expression B augmente, la sous-expression ne sera plus alors considérée comme en position d'exposant, mais comme aligné par rapport à A. La figure 8 illustre cet exemple.

Une autre approche que nous avons détaillée page 48 concernait le découpage par projections. Il a été utilisé avec succès par M. Okamoto pour l'analyse de notations typographiées, mais, Z. X. Wang et C. Faure [**?**][2] ont montré les limites d'un tel système dans le cas manuscrit.

[2] WANG Z.X. & FAURE C., Automatic perception of the structure of handwritten mathematical expressions (1990).

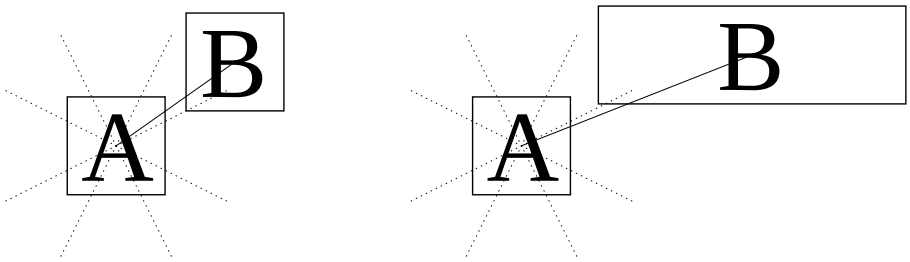


FIG. 8 – Détection des relations entre symboles, basée sur une notion d'angle

Nous avons opté pour une méthode moins restrictive que le découpage par projections, qui implique que les symboles soient parfaitement alignés donc difficilement adaptable au cas manuscrit. Le plan est subdivisé en neuf régions autour du symbole que l'on considère : les 8 directions principales (au dessus, en dessus, à gauche, à droite, en haut à gauche, ...), auxquelles s'ajoute la notion d'inclusion ("est à l'intérieur de"). Cette dernière est importante pour certains symboles comme la racine carrée par exemple. Les zones hachurées de la figure 9 illustrent les différentes régions dans lequelles doit se trouver le symbole afin d'être un candidat digne d'intérêt pour la création d'un lien avec le symbole représenté par le rectangle blanc. Cette zone de recherche, symbolisée par le rectangle hachuré, est proportionnelle à la taille du symbole par rapport auquel on effectue cette exploration. En effet, dans une notation mathématique, la taille du symbole indique la portée de celui-ci. Plus le symbole est grand, plus il englobe un nombre important de symboles, qui peuvent donc être plus éloignés. Par opposition, plus le symbole est petit, plus les symboles se rapportant à celui-ci seront proches de lui.

Plusieurs remarques sont à effectuer à propos de ce schéma. Les régions de recherche ne sont pas toutes équivalentes. Prenons un exemple ; pour la région "est à droite de", la zone de recherche est plus étendue en largeur que pour une recherche "en haut à droite". Ceci s'explique simplement par le fait même de l'écriture mathématique ; le symbole en position d'exposant est placé plus prêt qu'un symbole qui est sur la même ligne d'écriture que le symbole considéré. De même, on va chercher moins haut dans le cas d'une recherche des éléments "en haut à droite" que "en haut".

On peut aussi remarquer sur cette figure qu'une même région du plan peut correspondre à différentes positions. Par exemple, les régions "en haut à droite" et "à droite" se superposent. Ce découpage de l'espace permet une plus grande latitude dans la manière de typographier une notation mathématique, mais aussi de ne pas omettre certains liens dans le cas manuscrit où les variations peuvent être importantes dans l'écriture. L'évaluation de distances ou de l'alignement de deux symboles n'est pas absolu. Un seuil de tolérance paramétrable a été introduit, permettant ainsi une adaptation simplifiée au cas typographié et manuscrit de la construction du graphe.

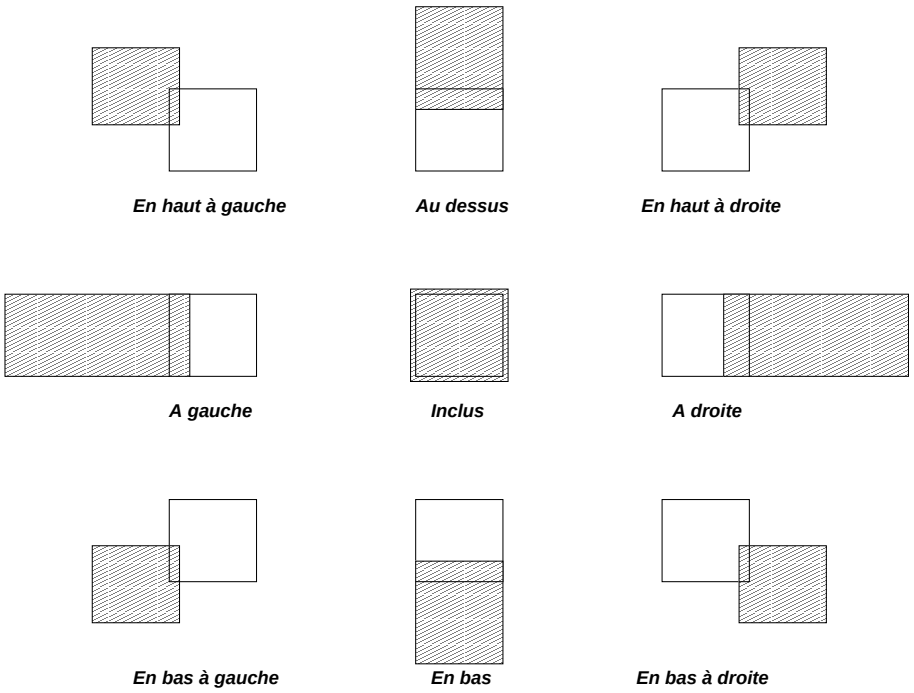


FIG. 9 – Analyse géométrique : définition des régions

4.3.3 Critères divers

Plusieurs types de critères sont donc pris en compte afin de déterminer si un lien doit être construit, et donc si un arc doit être ajouté ou non au graphe. Outre les critères purement géométriques que nous venons d'exposer, les informations à notre disposition sont incluses dans la prise de décision. Prenons deux exemples afin de mieux illustrer les critères en question.

- Dans le cas d'un lien horizontal, les critères suivants sont pris en compte :
  - l'alignement des lignes de référence des caractères,
  - l'alignement des centres des boîtes englobantes.
- Pour déterminer si un élément est en position d'exposant ("en haut à droire") :
  - la taille de l'élément

Chacun de ces critères entre en jeu dans l'évaluation de la probabilité d'un lien entre deux éléments. Un seuil permet de déterminer si l'arc est ajouté au graphe ou non.

Un autre paramètre est analysé afin de déterminer si un lien entre deux objets est valide, et donc si l'arc reliant les nœuds correspondants doit être construit ou pas : le type lexical.

4.3.4 Rôle du type lexical

L'analyse lexicale a permis de typer l'ensemble des symboles reconnus par le système de reconnaissance des caractères. Comme nous l'avons présenté, ce typage permet une abstrac-

tion supplémentaire dans le cadre de l'analyse structurelle, en créant des classes de symboles ayant un comportement identique. Une utilisation du type lexical, outre son emploi dans la grammaire, est une aide à la décision de la validité d'un lien dans la construction du graphe.

En pratique, certains opérateurs ou classes d'opérateurs ne peuvent être employés dans n'importe quel contexte. Un opérateur comme $<$ ne peut pas être placé en position d'exposant par rapport à un autre symbole. Dans la plupart des notations mathématiques courantes, $X^<$ (X exposant inférieur) n'a pas de sens syntaxique. Pour l'ensemble des lexèmes appartenant à la même classe lexicale que l'opérateur inférieur, ce type de relation n'a pas de sens. Nous spécifions donc qu'il n'est pas nécessaire de construire une telle liaison entre deux éléments, car elle ne peut pas être syntaxiquement valide. On adjoint ainsi des règles sur les relations qui ne sont pas acceptées pour chacun des types lexicaux.

Unité lexicale	Lexèmes	Type de liens interdits
oprel	$< \leq = > \geq \neq$ ...	liens diagonaux
accent	$\hat{}\ \tilde{}\ \rightarrow$	liens diagonaux, liens horizontaux
digit	0 1 2 ...	néant
letter	$[a-z]$ $[A-Z]$	néant
...	...	...

L'introduction de ces règles de contraintes sur les liens en fonction du type lexical a permis, dans un deuxième temps, de simplifier le graphe construit. En effet, la plupart des liens sont "réversibles" : x "est à gauche de" y est équivalent à y "est à droite de" x. Ceci a pour effet de multiplier par deux le nombre d'arcs dans le graphe. Grâce aux règles ci-dessus, il est possible de simplifier ce graphe en subdivisant les règles en deux catégories : le type d'arcs entrants sur un nœud et le type d'arcs sortants d'un nœud qui sont autorisés ou interdits.

Nota : On appelle **arc entrant** sur un nœud n, un arc orienté du type $A(x, n)$. Par opposition, on appelle **arc sortant** d'un nœud n, un arc orienté du type $A(n, x)$.

4.3.5 Conclusion

L'ensemble des paramètres géométriques et lexicaux que nous avons décrits sont pris en compte afin de déterminer si deux nœuds doivent être reliés par un certain type d'arc. L'algorithme pour la construction du graphe initial peut être résumé de la façon suivante :

```
procedure ContruireGrapheInitial(Ensemble des Noeuds en, resultat Graphe g)
debut
  g := GrapheVide
  pour chaque noeud n1 de l'ensemble des noeuds en faire
    pour chaque type d'arc a faire                                    5
      pour chaque noeud n2 de l'ensemble des noeuds faire
        si (n1<>n2) et Evaluation(n1, n2, a) alors
```

```
            ajouter l'arc a entre n1,n2 au graphe g
          finsi
        finpour                                                    10
      finpour
    finpour
    resultat g
fin
```

La fonction Evaluation, tient compte de l'ensemble des critères métriques, géométriques et lexicaux, afin de déterminer si la construction du lien doit être effectuée ou non (détails page 74).

Si n est le nombre de symboles composant l'expression mathématique que l'on souhaite reconnaître, la complexité de l'algorithme de construction du graphe initial (ConstruireGrapheInitial) est $\mathcal{O}(n^2)$. Toutefois, cet algorithme est quelque peu simpliste ; un gain important peut être réalisé. Nous allons présenter les optimisations effectuées dans le paragraphe suivant.

4.4 Optimisation

Il n'est pas nécessaire de tester un élément avec tous les autres afin de déterminer lequel se situe à sa "droite" ou bien "au dessus". Prenons un exemple concret pour illustrer la nécessité d'introduire une optimisation au niveau de la construction du graphe initial ; soit la formule suivante :

$$\sum_{I=(i_1,...,i_k)} P_I \sum_{j=1}^{n} (-1)^j \frac{p(m_{i_1},...,\hat{m}_{i_j},...,m_{i_k})}{p(m_{i_1},...,m_{i_k})} \delta^k_{m_1,...,m_n}(e_{i_1} \wedge ... \wedge \hat{e}_{i_j} \wedge ... \wedge e_{i_k}) = 0$$

Cette expression contient 116 symboles. Sans optimisation d'aucune sorte, le temps de calcul nécessaire à la construction du graphe initial atteint 35 secondes sur un PC, Pentium 200Mhz. Précisons tout de même que l'implémentation qui a permis de mesurer ce temps de calcul est réalisée dans un langage interprété.

Pour affiner cette approche, la notion de proximité décrite dans le paragraphe 4.3.1, ainsi que la notion de direction ont été prises en compte de manière plus précise. Afin de faire décroître la complexité de l'algorithme de calcul du graphe initial, nous avons tenté de limiter le nombre d'éléments à tester. La construction du graphe tient compte de la proximité des éléments par rapport à l'élément considéré. Le concept de voisinage d'un objet prend donc toute son importance. L'algorithme pour la construction du graphe initial devient alors le suivant :

```
procedure ContruireGrapheInitial(Ensemble des Noeuds en, resultat Graphe g)
debut
  g := GrapheVide
```

```
  pour chaque noeud n1 de l'ensemble des noeuds en faire
    pour chaque type d'arc a faire                                    5
      pour chaque noeud n2 voisin de n1 faire
        si (n1<>n2) et Evaluation(n1, n2, a) alors
          ajouter l'arc a entre n1,n2 au graphe g
        finsi
      finpour                                                         10
    finpour
  finpour
  resultat g
fin
```

Nous avons introduit la notion de subdivision du plan en sous-régions, afin de limiter le nombre de candidats possibles pour un type de lien dans une direction donnée. Le concept de voisinage se définit alors par l'inspection de tous les éléments appartenant aux sous-régions concernées. Ainsi, sur l'exemple présenté par la figure 10, pour déterminer si un élément est "à droite" de l'intégrale, il suffit de considérer les symboles qui sont dans les cases $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$. Si aucun élément n'est détecté dans cette zone, on pourra étendre la recherche aux cases $(0,2)$, $(1,2)$, etc. Une distance maximale de recherche est définie.

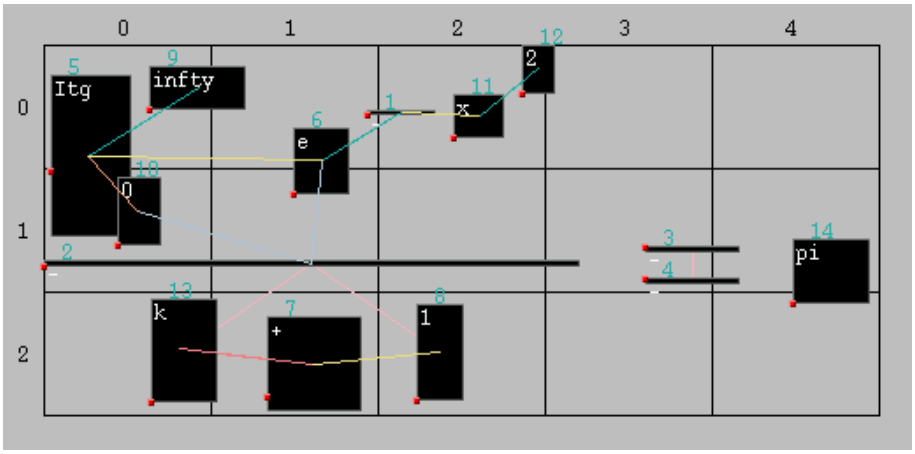


FIG. 10 – Subdivision du plan en sous-régions

Le nombre de subdivisions est calculé à partir de l'ensemble des symboles qui composent la formule. Une étude a été réalisée afin de déterminer le nombre optimum de subdivisions du plan, en fonction des éléments de la notation à identifier. La figure 11 représente le temps nécessaire au calcul du graphe initial en fonction du nombre de subdivisions du plan selon l'axe x (ici intervalle $[0, 35]$) et y (intervalle $[0, 15]$), pour la formule ci-dessus.

Cette étude, réalisée sur un vaste échantillon de notations mathématiques, a permis de déterminer l'intervalle de la subdivision pour un temps optimum de construction du graphe initial. Ainsi, la valeur optimisant le temps de calcul est $1,75$ fois la taille moyenne des boîtes englobantes des symboles de la notation.

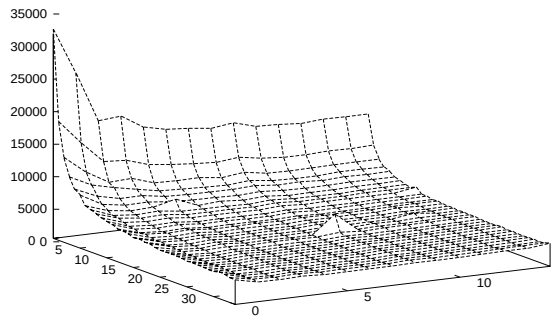


FIG. 11 – Optimisation du temps de construction du graphe initial

Avec la subdivision du plan que nous venons de présenter, le temps de calcul nécessaire à la construction du graphe initial de la formule citée ci-dessus, a été réduit à 3,6 secondes (soit un gain d'un facteur 9). Ce gain de performance n'est toutefois vraiment sensible que pour des notations composées de nombreux symboles.

4.5 Les limites de la construction du graphe

Comme nous l'avons souligné précédemment, le construction du graphe est une étape importante ; trop de liens peuvent nuire à la reconnaissance de la notation et le manque de liens peuvent aussi conduire à un échec lors de l'analyse. Dans le cas manuscrit, plus sujet aux variations dans la calligraphie, plusieurs erreurs de construction du graphe peuvent se produire :

- cas limite de détection en haut ou en bas à droite et aligné,
- symboles trop éloignés les uns des autres et donc non construction de liens dans le graphe (cas rarement rencontré),
- symboles sans liens logique trop proches d'un point de vue géométrique.

Illustrons, à l'aide de la formule suivante, cette dernière catégorie d'erreur qui peut aussi se produire dans le cas typographiés.

$$X^{X_{X_{X_{X_{X}}}}}_{X^{X^{X}}_{X_{X_{X_{X}}}}}$$

Le graphe construit (voir la figure 12), contient un lien qui amènent à une impasse lors de la phase de reconnaissance de la structure. D'un point de vue géométrique, les symboles en exposant de l'indice et en indice de l'exposant sont suffisamment prochent pour que l'analyse

géométrique juge bon de lier ces objets.

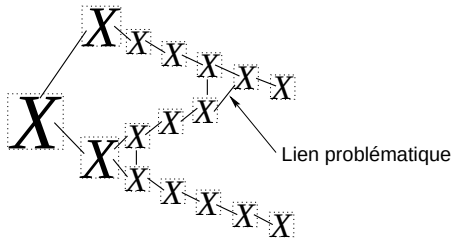


FIG. 12 – Problème de construction du graphe

Le système d'étiquetage des arcs du graphe tient aussi compte d'informations sur la taille des symboles pour éviter la construction de liens erronés pour les indices et les exposants ; la taille du (ou de) caractère en indice doit être inférieure ou égale au symbole de référence. Dans le cas de notre exemple, le système ne peut pas non plus utiliser l'information sur la taille des symboles pour détecter une erreur. En effet, sur cet exemple typographié par LaTeX, la taille des polices de caractères utilisées pour les indices ou exposants va en décroissant. Toutefois cette décroissance a un seuil minimum atteint au troisième niveau. Les symboles sont donc de taille identique et le système ne peut en déduire qu'il n'existe pas de lien de type "exposant" entre ces symboles. Afin de trouver une solution à ce type de problème, nous avons pensé à faire une analyse des cycles dans le graphe construit. Toutefois nous n'avons pas pu la mettre en pratique et ainsi vérifier si une telle solution pouvait résoudre la question.

Toutes les étapes que nous venons de détailler permettent d'obtenir un graphe renfermant l'ensemble des données nécessaires. Il reste toutefois à analyser le graphe obtenu, afin d'en déduire l'arbre de syntaxe de l'expression sous-jacente. Nous allons détailler la méthode retenue, basée sur l'utilisation d'une grammaires de graphes.

5 Rappels sur les grammaires

Nous avons jusqu'à présent décrit les avantages d'une structure comme le graphe pour formaliser des données. Les grammaires de graphes fournissent, quant à elles, un formalisme intéressant de description des manipulations structurelles de données multidimensionnelles [**?**][1].

Une grammaire formelle classique opère sur une suite de caractères, c'est à dire un espace à une dimension, à la topologie rudimentaire. Mais dans le cas de l'analyse d'expressions qui s'écrivent dans le plan, on gagne une dimension en même temps que l'on perd l'ordre total

[1] GLADKOFF S., *Grammaires de Graphes et Applications*, PhD Thesis (1982).

naturel de la droite et son caractère discret. Le problème qui était purement combinatoire et discret à une dimension, devient en plus géométrique et continu à deux dimensions.

La solution pour la reconnaissance de formules doit alors avoir deux aspects : un aspect géométrique et un aspect combinatoire. Si l'on adopte, comme cela semble naturel, l'approche des grammaires formelles, on note que le formalisme de la grammaire et l'algorithme de reconnaissance associé doivent, à eux deux, faire intervenir des concepts géométriques et combinatoires.

Cela nous a conduit à nous orienter vers une classe de grammaires spécialisées : une grammaire de graphes, localement dépendante du contexte et attribuée . Elle généralise les grammaires linéaires usuelles en étendant les liaisons "à droite" et "à gauche" entre les lexèmes du cas linéaire, aux liaisons relatives des rectangles des lexèmes : "en haut et à droite" , "en bas"...

5.1 Rappels sur les grammaires

Les définitions rappelées ci-dessous sont générales et concernent aussi bien les structures linéaires (analyse de texte), que les formes planes et elles peuvent même être éventuellement appliquées à la reconnaissance de structures multidimensionnelles. Nos propos seront cependant plus orientés sur la reconnaissance bidimensionnelle et les problèmes posés par la reconnaissance de structures géométriques.

5.1.1 Grammaires formelles

L'étude des langages formels a constitué, et constitue encore une part importante de l'informatique. Ce domaine de recherche a connu un engouement certain dès les années 1956, quand Noam Chomsky [**?**][2] a défini un modèle mathématique pour les grammaires en connexion avec ses études sur le langage naturel. Nous rappelons simplement ici quelques définitions utiles.

Définition 4 : Grammaire formelle

Une grammaire formelle est un quadruplet $G = (V_N, V_T, P, X)$ où :

- *V_N est un ensemble fini et non vide de symboles non-terminaux, appelé encore alphabet non-terminal,*
- *V_T est un ensemble fini et non vide de symboles terminaux appelé alphabet terminal. Ces ensembles N et T ont une intersection vide,*
- *P est un ensemble fini non vide de règles de production (ou règles de réécriture [**?**][1]),*

[2] CHOMSKY N., *Aspects of the Theory of Syntax* (1965).

[1] JOUANNAUD J.P. & LESCANNE P., La réécriture (1986).

– *X est un symbole non-terminal particulier, appelé axiome de la grammaire.*

Par convention, nous noterons $V = V_N \cup V_T$, où V est appelé le vocabulaire de la grammaire. Chaque production p de P est de la forme $\alpha \rightarrow \beta$, avec $\alpha_i \in V^+$ et $\beta_i \in V^*$. ■

On rappelle que V^* désigne l'ensemble des chaînes pouvant être construites en concaténant les éléments de V, y compris les chaînes vides, et V^+ en excluant les chaînes vides.

Chomsky propose une classification des grammaires formelles selon les restrictions faites sur la nature des règles de production de la grammaire. Il a ainsi défini trois types de grammaires, souvent nommées grammaires de type 0, 1, 2 et 3.

Définition 5 : Grammaire de type 0

Une grammaire de type 0 est sans aucune contrainte sur les règle de production. ■

Définition 6 : Grammaire de type 1 ou grammaire contextuelle

$$xAy \rightarrow xzy \text{ avec } x, y, z \in V^* \text{ et } A \in N$$

Le symbole A ne peut être écrit en z qu'en présence du contexte gauche x et du contexte droit y. ■

Définition 7 : Grammaire de type 2 ou grammaire hors contexte

$$A \rightarrow x \text{ avec } A \in N \text{ et } x \in V^*$$

Le symbole A peut-être réécrit en x quel que soit son contexte. ■

Définition 8 : Grammaire de type 3 ou grammaire régulière à contexte libre

$$A \rightarrow a \text{ ou } A \rightarrow aB \text{ où } a \in T \text{ et } A, B \in N$$ ■

5.1.2 Grammaires attribuées

L'idée d'introduire dans la grammaire des attributs permettant d'utiliser les informations sémantiques est due à Donald Knuth [**?**][2]. On parle dans ce cas de grammaire attribuée. Dans une grammaire attribuée, chaque production est en fait constituée de deux règles : une règle syntaxique et une règle sémantique, introduisant des propriétés spécifiques supplémentaires.

On retrouve la même définition que pour une grammaire formelle (Définition 4). Pour chaque élément $x \in V$, il existe un ensemble fini d'attributs $A(x)$, où chaque élément α de $A(x)$ contient un ensemble fini ou infini de valeurs possibles D_α. Cet ensemble représente un vecteur d'attributs, spécifiant les propriétés de l'objet (coordonnées ou dimensions par exemple). Chaque production se définit donc en deux parties :

[2] KNUTH D.E., Semantics of context-free languages (1968).

– une règle syntaxique de la forme :

$$x_0 \rightarrow x_1 x_2 ... x_m \text{ où } x_0 \in N, \text{ et } \forall i \in [1..m], x_i \in V$$

– une règle sémantique est constituée de contraintes sur les attributs et se définit comme un ensemble d'expressions :

$$\begin{aligned} \alpha_1 &\rightarrow f_1(\alpha_{11}, \alpha_{12}, ..., \alpha_{1n_1}) \\ \alpha_2 &\rightarrow f_2(\alpha_{21}, \alpha_{22}, ..., \alpha_{2n_2}) \\ &\vdots \\ \alpha_n &\rightarrow f_n(\alpha_{n1}, \alpha_{n2}, ..., \alpha_{nn_n}) \end{aligned}$$

où $\alpha_1, ..., \alpha_n = A(x_0) \cup A(x_1) \cup ... \cup A(x_m)$, et chaque α_{ij} $(1 \leq i \leq n, 1 \leq j \leq n)$ est un attribut d'un x_k pour $(0 \leq k \leq m)$. Les f_i pour $(1 \leq i \leq n)$ sont des opérateurs permettant l'une des trois formes suivantes :

– $f_i : D_{\alpha_{i1}} \times D_{\alpha_{i2}}, ..., D_{\alpha_{in_i}} \rightarrow D_{\alpha_i}$,

– f_i est une fonction sans variable, exprimée en fonction des valeurs $\alpha_{i1}, ..., \alpha_{in_i}$,

– f_i est un algorithme, qui prend en entrée les valeurs $\alpha_{i1}, ..., \alpha_{in_i}$, ainsi que toute autre information ou donnée, et qui retourne le résultat dans α_i.

Chaque symbole de la grammaire possède un ensemble d'attributs qui permettent de représenter ce que l'on désire : une chaîne de caractères, un nombre, un type, etc. La valeur de l'attribut est définie par une règle sémantique associée à la production utilisée. L'ensemble des attributs est partitionné en deux sous-ensembles appelés les **attributs synthétisés** et les **attributs hérités** de ce symbole. La valeur d'un attribut synthétisé en un nœud est calculée à partir des valeurs des attributs aux fils de ce nœud dans l'arbre syntaxique ; la valeur d'un attribut hérité en un nœud est calculé à partir des valeurs des attributs aux frères et au père de ce nœud.

Nous avons jusqu'à présent uniquement considéré les grammaires linéaires, c'est à dire des grammaires dans lesquelles les relations entre symboles ne sont constituées que de concaténations, chaque symbole étant connecté soit à sa gauche, soit à sa droite. Ce type de relation unaire ne permet pas de prendre en compte des notations bidimensionnelles (ou n-dimensionnelles) ; il faut pour cela définir les relations n-aires correspondantes. Il existe essentiellement deux possibilités d'obtenir une représentation plus appropriée à la classe de problèmes traitée (modélisation du plan ou de l'espace).

– La première consiste à choisir un formalisme ou une structure adaptée à la représentation de données multidimensionnelles ($n \geq 2$), comme un arbre ou un graphe.

– La seconde possibilité revient à introduire dans les règles de la grammaire des relations n-aires plus générales que la concaténation.

Deux approches, connues dans le domaine de la reconnaissance de formes, utilisent ce dernier concept : le langage PDL et les plex-grammaires.

Nous avons introduit la notion de graphe comme structure de données permettant la représentation de formes planes. Nous allons maintenant présenter les grammaires de graphes

qui manipulent ces graphes pour en extraire, par un système de réécriture, l'information souhaitée.

5.2 Grammaires de graphes

Pfaltz et Rosenfeld [**?**][1] ont étendu le concept de grammaire linéaire aux grammaires de graphes afin de résoudre des problèmes pratiques de traitement d'images (reconnaissance de chromosomes, de caractères manuscrits, ...). Les graphes manipulés par de telles grammaires sont ceux dans lesquels les nœuds désignent les formes ; celles-ci jouent le rôle de terminaux et de non-terminaux.

La diversité des champs d'application de ces grammaires a eu pour conséquence l'apparition de nombreux modèles de systèmes de réécriture de graphes. L'idée d'utiliser de telles grammaires apparaît entre autre dans les travaux de H. Bunke [**?**][2]. Plusieurs utilisations des grammaires de graphes peuvent être citées comme : [**?**][3] sur l'utilisation en général des grammaires de graphes, mais aussi [**?**][4] et [**?**][5] pour la reconnaissance de partitions musicales, [**?**][6] pour la reconnaissance de schémas électroniques. Quelques articles de synthèse sur le sujet sont aussi à noter comme [**?**][7], [**?**][8]. Le lecteur désireux de connaître de nombreuses autres applications pourra consulter [**?**][9] ou encore des recueils comme [**?**][10], [**?**][11], entièrement consacrés aux grammaires de graphes et à leur utilisation.

La plupart des modèles de grammaires de graphes sont basés sur l'opération de réécriture d'un graphe (ou d'un sous-graphe) et l'itération d'un tel processus. Les travaux de B. Courcelle [**?**][1] ont permis de simplifier le concept de réécriture de graphes grâce à une représentation de ceux-ci sous forme d'expressions algébriques.

Dans toutes les grammaires que nous venons d'étudier, une production $\alpha \rightarrow \beta$ est utilisée

[1] PFALTZ J. & ROSENFLED A., Web grammars (1969).

[2] BUNKE H., Graph grammars as a generative tool in image understanding (1982).

[3] PFEIFFER J.J., Using graph grammars for data structure manipulation (1990).

[4] BLOSTEIN D. & FAHMY H., A graph grammar for high-level recognition of music notation (1991).

[5] BLOSTEIN D., FAHMY H. & GRBAVEC A., Practical use of graph rewriting, Technical Report (1995).

[6] BUNKE H., Attributed programmed graph grammars and their application to schematic diagram interpretation (1982).

[7] NAGL M., A tutorial and bibliographical survey on graph grammars (1978).

[8] BLOSTEIN D. & FAHMY H., A survey of graph grammars : Theory and applications (1992).

[9] *Graph-Grammars and Their Applications to Computer Science and Biology* (1979).

[10] *Graph Grammars and their Application to Computer Science* (1986).

[11] *Graph Transformations in Computer Science* (1993).

[1] COURCELLE B., An axiomatic definition of context-free rewriting and its application to nlc graph grammars, Technical Report (1987).

pour remplacer une chaîne par une autre. Ainsi, la chaîne $\gamma\alpha\delta$ est remplacée par $\gamma\beta\delta$. Dans le cas où les données sont représentées par des graphes, il devient nécessaire de préciser, pour une règle de la forme $\alpha \rightarrow \beta$, comment remplacer le sous-graphe α d'un graphe ω, par un sous-graphe β. Il s'agit donc d'associer à chaque production $\alpha \rightarrow \beta$ une règle précisant ce remplacement, sachant qu'il est indépendant du graphe "receveur" ω.

Soit V un ensemble d'étiquettes, N_α et N_β l'ensemble des noeuds de α et β ; la règle de réécriture d'une grammaire de graphes est un triplet (α, β, ϕ) avec :

$$\phi : N_\beta x N_\alpha \rightarrow 2^V$$

L'application ϕ précise comment relier les nœuds de β à chaque nœud du graphe ω privé du sous-graphe α (noté $\omega - \alpha$).

Comme pour les grammaires formelles linéaires, il est possible d'augmenter chaque production d'un ensemble d'attributs. On parle alors de grammaires de graphes attribuées. Ces dernières permettent de concevoir des systèmes de reconnaissance puissants, dans lesquels concepts structurels, numériques et sémantiques sont fortement liés.

Nous utilisons une grammaire de graphes attribuée, localement dépendante du contexte [**?**][2] [**?**][3]. Une règle de la grammaire exprime le fait qu'un sous-graphe du graphe de données peut-être réduit à un sous-graphe (qui peut-être réduit à un nœud) si certaines conditions syntaxiques sont vérifiées. Les symboles terminaux représentent les caractères reconnus par le programme de reconnaissance des caractères et les non-terminaux représentent les morceaux de sous-formules déjà analysés.

Définition 9 : Règle syntaxique d'une grammaire de graphes

Une règle est un terme $G, C \rightarrow P$ avec :

- *P est un nœud, appelé la production de la règle.*
- *G est un graphe, appelé le modèle de la règle.*
- *C est un ensemble fini de graphes, appelé contexte de la règle.* ■

La grammaire étant attribuée, des règles sémantiques sont associées aux règles syntaxiques (voir page 82). Ces règles permettront de calculer un ensemble d'attributs lors de l'application de la règle syntaxique sur le graphe (étape de réécriture). Citons ici quelques exemples d'attributs synthétisés : la ligne de référence pour le nouvel objet, la taille de l'élément, ... Nous reviendrons plus en détail sur ce point dans la suite du discours.

Définition 10 : Grammaire de graphes

[2] LAVIROTTE S. & POTTIER L., Optical formula recognition (1997).

[3] LAVIROTTE S. & POTTIER L., Mathematical formula recognition (1997).

Une grammaire est un ensemble fini de règles. ■

Soit un graphe représentant une formule (les nœuds sont donc les symboles de la formules et les arcs les liens exprimant les relations géométriques entre les symboles) ; les règles sont utilisées pour réécrire le graphe en remplaçant le sous-graphe reconnu par un nœud dont la valeur est la sous-formule construite (arbre de syntaxe abstraite) sous la forme d'un terme. Ce processus utilise donc la reconnaissance et le remplacement d'un sous-graphe dans un graphe, ce que nous allons définir [**?**][1].

Rappelons tout d'abord les notions de substitution et de reconnaissance de modèles (*pattern matching*) appliquées aux termes.

Définition 11 : Substitution

Une substitution est un endomorphisme de $T(F,V)$, c'est à dire une application σ vérifiant $\sigma f(t_1,\ldots,t_n) = f(\sigma t_1,\ldots,\sigma t_n)$ pour tout f dans F et tous les termes $t_1,\ldots,t_n$. Une substitution σ est uniquement déterminée par sa restriction $\sigma_{|V}$ à l'ensemble des variables. ■

Définition 12 : Filtrage de termes

Un terme t filtre avec un terme t', noté $t \leq t'$ si et seulement si il existe une substitution σ telle que $\sigma t = t'$. ■

Le filtrage d'un ensemble fini de termes est défini par :

$$\{t_1,\ldots,t_n\} \leq \{t'_1,\ldots,t'_m\} \Leftrightarrow \exists\sigma\ \{\sigma t_1,\ldots,\sigma t_n\} = \{t'_1,\ldots,t'_m\}$$

Définition 13 : Réécriture d'un graphe par application d'une règle

Une règle $r : G, C \rightarrow P$ réécrit un graphe G_1 en un graphe G_2, noté $G_1 \rightarrow_r G_2$ si et seulement si il existe une substitution σ, un sous-graphe G' de G_1 (c'est à dire $G' \subset G_1$), tels que :

- *$\sigma G = G'$.*
- *pour tout graphe H dans le contexte C, il n'existe pas de substitution τ telle que $\tau_{|Var(G)} = \sigma_{|Var(G)}$ et $\tau H \subset G_1$.*
- *G_2 est obtenu par remplacement de G' dans σP, c'est à dire en retirant de G_1 tous les arcs de G' et en remplaçant dans G_1 tous les nœuds de G' par le nœud σN.* ■

Après une brève présentation des principaux éléments composant une grammaire de graphes, intéressons nous à son utilisation dans le cas particulier de la reconnaissance des expressions mathématiques.

[1] RAOULT J.C. & VOISIN F., Set-theoretic graph rewriting, Technical Report (1992).

6 Analyse structurelle

6.1 Grammaire de graphes et formules mathématiques

L'application des grammaires de graphes à la reconnaissance de formules mathématiques doit tirer bénéfice des spécificités du domaine étudié. Les notations mathématiques présentent des particularités qui doivent être prises en compte afin d'adapter au mieux l'utilisation des grammaires de graphes à ce contexte.

Étant donné un graphe $\mathcal{G}$, construit suivant la méthode exposée précédemment. Ce graphe va être réécrit successivement par les règles de la grammaire, afin de reconnaître l'arbre de syntaxe abstraite de la formule "représentée" par ce graphe. Chaque étape de réécriture correspond à l'identification d'une sous-expression de la formule mathématique globale. Chacune de ces règles doit ainsi exprimer l'identification d'une sous-expression, ce qui implique donc l'identification d'un opérateur mathématique et des opérandes qui se rapportent à cet opérateur. Nous pouvons donc découper la grammaire en règles correspondant aux opérateurs mathématiques que l'on souhaite reconnaître. Nous définissons alors une grammaire dite grammaire d'opérateurs [**?**][1]. Ces grammaires ont la propriété (parmi d'autres exigences essentielles) de ne pas avoir de production dont la partie droite est vide ni d'avoir deux non-terminaux adjacents. On peut facilement construire à la main des analyseurs efficaces par décalage-réduction, dans le cadre d'une utilisation sur des chaînes de caractères.

Nous allons maintenant décrire une technique d'analyse facile à implanter appelée analyse par priorité d'opérateurs. En tant que technique générale d'analyse, la priorité d'opérateurs a un certain nombre de désavantages. Par exemple, il est difficile de manipuler des unités lexicales comme le signe moins, qui a deux priorités différentes (selon son utilisation comme opérateur binaire ou unaire). Nous avons donc développé des méthodes utilisant la géométrie et le contexte d'un sous-graphe, afin de résoudre ces cas délicats. Présentons tout d'abord de manière générale les choix effectués pour l'organisation de cette grammaire.

Dans l'analyse de précédence par opérateurs, nous définissons donc trois relations de précédences disjointes $<$, $=$, $>$, entre certains couples de terminaux de la grammaire. On associe donc une priorité dans l'ordre d'application des règles. Cette priorité correspond à l'opérateur auquel la règle est associée. Nous pouvons donc en déduire les relations entre les couples de terminaux de la grammaire qui nous intéressent (les opérateurs mathématiques).

Toutefois, ce seul critère n'est pas suffisant dans le cadre de notre application. En effet, des critères géométriques doivent être pris en compte. Chaque changement de ligne de référence pour l'écriture d'une notation implique un changement de priorité dans l'analyse.

[1] AHO A.V. & ULLMAN J.D., *The Theory of Parsing Translation and Compiling* (1972).

Nous présentons dans les paragraphes suivants la méthode que nous avons développée afin de rendre les grammaires de graphes d'opérateurs mathématiques non ambiguës, et ainsi obtenir un analyseur syntaxique mieux adapté aux grammaires de graphes, dans le cadre de la reconnaissance de structures mathématiques.

6.2 Construction du contexte des règles

Un des problèmes principaux avec une telle grammaire d'opérateurs et les règles de réécriture est bien l'existence d'ambiguïtés. En effet, deux règles peuvent réécrire un graphe de deux manières différentes et donc aboutir à deux graphes différents. La suppression des ambiguïtés peut s'effectuer grâce à l'analyse de cas sur les modèles, l'utilisation de priorités des règles, ou bien encore la complétion du type Knuth-Bendix. Mais ces techniques s'appliquent difficilement à notre cas, c'est pourquoi nous utilisons un **contexte** pour les règles : étant donnée une grammaire de graphes contenant des ambiguïtés, notre but est d'ajouter un contexte d'application à ces règles, afin de supprimer les ambiguïtés, aussi systématiquement que possible.

6.2.1 Exemple

Pour illustrer les propos théoriques qui suivent, nous allons prendre un exemple de formule à reconnaître et une grammaire permettant d'effectuer cette analyse.

Soit la formule $a^2 + b$, décrite par le graphe $\mathcal{F}$.

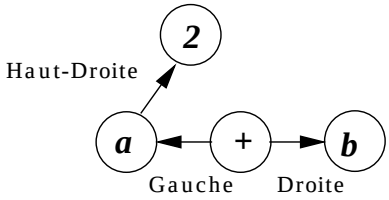


FIG. 13 – Le graphe représentant la formule $A^2 + B$

$$\begin{aligned}\mathcal{F} = \ & \{A(\textit{Haut-Droite}, N(Expr, 1, A), N(Expr, 2, 2)),\\ & A(Gauche, N(Plus, 3, +), N(Expr, 1, A)),\\ & A(Droite, N(Plus, 3, +), N(Expr, 4, B))\}\end{aligned}$$

Soit la grammaire $\mathcal{G}$ indépendante du contexte, avec deux règles permettant de reconnaître : pour l'une l'addition et pour l'autre l'opérateur implicite puissance.

– Opérateur addition $z + t$:

$$\begin{aligned}r_+ : \ & \{A(Gauche, N(Plus, i_+, +), N(Expr, i_z, z)),\\ & A(Droite, N(Plus, i_+, +), N(Expr, i_t, t))\}, \emptyset \rightarrow N(Expr, i_+, +(z, t))\end{aligned}$$

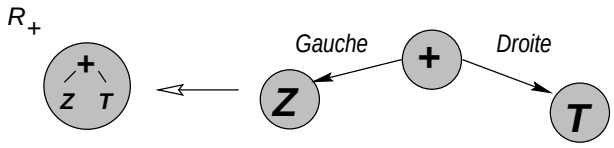


FIG. 14 – Règle pour l'addition

– Opérateur implicite puissance x^y :

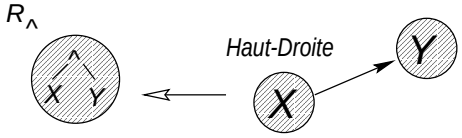


FIG. 15 – Règle pour l'exposant

$$r_\wedge : \{A(\textbf{Haut-Droite}, N(Expr, i_x, x), N(Expr, i_y, y))\}, \emptyset \rightarrow N(Expr, i_x, \wedge(x, y))$$

Pour la règle r_+ : la première partie est la description de la "production" de la règle, le nœud qui remplacera le graphe reconnu. Les éléments de ce nœud sont : le type lexical $Expr$, la valeur du nœud (qui est la construction de l'élément constitué de $+$ comme symbole racine de l'arbre, avec pour arguments z et t), et un identifiant du noeud créé. La deuxième partie est le modèle de graphe que l'on cherche à reconnaître. Ce graphe est constitué des deux arcs, avec pour chacun leur type (direction et poids) et les nœuds.

6.2.2 Superpositions dans l'application des règles

Nous allons préciser la technique employée. Quand deux règles peuvent s'appliquer sur deux sous-graphes disjoints, il n'y a pas d'ambiguïté sur l'application des règles : l'application des règles peut commuter. Les ambiguïtés apparaissent quand deux modèles de règles peuvent être appliqués sur un même sous-graphe, c'est à dire lorsque les modèles se superposent.

– Deux graphes G_1 et G_2 peuvent se superposer si et seulement si il existe σ_1 et σ_2 tels que $\sigma_1 G_1$ et $\sigma_2 G_2$ ont un nœud en commun. Nous notons $S(G_1, G_2)$ l'ensemble des couples de telles substitutions, appelées superpositions de G_1 et G_2.
– Soient deux règles $r_i : G_i, C_i \rightarrow P_i$, $i = 1, 2$, l'ensemble $A(r_1, r_2)$ des ambiguïtés de r_1 et r_2 est défini comme le sous-ensemble de $S(G_1, G_2)$ formé par les couples (σ_1, σ_2) tels que les deux règles peuvent s'appliquer sur le graphe $\sigma_1 G_1 \cup \sigma_2 G_2$, c'est à dire $\forall i = 1, 2,\ \forall H \in C_i$, il existe une substitution τ telle que $\tau_{|Var(G_i)} = \sigma_{i|Var(G_i)}$ et $\tau H \subset \sigma_1 G_1 \cup \sigma_2 G_2$.

Si nous reprenons notre exemple, il y a bien une ambiguïté car nous pouvons appliquer les deux règles à $\mathcal{F}$ avec au moins un nœud commun. Si nous appliquons r_+, les nœuds a,

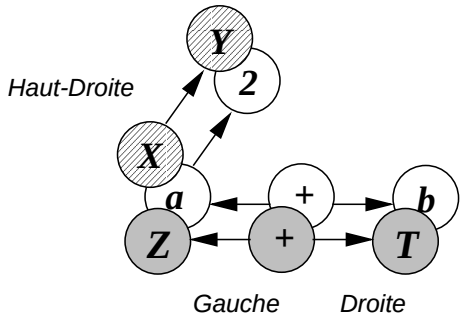


FIG. 16 – Superposition des modèles sur un graphe

$+$ et b sont regroupés et remplacés, et nous obtenons, par réécriture, un graphe représentant la formule $(a+b)^2$. Alors que si nous appliquons $r_\wedge$, nous obtenons alors un autre graphe représentant la formule $(a^2)+b$. Le bon choix est clairement d'appliquer la règle $r_\wedge$. Nous devons donc empêcher l'application de la règle r_+ dans ce cas. La figure 17 montre, sous forme de schémas, les quatre superpositions minimales des règles r_+ et $r_\wedge$.

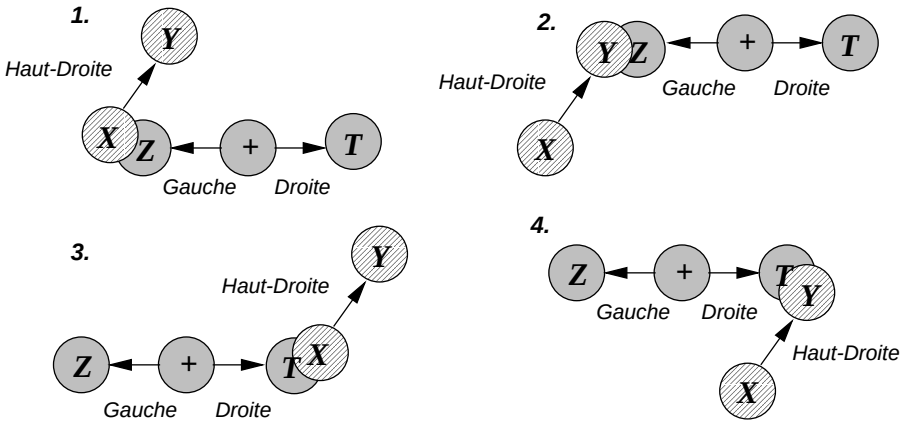


FIG. 17 – Toutes les superpositions des règles r_+ et $r_\wedge$

Après quelques considérations plus théoriques, des commentaires sur les résultats déduits de chacune de ces superpositions seront donnés.

L'ensemble $S(G_1, G_2)$ peut être infini, mais les superpositions minimales sont en nombre fini. Mais définissons tout d'abord un ordre sur les couples de substitutions :

Définition 14 : Ordre sur les couples de substitutions

$$(\sigma_1, \sigma_2) \leq (\sigma'_1, \sigma'_2) \Leftrightarrow \exists \tau, \sigma'_1 = \tau \circ \sigma_1, \quad \sigma'_1 = \tau \circ \sigma_1$$

■

Proposition 1

Étant donnés deux graphes G_1 et G_2, il existe un sous-ensemble fini S_0 de $S(G_1, G_2)$ tel que :

$$\forall c \in S(G_1, G_2),\ \exists c' \in S, c' \ll c.$$

■

PREUVE Il y a un nombre fini de façons d'associer les nœuds de G_1 et G_2 par couples, c'est à dire de construire les ensembles $\{(n_{i1}, n_{i2})\}_{i=1..m}$, où n_{ij} est un nœud de G_j.

Une superposition (σ_1, σ_2) de G_1 et G_2 correspond au moins à un de ces sous-ensembles $\{(n_{i1}, n_{i2})\}_{i=1..m}$, où nous avons $\forall i,\ \sigma_1 n_{i1} = \sigma_2 n_{i2}$.

Alors (σ_1, σ_2) est une solution d'un problème d'unification de termes, $\{n_{i1} = n_{i2}\}_{i=1..m}$. D'après le théorème de Robinson, un tel problème n'a pas de solution ou une solution unique (au changement de variables près) minimale (pour $\leq$), appelée "unificateur principal".

Il est alors clair que si S_0 est un ensemble d'unificateurs principaux (quand ils existent) de tous les problèmes $\{n_{i1} = n_{i2}\}_{i=1..m}$, S_0 à la propriété requise. □

Proposition 2

Il existe un ensemble de superpositions de deux graphes unique (au renommage de variables près), minimal (pour $\leq$) et fini. ■

PREUVE Prendre une superposition minimale dans l'ensemble précédent S_0. □

Nous allons noter $S_0(G_1, G_2)$ cet ensemble minimal.

A cause des contextes, l'ensemble des ambiguïtés de deux règles a une structure plus compliquée que l'ensemble des superpositions des modèles. Mais la propriété suivante va nous aider à mieux la décrire :

Proposition 3

Soient $r : G, C \to P$ une règle, $r' : G, \emptyset \to P$ la même règle avec un contexte vide, G_1, G_2 deux graphes, tels que $G_1 \to_{r'} G_2$ et $G_1 \not\to_r G_2$.

Soit ρ une substitution.

Alors $\rho G_1 \to_{r'} \rho G_2$ et $\rho G_1 \not\to_r \rho G_2$. ■

PREUVE L'hypothèse posée certifie l'existence de deux substitutions σ et τ, deux sous-graphes G' et G'' de G_1, et un contexte H dans C tels que :

$\sigma G = G'$

$\tau_{|Var(G)} = \sigma_{|Var(G)}$

$\tau H = G''$

En composant par ρ, nous avons :

$(\rho \circ \sigma)G = \rho G'$

$(\rho \circ \tau)H = \rho G''$

et $(\rho \circ \tau)_{|Var(G)} = \rho \circ (\tau_{|Var(G)}) = \rho \circ (\sigma_{|Var(G)}) = (\rho \circ \sigma)_{|Var(G)}$.

Ce qui implique $\rho G_1 \to_{r'} \rho G_2$ et $\rho G_1 \not\to_r \rho G_2$. □

Soient deux règles r_1 et r_2 avec comme modèles G_1 et G_2, et une superposition (σ_1, σ_2) de $S(G_1, G_2)$. Si nous retirons le contexte des règles, alors les deux applications des règles sur le graphe donnerons $\sigma_1 G_1 \cup \sigma_2 G_2$. Supposons maintenant que (σ_1, σ_2) ne soit pas une ambiguïté de r_1 et r_2. Alors une des deux règles (avec son contexte) ne s'applique pas à $\sigma_1 G_1 \cup \sigma_2 G_2$. La dernière proposition implique que pour chaque substitution τ, la superposition$(\tau \circ \sigma_1, \tau \circ \sigma_2)$, n'est pas une ambiguïté de r_1 et r_2.

Plus brièvement, nous avons donc :

Proposition 4

Si $c \in S(G_1, G_{2)} \setminus A(r_1, r_2)$ *et* $c \leq c'$ *alors* $c' \in S(G_1, G_2) \setminus A(r_1, r_2)$. ■

Cela signifie que le complément de $A(r_1, r_2)$ dans $S(G_1, G_2)$ a la même structure de "cône" que $S(G_1, G_2)$.

En particulier, nous avons :

Corollaire 1

Si les superpositions minimales des modèles de deux règles ne génèrent pas d'ambiguïtés pour les règles, alors les règles n'ont pas d'ambiguïtés. ■

6.2.3 Grammaire sans ambiguïté

Nous allons maintenant exploiter cette propriété pour définir une méthode qui, étant donnée une grammaire avec des ambiguïtés, ajoute un contexte aux règles, dans le but d'obtenir une nouvelle grammaire de graphes sans ambiguïté.

Soit $\mathcal{G} = \{r_1, \ldots, r_n\}$ une grammaire de graphes, avec $r_i : G_i, \emptyset \to P_i$.

Soient $(\sigma_{k1}^{ij}, \sigma_{k2}^{ij}), k = 1 \ldots a_{ij}$ les superpositions minimales de G_i et G_j. Comme les contextes sont vides au départ, ces superpositions correspondent aux ambiguïtés.

Supposons que pour chaque superposition minimale, c'est à dire quand deux règles peuvent s'appliquer simultanément sur une même partie de graphe, nous avons une méthode pour choisir la bonne règle à appliquer. Cela équivaut à donner une fonction C telle que $C(i, j, k) \in \{1, 2\}$: si $C(i, j, k) = 1$, ce qui signifie que l'on veut empêcher l'application de la règle r_j pour l'ambiguïté $(\sigma_{k1}^{ij}, \sigma_{k2}^{ij})$. Nous finalisons ce but en ajoutant au contexte $\sigma_{k1}^{ij} G_1$ de la règle r_j.

En effectuant cette opération pour toutes les règles et toutes les superpositions minimales, nous définissons une nouvelle grammaire $\mathcal{G}' = \{r'_1, \ldots, r'_n\}$, où :

$$r'_i : G_i, \bigcup_{j=1 \ldots n} \{\sigma_{k2}^{ij} G_j | k = 1 \ldots a_{ij},\ C(i, j, k) = 2\} \to P_i$$

D'après le corollaire 1, nous avons donc :

Proposition 5

Cette grammaire $\mathcal{G}'$ n'a pas d'ambiguïtés. ∎

Dans notre exemple illustré par la figure 17, pour la première superposition (1), nous ne voulons pas appliquer r_+ en premier, donc nous ajoutons juste le graphe modèle de la règle $r_\wedge$ au contexte de r_+. Ainsi, r_+ ne s'appliquera pas s'il y a une puissance sur son premier argument. La superposition 3 de la figure 17 est un cas identique à celui que nous venons de décrire. Donc l'ajout de ces deux contextes à la règle r_+ permet d'éviter l'application de la règle si un des arguments a un exposant. La superposition 2 implique l'ajout de r_+ dans le contexte de la règle $r_\wedge$, indiquant que dans ce cas de figure, la règle plus doit être appliquée avant la règle exposant. Enfin la dernière superposition n'a pas de sens mathématique à proprement parlé pour les règles de l'addition et de l'exposant. On peut donc ne pas tenir compte de cette superposition.

En réitérant cette opération pour chacune des superpositions, nous obtenons une nouvelle grammaire $\mathcal{G}'$, où l'application d'une règle dépend de son contexte :

$$\begin{aligned} r'_+ = \quad & \{A(Gauche, N(Plus, i_+, +), N(Expr, i_z, z)), \\ & A(Droite, N(Plus, i_+, +), N(Expr, i_t, t))\}, \\ & \{\{A(\textbf{Haut-Droite}, N(Expr, i_z, z), N(Expr, i_{u1}, u1))\}, \\ & \{A(\textbf{Haut-Droite}, N(Expr, i_t, t), N(Expr, i_{u2}, u2))\}\} \\ & \rightarrow N(Expr, i_+, +(z, t)) \end{aligned}$$

$$\begin{aligned} r'_\wedge = \quad & \{A(\textbf{Haut-Droite}, N(Expr, i_x, x), N(Expr, i_y, y))\}, \\ & \{\{A(Gauche, N(Plus, i_+, +), N(Expr, i_x, x)), \\ & A(Droite, N(Plus, i_+, +), N(Expr, i_{u1}, u1))\}, \\ & \{A(Gauche, N(Plus, i_+, +), N(Expr, i_{u2}, u2)), \\ & A(Droite, N(Plus, i_+, +), N(Expr, i_y, y))\}\} \\ & \rightarrow N(Expr, i_x, \wedge(x, y)) \end{aligned}$$

Cette grammaire simple (avec les deux règles) s'applique et aboutit à un résultat correct sur des formules comme $a^2 + b$, $a + b^2$, a^{b+c}, $x^{a+b} + y^{c+d}$, $x^{a+b^2} + y^{c^{i+j^2}+d}$, etc.

Sur notre exemple de grammaire, nous devons donc trouver une fonction C qui indiquera comment résoudre automatiquement le problème pour chaque ambiguïté. Nous avons défini de nouveaux critères afin d'obtenir cette fonction :

– la priorité mathématique de l'opérateur décrit dans la règle. Par exemple, la règle modélisant le produit (r_*) a une priorité plus grande que la règle de l'addition (r_+).

– une information graphique. Ce critère est important pour déterminer la priorité à droite pour l'application de la règle. Dans notre exemple, $\wedge$ a une priorité plus grande que $+$, mais ceci ne veut pas dire que la règle doit toujours s'appliquer avant l'autre. Cette propriété est vraie pour le cas "linéaire". Dans le cas de l'opérateur $\wedge$, il y a un parenthésage implicite pour les arguments et ceci est vrai pour tous les opérateurs qui ont des arguments sur "différents niveaux" comme : $x^{(y+z)}$, $x_{(y+z)}$, $\sum_{(i=1)}^{(n^2)} x_i$, $\frac{(x+y)}{(z+t)}$, etc.

Avec ces deux critères simples, nous sommes capables de définir, pour chaque règle de la grammaire, celle qui doit être appliquée à chaque cas pour lequel on a une ambiguïté.

6.3 Analyse structurelle

L'analyse syntaxique des langages bidimensionnels a été étudiée et est exposée par M. Tomita dans [**?**][1]. Nous ne présentons pas ici de considérations générales, mais uniquement les conséquences de l'analyse que nous avons présentée.

6.3.1 Analyse ascendante

L'approche envisagée précédemment est donc du type ascendante (*bottom-up*). Étant donné un graphe initial, l'application successive des règles de la grammaire, et donc la réécriture du graphe, va permettre l'analyse de chacune des sous-expressions composant la notation à reconnaître. Cette réécriture se poursuivra jusqu'à ce que le graphe ne contienne plus qu'un seul et unique nœud, qui contiendra la structure de l'expression mathématique.

La mise en œuvre des contextes assure que la grammaire est sans ambiguïté et donc qu'une règle ne s'applique que si aucune autre ne peut l'être au "même endroit" (avec au moins un nœud en commum). L'analyse peut donc s'effectuer sans retour arrière (*backtracking*).

En cas d'erreur, l'analyse s'arrêtera si aucune règle ne peut s'appliquer ; il restera alors un graphe contenant plus d'un seul nœud.

Pour une illustration de cette réécriture successive du graphe par application des règles, le lecteur pourra consulter l'annexe A. Elle contient les différentes étapes de la reconnaissance structurelle de la formule $n! = \prod_{i=1}^{n} i$. Nous allons détailler une de ces étapes de réécriture du graphe après avoir identifier qu'une règle peut s'appliquer ; la figure 18 illustre cette phase.

Sur la figure 18, partie gauche, le sous-graphe entouré est la partie identifiée par le modèle de la règle que l'on applique. Ce modèle est supprimé du graphe et remplacé par la production de la règle. Dans notre cas cette production peut-être un nouveau nœud contenant l'arbre de syntaxe abstraite résultant de la combinaison des différents éléments identifiés par

[1] TOMITA M., Parsing 2-dimensional language (1989).

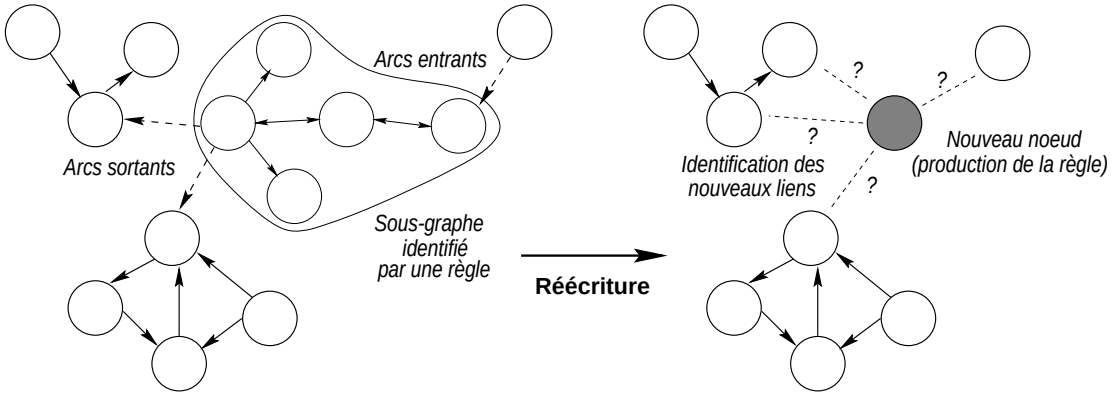


FIG. 18 – Réécriture d'un sous-graphe en un nœud

la règle.

Se pose alors le problème des arcs entrants et sortant du sous-graphe. La validité de ces arcs est à reconsidérer afin de connecter le nouvel élément (le nouveau nœud) au reste du graphe. La solution la plus simple serait de recalculer complètement le graphe incluant les modifications apportées par la réécriture. Toutefois, pour des raisons évidentes d'efficacité, une solution incrémentale est préférable. C'est la méthode que nous allons maintenant présenter.

6.3.2 Attributs synthétisés

La première étape est de synthétiser les attributs du nouveau nœud. Pour cela, les règles sémantiques telles que nous les avons introduites page 82, sont appliquées.

Un ensemble d'attributs, correspondant à des informations géométriques, est attaché à chacun des nœuds du graphe. On citera entre autres : la position géométrique, la taille relative de l'objet, la ligne de référence par rapport à laquelle le symbole est écrit, etc. Lors de la réécriture, un nouveau nœud est produit ; il est nécessaire de calculer la valeur de chacun de ses attributs. Les règles sémantiques permettent de spécifier la manière dont la valeur de l'attribut sera synthétisée (calculée à partir des fils, donc des valeurs positionnées sur les nœuds du sous-graphe que l'on remplace).

Comme nous l'avons présenté précédemment, l'attribut définissant l'enveloppe du symbole ou de la sous-expression, est synthétisé par calcul de la région rectangulaire englobant l'ensemble de la sous-expression reconnue. Les autres attributs, comme la taille ou la ligne de référence, seront calculés en fonction de l'opérateur considéré, donc de la règle sémantique attachée à une règle syntaxique donnée. Prenons un exemple :

– pour un opérateur s'écrivant de manière linéaire comme l'addition $a + b$, la nouvelle

ligne de référence sera la valeur moyenne des lignes de références pour chacun des objets composant l'opération.

– pour une division du type $\frac{a}{b}$, la nouvelle ligne de référence sera celle du trait de fraction.

Disposant de ces attributs pour le nouveau nœud, il est alors possible de faire une mise à jour locale du graphe, et donc de calculer les arcs reliant le nouveau nœud au reste du graphe.

6.3.3 Mise à jour incrémentale du graphe

Les informations géométriques, sous forme d'attributs des nœuds, permettent de déterminer quels sont les voisins géométriques du nœud. On peut donc appliquer au graphe l'algorithme de construction du graphe présenté précédemment ConstruireGrapheInitial en n'itérant le processus que sur le nouveau nœud.

```
procedure MiseAJourGraphe(Noeud n1, donnee−resultat Graphe g)
debut
  pour chaque type d'arc a faire
    pour chaque noeud n2 voisin de n1 faire
      si (n1<>n2) et Evaluation(n1, n2, a) alors
        ajouter l'arc a entre n1,n2 au graphe g
      finsi
    finpour
  finpour
  resultat g
fin
```

6.4 Optimisation

Une règle ne peut s'appliquer que si l'on vérifie que l'ensemble des sous-graphes contextuels n'est pas présent. L'augmentation de règles de la grammaire induit une augmentation des contextes d'une règle, même si les informations lexicales permettent de limiter les cas d'ambiguïtés. Fort de ce constat, nous avons tenté d'optimiser quelques peu l'application de la grammaire, afin qu'elle soit concrètement applicable à des formules de plusieurs dizaines de symboles, sans être pénalisé par un temps trop long d'exécution.

Étant donnée une règle, l'ensemble qui constitue le contexte doit être testé afin de savoir si l'on peut appliquer une règle et ainsi réécrire une partie du graphe. L'évaluation du contexte se limite à tester séquentiellement la présence ou non d'un sous-graphe avec un certain nombre de nœuds ou d'arcs communs avec le sous-graphe reconnu par la règle. Dès que l'un des contextes est reconnu, la règle ne peut pas s'appliquer. L'ordre des composants du contexte est donc important. Une organisation de celui-ci permet de diminuer le temps nécessaire pour décider si une règle peut ou non s'appliquer. Cette réorganisation des graphes constituant le contexte peut s'effectuer dynamiquement, en mettant en tête tout nou-

veau contexte qui fait échouer l'application de la règle. De cette manière, les contextes qui reviennent le plus souvent feront échouer l'application le plus tôt possible.

Cette optimisation a permis un gain de plus de 20% du temps nécessaire pour la reconnaissance d'une formule composée d'environ 25 symboles.

7 Conclusion

Ce chapitre a présenté les points principaux de l'architecture et des méthodes employées dans notre composant pour la reconnaissance structurelle des notations mathématiques. Le choix des grammaires de graphes pour l'analyse syntaxique s'est révélé être bien adapté au problème traité.

La réflexion menée sur l'application des grammaires de graphes au cas particulier de la reconnaissance des notations mathématiques, nous a conduit à l'enrichir de l'étude des contextes, obtenant ainsi une grammaire contextuelle. L'association entre les opérateurs à reconnaître et les règles de grammaires correspondantes, ainsi que les paramètres géométriques pris en compte ont permis de construire automatiquement ces contextes de règles. Il est ainsi facile pour un utilisateur de modifier la grammaire, apportant alors un degré supplémentaire d'évolutivité du composant *OFR*.

La dissociation des étapes d'analyses géométriques et syntaxiques permet une adaptation aux différents cas d'étude que nous avons rencontrés, à savoir la reconnaissance structurelle de notations typographiées et manuscrites.

Dans les chapitres suivants, certains aspects plus techniques sont évoqués. Notre expérience dans le domaine à même conduit à d'autres travaux. La méthode à base de grammaires de graphes, que nous avons présentée dans ce chapitre, à été utilisée avec succès dans des travaux de recherche sur une interface pour l'édition de formules mathématiques manuscrites (travaux réalisés par S. Smithies [**?**][1], Université de Otago, Nouvelle Zélande). Nous présenterons les prototypes que nous avons étudiés et utilisés et ceux que nous avons mis au point quant cela était nécessaire, afin d'effectuer les tests de validation de la méthode.

[1] SMITHIES S., *Freehand Formula Entry System*, Master's thesis (1999).

Chapitre IV

Applications et Validation

Comme nous l'avons exposé dans les deux chapitres précédents, *OFR* est un composant pour la reconnaissance structurelle de formules mathématiques destiné à être testé dans le cadre de diverses applications. Le prototype d'interface à la saisie d'expressions manuscrites pour les systèmes de calcul formel, que nous avons réalisé, est un exemple d'une telle application. Cependant, afin de valider la méthode utilisée dans le cadre de notations typographiées, nous avons été amenés à mettre en œuvre *OFR* avec un logiciel de reconnaissance de symboles mathématiques typographiés.

1 Notations mathématiques typographiées

L'architecture, que nous avons voulue souple, nous permet d'envisager l'utilisation d'un composant extérieur pour effectuer l'analyse de l'image et la l'identification des symboles. Cela permet d'éviter une perte de temps importante dans l'élaboration d'outils déjà existants. Nous avons donc étudié quelques-unes des applications disponibles, afin d'évaluer leurs aptitudes et fonctionnalités pour une utilisation dans le cadre particulier de notre étude : la reconnaissance des symboles constituant les notations mathématiques.

1.1 Un comparatif des logiciels pour la reconnaissance de symboles

L'évaluation et la comparaison des performances pour la reconnaissance des symboles est un sujet complexe. Quelques éléments plus "classiques" de comparaison peuvent être trouvés dans [**?**][1] et [**?**][2] ; [**?**][3] intègre un chapitre complet sur l'évaluation des performances en reconnaissance de documents. Plusieurs types de comparaisons sont envisageables, tant au niveau :

- de la méthode utilisée,
- des algorithmes sur lesquels reposent le processus de reconnaissance,
- des logiciels eux-mêmes.

Notre approche a été guidée par l'aspect pratique. Le but étant de réutiliser un logiciel effectuant la reconnaissance de symboles, nous n'avons pas considéré les méthodes ou algorithmes mis en jeu pour effectuer cette identification ; la qualité du résultat final (reconnaissance ou non du symbole, informations disponibles, etc) a été notre critère de choix principal. La liste des critères de comparaison utilisée, et que nous présentons plus en détail dans le paragraphe suivant, mélange des points importants et d'autres plus anecdotiques du point de vue de la reconnaissance de documents (comme les formats d'export des données par exemple). Toutefois, d'un point de vue pratique, tous ces critères ont pris part à notre évaluation des logiciels existants.

Ce paragraphe n'a pas pour but d'établir une liste exhaustive des logiciels pour la reconnaissance de caractères. Nous avons eu l'occasion de tester, au moins dans des versions d'évaluation, un certain nombre de logiciels permettant le traitement de l'image et la reconnaissance des symboles. Nous ne nous sommes pas cantonnés aux logiciels commerciaux, nous nous sommes également intéressés aux produits issus de la recherche.

Les systèmes commerciaux les plus connus que nous avons testés sont : *Cuneiform*, *OmniPage Pro*, *TextBridge Pro*, *TypeReader*, *ScanWorX*. Quelques logiciels expérimentaux de reconnaissance de caractères sont issus de la recherche, entre autres *Xocr* et *OCRchie*, deux logiciels permettant la reconnaissance de symboles typographiés. Nous les avons aussi intégrés à notre évaluation.

Lors du comparatif final, seuls les logiciels les plus intéressants, ou ceux faisant office de référence seront détaillés. Nous avons défini un ensemble de critères permettant leur évaluation en fonction de nos besoins spécifiques, à savoir l'intégration du composant dans notre étude sur les notations mathématiques.

---

[1] BAIRD H.S., Applications of multidimensional search to structural feature identification (1988).

[2] WILSON C.L., Evaluation of character recognition systems (1993).

[3] *Graphics Recognition : Algorithms and Systems* (1997).

1.1.1 Quelques critères

Nous présentons ici un ensemble de critères pour l'utilisation d'un logiciel de reconnaissance des caractères, dans le cadre particulier de l'identification des symboles mathématiques. Ces critères sont organisés par thèmes, du moins fondamental (formats traités) au plus important (analyse de document et reconnaissance des symboles). Le dernier thème, adaptabilité et évolutivité, regroupe les critères permettant éventuellement d'adapter le produit à nos besoins spécifiques si une fonctionnalité importante venait à manquer.

Formats traités

Ce paragraphe définit les points permettant d'évaluer le logiciel dans sa capacité à s'adapter aux données fournies et à restituer les données analysées.

- **Formats d'entrée (FE)** : quel est le type d'image que peut analyser le logiciel ? Ce critère, même s'il est important, n'en est pas pour autant primordial. De nombreux outils (tels que la bibliothèque *netpbm* sous *Unix*) permettent de convertir les images d'un format donné vers la plupart des formats connus. Seuls les formats propriétaires sont donc à éviter.
- **Formats de sortie (FS)** : sous quel format les données identifiées sont-elles fournies ? Il est souhaitable que le type des données fournies en sortie soit facilement analysable, ce qui signifie que les données produites ne doivent pas seulement être disponibles dans un format propriétaire (comme *Microsoft Word*), interdisant par là-même toute analyse des données.
- **Informations spatiales (IS)** : le format de sortie intègre-t-il les informations spatiales relatives aux symboles reconnus ? Les données concernant la boîte englobante d'un caractère seront privilégiées, mais tout autre type de données permettant de localiser le symbole de manière précise sera considérée comme acceptable.
- **Informations typographiques (IT)** : le format de sortie intègre-t-il les informations typographiques pour chacun des symboles ? Les données concernant la typographie utilisée sont nombreuses. Elles peuvent aller de la reconnaissance de la fonte utilisée, à des informations sur la calligraphie des symboles, comme italique ou gras, ou bien la taille de la fonte en points pour le caractère identifié.

Numérisation et traitement de l'image

Cette section regroupe les critères qui permettent de juger la simplicité et l'efficacité de la numérisation et des traitements préliminaires pour l'exploitation de l'image.

- **Seuil de numérisation automatique (SN)** : le seuil de numérisation est-il déterminé automatiquement ? Le seuil de numérisation, que nous avons présenté page 29, peut éventuellement être déterminé automatiquement pour les logiciels intégrant la possibilité d'effectuer la numérisation. Cette facilité donnée à l'utilisateur est plus un critère de confort d'utilisation qu'un critère déterminant dans notre choix.

- **Réduction du bruit (B)** : le logiciel intègre-t-il une étape de réduction du bruit ? La réduction du bruit (page 30) est l'identification des imperfections dues à l'étape de numérisation avec du matériel de faible qualité.
- **Réalignement (R)** : la détection d'une rotation et la correction sont-elles disponibles ? Une réorientation de l'image (page 31) est nécessaire sur les documents ayant subi une rotation lors de la numérisation. Cette étape permet d'obtenir de meilleurs résultats lors de la reconnaissance des symboles.

Analyse du document et reconnaissance des symboles

Ce paragraphe comprend tous les points relatifs à l'identification des différentes composantes d'un document et à l'ensemble des symboles reconnus par le logiciel. Nous n'avons pas intégré dans nos critères celui relatif à la reconnaissance de l'alphabet latin, tous les logiciels offrant cette possibilité pour un nombre de fonte plus ou moins grand.

- **Identification des composants (IC)** : les différents éléments d'un document sont-ils identifiés ? Un document est constitué de différents types d'éléments : zones de texte, images, logos, tableaux, graphiques, notations mathématiques, etc. L'étiquetage de ces différents composants, et donc leur localisation, permet d'effectuer des traitement spécifiques selon le type d'élément considéré.
- **Alphabet grec (G)** : le logiciel permet-il de reconnaître l'ensemble des symboles constituant l'alphabet grec ? Les symboles grecs sont fréquemment utilisés dans les notations mathématiques. Il est donc important que le logiciel soit capable d'identifier ces symboles, au même titre que l'alphabet latin.
- **Symboles mathématiques (SM)** : la reconnaissance des symboles mathématiques spécifiques est-elle correcte ? Outre les caractères latins et grecs, les notations mathématiques sont composées de symboles spécifiques (comme $\neq$, $\in$, $\forall$, ∞, $\int$, etc). Le logiciel doit bien entendu être capable d'identifier cette classe de symboles.

Adaptabilité et Évolutivité

Les critères suivants déterminent la possibilité de faire évoluer le logiciel afin de l'adapter à nos besoins spécifiques.

- **Apprentissage d'une fonte (AF)** : peut-on faire apprendre une nouvelle fonte ou de nouveaux symboles au système ? Dans le cas où le logiciel n'intègrerait pas l'ensemble des symboles que nous souhaitons être capables d'identifier, le logiciel doit permettre d'étendre l'ensemble des symboles reconnus.
- **Modification du format de sortie (M)** : Est-il possible de paramétrer le type des données fournies en sortie ? Dans le cas où le système ne fournit pas un format de sortie convenable, il doit être possible de l'adapter pour pouvoir disposer des informations qui nous sont utiles.
- **Disponibilité du code source (CS)** : Le code source de l'application est-il disponible et modifiable ? Si des modifications mineures sont nécessaires, il n'est possible

d'adapter réellement le logiciel que si l'on dispose du code source de l'application.

1.1.2 Protocole de comparaison

Les documents et formules qui servent d'échantillons de base sont tirés de la littérature et sont les plus variés possibles (contenant un maximum de symboles différents). Ces documents sont principalement composés d'éléments mathématiques, afin d'évaluer les logiciels selon les critères les plus importants pour notre étude. Nous avons donc tout particulièrement considéré la reconnaissance de l'alphabet grec, des symboles mathématiques, et la possibilité d'obtenir des informations sous un format complet (symbole reconnu et informations graphiques). Un exemple de document utilisé lors des tests est fournit en annexe C. Les documents ayant servit à l'établissement de ce comparatif sont au nombre de 2 :

- celui présenté en annexe C,
- et un document contenant les différents alphabets pour un test sur l'apprentissage de nouveaux motifs.

Chaque document était numérisé avec une résolution de 300 points par pouce. Un exemple de résultat fourni par le logiciel *OmniPage Pro 10* est présenté en annexe C. On ne peut que constater les problèmes rencontrés avec les notations mathématiques. Ils sont détaillés en annexe.

1.1.3 Tableau comparatif

Nous avons tenté de réaliser le test le plus significatif possible. Dans le tableau comparatif ci-dessous, les étoiles donnent les qualités relatives des différents logiciels, selon les critères que nous avons mentionnés. L'absence d'étoile signifie que le critère n'est pas vérifié ou qu'il n'a pas de sens par le logiciel considéré. Quand le critère est quantitatif, le nombre d'étoiles donne une idée du niveau atteint pour le logiciel considéré. Dans le cas de critères fonctionnels, la présence d'étoiles signifie que le critère est vérifié.

	Formats				Prétraitements			Reconnaissance			Adaptabilité		
	FE	FS	IS	IT	SN	B	R	IC	G	SM	AF	M	CS
Cuneiform	*			*									
Fine Reader	**			*	*		*	*			*		
OmniPage	**			*	*		*	*	*		*		
Recognita	**	*	*	*	*	*	*	*	*		*		
TextBridge	**	*	*	*	*	*	*	*					
Xocr	*											*	*
OCRchie	*	*	*			*	*				*	*	*

Ce tableau ne peut pas être considéré comme un test exhaustif des caractéristiques des logiciels de reconnaissance de caractères. Nous espérons toutefois que les éléments de com-

paraison qu'il présente démontrent que les logiciels de reconnaissance de symboles, qu'ils soient commerciaux ou issus de la recherche, ne proposent pas les fonctionnalités souhaitées dans le cadre de notre étude sur les notations mathématiques.

1.1.4 Conclusion

Ces tests sur les différents produits existants ont été réalisés une première fois au début de nos recherches (début 1996). Toutefois les résultats présentés ici ont été réactualisés avec les dernières versions de chacun de ces logiciels. On notera qu'à la première évaluation, ces applications ne permettaient pas l'apprentissage de nouveaux symboles par l'utilisateur. Même si cette fonctionnalité a été ajoutée depuis, permettant ainsi de faire un apprentissage des symboles mathématiques, le format d'export des résultats ne permet toujours pas d'obtenir les informations géométriques nécessaires.

OCRchie semblait la solution la plus proche de nos besoins, mais l'apprentissage d'une nouvelle fonte était très fortement lié aux caractères ASCII ; cela nous a donc empêché d'étendre le système à l'apprentissage de l'alphabet grec et des symboles mathématiques. Il est aussi fortement orienté dans le domaine de la reconnaissance de texte : recherche d'une structuration en ligne des éléments, et association des composantes connexes (voir le paragraphe 1.2.2).

On notera aussi la formation d'un consortium pour la réalisation d'une librairie gratuite de reconnaissance des caractères (création fin 1998) ; il pourrait être intéressant d'en réaliser une évaluation pour des travaux futurs. Nous n'avons malheureusement pas pu bénéficier de ces travaux au moment opportun.

Nous avons donc orienté une partie de nos recherches vers le développement d'un composant de reconnaissance des symboles mathématiques, qui puisse être réutilisable par d'autres, à l'avenir.

1.2 Développement d'un OCR dédié et adaptatif

Nous présentons dans ce paragraphe les principaux points du système d'identification des symboles que nous avons implémenté. Ce développement a fait l'objet de l'encadrement du stage de maîtrise de Donatello Bussacchini et Pierre-Marie Gandoin. Les méthodes utilisées et les algorithmes implémentés reprennent les techniques "classiques" utilisées pour l'identification de motifs (voir chapitre I, paragraphe 6.1). Nous avons utilisé un modèle combinant description syntaxique et statistique des formes : définition de vecteurs de caractéristiques, exploités par une technique de classification.

Bien que les traitements et les résultats fournis soient plus spécifiquement adaptés à notre étude particulière, l'ensemble des fonctionnalités conviendrait à une extension du système, la plus simple et la plus large possible. Nous n'avons pas intégré les éléments de pré-traitement de l'image, comme l'élimination du bruit, ce prototype n'ayant par pour but d'être un logiciel complet, mais plutôt orienté vers une expérimentation rapide. Son implémentation en *Java* et son architecture modulaire permettent toutefois d'envisager une extension et éventuellement une distribution du système. Le système complet représente environ 10.000 lignes de code *Java*.

Pour chaque forme tirée de l'image originale, on extrait un certain nombre de caractéristiques issues à la fois de la représentation vectorielle et de la représentation par points de la forme. Ces caractéristiques seront utilisées pour trouver la ou les formes les plus proches dans un dictionnaire construit lors de la phase d'apprentissage. Ces dictionnaires contiennent l'association entre un vecteur de caractéristiques de la forme et une description (chaîne alphanumérique) qui sera retournée comme résultat en cas d'identification.

1.2.1 Numérisation du document

La numérisation du document s'effectue en 400 points par pouce à l'aide du logiciel *xvscan*. Ce logiciel est basé sur *xv*, le célèbre logiciel de visualisation d'images sous *Unix*, auquel un module de numérisation des documents a été ajouté. L'utilisation de *xvscan* permet aussi de bénéficier des fonctions de traitement d'image de *xv*, comme la réduction du bruit dont nous avons déjà parlé. Nous obtenons ainsi une image "propre" de la notation que l'on souhaite analyser.

1.2.2 Extraction des composantes connexes

L'extraction des symboles se fait par identification des composantes connexes. Nous entendons par composante connexe un ensemble continu de points qui constituent une forme. Afin d'éviter un chargement complet de l'image à analyser, l'algorithme développé effectue une construction incrémentale.

```
pour chaque ligne de l'image faire
    isoler les segments noirs
    si uneetune seule region possede une frontiere avec le segment alors
        ajouter le segment a la region
    sinon si plus d'une region possede une frontiere avec le segment
        unifier toutes les regions en une seule
        ajouter le segment a la nouvelle region
    sinon
        creer une nouvelle region avec le segment
    finsi
finpour
```

Un détail est tout de même à souligner concernant la distinction entre symbole et composante connexe. Certains symboles sont composés de plusieurs composantes connexes. C'est par exemple le cas des lettres i et j, qui sont toutes deux composées d'une forme pour la partie principale, surmonté d'un point. Beaucoup de symboles mathématiques ont aussi la particularité d'être constitués de plusieurs composantes connexes. On citera entre autre : $\leq$, $\ll$, $\approx$, $\Im$, etc.

Dans la plupart des systèmes de reconnaissance des symboles que nous avons pu rencontrer, l'association des composantes connexes d'un symbole s'effectue à un bas niveau, c'est à dire lors de l'identification du caractère. Si cette solution est tout à fait satisfaisante dans le cas de la reconnaissance de texte (association de i et du .), ce n'est pas toujours vrai pour la reconnaissance structurelle des expressions mathématiques. Cette association dépend du contexte d'un symbole, elle est donc syntaxique et non statique. Prenons comme exemple la formule suivante :

$$\frac{a}{\overline{b}} = 1$$

L'identification du symbole d'égalité, constitué de deux traits horizontaux, ne peut pas être effectuée hors de son contexte syntaxique. Cette reconstitution des caractères ne doit donc pas être effectuée par le système d'identification des symboles mais doit être réalisée lors de l'analyse syntaxique.

Le système d'identification des symboles que nous avons développé n'est donc pas strictement équivalent à un système de reconnaissance des caractères. Le système renvoie comme résultat les composantes connexes constituant les notations. À charge pour le système de reconnaissance de la structure d'effectuer l'identification des symboles à partir des composantes connexes, en fonction de l'environnement, et donc du contexte d'un symbole.

1.2.3 Extraction des caractéristiques des symboles

Les différentes composantes connexes qui constituent la notation à analyser étant identifiées, il reste à caractériser chacune d'elle, afin de réaliser la reconnaissance proprement dite. Pour cela, on extrait deux types de représentation de l'image :

- une représentation point à point (*bitmap*),
- une représentation vectorielle.

La représentation point à point est une image en 256 niveau de gris, de dimensions 4 par 5, produite à partir de l'image de la composante connexe isolée. Une interpolation a été utilisée pour rendre cette représentation plus fiable dans le cas de formes non symétriques.

La représentation vectorielle est une suite de contours représentés par des points. C'est sur cette représentation vectorielle qu'un certain nombre de caractéristiques vont être calcu-

lées, en particulier la position des contours qui permet ainsi de déterminer la boîte englobante du symbole. Un traitement préliminaire de l'image est réalisé avant d'extraire d'effectuer la vectorisation. Plusieurs filtres sont appliqués (voir matrices 1) pour obtenir le contour vectoriel de la forme (voir figure 2).

$$\begin{pmatrix} 2 & 4 & 2 \\ 4 & 16 & 4 \\ 2 & 4 & 2 \end{pmatrix}$$

(a) Lissage

$$\begin{pmatrix} 2 & 4 & 2 \\ 4 & 4 & 4 \\ 2 & 4 & 2 \end{pmatrix}$$

(b) Épaississement

$$\begin{pmatrix} 0 & 4 & 0 \\ 4 & 32 & 4 \\ 0 & 4 & 0 \end{pmatrix}$$

(c) Extraction du contour

FIG. 1 – Filtres appliqués à l'image

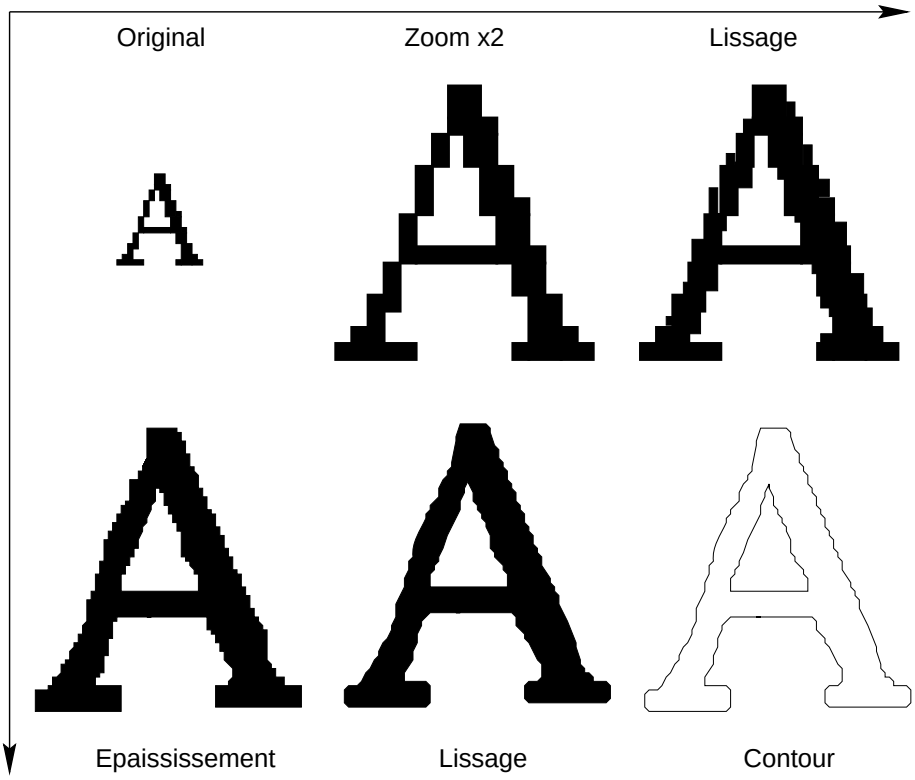
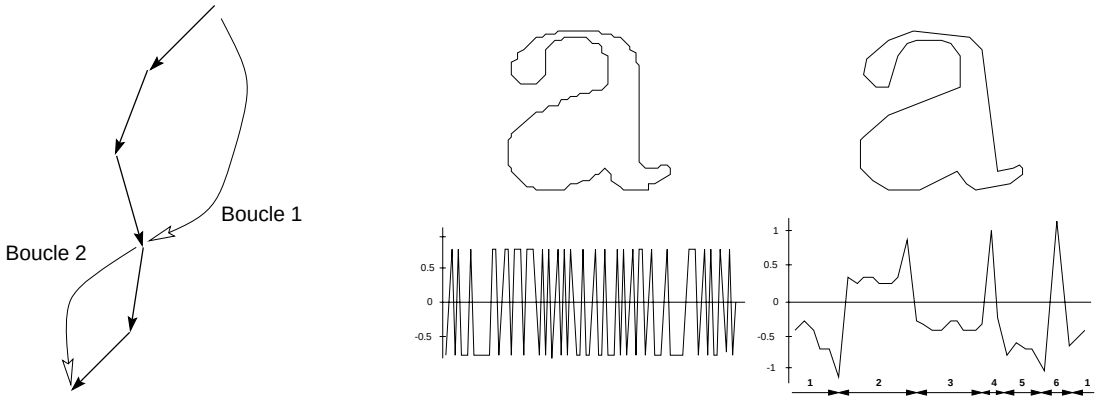


FIG. 2 – Extraction du contour vectoriel

On définit les boucles (voir IV.3(a)) comme l'angle formé par une suite de vecteurs dont les angles ont le même signe. La vectorisation permet de rendre les formes moins sensibles aux variations locales (bruit, mais aussi taille de la fonte du caractère, etc). Toutefois, le problème d'une représentation vectorielle infiniment précise, c'est à dire sans dégradation

par rapport à l'image originale) contient certaines particularités qui rendent la détection des boucles trop fine et donc inutilisable. Une dégradation du contour vectoriel permet de généraliser cette représentation et d'en déduire un critère, les boucles, exploitables comme caractéristiques de la forme (voir IV.3(b)).



(a) Définition (b) Représentation vectorielle et boucles

FIG. 3 – Extraction des boucles

L'utilisation d'un vecteur de caractéristiques pour chaque forme et leur classification est un principe très répandu dans le domaine de la reconnaissance de symboles. L'avantage est d'être assez simple à mettre en œuvre, mais elle suppose que les paramètres soient suffisamment discriminants.

Lors des premières tentatives, nous avions recherché un maximum de caractéristiques (vecteur principale, pattes, angles droits, points de rebroussement, etc.), mais les résultats obtenus étaient très sensibles aux petites variations, et ces caractéristiques n'étaient pas forcément très représentatives du caractère considéré ; les différences inter-classes n'étaient pas assez prononcées. Nous avons donc réduit le nombre et affiné les critères, afin d'avoir les classes d'objets les plus représentatives et donc les résultats les meilleurs possibles. L'ensemble des caractéristiques retenues comme vecteurs de données représentatives d'un symbole sont :

- la représentation bitmap (dégradée sur une image 4x5),
- le rapport largeur/hauteur de la forme,
- la représentation vectorielle (telle que nous l'avons présentée),
- le nombre de contours intérieurs,
- les boucles de la forme.

1.2.4 Apprentissage

L'apprentissage du système a consisté à générer, sous forme d'images, les symboles issus des polices utilisées pour les mathématiques dans LaTeX (voir annexe B). Le tableau

ci-dessous détaille l'ensemble des fontes utilisées pour l'apprentissage.

Fonte LaTeX	Taille(s) en points							
	5	6	7	8	9	10	12	17
cmb						*		
cmbx	*	*	*	*	*	*	*	
cmbxsl						*		
cmbxti						*		
cmex					*	*		
cmmi	*	*	*	*	*	*	*	
cmr		*	*	*	*	*	*	*
cmsl				*	*	*	*	
cmsy	*	*	*	*	*	*		
cmti			*	*	*	*	*	
cmu						*		
cmvtt						*		

Plus de 22.000 motifs (environ 250 symboles différents) ont été appris, couvrant l'ensemble des symboles mathématiques, des alphabets latin et grec, des chiffres, le tout pour plusieurs fontes (roman, sans-serif, etc), avec plusieurs types de variations (normal, gras, italique).

Les échantillons fournis lors de la phase d'apprentissage sont des images constituées d'un seul symbole, à une résolution de 400 et 600 points par pouce, non bruitées. Une description leur est associée lors de cette phase d'apprentissage (chaîne de caractères qui devra être retournée par le système lors de la reconnaissance).

On notera aussi qu'il est possible d'effectuer un rétro-contrôle des résultats, en forçant le réapprentissage d'un symbole que l'on considère comme souvent mal reconnu, augmentant par là-même son taux de reconnaissance.

1.2.5 Identification du caractère et résultat fourni

L'identification des caractères, ou plus exactement des composantes connexes constituant les caractères, s'effectue par recherche des caractéristiques les plus proches dans l'ensemble des symboles appris par le système. Cette recherche dans les dictionnaires, construits lors de la phase d'apprentissage, peut fournir plusieurs résultats si cela est nécessaire. Dans le cas où plusieurs symboles ont un vecteur de caractéristiques proche, l'ensemble des possibilités sera retourné par le système. À chacune de ces possibilités est associé un indice de confiance ou d'erreur, permettant ainsi au système d'identification de la structure de pouvoir éventuellement faire un choix parmi les alternatives, si cela s'avère nécessaire.

$$\sum_{n=0}^{\infty} \frac{z^n}{n!}$$

Pour la formule précédente, les informations fournies par le logiciel sont :

```
31 0 86 25 infty
5 50 114 166 Sigma
150 36 184 74 z 0.1 Z 0.7
192 17 227 42 n 0.15 u 0.9
146 106 234 110 hbar
155 149 199 187 n 0.1 Uparrow 1.2
210 127 219 169 exclam
210 177 219 186 dot 0.1 comma 0.4
0 201 35 226 n 0.15 u 0.9
42 203 83 206 hbar 0.08 bar 0.2
42 217 83 220 hbar 0.08 bar 0.2
92 188 117 227 0 0.2 O 0.3
```

Pour chaque composante connexe identifiée on retrouve les informations suivantes :

- les informations géométriques : les coordonnées haut-gauche, bas-droite de la région occupée par le motif,
- le symbole reconnu avec un taux de confiance (indication du taux d'erreur),
- d'éventuelles alternatives au symbole reconnu (avec leur taux de confiance respectif).

1.2.6 Évaluation des performances

Durant le protocole d'évaluation des performances, nous nous sommes intéressés à trois cas en particulier :

- la police a été apprise avec les bonnes dimensions (taille en points) et se trouve dans le dictionnaire,
- la police se trouve dans le dictionnaire mais pas dans les bonnes dimensions,
- enfin, la police ne se trouve pas dans le dictionnaire.

Le résultat a été jugé correct si le caractère a été reconnu. Une autre série de tests a consisté à considérer le caractère comme reconnu, mais indépendamment de sa casse (majuscule ou minuscule).

La police de caractères Times et l'ensemble des fontes mathématiques ont été apprises pour les tests d'évaluation. Les polices de caractères Helvetica et Roman n'ont pas été apprises, afin d'évaluer la méthode utilisée sur des éléments à priori non connus par le système. Les résultats obtenus sont les suivants :

- par comparaison des images (représentation en niveau de gris) :

Nom et taille de la police	Taux de reconnaissance
Times 12 pt	97%
Times 8 pt	95%
Helvetica 12 pt	45%

– par comparaison de la représentation vectorielle :

Nom et taille de la police	Taux de reconnaissance
Times 12 pt	93%
Times 8 pt	85%
Helvetica 12 pt	40%

– par utilisation mixte des deux méthodes :

Nom et taille de la police	Taux de reconnaissance
Times 12 pt	99,4%
Times 8 pt	98%
Helvetica 12 pt	52%
Roman 12 pt	92%

Après avoir mis en place ce module de reconnaissance des symboles typographiés, nous avons orienté nos recherches vers un composant qui réalise la reconnaissance de l'écriture manuscrite et des symboles mathématiques manuscrits les plus usuels.

2 Notations mathématiques manuscrites

Les pré-requis restent sensiblement les mêmes que dans le cas typographié, à savoir la reconnaissance de l'écriture manuscrite de symboles de l'alphabet latin, grec, des principaux symboles mathématiques et numériques. Dans le souci d'éviter un nouveau développement lourd, nous avons étudié les différentes possibilités de récupérer un composant capable d'effectuer la segmentation et la reconnaissance de symboles manuscrits. Nos recherches nous ont conduit à établir une collaboration avec l'Université de Duisburg, dont nous présentons ici le logiciel.

2.1 Présentation du système NeuroGraph

Le système a été développé par l'équipe dirigée par Gerhard Rigoll de l'Université de Duisburg ([**?**][1], [**?**][2], [**?**][3]). Il permet la reconnaissance d'environ une centaine de caractères ou symboles :

[1] RIGOLL G., KOSMALA A., ROTTLAND J. & NEUKIRCHEN C., A comparison between continuous and discrete density hidden markov models for cursive handwriting recognition (1996).

[2] KOSMALA A., ROTTLAND J. & RIGOLL G., Improved on-line handwriting recognition using context dependent hidden markov models (1997).

[3] KOSMALA A. & RIGOLL G., Recognition of on-line handwritten formulas (1998).

– l'alphabet en lettres majuscules et minuscules, et les chiffres : [A-Z], [a-z], [0-9],
– les symboles mathématiques principaux : $+ - \cdot : / \longrightarrow \hat{} \sqrt{} \sum \prod \int , ' = < > \leq \geq \rightarrow \leftrightarrow \approx ! \infty$,
– un sous-ensemble des lettres grecques les plus utilisées dans les notations mathématiques : $\alpha\ \beta\ \gamma\ \lambda\ \mu\ \Delta\ \pi\ \omega\ \epsilon\ \tau\ \phi$,
– et enfin un ensemble de caractères de type "parenthésage" : ( ) [ ] { }.

L'inconvénient majeur de cette application est qu'elle est prévue pour un écrivain unique. Le système, basé sur l'utilisation des chaînes de Markov (*HMM*), doit être entraîné par la saisie de chacun des symboles au moins 2 fois, écrits de manière distincte et isolés sur une tablette à digitaliser. À ce long processus se rajoute un ensemble de 100 formules de mathématique et de physique, utilisées pour entraîner le modèle, ainsi qu'un ensemble de 30 formules supplémentaires pour le tester. La figure 4, ci-dessous, donne un aperçu du système, que nous allons présenter plus en détail.

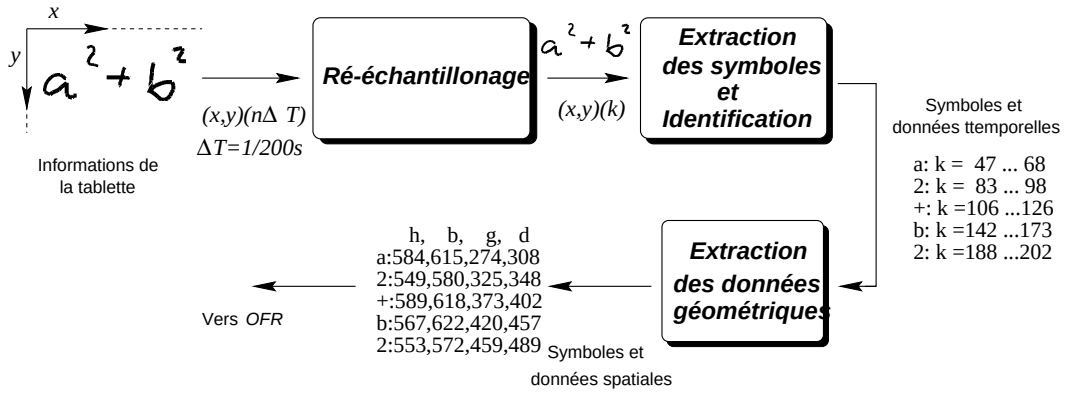


FIG. 4 – Présentation du système pour la reconnaissance des symboles manuscrits

2.2 Reconnaissance des symboles manuscrits

Les données, issues d'une tablette à digitaliser, sont capturées avec un taux d'échantillonnage de 200Hz. Il en résulte un flux d'informations constitué des coordonnées cartésiennes de la trajectoire du stylo, ainsi qu'une information relative à la pression du stylo sur la tablette. Cette dernière information est de type binaire ; on sait donc si le stylo était levé ou non par rapport à la tablette. Cette donnée sera entre autre utile pour effectuer la segmentation automatique des symboles.

Le premier traitement des données consiste à les rééchantillonner selon un vecteur de longueur constante (voir figure 5). Outre le fait de diminuer le volume des données (environ 1 :2 ou 1 :3), l'avantage principal de cette opération est de permettre l'abstraction du modèle de la vitesse et des accélérations lors de l'écriture, tout en préservant les données spatiales.

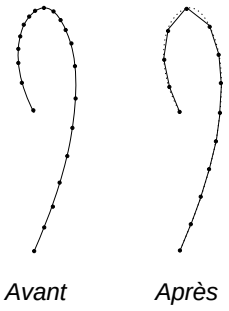


FIG. 5 – Rééchatillonnage des données

En effet, plusieurs expérimentations ont permis de montrer que cette information, dans le cas particulier des expressions mathématiques, introduit une perturbation importante, entraînant des erreurs sur la reconnaissance des symboles.

Plusieurs informations sont donc extraites des données disponibles :

- des informations en ligne :
  - l'orientation des vecteurs constituant la trajectoire décrite par le stylo $cos(\alpha), sin(\alpha)$,
  - la différence entre deux vecteurs consécutifs $cos(\Delta\alpha), sin(\Delta\alpha)$ avec $\Delta\alpha = \alpha(k) - \alpha(k-1)$,
  - l'information de pression du stylo $0|1$.
- des informations hors ligne :
  - l'image sous-échantillonnée de la trajectoire du stylo.

Toutes ces données forment des flux, présentés au système à base de chaînes de Markov.

Un modèle avec 12 états est utilisé pour les lettres majuscules et les grands symboles mathématiques, comme les sommes ou les produits. Les lettres minuscules et les opérateurs mathématiques plus petits sont reconnus par un modèle à 8 états ; les plus petits symboles comme, le point ou la virgule, par un système à 3 états.

Pour la reconnaissance proprement dite, l'algorithme de Viterbi est utilisé. Il est alors possible de déterminer la séquence d'états $\mathbf{q}^\star$ d'un ensemble de chaînes de Markov λ pour une séquence donnée $\mathbf{O}$.

$$P(\mathbf{O}, \mathbf{q}^\star|\lambda) = \max_{\mathbf{q}} P(\mathbf{O}, \mathbf{q}|\lambda) \tag{1}$$

Ceci peut être exploité efficacement, pour les notations mathématiques, en analysant l'alignement de chacune des trajectoires, avec la meilleure correspondance possible dans le modèle de Markov. Le résultat supplémentaire que l'on obtient par cette méthode est le point de départ et d'arrivée de la trajectoire d'un symbole. La figure 6 illustre la transformation des coordonnées temporelles dont on dispose en sortie de l'algorithme de décodage ou de reconnaissance (début et fin de la trajectoire d'un symbole).

En parcourant cette trajectoire et en isolant les minima-maxima en x et en y, on obtient les coordonnées de la boîte englobante du symbole identifié. On notera toutefois que, dans ce cas précis, le système développé ne permet pas de fournir d'alternatives à la reconnaissance

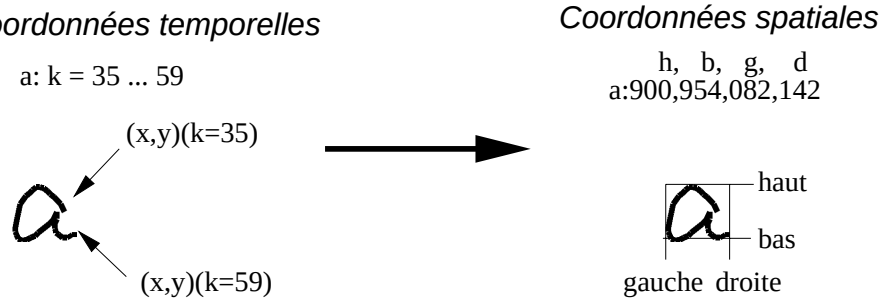


FIG. 6 – Transformation des coordonnées temporelles en coordonnées spatiales

d'un symbole.

On notera également que, contrairement au système d'analyse statique de l'analyse d'images, les données temporelles fournissent un certain nombre d'informations supplémentaires. Elles permettent par exemple de résoudre, dans certains cas, l'association des composantes connexes. Comme nous l'avons présenté dans le paragraphe 1.2.2, il est impossible d'associer certaines composantes connexes lors de l'analyse de l'image ; il est nécessaire d'effectuer cette analyse dans un contexte plus large, et il est donc indispensable de l'intégrer à l'analyse syntaxique. Ce n'est pas le cas pour l'écriture manuscrite, où nous disposons d'informations temporelles incorporées dans le modèle de reconnaissance des symboles. Il sera donc possible d'identifier directement l'ensemble des composantes connexes d'un symboles comme i ou $=$. Cette partie pourra alors être déchargée du système d'analyse syntaxique dans le cas manuscrit.

Les informations minimales nécessaires à la reconnaissance de la structure d'une notation mathématique étant réunies, elles peuvent être envoyées au processus *OFR* pour effectuer ce travail. Afin de valider et de tester l'interaction de l'ensemble de ces composants, nous avons développé un prototype d'interface permettant l'intégration et la communication entre ces différents modules.

3 Irma : Dualité du système développé

Pour valider notre étude sur l'utilisation d'une méthode à base de grammaires de graphes, nous avons réalisé un prototype intégrant la plupart des principes exposés durant notre étude sur la reconnaissance des expressions mathématiques. Ce prototype, appelé *Irma*, est basé sur le composant *OFR*, et constitue une plate-forme d'expérimentation très souple de reconnaissance des notations mathématiques typographiées et manuscrites. Cette application intègre tous les composants que nous avons présentés dans les chapitres et paragraphes précédents à savoir :

– le logiciel de reconnaissance des caractères et symboles mathématiques typographiés,

– le système de reconnaissance des caractères et symboles manuscrits,
– le composant *OFR* pour l'analyse géométrique et structurelle des notations mathématiques.

La souplesse de la méthode à base de grammaires de graphes permet cette adaptation aux deux cas d'étude : notations mathématiques typographiées [**?**][1] et manuscrites [**?**][2]. Cette dualité du système *Irma* (Interface pour la Reconnaissance des notations MAthématiques) a été possible grâce à l'architecture logicielle que nous avons mise en place, ainsi qu'à l'ensemble des paramètres qui permettent d'adapter le comportement d'*OFR*.

3.1 Adaptation au format des données

Les moyens de paramétrer l'ensemble des données en entrée démontrent la faculté d'adaptation et la souplesse du composant *OFR*. Cette paramétrisation se caractérise par plusieurs points :

– **unification des données** : les données fournies au système effectuant la reconnaissance structurelle sont issues de logiciels ayant tous un format de communication des données différent. Une certaine unification des données attendues est donc nécessaire. Les informations impératives, que nous avons présentées dans le chapitre III, sont le symbole identifié et la région occupée par celui-ci. D'autres informations peuvent être nécessaires pour effectuer l'analyse géométrique. Ce sont la taille relative du symbole et la ligne de référence par rapport à laquelle le caractère s'écrit. Ces dernières sont extrapolées si le système de reconnaissance des symboles ne fournit pas ces informations.
– **règles lexicales** : le typage lexical permet lui aussi d'unifier et de simplifier la grammaire de graphes. Il permet d'abstraire les symboles et de généraliser les règles de la grammaire.
– **paramétrage géométrique** : la construction géométrique doit être adaptable aux différentes "typographies" que l'on peut rencontrer dans les variations des notations. Cela va des variations très subtiles entre deux typographies différentes (positionnement d'un exposant qui peut être plus ou moins élevé), aux variations très importantes entre une notation typographiée et une manuscrite. Un ensemble de paramètres et de seuils de tolérance permettent au système de s'adapter aux différents cas rencontrés.

3.2 Adaptation aux notations reconnues

– **règles syntaxiques** : comme nous avons déjà pu le mentionner, il n'est pas possible de prétendre reconnaître l'ensemble des notations mathématiques. Celles-ci n'étant

[1] LAVIROTTE S. & POTTIER L., Mathematical formula recognition using graph grammar (1998).

[2] KOSMALA A., LAVIROTTE S., POTTIER L. & RIGOLL G., On-line handwritten formula recognition using hidden markov models and context dependent graph grammars (1999).

que partiellement standardisées, et des notations personnelles ne pouvant être incluses d'emblée, il est nécessaire de pouvoir spécifier le schéma de nouvelles notations. Les règles syntaxiques, à base de graphes, permettent d'étendre ou de modifier, selon les besoins, l'ensemble des notations reconnues.

3.3 Adaptation du format de sortie

- **sémantique** : une fois l'arbre de syntaxe abstraite construit, par application de la grammaire de graphes sur le graphe généré, à partir des données fournies par le logiciel d'identification des symboles, des informations sémantiques doivent permettre une exploitation du résultat obtenu. Comme nous l'avons présentée page 56, l'interprétation d'un même arbre de syntaxe peut-être différente selon la sémantique des objets. Nous n'avons pas pu effectuer tous les tests souhaités dans ce domaine du fait du caractère isolé des formules analysées. Plusieurs conventions sur la sémantique sont donc intégrées à certaines objets mathématiques : A, B, C, D, ... symbolisent des matrices, f, g, ... des symboles fonctionnels, t, x, y, z, ... des variables, a, b, c, d, e, des constantes etc.
- **communication des résultats** : disposant d'un arbre décoré avec les informations sémantiques synthétisées, la communication du résultat sous diverses formes est possible : LaTeX (dans ce cas, les informations sémantiques sont superflues), ou tout système intégrant le protocole *OpenMath*.

3.4 Communication à l'aide d'OpenMath

L'intégration d'*OpenMath* à *Klone* permet de communiquer les résultats obtenus à d'autres applications utilisant ce protocole. *OpenMath* est en passe de devenir un standard incontournable pour la communication des expressions mathématiques inter-applications. L'intégration d'*OpenMath* nous permet de communiquer les résultats obtenus à tous les systèmes intégrant cette bibliothèque de communication des expressions mathématiques. C'est le cas, par exemple, de *Mathematica*, *Reduce*, *Maple*, d'autres systèmes devant suivre très prochainement.

La figure 7 présente l'utilisation du protocole *OpenMath* pour la communication d'expressions mathématiques. L'application *Irma*, en haut, a reçu les symboles reconnus de la part du logiciel chargé d'effectuer cette tâche. Après analyse de cette notation par la grammaire de graphe, *Irma* dispose de l'arbre de syntaxe abstraite de la formule. Cette structure peut être convertie, par simple parcours de l'arbre, en une quelconque autre représentation, dont *OpenMath*. Sous sa forme ascii, ce protocole se présente sous une forme *XML*, avec des balises pour notifier les variables (<OMV>), les entiers (<OMI>), les symboles (<OMS>), etc. Afin de montrer que c'est bien la formule avec sa sémantique qui est communiquée à *Mathematica*, et pas seulement une représentation de la formule, l'exemple montre que l'on

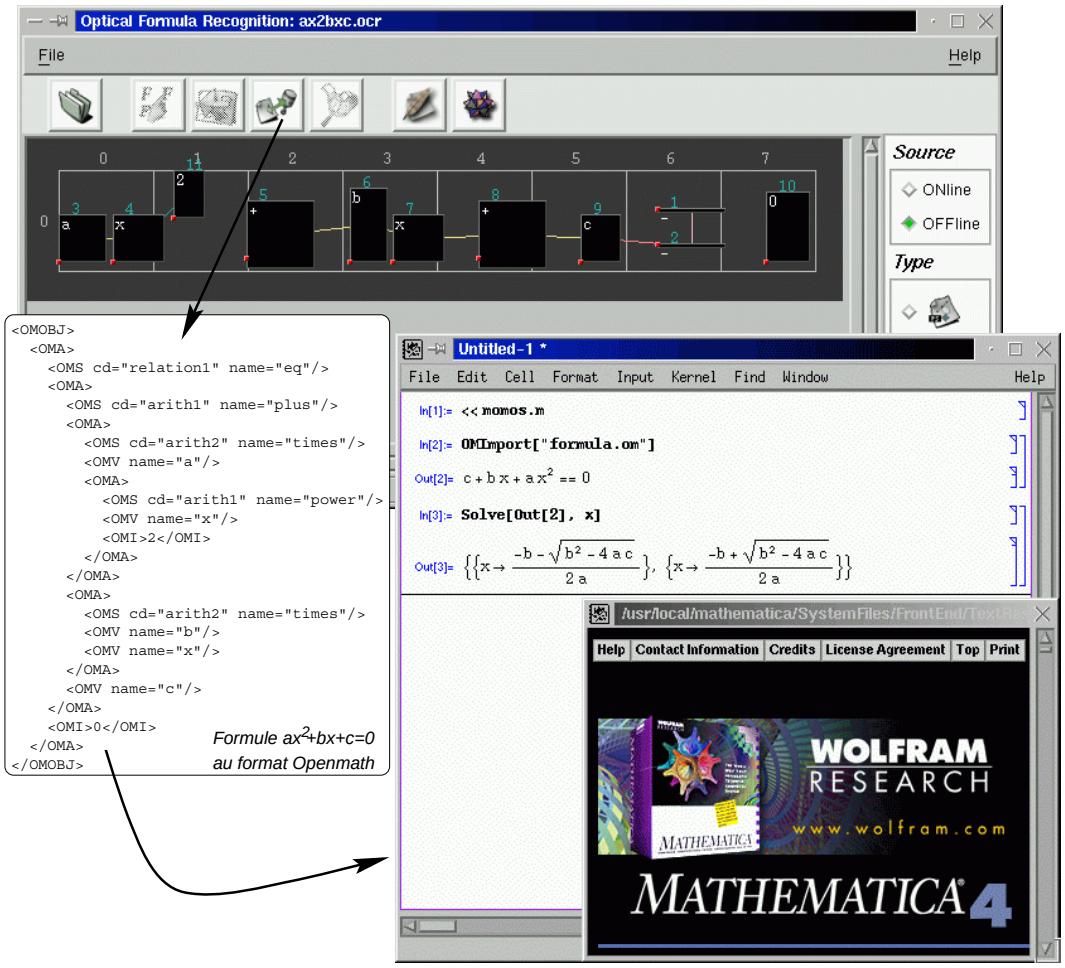


FIG. 7 – Communication entre Irma et Mathematica via OpenMath

peut faire une recherche des solutions de cette expression.

Couplé au système de reconnaissance de l'écriture manuscrite, *Irma* peut être utilisé comme une interface à la saisie d'expressions mathématiques beaucoup plus conviviale et intuitive pour l'utilisateur. Comme nous l'avons montré dans le chapitre I, les systèmes de calcul formel, ou, dans un cadre plus général, les interfaces à la saisie de notations mathématiques, sont limités à deux techniques principales : l'utilisation d'une syntaxe linéaire ou de palettes de modèles. L'utilisation d'une interface manuscrite à la saisie de notations mathématiques est une alternative des plus intéressantes et semble très apprécié des utilisateurs, en particulier ceux qui ne sont pas familiers des interfaces existantes. C'est ce qui ressort d'une étude faite par S. Smithies [**?**][1], sur la saisie des notations mathématiques manuscrites.

4 Validation

Comme nous l'avons exposé précédemment, *Irma* intègre l'ensemble des composants que nous avons présentés, à savoir : le système de reconnaissance de l'écriture typographié, le logiciel de reconnaissance des symboles manuscrits et le composant *OFR*. Ce prototype nous a permis de faire une évaluation de la méthode proposée, à la fois pour la reconnaissance structurelle de notations mathématiques typographiées, mais aussi pour les expressions manuscrites.

4.1 La grammaire de graphes

Pour effectuer ces tests de validation, nous avons développé la grammaire de graphes la plus significative possible quant aux notations identifiées. Cette grammaire regroupe donc un échantillon des différents types de notations mathématiques. Nous entendons par différents types de notations la variété des opérateurs mathématiques (opérateur infixe, préfixe, etc) et des représentations graphiques (linéaire, bidimensionnelle, etc). Nous pouvons donc regrouper les notations reconnues en plusieurs catégories :

- **linéaires** :
  - préfixe : moins unaire ($-x$),
  - postfixe : notation factorielle ($x!$),
  - infixe : l'addition, la soustraction, la multiplication, les égalités ou inégalités, etc ($x + y$, $x - y$, $x \times y$, $x = y$, $x < y$, ...)
  - implicite : la multiplication implicite ($2x$),
  - parenthèses : l'ensemble des symboles qui permettent de regrouper des éléments comme les (x), $[x]$, $\{x\}$, etc.

[1] SMITHIES S., *Freehand Formula Entry System*, Master's thesis (1999).

- **verticales** : les fractions ($\frac{x}{y}$)
- **bidimensionnelles** :
  - explicites : les sommes ou produits, les composants binomiaux, etc. ($\sum_{x}^{y}$, $\prod_{x}^{y}$, C_n^p, ...).
  - implicites : les indices, les exposants, les matrices, etc (x^y, x_y, $\begin{pmatrix} x & y \\ y & x \end{pmatrix}$, ...)

La grammaire est constituée d'une trentaine de règles permettant d'analyser toutes les notations que nous avons présentées. Chacune de ces règles est constituée de plusieurs éléments :

- **la règle syntaxique** : elle se décompose de la manière suivante :
  - le modèle : c'est le sous-graphe que l'on cherche à identifier,
  - le contexte : c'est un ensemble de sous-graphes pouvant se trouver en superposition avec une partie du sous-graphe du modèle ; si tel est le cas, la règle ne peut pas s'appliquer,
  - la production : c'est un nœud qui remplace le sous-graphe modèle que l'on a identifié ; cette partie de la règle permet de spécifier l'arbre que l'on construit à partir des éléments du modèle,
  - la priorité : elle correspond à la priorité de l'opérateur mathématique que la règle modélise,
  - l'associativité : elle permet de spécifier le type d'associativité de l'opérateur (associatif à gauche, à droite, etc),
  - l'arité : elle représente l'arité de l'opérateur considéré (unaire, binaire, n-aire)
- **la règle sémantique** : à chaque règle syntaxique est associée une règle sémantique qui permet de spécifier la manière dont les attributs de la production seront calculés.

On notera que le contexte ne doit pas être spécifié, mais est déduit automatiquement grâce à l'ensemble des informations qui composent la règle : propriétés géométriques du modèle, priorité, associativité, arité.

On distingue deux types de règles dans la grammaire : les règles correspondant à l'identification d'un opérateur et de ses opérandes, et les règles reconstituant les symboles ou entités mathématiques. Examinons, tout d'abord, les règles permettant d'identifier les opérations mathématiques très classiques comme : la multiplication, l'addition ou bien encore la factorielle.

Num	Opérateur	Modèle	Production	Prio.	Assoc.	Arité
1	$X!$	Excl A -g-> Expr X	Expr "fact(X)"	115	gauche	unaire
2	$X * Y$	Expr X <-g- Mult A -d-> Expr Y	Expr "mult(X,Y)"	50	gauche	n-aire
3	$X + Y$	Expr X <-g- Plus A -d-> Expr Y	Expr "plus(X,Y)"	10	gauche	n-aire

Ces règles sont toutes composées de plusieurs éléments : le modèle, la production, l'arité de l'opérateur représenté, sa priorité, etc. Le modèle est le sous-graphe que l'on cherche à identifier dans le graphe construit à partir des données. Si nous prenons l'exemple de la règle 3, le modèle est constitué de trois nœuds, identifiés par les variables X, Y et A, X et Y étant de type Expression et A de type Plus, et de deux arcs qui relient ces nœuds. La production de la règle est un nœud de type Expression, dont la valeur sera produite en construisant l'arbre ayant pour racine l'opérateur plus, et pour sous-arbres les valeurs associées à X et à Y.

Complémentairement à ces règles d'identification des opérations, nous avons un ensemble de règles, permettant de réaliser l'assemblage des symboles constitués de plusieurs composantes connexes. Comme nous l'avons présentée dans le paragraphe 1.2.2, cette analyse doit s'effectuer lors de l'analyse syntaxique de la formule ; un même symbole peut avoir une signification différente selon son contexte graphique (exemple du trait horizontal). Dans le cas manuscrit, ces règles sont pour la plupart inutiles, ce regroupement des composantes connexes étant réalisé par le composant de reconnaissance des symboles, grâce aux informations temporelles.

Ces règles servent donc à reconstituer les symboles ayant un sens syntaxique. Cette association peut être assez évidente dans les cas suivants : $\imath$ et $\cdot$ pour le i, $\jmath$ et $\cdot$ pour le j, etc. Elle peut également porter sur l'association de symboles plus ambigus, leur utilisation étant très répandue dans les notations. C'est le cas tout particulièrement du trait horizontal, qui intervient dans plusieurs symboles ($\leq$, $\geq$, $=$, $\simeq$, etc.) et dans diverses notations (moins unaire, moins binaire, trait de fraction, conjugué, etc.).

Examinons quelques exemples pour illustrer ce type de règles. Le tableau ci-dessous fournit les règles permettant d'identifier les symboles, comme le point d'exclamation (règle 6), le symbole égal (7), mais aussi permettant d'effectuer la réunion d'une lettre et de son point ($\imath$ et $\cdot$). Cette même règle pourra aussi identifier la notation classique de la dérivée première (une lettre surmontée d'un point), l'interprétation étant laissée à l'analyse sémantique de l'arbre construit.

Num	Opérateur	Modèle	Production	Prio.	Assoc.	Arité
4	$\dot{\square}$	Lett X -h-> Dot Y	Lett "dot(X)"	203	non	unaire
5	…	Dot X -d-> Dot Y -d-> Dot Z	Elip X "..."	202	non	unaire
6	!	VBar X -b-> Dot Y	Excl Y " !"	201	non	unaire
7	=	HBar X -h-> HBar Y	Equa X "="	200	non	unaire

Comme nous l'avons déjà mentionné, une règle sémantique est associée à chacune des règles syntaxiques. Dans le tableau ci-dessous, nous présentons les règles sémantiques associées aux règles syntaxiques ci-dessus. Ces règles permettent de synthétiser les attributs des nœuds, comme la position de la ligne de référence de l'expression ou bien encore la taille des symboles de l'expression. Pour la règle numéro 6, la nouvelle ligne de référence de l'objet sera déduite du symbole identifié par Y, la taille de la fonte sera la fonte maximum des symboles identifiés par les variables X et Y.

Num	Règles sémantiques	
4	baseline=Baseline(X)	font=Font-max(X)
5	baseline=Baseline-average-y(X,Y,Z)	font=Font-max(X,Y,Z)
6	baseline=Baseline(Y)	font=Font-max(X,Y)
7	baseline=Baseline-middle(X,Y)	font=Font-max(X,Y)

Comme nous l'avons déjà mentionné, la reconnaissance des symboles intervient à part entière dans l'identification des notations typographiées. Pour preuve, examinons les trois règles suivantes :

Num	Opérateur	Modèle	Production	Prio.	Assoc.	Arité
7	$=$	HBar X -h-> HBar Y	Equa X "="	200	non	unaire
8	$\frac{X}{Y}$	Expr X <-h- HBar A -b-> Expr Y	Expr "frac(X,Y)"	100	non	binaire
9	$\overline{X}$	Expr X -h-> HBar A	Expr "conj(X)"	150	non	unaire

Ces deux règles permettent d'identifier le symbole égal, les fractions, et la notation conjugué. Le trait horizontal est le point commun entre toutes ces notations. La construction automatique du contexte va introduire la notion de contexte dans chacune de ces règles, comme nous l'avons expliqué page 87. Celle-ci aura pour effet d'ajouter à la règle 7 la recherche de symboles au-dessus et au-dessous, et en cas de présence de ces symboles, empêchera l'application de la règle. Prenons l'exemple suivant :

$$\frac{X}{\overline{Y}} = 1$$

Cette formule signifie : X divisé par conjugué de Y est égal à 1. Même le lecteur peut douter à la lecture de cette formule, mais le contexte des symboles permet de résoudre sans ambiguïté l'interprétation de cette notation. La différence de longueur des traits horizontaux entre X et Y ainsi que l'éloignement des traits ne sont pas des critères suffisamment significatifs

pour différencier à coup sûr ces deux cas. Par contre, la présence de symboles au-dessus et au-dessous permet d'analyser correctement la notation.

Pour un symbole autant utilisé dans les notations mathématiques que le trait horizontal (moins unaire, moins binaire, fraction, conjugué, égalité, etc.), la construction automatique des contextes des règles introduit un nombre important de contextes à vérifier pour pouvoir appliquer la règle. Ceci entraîne évidemment un surcoût pour l'application d'une règle, mais évite un système contraignant de retours arrières dans l'analyse.

La grammaire construite peut toutefois être très concise, tout en restant très complète. Le système de substitution des termes, élément de base de chacun des composants du graphe (voir les définitions d'un nœud, arc, graphe page 70), permet cette expressivité.

Num	Opérateur	Modèle	Production	Prio.	Assoc.	Arité
10	(X) ou $[X]$ ...	Par X G <-g- Expr A -d-> Par Y D	Expr "GD(X)"	115	non	unaire
11	$X < Y$ ou $X \geq Y$ ...	Expr X <-g- Comp A C -d-> Expr Y	Expr "C(X)"	5	non	binaire

Le première règle permet d'identifier les notations de type "parenthèsage", alors que la deuxième est une règle générique de reconnaissance des opérateurs d'égalité ou d'inégalité. Le type de l'opérateur analysé est récupéré grâce à la substitution de C par l'opérateur identifié. Cette unique règle permet ainsi d'identifier de nombreuses notations comme $=$, $<$, $>$, $\leq$, $\geq$, mais aussi $\approx$, $\neq$, etc.

4.2 Résultats

Les résultats obtenus grâce à l'ensemble des composants que nous avons présentés sont de deux types :

- la reconnaissance structurelle de formules mathématiques typographiées à l'aide d'un système quelconque (LaTeX par exemple). L'adaptation à un autre formateur d'expressions mathématiques est très simple. Il suffit d'adapter les paramètres géométriques à la typographie. Le système de reconnaissance des symboles développé permettant l'apprentissage de nouveaux symboles, il pourra être réutilisé. On notera toutefois qu'un utilisateur pourra aisément remplacer ce module de reconnaissance par un autre de son choix.
- la reconnaissance structurelle des notations manuscrites à l'aide du système *Neurograph* que nous avons présenté. Ce système étant dépendant du scripteur, l'évaluation des résultats a été assez limitée. Le temps d'apprentissage, nécessaire au système à base de chaînes de Markov, est lourd et contraignant (voir paragraphe 2.1). Étant données ces contraintes, nous n'avons pas trouvé beaucoup de candidats pour réaliser une évaluation aussi poussée que ce que nous aurions souhaité.

4.3 Quelques exemples caractéristiques

Afin de mettre en lumière le potentiel d'*OFR*, nous présentons ici quelques exemples sélectionnés pour leur intérêt. Deux critères ont été retenus :

- l'intérêt d'un point de vue mathématique (problème de résolution des priorités des opérateurs),
- la difficulté liée à l'arrangement géométrique des symboles qui composent la notation (proximité des symboles, faible taille, type de liaisons entre les symboles difficile à définir, ...).

Dans un souci de clarté, nous avons regroupé les exemples en différentes catégories selon le type d'opérateurs utilisés dans la notation.

Présentons tout d'abord quelques exemples de notations mathématiques simples. Le premier type de difficultés liées à la reconnaissance structurelle des expressions mathématiques est l'analyse correcte des expressions, en respectant les priorités des opérateurs qui les composent. Vient ensuite le problème de l'identification des unités lexicales. La grammaire graphique doit intégrer l'association des symboles pour reconstituer les entités en fonction du contexte. L'analyse de la formule IV.8(a) pourra donc être différente selon que, la grammaire considère mc comme une entité lexicale ou si les variables utilisées dans les notations sont réduites à un seul symbole. Seule une information extérieure pourra déterminer le bon cas et donc le type de grammaire à utiliser. La deuxième notation IV.8(b) est non ambiguë car le point marque bien la multiplication entre les variables m et c. Pour la troisième, il y a bien multiplication implicite entre l'entier 2 et la lettre p.

$$E = mc^2 \qquad E = m \cdot c^2 \qquad 2p^{x+2} + 1$$

(a) (b) (c)

FIG. 8 – Notations simples

Les formules trigonométriques ne posent pas de problèmes particuliers, mis à part quelques difficultés de priorité et de portée des opérateurs. La figure 9 donne quelques exemples de formules reconnues par le système.

$$\tan z = \frac{\sin z}{\cos z} \qquad \cos^2 z + \sin^2 z = 1 \qquad \sin z_1 - \sin z_2 = 2\cos\left(\frac{z_1 + z_2}{2}\right)\sin\left(\frac{z_1 - z_2}{2}\right)$$

(a) (b) (c)

FIG. 9 – Notations trigonométriques

Les opérateurs ternaires, comme la somme, le produit ou l'intégrale peuvent se présenter sous plusieurs formes : l'opérateur et son argument ($\sum \square$), l'opérateur avec une seule borne ($\sum_{\square} \square$) ou bien avec deux ($\sum_{\square}^{\square} \square$). La grammaire doit permettre la reconnaissance de toutes ces formes. Une difficulté supplémentaire se situe aussi au niveau de la localisation des bornes : elles peuvent être au-dessus et au-dessous de l'opérateur ou bien en position d'indice ou d'exposant comme c'est la cas de l'intégrale dans l'exemple IV.10(c) de la figure 10. Enfin, les bornes de l'opérateur peuvent avoir une largeur supérieure à la largeur de l'opérateur, induisant des erreurs possibles sur les relations entre les symboles constituant la borne et ceux de l'opérande. L'ensemble de ces formules est correctement reconnu par notre système.

$$\sum_{0 \leq i \leq n} x^i + 1 \qquad \prod_{i=0}^{n} x^i + 1 \qquad \int_0^{\infty} g + \sum_a^b f$$

(a) (b) (c)

$$\int_a^b \left((1+x)^2 (x-1) + e^{x^2} \right) dx$$

(d)

FIG. 10 – Notations de type somme, produit, intégrale

Les notations du type indice ou exposant sont parmi les plus difficiles à identifier, car elles sont uniquement basées sur l'agencement des opérandes. Aucun opérateur n'est utilisé pour marquer la liaison entre les symboles. Les exemples de la figure 11 ci-dessous illustrent bien la complexité potentielle due à la combinaison des notations indices exposants. Les possibilités de ces combinaisons indices/exposants vont du cas le plus simple et le plus courant (expression IV.11(a)) au cas extrême de la figure IV.11(e). Ce dernier exemple, même s'il ne correspond pas à un cas concret, illustre bien la limite du système *OFR*. Celui-ci bute sur la reconnaissance de cette dernière expression, ne déterminant pas correctement l'ensemble des relations entre les symboles (confusion entre les indices de l'exposant et les exposants de l'indice). Les autres expressions sont, elles, correctement appréhendées par le système.

Voici maintenant des formules un peu plus "complexes", faisant intervenir un panel plus large de types de notations. La complexité du graphe construit, et plus particulièrement pour l'expression IV.12(c), induit un temps d'application de la grammaire important. La reconnaissance de cette formule peut être visualisée grâce à une petite animation, en consultant la page suivante : http ://www.inria.fr/cafe/Stephane.Lavirotte/Ofr.

Enfin, les notations matricielles sont prises en compte par *OFR*. La plupart des expériences passées ont buté sur l'extension de leur méthode au cas des notations implicites,

$x_1^2 + x_2^2 + ... + x_n^2$ (a)

C_{n+1}^{2p+1} (b)

$4x_{i+j^2}^{a+b_3}$ (c)

$x_{a_1^{2b^5+c}}^{4g+e^6} y_7^{\alpha}\beta$ (d)

$X^{X_{X_{X_{X_X}}}}_{X_{X^{X^X}_{X_{X_{X_X}}}}}$ (e)

FIG. 11 – Notations de type indice, exposant

$\frac{\frac{x^{-201}+y^{523}}{abce}}{(x^2+y^2)(x^3+y^3)}$ (a)

$S(2m) = (-1)^{m+1} B_{2m} \frac{(2\pi)^{2m}}{2(2m)!}$ (b)

$\left(\frac{1}{x^2+1}\right)^{(n)} = (-1)^n.n!\frac{\sum_{0\leq 2p\leq n}(-1)^p C_{n+1}^{2p+1} X^{n-2p}}{(X^2+1)^{n+1}}$ (c)

FIG. 12 – Formules plus complexes

comme les notations matricielles. La figure 13 fournit trois exemples. Notons toutefois que dans le cas manuscrit, ce type de notation obtient un taux de reconnaissance plus faible que dans le cas typographié. La différence est due aux imprécisions de la segmentation des sous-expressions.

$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ (a)

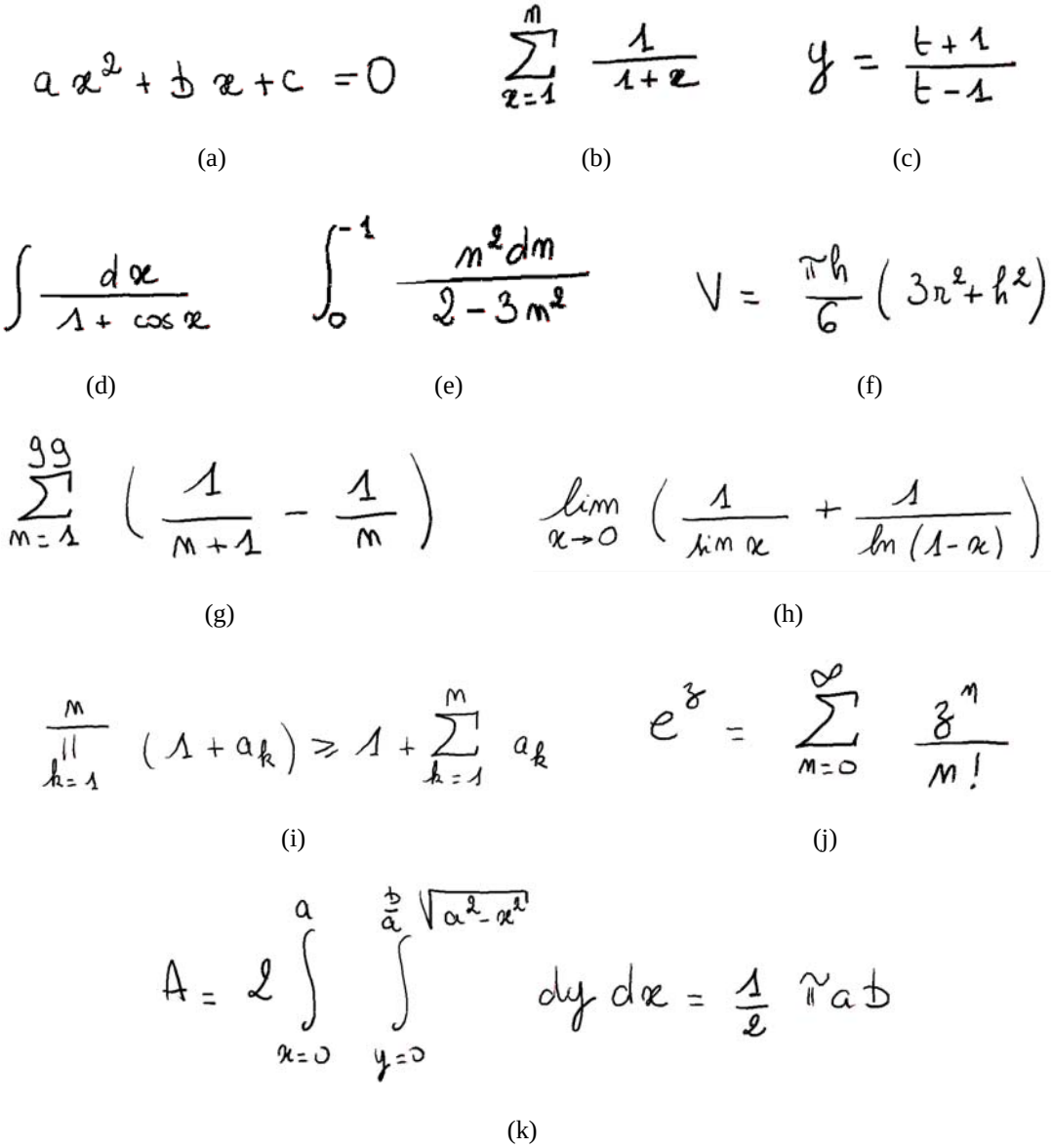
$\begin{pmatrix} y^3 & 4 \\ x & z+2 \\ z^3 & 4 \end{pmatrix}$ (b)

$D\begin{pmatrix} 1 & x_0 & ... & x_0^n \\ 1 & x_1 & ... & x_1^n \\ ... & & & \\ 1 & x_n & ... & x_n^n \end{pmatrix} = \prod_{i<j}(x_i - x_j)$ (c)

FIG. 13 – Notations matricielles

Pour conclure sur la présentation de ces résultats, voici un ensemble de formules manuscrites faisant intervenir les différents types de notations que nous avons présentées (figure 14), toutes reconnues par le système *OFR*.

Les problèmes rencontrés pour la reconnaissance de ces notations sont du même ordre que dans le cas typographié. Les contraintes sur le positionnement ont été assouplies, afin de satisfaire les aléas du placement des symboles inhérents au cas manuscrit. Quelques problèmes subsistent, principalement dus à la détection des relations entre les symboles. Lors de l'analyse géométrique, l'ajout au graphe de liens erronés peut entraîner une mauvaise



(a) (b) (c)

(d) (e) (f)

(g) (h)

(i) (j)

(k)

FIG. 14 – Exemples divers de formules manuscrites

reconnaissance ou un échec. Les problèmes de cas limite, seuil de détection entre des symboles alignés ou en position d'indice ou d'exposant, sont plus nombreux que dans le cas typographié. Ceux-ci ne permettent pas une identification aussi précise des notations que celles présentées figure 11, qui elles, sont correctement analysées dans le cas typographié. Enfin, pour conclure sur les résultats de la reconnaissance des notations manuscrites, nous précisons aussi qu'il peut être nécessaire d'adapter le système à l'utilisateur.

Les quelques exemples que nous fournissons dans ce paragraphe sont donnés à titre indicatif. Mais, profitant de ce propos sur les notations identifiées par *OFR*, nous tenons à souligner à nouveau la nécessité d'un vrai travail de classement des expressions pertinentes, qui devrait être mis au point par la communauté des personnes travaillant ou ayant travaillé sur le sujet. Ce type de travail, qui a été réalisé pour la reconnaissance des symboles manuscrits par exemple, permettrait de mettre au point une bibliothèque de formules mathématiques caractéristiques et intéressantes sur le plan mathématique ou de la représentation. Une telle librairie d'exemples de tests permettrait aussi de comparer plus précisément les différentes approches existantes, tout en facilitant le travail de mise au point d'un système.

5 Conclusion

Dans le cadre d'un travail de recherche, il n'a pas été possible d'investir assez de temps dans la réalisation d'un composant finalisé. Nous avons concentré nos efforts sur l'utilisation et l'amélioration des grammaires de graphes, appliquées au cas particulier de la reconnaissance des formules mathématiques.

Afin de disposer de plus de temps pour valider la méthode, nous avons tenté de réutiliser des composants existants pour effectuer la reconnaissance des symboles composant les notations mathématiques. Nous avons donc étudié les différents logiciels disponibles et la possibilité d'apprentissage, afin d'intégrer l'ensemble des symboles nécessaires. Cette recherche, pour le cas typographié, s'est révélée infructueuse, et nous a donc conduit à développer un composant spécifique réutilisable par d'autres, pour des études futures. Dans le cas manuscrit, la recherche a été fructueuse, entraînant une collaboration avec l'Université de Duisburg et l'obtention de résultats encourageants.

Chapitre V

Implémentation d'OFR

Le but de notre recherche est la mise au point de techniques et d'algorithmes afin de réaliser la reconnaissance structurelle de formules mathématiques. Dans cette optique, le choix d'un langage pour l'implémentation ne se base pas sur les performances, mais plutôt sur la facilité de développement. Cette souplesse de développement repose sur plusieurs critères, comme la disponibilité de bibliothèques de fonctions, la qualité de la documentation, mais aussi et surtout dans notre cas, la capacité à intégrer les programmes utiles à la réalisation du prototype complet.

En effet, la mise au point d'algorithmes de reconnaissance structurelle d'expressions mathématiques suppose, comme nous l'avons déjà mentionné, que l'on dispose des symboles composant la formule, ainsi que de leur position. Dans un souci d'économie de temps et pour bénéficier des meilleurs outils possibles, il est bien entendu souhaitable d'utiliser des logiciels existants nous fournissant ces informations, qui sont un pré-requis à notre travail.

Le langage choisi doit donc permettre d'intégrer facilement des programmes extérieurs se chargeant de réaliser les tâches "annexes". L'efficacité de ce langage en termes de performances n'est pas primordiale, mais il doit permettre un développement rapide. En effet, la mise au point de solutions algorithmiques suppose une réécriture parfois importante. Une solution précédemment retenue devenant obsolète, il est alors parfois nécessaire de réécrire une partie importante du code source. Ou bien, plus simplement, pour tester de nouvelles fonctionnalités, il est appréciable de ne pas avoir à investir un temps important dans im-

plémentation d'un programme qui ne sera peut-être pas conservé pour la version finale de l'algorithme.

Tous ces critères nous ont fait nous intéresser aux langages de script, qui présentent toutes ces caractéristiques. Dans ce chapitre, nous présenterons les langages de script, nous les comparerons et nous justifierons nos choix.

1 Langages de script

Durant les quinze dernières années, un changement fondamental est apparu dans la manière dont on programme un ordinateur. Ce changement est une transition, de la programmation dans des langages de bas niveau (langages système) comme *C* ou *C++*, vers une programmation à l'aide de langages de script comme *Perl* ou *Tcl*.

Les langages de script, comparativement aux langages système, sont conçus pour réaliser des tâches différentes, ce qui implique des caractéristiques particulières à ces langages. Les langages de programmation système sont conçus pour construire des structures de données et des algorithmes à partir de rien, en se basant sur les primitives de bas niveau d'un ordinateur, comme un mot mémoire. À l'opposé, les langages de script sont étudiés pour "agglomérer" un ensemble de composants puissants, pré-existants.

Afin de comprendre les différences entre langages de script et langages de programmation système, il est important de comprendre comment les langages ont évolué.

1.1 Bref historique des langages

Les langages système ont été introduits comme une alternative aux langages assembleurs, pour lesquels les programmeurs sont obligés de gérer des détails de programmation de bas niveau comme l'allocation d'un registre ou l'appel de procédures. Il en résulte qu'il est très difficile d'écrire et surtout de maintenir des programmes de grande taille en assembleur. À la fin des années 1950, des langages de programmation de plus haut niveau sont apparus comme *Fortran* et *Algol*. Dans ces langages, une instruction ne correspond plus exactement à une instruction machine ; un compilateur transcrit chacune des instructions en une séquence d'instructions binaires. Plusieurs langages bien connus ont découlé de *Algol*, comme *PL/1*, *Pascal*, *C*, *C++*. Les langages de programmation système sont moins performants que l'assembleur, mais ils permettent un développement plus rapide et une maintenance des programmes plus commode.

Les langages de script comme *Perl*, *Python*, *Tcl*, *Visual Basic* ou encore les shells Unix représentent un style différent de programmation par rapport aux langages système. Les langages de script supposent qu'il existe déjà une collection de composants écrits dans d'autres

langages. Les langages de script ne sont pas conçus pour écrire une application à partir de rien mais plutôt pour assembler des composants. Ils sont d'ailleurs souvent référencés comme "langages d'intégration de composants" ou "langages d'assemblage de composants".

Les langages de script et système sont complémentaires et apparaissent sur la majorité des plates-formes de développement, et ce depuis les années 1960. Ces langages sont typiquement utilisés ensemble dans le cadre suivant : les composants sont développés à l'aide de langages de programmation système et assemblés avec les langages de script. Cependant, plusieurs faits, comme des machines plus rapides, de meilleurs langages de script, l'augmentation de l'importance des interfaces graphiques et de l'architecture en composants, ainsi que la croissance d'Internet, ont considérablement accru l'utilisation des langages de script.

Non seulement les langages de programmation et leurs utilisations ont fondamentalement changé, mais une évolution dans la conception de l'architecture logicielle est aussi à constater. Plusieurs styles d'architectures ont émergé et se sont imposées comme des standards de programmation [**?**][1]. Les plus connus et utilisés sont schématisés à l'aide de la figure 1 ; cette représentation est issue de http ://www-ast.tds-gn.lmco.com/arch/guide.html.

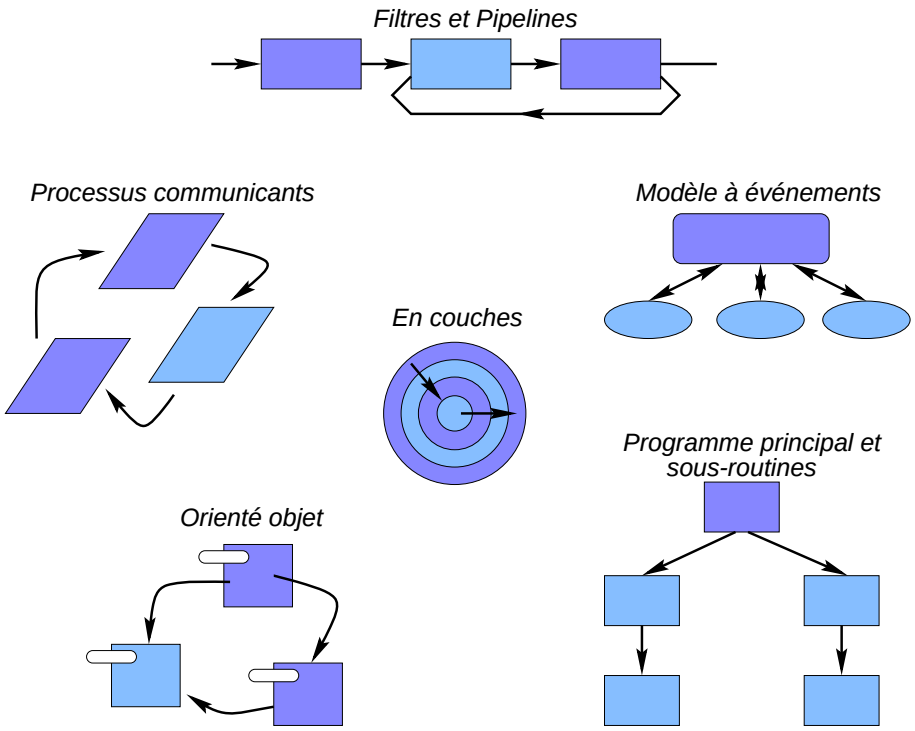


FIG. 1 – Styles d'architecture logicielle

[1] GARLAN D. & SHAW M., An introduction to software architecture (1993).

L'architecture en composants spécialisés que nous avons présentée est du type filtres, pipeline. Avec ce type d'architecture, la classe de langage la plus adaptée est bien sûr celles des langages de script. Ils permettent de faire la clue entre les différents filtres ou programmes. L'exemple le plus connu de ce type d'architecture est le shell Unix. Unix fourni une notation pour connecter les différents composants (représentés par des processus) et un mécanisme pour mettre en œuvre les pipes. Les systèmes à base de filtres ou de pipelines ont plusieurs propriétés appréciables :

- une conception facilitée : chaque composant peut être vu comme une boîte noire,
- la possibilité de réutiliser les composants : sortir un composant de son contexte global est simple,
- le système peut facilement être maintenu et étendu : possibilité d'ajouter de nouveaux "filtres" dans la chaîne,
- il autorise, dans certains cas, l'exécution concurrentielle des processus.

Mais ce type de système a aussi ses désavantages :

- la nécessité pour chaque filtre d'analyser et de produire les données dans un format particulier ;
- une perte éventuelle de performances due à l'analyse et à la génération des données ;
- il n'est pas adapté à une application intéractive, mais plutôt à une chaîne de calcul pour aboutir à un résultat en fonction d'une entrée.

Chaque composant entrant dans la chaîne de calcul peut avoir sa propre architecture logicielle (programme principal et sous-routines, orienté objet, en couches, etc).

1.2 Quelques caractéristiques des langages

Les différents systèmes que nous présenterons par la suite sont tous basés sur des langages interprétés, nombreux et variés. Nous présentons ici quelques points important permettant d'identifier quelques classes de langages.

1.2.1 Machine virtuelle

Parmi la multitude de langage interprétés, certains n'exécutent pas le code écrit dans le langage lui-même, mais un code intermédiaire, généré lors d'une phase de pré-compilation. Le moteur d'évaluation de ce code intermédiaire est appelé machine virtuelle.

Deux choix de compilation peuvent être mis en œuvre : beaucoup d'implémentations de *Scheme*, par exemple, génèrent du code intermédiaire lors du chargement du fichier, l'utilisation du code étant alors transparente pour le programmeur (*byte-compile*). D'autres au contraire, comme *Java*, chargent des fichiers pré-compilés, ce qui contraint le programmeur à exécuter une phase de compilation par lui-même.

L'utilisation de code intermédiaire a pour intérêt d'accroître la vitesse d'exécution et permet une optimisation du code par le compilateur. À titre d'exemple, l'interprète *Tcl* sans directives de pré-compilation (version antérieure à 8.0), était mille fois plus lent qu'un programme *C* équivalent. Avec la pré-compilation, les nouvelles versions de *Tcl* permettent de gagner un facteur deux lors de l'exécution du code généré. Notons tout de même dès à présent que les performances de *Tcl* sont largement insuffisantes. Un langage interprété comme *Self* hautement optimisé, n'est que deux à trois fois plus lent qu'un programme *C* compilé et optimisé.

Le passage par une représentation intermédiaire permet ainsi de concilier à la fois la vitesse d'exécution du code compilé et la portabilité du code source.

1.2.2 Typage

La vérification statique des types est sujette à discussion. Certains prétendent qu'elle est nécessaire à la qualité logicielle, puisque le compilateur est capable de détecter des erreurs avant l'exécution du programme. La phase de compilation permet, en outre, de détecter des erreurs qui n'apparaissent pas à l'exécution si le jeu de test utilisé ne couvre pas toutes les branches du programme. D'autres, au contraire, estiment que le typage est un obstacle à la facilité de développement, et que son absence fait partie des points forts d'un langage [**?**][1].

Notons que la mise en œuvre du typage nécessite l'utilisation d'un compilateur ou tout au moins d'une phase de compilation dans l'interprète, comme dans *ML*. Mais les impératifs de qualité étant pris de plus en plus en compte, les langages modernes se tournent vers un typage fort. C'est par exemple le cas de *Java* qui utilise un typage fort, à la manière de *C++* ou *Ada*. Si ce typage fort est souhaitable dans le cadre d'un développement bien ciblé, il nous semble moins primordial, voir même un "obstable", à une activité de recherche où les données et les structures peuvent être pronfondement modifiées tout au long du travail. En effet, le changement des types dans une structure de données d'un langage fortement typé peut entraîner des modifications conséquentes et avoir des répercussions sur une grande partie de l'implémentation.

1.2.3 Modèle objet

La présence ou l'absence d'un modèle objet dans le système, spécifie la possibilité de créer de nouvelles classes d'objets, que ce soit en utilisant un modèle classe-instance, prototype-instance ou autre. La simple inversion syntaxique de *Tcl*, permettant par exemple de déclencher une fonction pré-définie sur un objet, ne sera pas considérée comme une caractéristique orientée objet. Car même si cela peut être assimilé à un appel de méthode, l'absence de modèle objet de ce langage est tout de même bien réelle.

[1] OUSTERHOUT J.K., *Tcl and Tk Toolkit* (1994).

1.3 Conclusion

Après avoir identifié les différences entre langages système et langages de script, et après avoir mis en évidence les caractéristiques des langages de script, nous allons établir une liste de critères permettant une comparaison des différents langages de script existants. Cette évaluation nous permettra de choisir le langage le plus adapté à notre étude.

2 Critères de choix pour un langage de script

Tout au long du développement d'*OFR*, nous avons souhaité que le résultat final soit utilisable et permette l'intégration des divers programmes nécessaires à la réalisation de notre plate-forme d'évaluation. Cette possibilité de combinaison de composants de base, dédiés à la réalisation d'un tâche précise, réduit le temps de développement. Nous avons toujours essayé de garder présent à l'esprit ce souci de réutiliser des composants existants afin de ne pas investir du temps inutilement dans des problèmes déjà résolus.

Nous nous sommes donc orientés vers un développement par regroupement de composants ce qui implique l'utilisation d'un langage de script. Toutefois, en plus de cette intégration, le langage de script doit aussi nous permettre d'effectuer des traitements de haut niveau, voire d'implémenter une partie de l'algorithmique nécessaire. Dans cette optique, il est donc important que l'interprète ne nuise pas à la qualité et à l'efficacité globale du produit. Cependant, les performances pures ne sont pas un souci primordial, même si elles doivent rester correctes, permettant ainsi d'effectuer des tests sur des cas réels.

2.1 Efficacité et fiabilité

Le langage doit être efficace en termes de ressources, mais aussi de consommation mémoire et de vitesse d'exécution. Toutefois, les langages interprétés ne peuvent pas rivaliser en terme de vitesse d'exécution avec des langages compilés, comme *C* ou *C++*. Pour pallier un manque d'efficacité, à un endroit bien déterminé, le langage doit permettre d'optimiser la partie de code identifiée et de la remplacer par une ou plusieurs fonctions équivalentes écrites dans un langage compilé comme le langage *C*.

Plus spécifiquement encore, il est important que ce langage possède des fonctionnalités de haut niveau, en ce qui concerne la gestion de la mémoire, afin d'éviter au programmeur d'avoir à gérer la mémoire comme dans le cas des langages système. Il doit donc intégrer un ramasse-miettes (*garbage collector*).

2.2 Environnement de développement

L'intérêt majeur des systèmes de script réside dans leur aptitude à réduire les temps de développement. Un schéma de développement classique comprend un grand nombre de cycles

“Édition / Compilation / Exécution”, dans lesquels le programmeur fait une modification ou une correction et en observe l’effet.

La fiabilité et l’efficacité d’un programme ne relèvent pas uniquement du langage dans lequel il est écrit. Les conditions dans lesquelles il est développé et testé sont elles aussi cruciales. Que ce soit pour le programmeur principal ou pour les utilisateurs occasionnels, l’environnement de développement du langage choisi doit être complet. La présence d’un débogueur symbolique est indispensable. Les débogueurs symboliques permettent de suivre le déroulement d’un programme d’après ses sources, durant son exécution. Ils autorisent aussi la consultation des variables d’après leur nom.

Un autre point important est la présence d’une bonne bibliothèque de fonctions prédéfinies. La présence d’éditeurs ou d’un mode d’édition adapté au langage est souhaitable.

Une autre fonctionnalité, présentant un intérêt non négligeable, est la possibilité de faire un test de performance (*profiling*). Ceci permet de mettre en évidence le temps passé dans chacune des fonctions, et d’en déduire les zones de programme à optimiser, pour gagner de manière significative en temps d’exécution.

2.3 Extension possible

Pour être utilisé dans le but d’ajouter des fonctions ou fonctionnalités au langage existant, le langage interprété doit permettre un bon interfaçage avec un langage du type système comme le langage *C*. L’appel depuis les modules additifs des fonctions primitives écrites en *C* doit rester simple. Le passage d’éventuelles structures de données entre le langage interprété et le *C* doit s’effectuer le plus naturellement possible.

Deux méthodes différentes peuvent être utilisées afin d’ajouter des fonctionnalités à un langage de script :

– si l’on dispose du code source du langage, il est alors possible d’en compiler une nouvelle version en y ajoutant les services souhaités. Ceci produira un nouveau programme, enrichi de nouvelles fonctions ;
– si le langage choisi le permet, de charger dynamiquement une librairie contenant les extensions souhaitées.

La deuxième solution est préférable car, dans le cas où l’on souhaite intégrer une version plus récente de la librairie, le travail nécessaire est moindre. De plus, il n’est pas obligatoire de disposer des sources du langage pour effectuer cette tâche, même si l’on peut disposer facilement du code source des différents langages de script que nous avons étudiés.

2.4 Sauvegarde des données dans le langage

Parmi les avantages d’un langage de script figure celui de pouvoir effectuer la sauvegarde de données dans le langage lui-même, évitant par là-même l’implémentation d’un format de sauvegarde particulier et/ou d’un analyseur syntaxique pour la relecture des données. Cette

possibilité, même si elle paraît anecdotique, permet une économie importante en termes de développement, surtout dans le cas de la sauvegarde de structures de données complexes.

Cette possibilité s'est avérée très intéressante pour sauvegarder la structure des grammaires de graphes. La sérialisation *Java* présente l'intérêt de fournir un mécanisme de sauvegarde des données. Mais on soulignera toutefois que la sérialisation en *XML* n'est pas encore standard, et la sérialisation en binaire présente l'inconvénient de ne pas être compatible si l'on modifie les champs de la classe considérée. Cela rend donc cette solution fragile aux modifications et évolutions du programme.

2.5 Une interface graphique

Enfin, dans l'optique de mettre en place une plate-forme de tests pour les différents développements effectués, la réalisation d'une interface graphique est souhaitable. Le langage de script doit donc permettre l'accès à une boîte à outils graphiques (*graphical toolkit*).

Une des fonctionnalités que doit fournir cette boîte à outils graphiques, est la possibilité d'avoir accès à un objet permettant la visualisation et la manipulation de graphiques en deux dimensions (*canvas*).

La présence d'un tel composant nous a permis une visualisation directe des résultats de la reconnaissance des caractères (symboles reconnus et position), de la construction du graphe de données, et même la visualisation de la réécriture du graphe par applications successives des règles de la grammaire.

3 Une comparaison des langages de script

Après avoir réalisé cette liste de critères, nous avons cherché un langage s'y conformant le plus parfaitement possible. Le comparatif du tableau montre les différents langages interprétés auxquels nous nous sommes intéressés, et les critères que nous avons évalués. Seuls les plus proches de nos critères ont été retenus.

3.1 Langages de script testés

- **Langages fonctionnels** :
  - *STk* [**?**][1] est un système développé par Erick Gallesio. C'est un interpréteur *Scheme* conforme à la norme R4RS qui utilise la boîte à outils *Tk*. *STk* peut être vu comme un interprète *Tcl/Tk*, dans lequel on a remplacé *Tcl* par un interprète *Scheme* "compatible", c'est à dire qu'aucune modification n'intervient dans le noyau *Tk*, bien que celui-ci fasse appel directement à l'interprète *Tcl*. Toutefois, c'est son modèle objet qui donne à *STk* son originalité et son intérêt.

[1] GALLESIO E., Stklos : A scheme object-oriented system dealing with the tk toolkit (1994).

  - *Guile* [**?**][1] est un dialecte *Scheme* développé par la (*Free Software Foundation*). Le but de ses travaux est de fournir un interprète embarqué à la communauté UNIX. *Guile* est très peu utilisé par la communauté de développeurs. Même les projets *GNU*, comme *Gimp*, qui embarque un interprète *Scheme*, ne l'utilisent pas. Ce manque de participants à la vie du langage nous paraît un obstacle important à son utilisation. C'est pourquoi nous ne l'avons pas retenu pour nos développement.
  - *Klone* [**?**][2] est un dialecte *Lisp* très proche de *Common Lisp*. La documentation est accessible en ligne sous l'interprète ou sous forme hyper-texte. Les performances de l'interprète *Klone* sont très bonnes, comme le montre le tableau comparatif page 139. On peut aussi noter la présence d'un débogueur avec retour aux sources, bien plus élaboré que ceux d'autres langages de script. Toutes les fonctions système sont disponibles et les bibliothèques sont assez riches. Une boîte à outils graphiques, basée sur *Motif*, est disponible et à été ajoutée à *Klone* par Jean-Michel Léon [**?**][3] (voir paragraphe 4.2).
- **Langages procéduraux** :
  - *Tcl/Tk* [**?**][4] est le langage de script le plus populaire et certainement le plus utilisé, dans le monde Unix au moins, même si cette popularité tend à s'estomper. Il a été développé par John K. Ousterhout, de l'Université de Berkeley en Californie, vers la fin des années 1980. Il est basé sur le langage *Tcl* et la boîte à outils *Tk*. *Tcl* est un langage similaire aux shells Unix, dont l'unique type manipulé est la chaîne de caractères. Cela signifie que chaque valeur, chaque identificateur, chaque résultat d'une évaluation est une chaîne. Ceci induit deux conséquences. La première est qu'il est très simple de développer dans ce langage. Mais en contrepartie, les performances deviennent rapidement catastrophiques dès lors que l'on manipule des nombres, et les structures de données sont assez limitées. Même si, récemment, l'introduction d'une phase de pré-compilation transparente pour le programmeur a permis de pallier une partie de cette inefficacité du langage, il n'en reste pas moins plus lent que la plupart de ses concurrents. On peut toutefois noter que *Tcl/Tk* est disponible sur Windows, Macintosh et Unix.
- **Langages orientés objets** :
  - *Python* [**?**][5] est un langage de script orienté objet. Il a été développé par Guido van Rossum du CWI à Amsterdam, mais est maintenant géré par un Consortium. Une très large communauté de programmeurs a adopté ce langage. Il a été conçu pour l'interface avec *C* ou *C++*. Seule sa syntaxe avec une indentation significative

[1] GALASSI M., *Guile User Manual* (1996).

[2] NAHABOO C., *Klone Reference Manual* (1995).

[3] LÉON J.M., *Contribution à la Simplification des Boîtes à Outils Graphiques : Application aux Langages de Scripts*, PhD Thesis (1995).

[4] OUSTERHOUT J.K., *Tcl and Tk Toolkit* (1994).

[5] *Python Reference Manual* (1999).

qui remplace, pour l'écriture de structures de contrôle, l'utilisation de délimiteurs de blocs, est un peu déroutante. Désormais, une très large communauté a contribué à l'écriture de modules pour *Python*, ce qui en fait maintenant un langage avec une bibliothèque très importante et bien structurée. Il est à souligner que ce n'était pas le cas lors du choix du langage pour l'implémentation d'*OFR* ; le langage n'en était qu'à ses balbutiements. Aujourd'hui, *Python* nous semble un très bon candidat. Plusieurs boîtes à outils graphiques sont disponibles (*Tk*, *wxWindows* par exemple). *Python* est aussi disponible sous Windows.

– *Perl* [**?**][1] est aussi un langage de script orienté objet. Là aussi, malgré une bonne efficacité et un interfaçage avec *C* assez facile, la syntaxe reste le point faible. Les programmes *Perl* sont peu lisibles, ce qui est très gênant pour assurer une bonne évolutivité du code. La communauté de développeurs *Perl* est très grande, mais l'utilisation est souvent limitée à de petits scripts, et son utilisation pour des projets plus gros est plus contestable.

Nous mentionnerons aussi d'autres langages de script que nous n'avons pas retenu pour notre comparatif, en expliquant, pour chacun, le point qui nous semblait un obstacle majeur à son utilisation.

– *Visual Basic* : il est uniquement disponible sous *Windows* et c'est un produit commercial. De plus, sur un système comme *Windows*, assez peu de composants gratuits sont disponibles. Tous les systèmes de reconnaissance de caractères sont payants, et il est impossible d'obtenir une documentation sur l'implémentation ou bien encore le code source d'une application, afin d'y rajouter les éventuelles fonctionnalités manquantes. Tous ces problèmes font de *Windows* un environnement trop fermé pour une activité de recherche.
– *Java Script* : assez peu d'implémentations sont disponibles, et le manque de bibliothèques standards proposées avec ce langage nous semble un frein important à son utilisation.

3.2 Conclusion

Les critères portés dans ce tableau sont un récapitulatif des différentes fonctionnalités que nous venons de décrire et qui nous paraîssent importantes pour un langage de script.

Efficacité	L'exécution est-elle rapide pour un langage de script ?
GC	Dispose-t-il d'un ramasse-miettes ?
Bibli.	Les bibliothèques fournies avec le langage sont-elles riches ?
Ext.	Est-il possible de charger dynamiquement une librairie ?
GUI	La boîte à outils supportée : Tk, Motif, wxWindows, ...

[1] SCHWARTZ L. & SCHWARTZ R., *Programming Perl* (19950-93).

Plus le nombre d'étoiles est important, plus le critère est considéré comme de bonne qualité. L'absence d'étoile signifie que la caractéristique n'est pas présente.

Nom	Efficacité	GC	Bibli.	Ext.	GUI
STk	**	*	**	*	*Tk*
Guile	*	*	**	*	*Tk*
Klone	***	*	***	*	*Motif*
Tcl	*		***	*	*Tk*
Python	**	*	***	*	*Tk, wxWin*
Perl	**		***	*	*Tk*

D'autres critères plus subjectifs ont aussi été pris en compte, comme la facilité de debogage du langage avec la présence ou non d'un débogueur symbolique, la qualité de la documentation, la communauté des utilisateurs, etc.

4 Choix d'un langage

L'interprète *Lisp Klone* nous a paru être le choix le plus adapté à nos besoins. Nous disposons, de plus, d'un excellent support technique car *Klone* est activement développé au sein de l'équipe *KOALA* sur le site même de l'*INRIA Sophia-Antipolis*. Ce dernier critère a aussi beaucoup compté dans notre choix. Les conclusions sur l'emploi d'un langage de script, et plus particulièrement sur l'utilisation de *Klone*, sont données dans le paragraphe suivant.

4.1 Klone

Lisp présente plusieurs avantages, en particulier sa syntaxe très homogène. Certains ont pourtant regretté sa lisibilité médiocre. Le nombre élevé de parenthèses et une indentation mal standardisée sont les points qui rebutent le plus les novices. Heureusement, le *Lisp* est largement connu au sein de la communauté scientifique, peut-être du fait de son ancienneté (le *Lisp*, apparu en 1964, est le résultat des travaux de *Mc Carthy*).

Le langage *Klone* [**?**][1] est le descendant de *WOOL*, un dialecte de *Lisp* utilisé dans *GWM* (*Generic Window Manager*), un gestionnaire de fenêtres programmable et personnalisable. Développé depuis 1988 *WOOL* a été très utilisé et débogué par la communauté des utilisateurs de *GWM*. *Klone* a aussi servit de base à de nombreux développements, dont certains à caractère industriel, par l'entreprise *BULL*, ce qui démontre sa fiabilité et son efficacité. À titre de comparaison pour l'efficacité, le parseur *XML* de *Klone* est plus rapide que celui de *Java*. Cela peut-être nuancé par le fait que l'implémentation *Klone* est basée sur l'utilisation

[1] NAHABOO C., *Klone Reference Manual* (1995).

d'expressions régulières qui sont optimisées, ce qui n'est pas le cas de *Java* qui ne les intègre pas en standard.

En choisissant *Klone*, nous disposons tout d'abord d'un langage de script performant, qui permet une intégration facile des différents composants nécessaires à notre réalisation. Mais il s'est aussi avéré que l'interprète *Klone* est assez performant pour gérer l'implémentation d'*OFR*. Dans un souci d'homogénéité, nous avons donc décidé d'employer *Klone* non seulement pour servir de lien entre les différents composants, mais aussi comme le langage de développement de l'algorithmique liée à la reconnaissance structurelle des formules mathématiques. Il est incontestable que ce choix nous a permis de faire des économies en termes de temps de développement, ainsi que dans la quantité de code à écrire ; la qualité et l'abondance des bibliothèques fournies ont aussi été très appréciées.

L'environnement de développement est de bonne qualité, même si le débogueur n'est pas encore aussi confortable que celui disponible pour le langage *C*. Nous avons tout particulièrement apprécié la documentation en ligne, disponible depuis l'interpréteur. Le mécanisme d'autodoc permet aussi d'intégrer ses propres fonctions au système de documentation, ce qui est un plus incontestable.

Il semble donc que l'emploi d'un langage de script et plus particulièrement de *Klone* se soit révélé un bon choix pour nos travaux.

4.2 Klm : une extension graphique de Klone

L'extension graphique de *Klone*, développée par Jean-Michel Léon [**?**][1] et baptisée *Klm*, nous permet d'avoir accès à une boîte à outils graphiques. Outre les composants standards de *Motif* (boutons, menus, etc.), l'auteur a ajouté un nouveau composant graphique ou **widget** (*widget : WInDow GadgET*) permettant l'intégration des graphiques à deux dimensions. Ce nouveau service nous aura été très utile pour la mise au point des algorithmes graphiques concernant la détection des relations entre les symboles composant une expression.

Afin de convaincre le lecteur de l'économie de programmation réalisée en choisissant un langage de script pour implémenter également la partie graphique, voici un exemple ; il concerne l'affichage d'une valeur entière dans un label *Motif*. Le code *C* permettant cette action est le suivant :

```
void SetIntegerValue(Widget label, int n)
{
    Arg arg;
```

[1] LÉON J.M., *Contribution à la Simplification des Boîtes à Outils Graphiques : Application aux Langages de Scripts*, PhD Thesis (1995).

```
    XmString xmvalue;
    char svalue[16];
    sprintf(svalue, "%d", n);
    xmvalue = XmStringCreateSimple(svalue);
    XtSetArg(arg, XmNlabelString, xmvalue);
    XtSetValue(label, &arg, 1);
    XmStringFree(xmvalue);
}
```

Alors qu'en *Klm*, le code correspondant s'écrit tout simplement :

```
(setq n 1234)
(label :labelString n)
```

La quantité de code pour effectuer une même action est moindre en *Klm*, minimisant par là-même les sources d'erreurs. On peut noter que ce type d'économie réalisée n'est pas spécifique à *Klm* mais est commune à tous les langages de script que nous avons décrits dans ce chapitre.

4.3 Communication avec d'autres systèmes : OpenMath

Le but de nos travaux étant la réalisation d'un prototype de reconnaissance structurelle d'expressions mathématiques, il nous a été nécessaire de mettre en place une structure de données de type arborescente, pour stocker les résultats obtenus. Une classification des objets mathématiques a été développée dans [**?**][1]. Même si ce type de représentation est intéressant, il ne résout pas le problème de communication des objets mathématiques entre différentes applications. Des travaux récents sur le sujet ont conduit à la création d'une librairie standard pour résoudre ce type de problèmes.

Le projet européen Esprit 24.969 intitulé *OpenMath* (*OpenMath : Accessing and Using Mathematical Information Electronically*) a pour but la définition d'un standard de communication d'objets mathématiques et la réalisation d'un certain nombre d'applications l'utilisant. Parmi ces applications, sont prévus des outils de consultation de documentations techniques et de bases de données d'articles mathématiques , des versions de systèmes de calcul formel *Axiom* et *GAP* utilisant *OpenMath*, et des logiciels d'enseignement à distance à travers le Web. Différents partenaires contribuent à ce projet dont l'*INRIA*, *NAG* (fournisseur de logiciels de calcul scientifique), *Springer-Verlag* (éditeur scientifique). Le lecteur pourra consulter le site officiel sur *OpenMath* à l'adresse suivante :

http ://www.nag.co.uk/projects/OpenMath.html.

[1] TOUMIT J., GARCIA-SALICETTI S. & EMPTOZ H., A hierarchical and recursive model of mathematical expressions for automatic reading of mathematical documents (1999).

L'*INRIA*, par l'intermédiaire des projets *SAFIR* et *CAFE*, est intervenu principalement dans la définition, la conception et l'écriture des bibliothèques (en *C* et *Java*) d'*OpenMath*.

L'architecture d'*OpenMath* est décrite à l'aide de la figure 2 et résume bien les interactions entre les différents composants *OpenMath*. Comme le montre la figure 2 ci-dessous, il y a trois niveaux de représentation pour un objet mathématique.

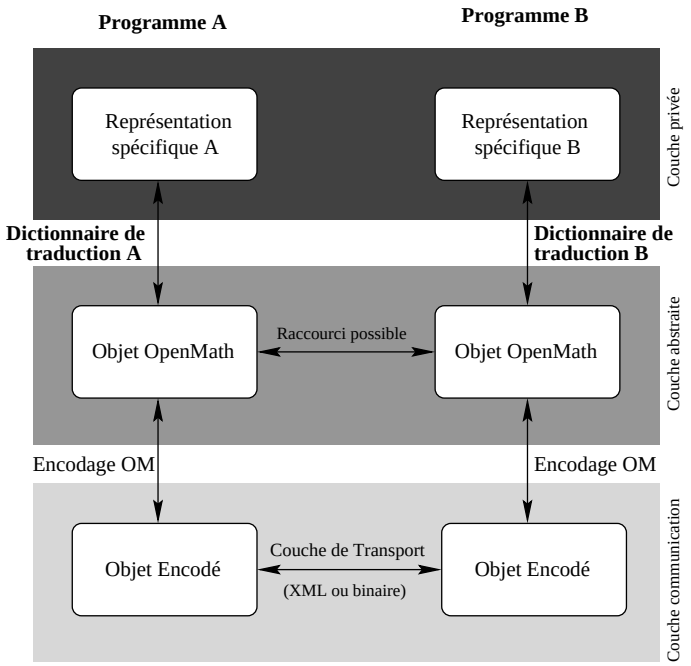


FIG. 2 – Architecture d'*OpenMath*

La couche privée est la représentation interne utilisée par l'application. La couche abstraite est la représentation de l'objet mathématique en tant qu'objet *OpenMath*. La troisième est la couche de communication, qui permet de transformer un objet *OpenMath* en un flux d'octets. Typiquement, une application manipule en interne sa propre représentation des objets mathématiques, elle peut les convertir en objets *OpenMath* et les transmettre à d'autres applications, en les convertissant en flux de données.

Les objets *OpenMath* sont la représentation d'entités mathématiques qui peuvent être transmises tout en conservant leur sémantique. Des dictionnaires (*CD : Content Dictionaries*) sont utilisés pour assigner la sémantique formelle où informelle à chacun des symboles constituant l'expression mathématique que l'on souhaite transmettre.

La possibilité d'ajouter de nouvelles fonctions implémentées en *C* à l'interprète *Klone* [**?**][1] nous a permis de bénéficier des fonctionnalités de la librairie *OpenMath*. La communication vers d'autres systèmes était, en effet, fondamentale dans l'optique de se servir d'*OFR*

[1] NAHABOO C., *Klone C Interface Manual* (1994).

comme d'une interface pour un système de calcul formel, ou tout simplement pour communiquer les résultats obtenus. L'ajout de la librairie *OpenMath* s'effectue par chargement dynamique dans l'interprète *Klone*.

5 Implémentations

L'implémentation d'*Irma*, qui renferme le composant *OFR*, représente environ 15.000 lignes de *Klone*. Elle peut se subdiviser en plusieur catégories :

- l'analyse des données fournies par l'OCR manuscrit ou typographié (1500 lignes),
- l'analyse géométrique (4500 lignes),
- l'analyse syntaxique à l'aide de la grammaire de graphes (6500 lignes),
- la communication openmath (1000 lignes),
- l'interface graphique (1000 lignes).

Le composant *OFR* est constitué de l'analyse géométrique et de l'analyse syntaxique à l'aide de la grammaire de graphe.

La possibilité de sauvegarder un certain nombre de données dans le langage de script lui-même, a permis d'économiser la spécification d'un format pour les données, ainsi que l'écriture de morceaux de programmes pour la génération et la relecture des données. C'est par exemple le cas de la grammaire de graphes.

6 Conclusion

Dans le cadre d'un travail de recherche, il n'est pas toujous possible d'investir assez de temps pour assurer la portabilité dans tous les environnements de développement. Un autre avantage d'avoir utilisé un langage interprété est que, si l'interprète est disponible sur une autre plate-forme, il n'est pas nécessaire de consacrer du temps à la portabilité du code. C'est le cas de *Klone*, donc *OFR* est aussi disponible sous plusieurs environnements (comme *Unix* et *Windows*). Toutefois, on notera que l'extension graphique *Klm* ne marche que sous *Unix* ; on perdra donc l'interface de contrôle sous l'environnement *Windows*.

Comme nous l'avons présenté dans ce chapitre, nous avons tiré plusieurs avantages à utiliser un langage de script comme *Klone*. Au moment du choix d'un langage pour l'implémentation, l'utilisation d'un langage de script, et de *Klone* en particulier, est apparue comme intéressante par rapport aux autres langages cités dans ce chapitre. Nous avons pu en tirer plusieurs avantages, comme la possibilité de sauvegarder des structures de données de haut niveau très facilement. La possibilité de manipuler et de programmer certains algorithme dans un langage compilé et efficace comme *C*, même si nous ne l'avons pas exploitée, n'en ai pas moins négligeable. Toutefois, au jour de la rédaction, la situation a beaucoup changée.

L'engouement pour *Java* ainsi que ses performances tout à fait intéressantes, la portabilité sur la plupart des systèmes et les nombreuses biliothèques à disposition, nous conduisent à penser que *Java* est actuellement un choix incontournable. Nous recommenderions toutefois de ne pas négliger l'aspect langage de script qui présente beaucoup d'avantages. Plusieurs solutions sont alors possibles : DynamicJava, Rhino, Iava, JPython, etc.

L'utilisation d'un langage de script nous a permis un gain de temps considérable en termes de développement. Le principal gain est surtout dû à la possibilité d'intégrer des programmes extérieurs à notre application, et en particulier un logiciel de reconnaissance des caractères typographiés ou manuscrits. Nous n'aurions pas pu tester ces différents cas sans cette possibilité.

Conclusion

Nous avons présenté dans ce mémoire : la conception, la réalisation et l'utilisation d'*OFR*, un composant de reconnaissance structurelle de formules mathématiques. Intégrable dans une application, *OFR* est au cœur de *Irma*, un prototype de reconnaissance d'expressions mathématiques typographiées et manuscrites. Aujourd'hui, la saisie des notations mathématiques pour une manipulation informatique est encore plus importante que par le passé. L'un des principaux facteurs de cette expansion est, entre autres, l'utilisation croissante des systèmes de calculs formels avec des champs d'application de plus en plus larges. Les utilisateurs de ces systèmes, ou bien encore d'éditeurs de documents scientifiques, sont de plus en plus nombreux et appartiennent à des milieux très divers. Paradoxalement, même si des systèmes comme *Maple* ou *Mathematica* sont maintenant utilisés par le plus grand nombre, très peu d'efforts ont été fournis sur la qualité des interfaces, en particulier sur les modes de saisie des expressions mathématiques. Ce constat est identique dans le domaine de l'édition de documents mathématiques, où des outils comme LaTeX sont réservés à des utilisateurs avertis. Un tel déficit de l'interface homme-machine en terme d'ergonomie peut considérablement ralentir l'utilisateur expérimenté, et bien souvent rebuter le néophyte. Ce constat, qui a motivé nos travaux, est encore vrai à l'heure de leur conclusion. Le composant *OFR* et l'interface expérimentale qui a servi à sa validation peuvent donner un avant-goût des possibilités de récupération ou d'édition des expressions mathématiques.

Le développement d'une plate-forme d'expérimentation pour évaluer nos idées en matière de reconnaissance structurelle des expressions mathématiques, nous a conduit à étudier les systèmes d'identification des symboles typographiés et manuscrits usités dans les notations mathématiques. Nous avons souhaité, dans la mesure du possible, réutiliser des

composants logiciels existants effectuant cette tâche, nécessaire à notre étude.

L'absence d'un tel composant, dans le cas typographié s'est rapidement imposée. Nous avons donc été contraints à la réalisation d'un système de reconnaissance des symboles utilisés dans les notations typographiées. Même si celui-ci n'implémente pas tous les traitements d'images souhaitables, comme le nettoyage ou la réorientation, l'ensemble des fonctionnalités indispensables est présent. De nouvelles expérimentations devraient avoir lieu, notamment dans le domaine de la correction d'erreurs ou bien encore de la recherche d'informations sémantiques. Pour faciliter de telles expériences, il nous a semblé judicieux de fournir aux futurs intervenants la base logicielle qui nous a fait défaut. Notre système de reconnaissance des symboles typographiés a été conçu en gardant cet objectif présent à l'esprit. Certaines de ses caractéristiques en sont le reflet, comme la possibilité d'ajouter de nouveaux symboles à l'ensemble des symboles déjà connus par le système.

Nos recherches ont été plus fructueuses dans la cas manuscrit, où nous avons pu bénéficier du travail de l'équipe allemande de l'Université de Duisburg, dirigée par le professeur Gerhard Rigoll. Leur système, même s'il reste quelque peu incomplet du point de vue de notre architecture, effectue la segmentation et la reconnaissance de symboles employés dans les notations mathématiques. Cette collaboration a permis une expérimentation de notre méthode de reconnaissance structurelle des expressions mathématiques, dans le cadre de notations manuscrites.

OFR est le composant logiciel que nous avons développé afin d'effectuer une reconnaissance structurelle des notations mathématiques. La méthode que nous avons étudiée et enrichie est basée sur l'utilisation de grammaires de graphes. Plusieurs expérimentations doivent être effectuées, et ce composant doit donc être capable de s'adapter au plus grand nombre, tant au niveau des outils utilisés pour effectuer la reconnaissance des caractères, que du type de notations identifiables par le système. Ces caractéristiques ont bien entendu influencé le développement sur plusieurs points.

L'un des choix architecturaux que nous avons été amenés à faire est celui d'utiliser un langage nous permettant à la fois une intégration des différents composants logiciels, et un développement rapide et évolutif de la méthode retenue, à base de grammaires de graphes. L'utilisation d'un langage de script s'est imposée en raison de ses nombreux avantages, comme les structures de données de haut niveau, l'intégration d'autres composants, la sauvegarde de données dans le langage lui-même, etc.

Un des points forts d'*OFR* est sa capacité d'adaptation, non seulement à différents types de logiciels qui effectuent la reconnaissance des symboles (typographiés ou manuscrits), mais aussi et surtout au type de notations reconnues par le système. C'est, en effet, un des points les plus novateurs dans le domaine, la plupart des systèmes existants étant fermés et ne permettant pas d'ajouter une nouvelle notation à identifier. Ils sont limités à des notations de type linéaire et leur extension à l'identification de notations réellement bidimensionnelles comme les matrices n'est pas possible. Notre approche, basée sur une analyse bidimension-

nelle des expressions mathématiques, est subdivisée en deux étapes : une analyse lexicale et une analyse syntaxique. Chacunes de ces deux analyses peut être paramétrée, ce qui permet l'adaptation du système aux besoins particuliers. La reconnaissance d'une nouvelle notation mathématique s'effectue en ajoutant une règle à la grammaire de graphes. Cette souplesse permet d'adapter assez rapidement le système à ses propres besoins.

Enfin, il nous reste à évoquer le mécanisme de transmission des résultats, avec l'utilisation d'*OpenMath*. Toujours dans un souci d'intégration de composants dédiés, nous avons inclus ce protocole de communication des objets mathématiques à *Klone*. Cette librairie, développée dans le cadre d'un projet européen, permet la communication des objets mathématiques, tout en conservant la sémantique des éléments constituant la formule. Plusieurs systèmes de calculs formels intègrent déjà cette librairie, comme *Mathematica* par exemple, ce qui permet donc d'envisager une diffusion des résultats issus de la reconnaissance à un grand nombre de systèmes. Entre autres applications, nous avons envisagé l'utilisation d'*OFR* en tant qu'interface à la saisie d'expressions mathématiques.

Plusieurs problèmes subsistent ou n'ont pas pu être abordés lors de ces recherches. On pourra citer l'intégration à la grammaire de graphes d'une possibilité de corriger les erreurs dues au logiciel de reconnaissance des caractères, et donc d'avoir un rétro-contrôle sur l'identification des symboles dans la méthode d'analyse de la syntaxe. Toutefois, ce type de corrections ne pourra pas être aussi efficace que dans le cas de la reconnaissance de textes, le redondance des informations n'étant a priori pas suffisante pour une correction automatique de tous les types d'erreurs.

La césure est utilisée dans le cas où l'expression est plus large que la page sur laquelle elle est imprimée ou bien, dans le cas d'une saisie manuscrite, si la surface de saisie n'est pas suffisante. Ces différents cas de figures, qui ne sont pas pris en compte par notre méthode, n'ont pas été analysés dans les différents travaux que nous avons référencés. Il est pourtant important d'intégrer cette possibilité à une application concrète. On soulignera donc la nécessité d'étudier l'analyse correcte des expressions scindées sur plusieurs lignes.

Si les actions de lecture, de reconnaissance et d'interprétation des notations mathématiques peuvent sembler simples et naturelles pour un lecteur humain, il s'avère que ces activités sont difficiles à décrire et à formaliser au sein d'une application informatique. Nous avons présenté dans ce mémoire la méthode basée sur l'utilisation d'une grammaire de graphes et sur l'architecture logicielle mise en place pour effectuer des tests dans les cas d'expressions manuscrites et typographiées. Cette approche globale du problème de reconnaissance des expressions mathématiques permet la reconnaissance non seulement des notations du type équations, mais aussi des formes bidimensionnelles comme les notations matricielles. L'introduction de l'interprétation sémantique et la communication des résultats permettent également de faire de l'outil développé une réelle plate-forme d'expérimentation.

Annexe A

Exemple de reconnaissance d'une formule avec Irma

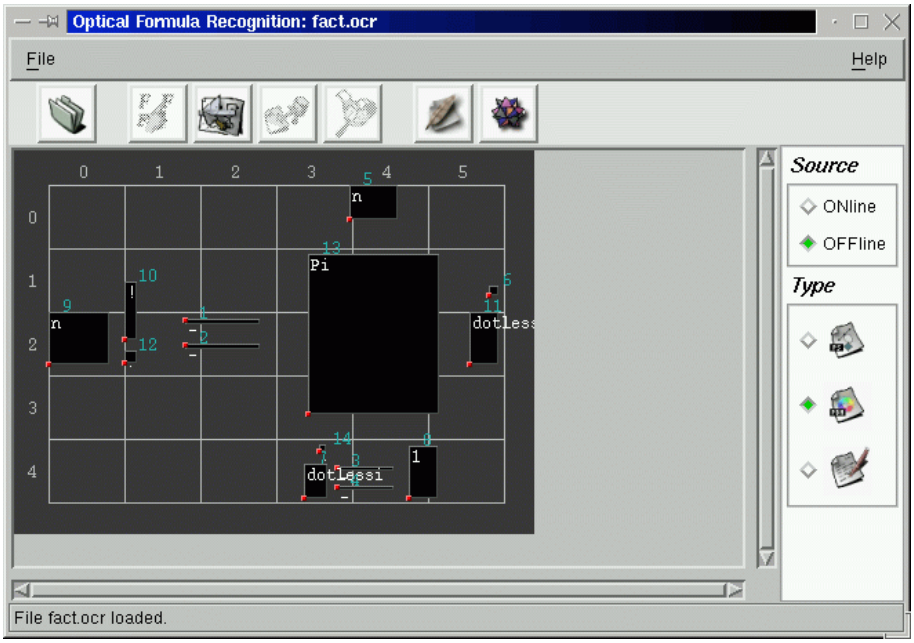
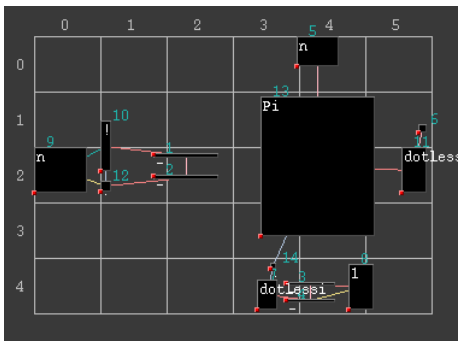
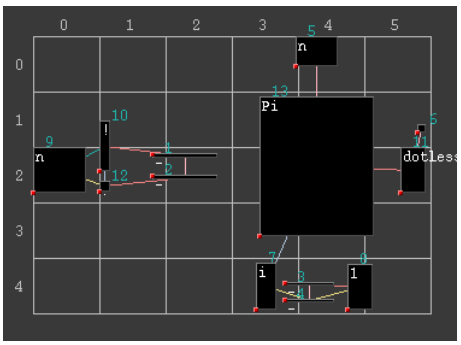


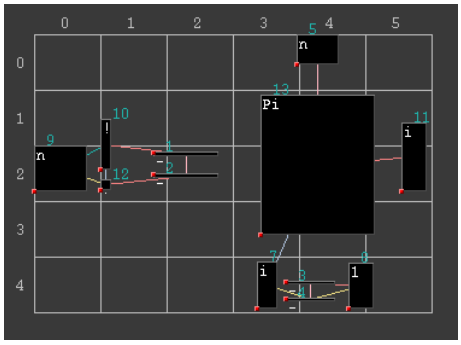
FIG. 1 – Irma : Symboles reconnus par l'OCR



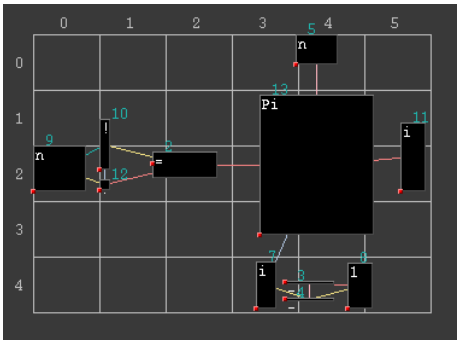
(a) Construction du graphe initial



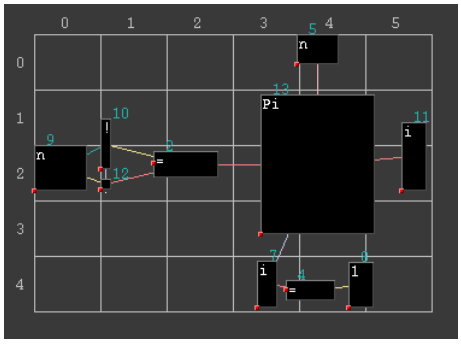
(b) Regroupement du symbole i ($\imath$ et $\cdot$)



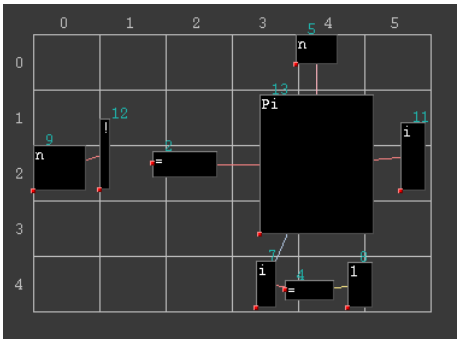
(c) Regroupement du symbole i ($\imath$ et $\cdot$)



(d) Regroupement du symbole d'égalité

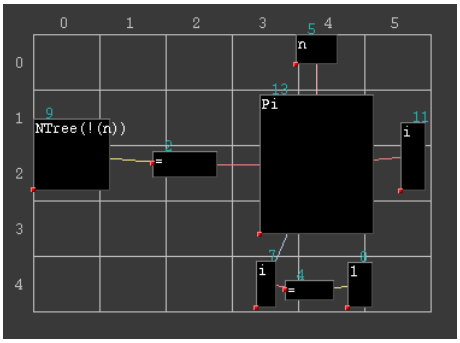


(e) Regroupement du symbole d'égalité

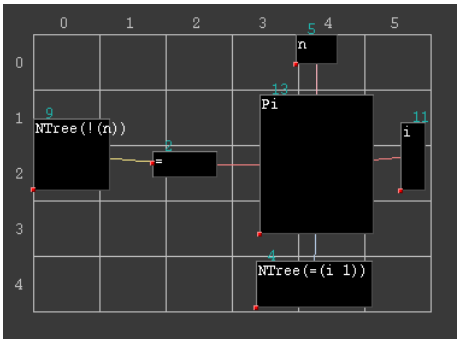


(f) Regroupement du symbole factoriel (| et $\cdot$)

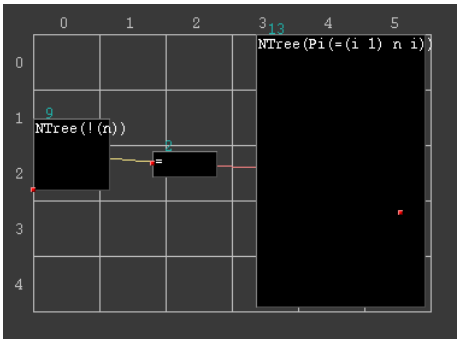
FIG. 2 – Reconnaissance de la notation $n! = \prod_{i=1}^{n} i$



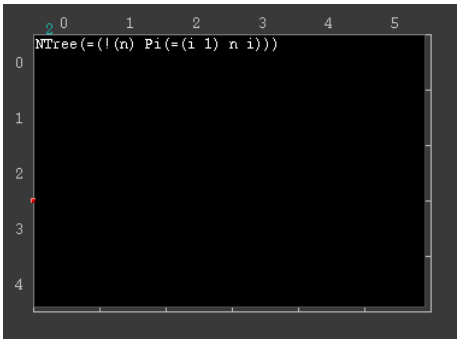
(a) Règle factorielle



(b) Règle égalité



(c) Règle produit



(d) Règle égalité

FIG. 3 – Reconnaissance de la notation $n! = \prod_{i=1}^{n} i$ (suite)

Annexe B

Variété des symboles

Cette annexe présente l'ensemble des symboles (principalement de LaTeX) qui sont utilisés dans les textes mathématiques. Pour une plus grande clarté, ils ont été classés par catégories. Les caractères les plus couramment utilisés et reconnus dans le cadre de l'analyse de texte (une soixantaine), sont regroupés afin de bien souligner la faible proportion qu'ils représentent par rapport à l'ensemble des symboles potentiellement utilisés dans les notations mathématiques (200 symboles supplémentaires, soit un total de plus de 250 symboles).

1 Exemples de polices et de variations

Les symboles peuvent être typographiés dans différentes fontes, chacun pouvant avoir subit une variation (italique, gras, etc). Voici quelques exemples de fontes et de variations.

Texte standard : abcdef ABCDEF 01234 ,. ! $\alpha\beta\gamma\delta\epsilon$ $\Gamma\Delta\Sigma\Pi$
Texte italique : abcdef ABCDEF 01234 ,. ! $\alpha\beta\gamma\delta\epsilon$ $\mathit{\Gamma\Delta\Sigma\Pi}$
Texte gras : abcdef ABCDEF 01234 ,. ! $\alpha\beta\gamma\delta\epsilon$ $\mathbf{\Gamma\Delta\Sigma\Pi}$
Texte sans-serif : abcdef ABCDEF 01234 ,. ! $\alpha\beta\gamma\delta\epsilon$ $\Gamma\Delta\Sigma\Pi$
`Texte Courier : abcdef ABCDEF 01234 ,. !` $\alpha\beta\gamma\delta\epsilon$ $\Gamma\Delta\Sigma\Pi$
Texte Roman : abcdef ABCDEF 01234 ,. ! $\alpha\beta\gamma\delta\epsilon$ $\Gamma\Delta\Sigma\Pi$
TEXTE PETITES MAJUSCULES : ABCDEFGHIJKLMNOP ABCDEF

Texte italique droit : abcdef ABCDEF 01234 ,. !

Texte Caligraphique : $\mathcal{ABCDEFGHIJKLMNOP}$

On peut rencontrer plusieurs variations au sein d'une même formule. Les fonctions du type sinus, cosinus, logarihtme, etc, sont très souvent en fonte normale alors que les variables sont en italique.

$$\sin^2\theta + \cos^2\theta = 1$$

2 Symboles textuels les plus courants

2.1 Alphabet latin

a	h	o	u
b	i	p	v
c	j	q	w
d	k	r	x
e	l	s	y
f	m	t	z
g	n		

TAB. 1 – Lettres latines minuscules

A	H	O	U
B	I	P	V
C	J	Q	W
D	K	R	X
E	L	S	Y
F	M	T	Z
G	N		

TAB. 2 – Lettres latines majuscules

2.2 Accentuations

´ `acute` ` `grave` ˆ `hat` ¨ `ddot`

TAB. 3 – Accentuations

2.3 Chiffres

0 1 2 3 4 5 6 7 8 9

TAB. 4 – Chiffres

2.4 Ponctuations

. , ; : ! ? ( )

TAB. 5 – Ponctuations

3 Symboles mathématiques les plus courants

3.1 Alphabet grec

α	`alpha`	θ	`theta`		`o`	τ	`tau`
β	`beta`	ϑ	`vartheta`	π	`pi`	υ	`upsilon`
γ	`gamma`	ι	`iota`	ϖ	`varpi`	ϕ	`phi`
δ	`delta`	κ	`kappa`	ρ	`rho`	φ	`varphi`
ϵ	`epsilon`	λ	`lambda`	ϱ	`varrho`	χ	`chi`
ε	`varepsilon`	μ	`mu`	σ	`sigma`	ψ	`psi`
ζ	`zeta`	ν	`nu`	ς	`varsigma`	ω	`omega`
η	`eta`	ξ	`xi`				

TAB. 6 – Lettres grecques minuscules

Γ	`Gamma`	Λ	`Lambda`	Σ	`Sigma`	Ψ	`Psi`
Δ	`Delta`	Ξ	`Xi`	Υ	`Upsilon`	Ω	`Omega`
Θ	`Theta`	Π	`Pi`	Φ	`Phi`		

TAB. 7 – Lettres grecques majuscules

3.2 Opérateurs mathématiques

$\pm$	`pm`	$\cap$	`cap`	$\setminus$	`setminus`	$\oplus$	`oplus`
$\mp$	`mp`	$\cup$	`cup`	$\diamond$	`diamond`	$\ominus$	`ominus`
$\times$	`times`	$\uplus$	`uplus`	$\bigtriangleup$	`bigtriangleup`	$\otimes$	`otimes`
$\div$	`div`	$\sqcap$	`sqcap`	$\bigtriangledown$	`bigtriangledown`	$\oslash$	`oslash`
$/$	`/`	$\sqcup$	`sqcup`	$\lhd$	`lhd`	$\odot$	`odot`
$*$	`ast`	$\vee$	`vee`	$\rhd$	`rhd`	$\amalg$	`amalg`
$\cdot$	`cdot`	$\wedge$	`wedge`	$\unlhd$	`unlhd`		

TAB. 8 – Opérateurs binaires

$\leq$	`leq`	$\geq$	`geq`	$\equiv$	`equiv`	$\models$	`models`
$\prec$	`prec`	$\succ$	`succ`	$\sim$	`sim`	$\perp$	`perp`
$\preceq$	`preceq`	$\succeq$	`succeq`	$\simeq$	`simeq`	$\mid$	`mid`
$\ll$	`ll`	$\gg$	`gg`	$\asymp$	`asymp`	$\parallel$	`parallel`
$\subset$	`subset`	$\supset$	`supset`	$\approx$	`approx`	$\bowtie$	`bowtie`
$\subseteq$	`subseteq`	$\supseteq$	`supseteq`	$\cong$	`cong`	$\Join$	`Join`
$\sqsubset$	`sqsubset`	$\sqsupset$	`sqsupset`	$\neq$	`neq`	$\smile$	`smile`
$\sqsubseteq$	`sqsubseteq`	$\sqsupseteq$	`sqsupseteq`	$\doteq$	`doteq`	$\frown$	`frown`
$\in$	`in`	$\ni$	`ni`	$\propto$	`propto`		
$\vdash$	`vdash`	$\dashv$	`dashv`				

TAB. 9 – Opérateurs relationnels

$\leftarrow$	leftarrow	$\longleftarrow$	longleftarrow	$\uparrow$	uparrow
$\Leftarrow$	Leftarrow	$\Longleftarrow$	Longleftarrow	$\Uparrow$	Uparrow
$\rightarrow$	rightarrow	$\longrightarrow$	longrightarrow	$\downarrow$	downarrow
$\Rightarrow$	Rightarrow	$\Longrightarrow$	Longrightarrow	$\Downarrow$	Downarrow
$\leftrightarrow$	leftrightarrow	$\longleftrightarrow$	longleftrightarrow	$\updownarrow$	updownarrow
$\Leftrightarrow$	Leftrightarrow	$\Longleftrightarrow$	Longleftrightarrow	$\Updownarrow$	Updownarrow
$\mapsto$	mapsto	$\longmapsto$	longmapsto	$\nearrow$	nearrow
$\hookleftarrow$	hookleftarrow	$\hookrightarrow$	hookrightarrow	$\searrow$	searrow
$\leftharpoonup$	leftharpoonup	$\rightharpoonup$	rightharpoonup	$\swarrow$	swarrow
$\leftharpoondown$	leftharpoondown	$\rightharpoondown$	rightharpoondown	$\nwarrow$	nwarrow
$\rightleftharpoons$	rightleftharpoons	$\leadsto$	leadsto		

TAB. 10 – Flèches

$\aleph$	aleph	$\Im$	Im	$\bot$	bot	$\natural$	natural
$\hbar$	hbar	$\mho$	mho	$\|$	|	$\sharp$	sharp
$\imath$	imath	$\prime$	prime	$\angle$	angle	$\backslash$	backslash
$\jmath$	jmath	$\emptyset$	emptyset	$\forall$	forall	∂	partial
ℓ	ell	∇	nabla	$\exists$	exists	∞	infty
$\wp$	wp	$\surd$	surd	$\neg$	neg		
$\Re$	Re	$\top$	top	$\flat$	flat		

TAB. 11 – Symboles divers

$\sum$	sum	$\bigcap$	bigcap	$\bigodot$	bigodot
$\prod$	prod	$\bigcup$	bigcup	$\bigotimes$	bigotimes
$\coprod$	coprod	$\bigsqcup$	bigsqcup	$\bigoplus$	bigoplus
$\int$	int	$\bigvee$	bigvee	$\biguplus$	biguplus
$\oint$	oint	$\bigwedge$	bigwedge	$\sqrt{}$	sqrt

TAB. 12 – Symboles de taille variable

(	left(	)	right)
[	left[	]	right]
{	left{	}	right{
⌊	lfloor	⌋	rfloor
⌈	lceil	⌉	rceil
⟨	langle	⟩	rangle
|	vbar	‖	dbvbar

TAB. 13 – Délimiteurs

ˆ	hat	`	grave	˙	dot
ˇ	check	˜	tilde	¨	ddot
˘	breve	¯	bar	-	overline
´	acute	→	vec	_	underline

overbrace

underbrace

TAB. 14 – Accentuations

Annexe C

Un document de test

Cette annexe fournit un exemple de document utilisé lors du comparatif des performances sur les résultats des logiciels pour la reconnaissance des caractères. Cet exemple est issu du livre [**?**][1], chapitre V, page 420.

Ce document est intéressant pour nos tests de comparaison des résultats fournis par les logiciels de reconnaissance des symboles, car il contient :

- des zones de texte,
- des titres de paragraphes,
- des formules incluses dans une zone textuelle,
- des formules isolées,
- une grande variété de symboles mathématiques,
- quelques petits et grands symboles.

[1] *Standard Mathematical Tables and Formulae* (1996).

Example

For the equation $u_x + x^2 u_y = -yu$ with $u = f(y)$ when $x = 0$, the corresponding equations are

$$\frac{\partial x}{\partial s} = 1, \qquad \frac{\partial y}{\partial s} = x^2, \qquad \frac{du}{ds} = -yu.$$

The original initial data can be written parametrically as $x(s = 0, t) = 0$, $y(s = 0, t_1) = t_1$, and $u(s = 0, t_1) = f(t_1)$. Solving for x results in $x(s, t_1) = s$. The equation for y can then be integrated to yield $y(s, t_1) = \frac{s^3}{3} + t_1$. Finally, the equation for u is integrated to obtain $u(s, t_1) = f(t_1) \exp\left(-\frac{s^4}{12} - st_1\right)$. These solutions constitute an implicit solution of the original system.

In this case, it is possible to eliminate the s and t_1 variables analytically to obtain the explicit solution: $u(x, y) = f\left(y - \frac{x^3}{3}\right) \exp\left(\frac{x^4}{4} - xy\right)$.

5.7.6 EXACT SOLUTIONS OF LAPLACE'S EQUATION

1. If $\nabla^2 u = 0$ in a circle of radius R and $u(R, \theta) = f(\theta)$, for $0 \le \theta < 2\pi$, then $u(r, \theta)$ is

$$u(r, \theta) = \frac{1}{2\pi} \int_0^{2\pi} \frac{R^2 - r^2}{R^2 - 2Rr\cos(\theta - \phi) + r^2} f(\phi)\, d\phi.$$

2. If $\nabla^2 u = 0$ in a sphere of radius one and $u(1, \theta, \phi) = f(\theta, \phi)$, then

$$u(r, \theta, \phi) = \frac{1}{4\pi} \int_0^{\pi} \int_0^{2\pi} f(\Theta, \Phi) \frac{1 - r^2}{(1 - 2r\cos\gamma + r^2)^{3/2}} \sin\Theta\, d\Theta\, d\Phi,$$

where $\cos\gamma = \cos\theta\cos\Theta + \sin\theta\sin\Theta\cos(\phi - \Phi)$.

3. If $\nabla^2 u = 0$ in the half plane $y \ge 0$, and $u(x, 0) = f(x)$, then

$$u(x, y) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{f(t)y}{(x - t)^2 + y^2}\, dt.$$

4. If $\nabla^2 u = 0$ in the half space $z \ge 0$, and $u(x, y, 0) = f(x, y)$, then

$$u(x, y, z) = \frac{z}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{f(\zeta, \eta)}{\left[(x - \zeta)^2 + (y - \eta)^2 + z^2\right]^{3/2}}\, d\zeta\, d\eta.$$

5.7.7 SOLUTIONS TO THE WAVE EQUATION

Consider the wave equation $\frac{\partial^2 u}{\partial t^2} = \nabla^2 u = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2}$, with $\mathbf{x} = (x_1, \ldots, x_n)$ and the initial data $u(0, \mathbf{x}) = f(\mathbf{x})$ and $u_t(0, \mathbf{x}) = g(\mathbf{x})$. When n is odd (and $n \ge 3$), the solution is

$$u(t, \mathbf{x}) = \frac{1}{1 \cdot 3 \cdots (n - 2)} \left\{ \frac{\partial}{\partial t} \left(\frac{\partial}{t\, \partial t} \right)^{(n-3)/2} t^{n-2} \omega[f; \mathbf{x}, t] + \left(\frac{\partial}{t\, \partial t} \right)^{(n-3)/2} t^{n-2} \omega[g; \mathbf{x}, t] \right\}, \tag{5.7.2}$$

FIG. 1 – Page de test pour la comparaison des résultats des logiciels d'OCR

Les resultats obtenus avec le logiciel *OmniPage Pro 10*, sur cette page de test, montrent bien que les symboles mathématiques et les symboles grecs ne sont absolument pas reconnus. De plus, les zones contenant des informations mathématiques ne sont pas détectées comme telles, mais comme du texte que le logiciel n'arrive pas à interpréter. La structure du document (titre de paragraphe, en-tête et pied de page, etc) sont correctement identifiés.

```
420     Chapter 5 Continuous Mathematics

Example For the equation ux + xzuY - -yu with u = f (y) when x = 0, the corresponding
equations are

8x      8y      z       du
        8s = 1' 8s = x '        ds = -Yu.

The original initial data can be written parametrically as x (s = 0, t) = 0, y (s =
0, tl) = tj, and u (s = 0, tj) = f (tj ). Solving for x results in x (s, ti) = s. The equa-
tion for y can then be integrated to yield y(s, t,) = 3 + tj. Finally, the equation for

u is integrated to obtain u(s, tl) = f (ti) exp(-i2 - st,). These solutions constitute
an implicit solution of the original system.
In this case, it is possible to eliminate the s and tj variables analytically to obtain
        x3      xa      1
the explicit solution: u (x, y) = f (y - 3 ) expC 4 - XY I.


5.7.6 EXACT SOLUTIONS OF LAPLACE'S EQUATION
1. If Vzu = 0 in a circle of radius R and u(R, 0) = f (O), for 0 _< 0 < 27r, then
u(r, 9) is
        1       271     R2 - r2
u (r, B) = 27r f R2 - 2Rr cos(B - 0) + rz f (~) d~.
2. If Vzu = 0 in a sphere of radius one and u(1, B, 0) = f (O, 0), then
        1 f7r z"        1 - r 2
u(r, 0, 0) =    f (O, (D)       sin 9d6d(t),
4n ~    (1 - 2r cos y + rz)3/z
where cos y = cos 0 cos O + sin B sin O cos(o - (D).
3. If Vzu = 0 in the half plane y > 0, and u (x, 0) = f (x), then
1       °°      f(t)y
u(x, Y) = n f . (x - t)z + yz dt.
4. If Vzu = 0 in the half space z > 0, and u (x, y, 0) = f (x, y), then

u (x, Y, z) = Z ~~      f (~, 77)       3/z d~ dq.

27C ~ f_ [(x - ~)z -1- (Y - n)z + zz]

5.7.7 SOLUTIONS TO THE WAVE EQUATION
Consider the wave equation ate = Vzu = ax z + + ate, with x = (xi, . . . , xn)
and the initial data u(0, x) = f (x) and ut (0, x) = g (x). When n is odd (and n >_ 3),
the solution is
        (n-3)/2
u (t, x) = 1 . 3 . . 1(n - 2) { 8t (t at)       tn-z~~. f' x' t]

(n-3)/2
+ (t at) to z,)rg> x, t]~~
JJ      (5.7.2)
```

L'ensemble des logiciels testés dans le chapitre IV offrent des résultats similaires, à quelques petites nuances près :

– erreurs sur la structure du document,

– plus ou moins d'erreurs sur la reconnaissance des symboles,
– etc.

Quoiqu'il en soit, les remarques et les resultats présentés pour le logiciel*OmniPage Pro 10* restent les mêmes pour les notations mathématiques :

– non reconnaissance des symboles mathématiques,
– non reconnaissance des symboles de l'alphabet grec,
– non identification des zones contenant des formules.

Ces remarques nous permettent d'aboutir à la conclusion qu'aucun des logiciels testés ne peut être utilisé dans le cadre précis de nos recherches.

Références bibliographiques

La bibliographie présentée ici, a été organisée en différentes parties, afin d'en faciliter la lecture. Les références les plus importantes à notre sens ont été commentées donnant ainsi un aperçu du contenu de l'article ou du livre cité. Nous espérons que ce travail aidera le lecteur dans sa recherche d'informations complémentaires.

Analyse d'image

[KJH95] Tapas Kanungo, M. Y. Jaisimha, and Robert M. Haralick. A Methodology for Quantitative Performance Evaluation of Detection Algorithms. In *IEEE Transactions on Image Processing*, volume 4 (12), pages 1667–1674. IEEE Computer Society Press, December 1995.

[KT95] Rangachar Kasturi and Karl Tombre, editors. *Graphics Recognition : Methods and Applications*. Number 1072 in Lecture Notes in Compter Science. Springer-Verlag, 1995.

[LCP90] Sang Uk Lee, Seok Yoon Chung, and Rae Hong Park. « A comparative performance study of several global thresholding techniques for segmentation ». *Computer Vision, Graphics, and Image Processing*, 52(2) :171–190, November 1990.

[SSWC88] P. K. Sahoo, S. Soltani, A. K. C. Wong, and Y. C. Chen. A Survey of Threhloding Techniques. In *Computer Vision Graphics Image Processing*, volume 41, pages 233–260. Academic Press, 1988.

[TC97] Karl Tombre and Atul K. Chhabra, editors. *Graphics Recognition : Algorithms and Systems*. Number 1389 in Lecture Notes in Compter Science. Springer-Verlag, 1997.

[WWL92] Wen-Yen Wu, Mao-Jiun J. J. Wang, and Chih-Ming Liu. « Performance Evaluation of Some Noise Reduction Methods ». *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 54(2) :134–146, March 1992.

Reconnaissance de motifs et reconnaissance de caractères

[ADPF98] Adnan Amin, Dov Dori, Pavel Pudil, and Herbert Freeman, editors. *Advances in Pattern Recognition*, volume 1451 of *Lecture Notes in Computer Sciences*. Springer-Verlag, 1998.

[Bai88] H. S. Baird. « Applications of Multidimensional Search to Structural Feature Identification ». In G. Ferraté, editor, *Syntactic and Structural Pattern Recognition*, volume 45 of *F : Computer and Systems Sciences*, pages 137–149. Springer-Verlag, 1988.

[BS82] B. Bartsch-Spörl. Grammatical Inference of Graph Grammars for Syntactic Pattern Recognition. In Harmud Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors, *Graph Grammars and their Application to Computer Science*, volume 153 of *Lecture Notes in Computer Sciences*, pages 1–7. Springer-Verlag, October 1982.

[Bun86] Horst Bunke. « Hybrid Approaches ». In G. Ferraté, T. Pavlidis, A. Sanfeliu, and H. Bunke, editors, *Syntactic and Structural Pattern Recognition*, volume 45 of *Computer and Systems Sciences*, pages 335–361, October 1986.

[Bun90] Horst Bunke. Hybrid Pattern Recognition Methods. In H. Bunke and A. Sanfelui, editors, *Syntactic and Structural Pattern Recognition : Theory and Applications*, pages 308–347. World Scientific, 1990.

[Bun93] Horst Bunke. Structural and Syntactic Pattern Recognition. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *Handbook of Pattern Recognition & Computer Vision*, pages 163–209. World Scientific, 1993.

[CJ83] R. G. Casey and C. R. Jih. A Processor-Based OCR System. In *IBM Journal of Research and Development*, volume 27 (4), pages 386–399. July 1983.

[DF85] H. S. Don and K. S. Fu. A Syntactic Method for Image Segmentation and Object Recognition. In *Pattern Recognition*, volume 18(1), pages 73–87. 1985.

[DHH+97] Andreas Dengel, Rainer Hoch, Franck Hönes, Thorsten Jäger, Michael Malburg, and Achim Weigel. Techniques for Improving OCR Results. In Horst Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis (ICDAR)*, Chapitre 8, pages 227–258. World Scientific, 1997.

[Elm96] A. J. Elms. « *The Representation and Recognition of Text Using Hidden Markov Models* ». PhD Thesis, University of Surrey, Guildford, Surrey, Grande-Bretagne, April 1996.

[Fu83] K. S. Fu. A Step Towards Unification of Syntactic and Statistical Pattern Recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, volume 5(2). IEEE Computer Society, March 1983.

[GT78] Rafael C. Gonzalez and Michael G. Thomason. *Syntactic Pattern Recognition : an Introduction*. Addison-Wesley Publishing Compagny Inc, 1978.

[JK94] F. Jenkins and J. Kanai. « The use of synthesized images to evaluate the performance of optical character recognition devices and algorithms ». In *SPIE Proceedings, Document Recognition*, pages 194–203, San Jose, California, USA, 1994. SPIE.

[Man86] J. Mantas. « An Overview of Character Recognition Methodologies ». *Pattern Recognition*, 19 :425–430, 1986.

[MLP93] Angelo Marcelli, Natasha Likhareva, and Theo Pavlidis. « A Structural Indexing Method for Character Recognition ». In IEEE Computer Society Press, editor, *Proceedings of The Second International Conference on Document Analysis and Recognition (ICDAR)*, pages 175–178, Tsukuba, Japon, October 1993.

[Nag82] George Nagy. « Optical Character Recognition - Theory and Practice ». In P.R. Krishnaiah and L.N. Kanals, editors, *Handbook of Statistics*, volume 2, pages 621–649. North-Holland Publishing Compagny, 1982.

[Nag92] George Nagy. « Teaching a Computer to Read ». In *Proceedings of the 11th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 225–229, The Hagues, Pays-Bas, September 1992. IEEE Computer Society Press.

[Pav80] Theodosios Pavlidis. *Structural Pattern Recognition*. Springer-Verlag, 2 edition, 1980.

[PWR96] Petra Perner, Patrick Wang, and Azriel Rosenfeld, editors. *Advances in Structural and Syntactical Pattern Recognition*, volume 1121 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[SF82] A. Sanfelui and K. S. Fu. Tree-Graphs Grammars for Pattern Recognition. In Harmud Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors, *Graph Grammars and their Application to Computer Science*, volume 153 of *Lecture Notes in Computer Sciences*, pages 8–19. Springer-Verlag, October 1982.

[Sko90] E. Skordalakis. « *Syntactical and Structural Pattern Recognition : Theroy and Applications* », Chapitre 18, pages 500–532. World Scientific, 1990.

[TF80] W. H. Tsai and K. S. Fu. Attributed Grammar : A Tool for Combining Syntactic and Statistical Approaches to Pattern Recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, volume 10 (12), pages 873–885. IEEE, 1980.

[Tsa90] W. H. Tsai. « *Syntactic and Structural Pattern Recognition : Theory and Applications* », Chapitre 12, pages 349–366. World Scientific, 1990.

[Wil93] C. L. Wilson. « Evaluation of character recognition systems ». pages 485–496, 1993.
ftp ://sequoyah.nist.gov/pub/papers_preprints/eval_ocr_sys.ps.Z.

Reconnaissance de l'écriture manuscrite

[AGG95] E. Anquetil, G.Lorette, and P. Gentric. « Reconnaissance en Ligne d'Écriture Cursive par Chaînes de Markov Cachées ». In *Traitement du Signal*, volume 12, pages 575–583, 1995.

[Ber75] Marc Berthod. « *Une Méthode Syntaxique de Reconnaissance des Caractères Manuscrits en Temps Réel avec un Apprentissage Continu* ». PhD Thesis, Université Paris VI, 1975.

[BK88] G. Baptista and K. M. Kulkarni. « A High Accuracy Algorithm for Recognition of Handwritten Numerals ». In Pergamon Press, editor, *Pattern Recognition*, volume 21 of *4*, pages 287–291, 1988.

[BS97] Abdel Belaïd and G. Saon. « Utilisation des processus markoviens en reconnaissance de l'écriture ». In *Traitement du Signal*, 1997.
Localisation du document : http ://www.loria.fr/equipes/read/bibliographie.html.

[GSD97] Isabelle Guyon, Markus Schenkel, and John Denker. Overview and Synthesis of On-Line Cursive Handwritten Recognition Techniques. In Horst Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, Chapitre 7, pages 183–225. World Scientific, 1997.

[KRR97] A. Kosmala, J. Rottland, and G. Rigoll. « Improved On-Line Handwriting Recognition Using Context Dependent Hidden Markov Models ». In *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR)*, pages 641–644, Ulm, Allemagne, 1997.
http ://www.fb9-ti.uni-duisburg.de/veroeffentlichungen.html.

[Kun97] Amlan Kundu. Handwritten Word Recognition using Hidden Markov Model. In Hort Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, Chapitre 6, pages 157–182. World Scientific, 1997.

[Lor96] G. Lorette. « Le Traitement Automatique de l'Écrit et du Document.État de la Recherche ». In *Sciences de l'Information*, volume 33, pages 214–217, July 1996.

[LWL96] Stefan Lehmberg, Hans-Jürgen Winkler, and Manfred Lang. « A Soft-Decision Approach for Symbol Segmentation within Handwritten Mathematical Expressions ». In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 6, pages 3434–3437, Atlanta, Etats-Unis, 1996.

Présentation d'un système permettant la segmentation des caractères et symboles composant les formules mathématiques. La méthode est basée sur l'apprentissage de symboles, l'extraction de données représentatives et la construction d'un modèle. Les performances annoncées sont bonnes, et les erreurs persistantes sont principalement dues à la connectivité de deux symboles ou au non regroupement des différentes parties composant un caractère.

[RKRN96] G. Rigoll, A. Kosmala, J. Rottland, and Ch. Neukirchen. « A Comparison Between Continuous and Discrete Density Hidden Markov Models for Cursive Handwriting Recognition ». In *Proceedings of IEEE Interntional Conference on Pattern Recognition (ICPR)*, pages 205–209, 1996.

http ://www.fb9-ti.uni-duisburg.de/veroeffentlichungen.html.

[TK99] Sergios Theodoridis and Konstantinos Koutroumbas, editors. « *Pattern Recognition* », Chapitre 9 : Context-Dependent Classification, pages 307–337. Academic Press, 1999.

[Win96] Hans-Jürgen Winkler. « HMM-Based Handwritten Symbol Recognition Using On-Line and Off-Line Features ». In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 6, pages 3438–3441, Atlanta, Etats-Unis, 1996.

Analyse de la structure de documents

[BC97] Abdel Belaïd and Yannick Chenevoy. « Document Analysis for Retrospective Conversion of Library Reference Catalogues ». In IEEE Computer Society Press, editor, *Proceedings of The Fourth International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, page 432, Ulm, Allemagne, August 1997.

Localisation du document : http ://www.loria.fr/equipes/read/bibliographie.html.

[BCP97] Abdel Belaïd, Yannick Chenevoy, and F. Parmentier. « Panorama des méthodes structurelles en analyse et reconnaissance de documents ». In *Journées d'Études sur les Documents*, 1997.

Localisation du document : http ://www.loria.fr/equipes/read/bibliographie.html.

[CB94] Yannick Chenevoy and Abdel Belaïd. « Low-Level Structural Recognition of Documents ». In *Third Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, Etats-Unis, 1994.

Localisation du document : http ://recife.u-bourgogne.fr :8081/˜chenevoy/biblio.html.

[Che92] Yannick Chenevoy. « *Reconnaissance structurelle de documents imprimés : études et réalisations.* ». PhD Thesis, Centre de Recherche en Informatique de Nancy, December 1992.

Localisation du document : http ://recife.u-bourgogne.fr :8081/˜chenevoy/these.html.

[Fat97] Richard J. Fateman. « How to Find Mathematics on a Scanned Page ». September 1997.

Localisation du document : http ://HTTP.CS.Berkeley.EDU/˜fateman/papers/see-eqns.ps Description d'une méthode à base d'heuristiques permettant de localiser des formules mathématiques insérées dans du texte. Quelques idées sont intéressantes et beaucoup de problèmes sont soulevés.

[FT98] Richard J. Fateman and Taku Tokuyasu. « A Suite of Programs for Documents Structuring and Image Analysis using Lisp ». February 1998.

Localisation du document : http ://HTTP.CS.Berkeley.EDU/˜fateman/papers/suite.ps.

[JJ97] Anil K. Jain and Bin Ju. « Page Segmentation Using Document Model ». In IEEE Computer Society, editor, *Proceedings of Fourth International Conference on document Analysis and Recognition (ICDAR)*, volume 1, pages 34–38, Ulm, Allemagne, August 1997.

[KNSV93] Mukkai Krishnamoorthy, George Nagy, Sharad Seth, and Mahesh Viswanathan. Syntactic Segmentation and Labeling of Digitized Pages from Technical Journals. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 15, pages 737–747. IEEE Computer Society, July 1993.

[OK97] Laurence O'Gorman and Rangachar Kasturi. *Executive Briefing : Document Image Analysis*. IEEE Computer Society, 1997.

[WWC82] Friedrich M. Wahl, Kwan Y. Wong, and Richard G. Casey. Block Segmentation and Text Extraction in Mixed Text/Image Documents. In *Computer Graphics and Image Processing*, volume 20. Academic Press, December 1982.

Grammaire et réécriture de graphes

[BF91] Dorothea Blostein and Hoda Fahmy. « A Graph Grammar for High-Level Recognition of Music Notation ». In *Proceedings of First International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 70–78, Saint Malo, France, October 1991. IEEE Computer Society.

[BF92a] Dorothea Blostein and Hoda Fahmy. « Graph Grammar Processing of Uncertain Data ». In Horst Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Machine Perception and Artificial Intelligence*, pages 373–382. World Scientific, August 1992.

[BF92b] Dorothea Blostein and Hoda Fahmy. « A Survey of Graph Grammars : Theory and Applications ». In *Proceedings of the 11th International Conference on Pattern Recognition (ICPR)*, volume The Hague, Pays-Bas. IEEE Computer Society, September 1992.

[BFG95] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. « Practical Use of Graph Rewriting ». Technical Report 95-373, Queen's University, Kingston Department of Computing and Information Science, Ontario, Canada, January 1995.

[Bun82a] Horst Bunke. Attributed Programmed Graph Grammars and their Application to Schematic Diagram Interpretation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, volume 4, pages 574–582. IEEE Computer Society, November 1982.

[Bun82b] Horst Bunke. Graph Grammars as a Generative Tool in Image Understanding. In Harmud Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors, *Graph Grammars and their Application to Computer Science*, volume 153 of *Lecture Notes in Computer Sciences*, pages 8–19. Springer-Verlag, October 1982.

[CEG79] V. Claus, H. Ehrig, and G.Rosenberg, editors. *Graph-Grammars and Their Applications to Computer Science and Biology*, Lecture Notes in Computer Science, 1979.

[Cou87] B. Courcelle. « An Axiomatic Definition of Context-Free Rewriting and its Application to NLC Graph Grammars ». Technical Report, Université de Bordeaux, February 1987.

[Dör95] Heiko Dörr. Efficient Graph Rewriting and its Implementation. In Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen, editors, *Lecture Notes in Computer Science*, volume 922. Springer-Verlag, 1995.

[ENR86] Harmud Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors. *Graph Grammars and their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Sciences*. Springer-Verlag, December 1986.

[Gla82] Serge Gladkoff. « *Grammaires de Graphes et Applications* ». PhD Thesis, Université Paris VI, June 1982.

[Nag78] Manfred Nagl. A Tutorial and Bibliographical Survey on Graph Grammars. In *Workshop on Graph-Grammars and their Application to Computer Science and Biology*, volume 73 of *Lectures Notes in Computer Science*, pages 70–126. Springer-Verlag, Bad Honnef, November 1978.

[Pfe90] Joseph J. Pfeiffer. « Using Graph Grammars for Data Structure Manipulation ». In *IEEE Workshop on Visual Languages*, pages 42–53, Skokie, Illinois, Etats-Unis, October 1990. IEEE Computer Society Press.

[PR69] John Pfaltz and Azriel Rosenfled. « Web Grammars ». In *Proceedings of First International Joint Conference on Artificial Intelligence*, pages 609–619, Washington, 1969.

[RV92] Jean-Claude Raoult and Frédéric Voisin. « Set-Theoretic Graph Rewriting ». Technical Report, IRISA, April 1992.

[SE93] Hans Jürgen Schneider and Hartmut Ehrig, editors. *Graph Transformations in Computer Science*, volume 776 of *Lecture Notes in Computer Sciences*. Springer-Verlag, January 1993.

[Zün94] Albert Zündorf. Graph Pattern Matching in PROGRES. In J. Cuny, H. Ehrig, G. Engels, and G. Rosenberg, editors, *Lecture Notes in Computer Science*, volume 1073, pages 454–468. Springer-Verlag, Williamsburg, Etats-Unis, November 1994.

http ://www-i3.informatik.rwth-aachen.de/research/progres/ .

Reconnaissance de formules mathématiques

[And67] Robert H. Anderson. Syntax-Directed Recognition and Hand-Printed Two-Dimensional Mathematics. In M. Klerer and J. Reinfelds, editors, *ACM Symposium on Interactive Systems for Experimental Applied Mathematics*, pages 436–459. Academic Press, Washington (DC), Etats-Unis, August 1967.

Cet article est un "résumé" de la thèse de doctorat [?] sur la reconnaissance syntaxique de formules mathématiques manuscrites. Pas de nouvelles idées développées, et mêmes exemples que ceux de la thèse.

[And68] Robert H. Anderson. « *Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics* ». PhD Thesis, Harvard University, Cambridge, January 1968.

L'un des premiers travaux sur la reconnaissance de formules mathématiques (manuscrites qui plus est). La méthode mise en place définit une grammaire de coordonnées (grammaire graphique) et utilise une analyse descendante. Les exemples sont nombreux et illustrent bien le propos. Les règles de la grammaire sont en annexes.

[And77] Robert H. Anderson. « Two Dimensional Mathematical Notation ». In K. S. Fu, editor, *Syntactic Pattern Recognition Applications*, pages 147–177. Springer-Verlag, 1977.

Cet article reprend les conclusions de la thèse de doctorat [?] et fournit en annexes les règles de la grammaire de coordonnées introduite, sous forme graphique. Pas de nouveaux éléments.

[Bel79] Abdelwahed Belaïd. « *Reconnaissance Structurelle de Caractères Manuscrits et de Formules Mathématiques* ». PhD Thesis, Université de Nancy, October 1979.

Description d'un système pour la reconnaissance de formules mathématiques manuscrites. La reconnaissance des symboles utilise la vectorisation du tracé puis la comparaison de ce modèle avec les différentes séquences stockées dans un arbre de décision, lors de la phase d'apprentissage. Un ensemble de caractères possibles sont retournés par le logiciel et la détermination se fait lors de l'étape suivante. La reconnaissance structurelle est du type approche syntaxique et utilise une grammaire de coordonnées. L'analyse est à la fois descendante et montante. L'auteur insiste sur l'identification des symboles en fonction du contexte.

[BG95] Dorothea Blostein and Ann Grabvec. « Mathematics Recognition Using Graph Rewriting ». In *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR)*, pages 417–421, Montréal, Canada, August 1995. IEEE Computer Society Press.

Description de Expresso, un système dédié à la reconnaissance de partitions musicales, adapté à la reconnaissance de formules mathématiques. Quelques exemples de formules simples sont donnés et montrent les difficultés de la reconnaissance de la structure d'une expression mathématique. La méthode utilisée est à base de réécriture de graphes. Quelques exemples simples de formules reconnues sont fournis. Les auteurs insistent sur le temps, assez important, de calcul nécessaire.

[BG97] Dorothea Blostein and Ann Grbavec. Recognition of Mathematical Notation. In Horst Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, Chapitre 21, pages 557–582. World Scientific, 1997.

Historique sur les différentes techniques mises en oeuvre. Les auteurs explicitent bien les différents points qui rendent la reconnaissance d'expressions mathématiques difficile. La bibliographie est complète et contient toutes les références importantes. Très bon état de l'art.

[BH84] Abdelwaheb Belaïd and Jean-Paul Haton. A Syntactic Approach for Handwritten Mathematical Formula Recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, volume 6, pages 105–111. IEEE Computer Society, January 1984.

Article qui résume le contenu de la thèse [?].

[Cha70] Shi-Kuo Chang. « A Method for the Structural Analysis of Two-Dimensional Mathematical Expressions ». *Information Sciences*, 2 :253–272, 1970.

[Cho89] P. Chou. « Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar ». In *Proceedings SPIE Conference on Visual Communications and Image Processing IV*, pages 852–863, Philadelphie, Etats-Unis, November 1989.

Présentation d'une méthode originale utilisant une grammaire probabiliste permettant l'analyse d'images bruitées contenant des expressions mathématiques. Les résultats présentés portent sur des équations simples. Aucun test n'a été réalisé sur des structures bidimensionnelles plus complexes, comme les matrices, les racines carrées, les équations sur plusieurs lignes.

[DCdlM91] Yannis A. Dimitriadis, Juan Lopez Coronado, and Cristina de la Maza. « A New Interactive Mathematical Editor, Using On-Line Handwritten Symbol Recognition, and Error Detection-Correction with an Attribute Grammar ». In *Proceedings of First International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 885–893, Saint Malo, France, October 1991. IEEE Computer Society Press.

L'idée générale est de mettre en place un système de type interface interactive pour les formules mathématiques manuscrites. La reconnaissance structurelle mise en place utilise une grammaire attribuée, avec une analyse ascendante, ainsi qu'un système pour compenser les erreurs possibles lors de la reconnaissance des symboles. Beaucoup d'informations techniques sont données en annexes, ainsi que des exemples de ce qui est reconnu.

[Fat97] Richard J. Fateman. « More Versatile Scientific Documents Extended Abstract ». In *Proceedings of Fourth International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 1107–1110, Ulm, Allemagne, August 1997. IEEE Computer Society Press.

Dans cet article, l'auteur s'interroge sur les besoins et les applications possibles de la reconnaissance d'expressions mathématiques. Il définit plusieurs niveaux pour repré-

senter les éléments reconnus : une description purement visuelle ou typographique, une description syntaxique ou structurelle et une description sémantique. Le discours est toutefois un peu confus.

[FB94] Richard J. Fateman and Benjamin P. Berman. « Optical Character Recognition for Typeset Mathematics ». In ACM Press, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 348–353, July 1994.

Présentation d'un logiciel dédié à la reconnaissance de symboles et caractères composant les formules. La méthode est basée sur l'apprentissage et la comparaison d'images à l'aide de la distance de Hausdorff. Une intervention humaine est nécessaire pour corriger manuellement les erreurs et décrire les caractéristiques du symbole (taille, fonte italique, . . .). Aucune description n'est faite quant à la méthode utilisée pour la reconnaissance structurelle.

[FST+99] R. Fukuda, I. Sou, F. Tamari, X. Ming, and M. Suzuki. « A Technique of Mathematical Expression Structure Analysis for the Handwriting Input System ». In *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR)*, pages 131–134, Bangalore, Inde, September 1999. IEEE Computer Society Press.

[FT96] Richard J. Fateman and Taku Tokuyasu. « Progress in Recognizing Typeset Mathematics ». In *Proceedings of SPIE*, San José, Californie, Etats-Unis, 1996.

Description technique du logiciel de reconnaissance de caractères mathématiques développé. La reconnaissance structurelle est basée sur une analyse récursive descendante du découpage de la formule en boîtes horizontales et verticales. La méthode présentée est très spécifique à la reconnaissance d'équations de type intégrales, et non adaptable à d'autres notations (comme les matrices).

[FTBM96] Richard J. Fateman, Taku Tokuyasu, Benjamin P. Berman, and Nicholas Mitchell. Optical Character Recognition and Parsing of Typeset Mathematics. In *Journal of Visual Communication and Image Representation*, volume 7, pages 2–15. Academic Press, March 1996.

Synthèse détaillée des articles [?] et [?] à laquelle quelques exemples pertinents ont été ajoutés. La méthode reste restreinte à la reconnaissance d'équations "linéaires" et optimisée pour les intégrales.

[HHP95] Jaekyu Ha, Robert M. Haralick, and Ihsin T. Phillips. « Understanding Mathematical Expressions from Document Images ». In *Proceedings of the Third International Conference on Document Analysis and Recognition*, pages 956–959, Montréal, Canada, August 1995. IEEE Computer society Press.

Reprise de l'idée développée dans [?] : subdivisions successives de l'image en sous-éléments par projections horizontales et verticales (approche descendante). L'analyse structurelle est quant à elle ascendante, mais très peu de précisions sont apportées sur la méthode exacte qui est utilisée pour passer de la représentation graphique à l'arbre de syntaxe de l'expression.

[KBA99] A. Kacem, A. Belaïd, and M. Ahmed. « EXTRAFOR : Automatic EXTRAction of Mathematical FORmulas ». In *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR)*, pages 107–110, Bangalore, Inde, September 1999. IEEE Computer Society Press.

Présentation d'un système d'extraction de formules mathématiques dans un document typographié. Cette méthode est basée sur l'utilisation de logique floue. Elle permet d'identifier à la fois les notations isolées du texte, mais aussi l'utilisation de symboles mathématiques ou d'expressions mathématiques inclus dans des paragraphes de texte.

Les résultats présentés sont très intéressants (près de 100% pour les formules isolées et plus de 90% pour les notations incluses dans un paragraphe)..

[KLPR99] Andreas Kosmala, Stéphane Lavirotte, Loïc Pottier, and Gerhard Rigoll. « On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars ». In *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR)*, pages 107–110, Bangalore, Inde, September 1999. IEEE Computer Society Press.

Adaptation au cas manuscrit de la méthode pour la reconnaissance d'expressions mathématiques typographiées présentée dans [?]. Première utilisation d'une grammaire de graphes pour la reconnaissance de formules mathématiques manuscrites..

[KR98] Andreas Kosmala and Gerhard Rigoll. « Recognition of On-Line Handwritten Formulas ». In *6th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korée, 1998.

Description d'une application permettant la reconnaissance de formules manuscrites. La reconnaissance des symboles se fait grâce à une méthode à base de HMMs (Hidden Markov Models) avec un apprentissage du style d'écriture. La reconnaissance structurelle fait des pré-supposés sur la manière dont la formule est écrite (ordre d'écriture des bornes d'intégrales, indices, exposants,. . .). Pas de possibilité d'étendre la technique à d'autres notations.

[Lit95] Richard Henry Littin. « Mathematical Expression Recognition : Parsing Pen/Tablet Input in Real-Time Using LR Techniques ». Master's Thesis, University of Waikato, Hamilton, New Zealand, March 1995.

[LL93] His-Jian Lee and Min-Chou Lee. « Understanding Mathematical Expressions in a Printed Document ». In *Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR)*, pages 502–505, Tsukuba, Japon, October 1993. IEEE Computer Society Press.

Présentation d'un composant logiciel, comprenant la reconnaissance des symboles (méthode de comparaison d'images sur un ensemble de 13 points caractéristiques), et une partie reconnaissance structurelle basée sur le regroupement des symboles en fonction de la localisation. La méthode proposée semble trop spécifique pour être adaptée à la reconnaissance de notations réellement bidimensionnelles (comme les matrices).

[LL94] Hsi-Jian Lee and Min-Chou Lee. Understanding Mathematical Expressions using Procedure-Oriented Transformation. In *Pattern Recognition*, volume 27, pages 447–457. Pergamon, 1994.

Cet article reprend les principes énoncés dans une publication précédente [?], en ajoutant des exemples pour l'illustration. Une méthode de détection et de correction pour les symboles mal séparés lors de la reconnaissance des symboles.

[LP97a] Stéphane Lavirotte and Loïc Pottier. « Mathematical Formula Recognition ». In *International Workshop on Retrodigitalization of Mathematical Documents*, Essen, Allemagne, December 1997.

[LP97b] Stéphane Lavirotte and Loïc Pottier. « Optical Formula Recognition ». In *Proceedings of Fourth International conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 357–361, Ulm, Allemagne, August 1997. IEEE Computer Society Press.

Présentation d'une méthode permettant d'utiliser les grammaires de graphes pour la reconnaissance de formules mathématiques. Cette méthode repose sur la complétion automatique des règles par une méthode de type Knuth-Bendix. Démonstration théorique de la méthode (propositions et preuves).

[LP98] Stéphane Lavirotte and Loïc Pottier. « Mathematical Formula Recognition using Graph Grammar ». In *Document Recognition V*, volume 3305, pages 44–52, San José, Californie, Etats-Unis, January 1998. SPIE - The International Society for Optical Engineering.

Description du processus complet pour la reconnaissance structurelle de formules, avec la mise en application des résultats théoriques de [?].

[LW95] Hsi-Jian Lee and Jiumn-Shine Wang. « Design of a Mathematical Expression Recognition System ». In *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR)*, pages 1084–1087, Montréal, Canada, August 1995. IEEE Computer Society Press.

Présentation d'un système complet permettant l'extraction et l'analyse de formules mathématiques incluses dans un document imprimé. La reconnaissance des caractères s'effectue sur le modèle classique de la comparaison de données caractéristiques des symboles. L'analyse structurelle est celle employée dans [?] et n'est pas assez généraliste. Un cas particulier est mis en place pour reconnaître les matrices. Une dernière étape permet d'effectuer des corrections pour les symboles mal reconnus.

[MV98] Erick G. Miller and Paul A. Viola. « Ambiguity and Constraint in Mathematical Expression Recognition ». In American Association of Artificial Intelligence, editor, *Proceedings of the 15th National Conference of Artificial Intelligence*, pages 784–791, Madison, Wisconsin, 1998.

http ://www.ai.mit.edu/people/emiller/OQE_slides/ .

[OM91] Masayuki Okamoto and Bin Miao. « Recognition of Mathematical Expressions by Using the Layout Structure of Symbols ». In *Proceedings of First International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 242–250, Saint Malo, France, October 1991. IEEE Computer Society.

Méthode originale basée sur la segmentation de l'expression mathématique par projections horizontales et verticales recursives. L'analyse est donc descendante et construit un arbre de la structure graphique qui est transformé en structure syntaxique à l'aide de quelques heuristiques. Quelques exemples montrent les limites de cette méthode, mais celle-ci est toutefois adaptée à la reconnaissance de matrices.

[OM92] Masayuki Okamoto and A. Miyazawa. An Experimental Implementation of a Document Recognition System for Papers Containing Mathematical Expressions. In H.S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 36–53. Springer-Verlag, Heidelberg, Allemagne, 1992.

Présentation du processus complet d'analyse d'un document contenant des formules mathématiques, avec la segmentation, la détection d'une rotation éventuelle, . . . La méthode de reconnaissance des expressions mathématiques est la même que celle présentée dans [?].

[OT95] Masayuki Okamoto and Hashim M. Twaakyondo. « Structure Analysis and Recognition of Mathematical Expressions ». In *Proceedings of Third International Conference on Document Analysis and Recognition (ICDAR), Montréal (Canada)*, pages 430–437, August 1995.

C'est globalement la même méthode que celle présentée dans [?] et [?]. Quelques améliorations et exemples supplémentaires détaillés illustrent bien le propos.

[Pot94] Loïc Pottier. « Reconnaissance de Formules Planes ». Technical Report, INRIA Sophia-Antipolis, projet SAFIR, March 1994.

[Smi99] Steve Smithies. « Freehand Formula Entry System ». Master's Thesis, University of Otago, Dunedin, New Zealand, May 1999.

[TGSE99] J. Toumit, S. Garcia-Salicetti, and H. Emptoz. « A Hierarchical and Recursive Model of Mathematical Expressions for Automatic Reading of Mathematical Documents ». In *Proceedings of the*

Fifth International Conference on Document Analysis and Recognition (ICDAR), pages 119–122, Bangalore, Inde, September 1999. IEEE Computer Society Press.

[WF88] Zi Xiong Wang and Claudie Faure. « Structural Analysis of Handwritten Mathematical Expressions ». In *Proceedings of the 9th International Conference on Pattern Recognition (ICPR)*, pages 32–34, Rome, Italie, November 1988. IEEE Computer Society Press.

La méthode présentée permet d'anoter les relations entre des symboles manuscrits proches, afin de différencier des symboles alignés, en position d'exposants ou d'indices. La détermination se fait suivant le contexte et une probabilité calculée lors d'une phase d'apprentissage. Les éléments ont été isolés par projections.

[WF90] Zi Xiong Wang and Claudie Faure. « Automatic Perception of the Structure of Handwritten Mathematical Expressions ». In *Computer Processing of Handwriting*, pages 337–361. World Scientific, 1990.

Cet article présente une application de reconnaissance d'expressions mathématiques manuscrites. Les exemples proposés montrent les limitations des projections pour la segmentation en sous-expressions dans le cas manuscrit. Les principes de reconnaissance de l'alignement des symboles présentés dans [?] sont développés et illustrés.

[ZSST96] Yanjie Zhao, Tetsuya Sakurai, Hiroshi Sugiura, and Tatsuo Torii. « A Methodology of Parsing Mathematical Notation for Mathematical Computation ». In ACM Press, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 292–300, July 1996.

Langages de script

[Cor99] Corporation for National Research Initiatives. « *Python Reference Manual* », 1.5.2 edition, July 1999.

[Gal94] Eric Gallesio. « STklos : A Scheme Object-Oriented System Dealing with the Tk Toolkit ». In *Proceedings of Xhibition'94*, San José, Californie, 1994.

[Gal96] Mark Galassi. « *Guile User Manual* ». Free Software Foundation, 1996.

[Léo95] Jean-Michel Léon. « *Contribution à la Simplification des Boîtes à Outils Graphiques : Application aux Langages de Scripts* ». PhD Thesis, Université de Nice Sophia-Antipolis, December 1995.

[Nah94] Colas Nahaboo. « *Klone C Interface Manual* ». Koala Project, Bull Research, INRIA Sophia-Antipolis, France, 1.0 edition, April 1994.

ftp ://koala.inria.fr/pub/Klone/klone-C.ps.gz.

[Nah95] Colas Nahaboo. « *Klone Reference Manual* ». Koala Projet, Bull Research, INRIA Sophia-Antipolis, France, 0.9i edition, March 1995.

ftp ://koala.inria.fr/pub/Klone/klone-ref.ps.gz.

[Ous94] John K. Ousterhout. *Tcl and Tk Toolkit*. Addison-Wesley Publising, 1994.

[SS93] Larry Schwartz and Randal Schwartz. *Programming Perl*. O'Reilly & Associates, 19950-93.

Divers

[ADG99] Olivier Arsac, Stéphane Dalmas, and Marc Gaëtano. « The Design of a Customizable Component to Display and Edit Formulas ». In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 283–290, Vancouver, Canada, July 1999. ACM Press.

[Ars97] Olivier Arsac. « *Interfaces Homme Machine pour le Calcul Formel* ». PhD Thesis, Université de Nice Sophia-Antipolis, July 1997.

[ASU91] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilateurs : Principes, Techniques et Outils*. InterEditions, 4 edition, 1991.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing Translation and Compiling*, volume 1 : Parsing of *Automatic Computation*. George Forsythe, 1972.

[Blo95] Dorothea Blostrein. General Diagram-Recognition Methodologies. In Rangachar Kasturi and Karl Tombre, editors, *Lectures Notes in Computer Science*, volume 1072, pages 106–122. Springer-Verlag, University Park, Etats-Unis, August 1995.

[Car84] Stuart K. Card. « Human Limits and VDT Computer Interface ». In J. Bennett, D. Case, J. Sandelin, and M. Smith, editors, *Visual Display Terminal*, page 117, Englewood Cliffs, 1984. Prentice-Hall.

[CC91] G. Costagliola and S. K. Chang. « Parsing 2-D Languages with Positional Grammars ». In *Second International Workshop on Parsing Technologies (IWPT)*, pages 235–243, Cancun, Mexique, February 1991.

[CEB78] S. K. Card, W.K. English, and B.J. Burr. « Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys and Text Keys for Text Selection on a CRT ». *Ergonomics*, 21 :601–613, 1978.

[CGG+88] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan, and S.M. Watt. *Maple Reference Manual*. Watcom Publications Limited, Ontario, Canada, 5 edition, 1988.

[CGN+90] C. Crimi, A. Guercio, G Nota, G. Pacini, and M. Tucci G. Tortora. « Relation Grammars for Modeling Multi-Dimensional Structures ». In *Workshop on Visual Languages*, pages 168–173, Skokie, Illinois, Etats-Unis, October 1990. IEEE Computer Society Press.

[Cho65] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.

[CMN80] Stuart K. Card, Thomas P. Moran, and Allen Newell. « The Keystroke Level Model for User Performance time with Interactive Systems ». *Communication of ACM*, 23 :396–410, 1980.

[Col92] Suzanne Collin. « *Interprétation de la Cotation des Dessins Techniques par Analyse Syntaxique* ». PhD Thesis, Institut National Polytechnique de Lorraine, January 1992.

[DGH96] Stéphane Dalmas, Marc Gaëtano, and Claude Huchet. A Deductive Database for Mathematical Formulas. In Jacques Calmet, editor, *Design and Implementation of Symbolic Computation Systems*, volume 1128 of *Lecture Notes in Computer Science*, pages 287–296. Spinger-Verlag, September 1996.

[DT93] D. Dori and K. Tombre. « Paper Drawings to 3D CAD : A Proposed Agenda ». In *Proceedings of Second International Conference on Document Analysis and Recognition (ICDAR)*, pages 866–869, Tsukuba, Japon, October 1993.

[EF95] T. H. Einwohner and Richard J. Fateman. « Searching Techniques for Integral Tables ». In ACM Press, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, Montréal, Canada, July 1995.

[Gol91] Eric J. Golin. « *A Method for the Specification and Parsing of Visual Languages* ». PhD Thesis, Brown University, May 1991.

[GS93] David Garlan and Mary Shaw. « An Introduction to Software Architecture ». In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Compagny, 1993.

[Hig93] Nicholas J. Higham. *Handbook of Writing for the Mathematical Sciences*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1993.

[HU69] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley Publishing Company, 1969.

[JL86] Jean-Pierre Jouannaud and Pierre Lescanne. « La Réécriture ». In *Techniques et Sciences Informatiques*, pages 433–452, 1986.

[KIA84] K. Kubota, O. Iwaki, and H. Arakawa. « Document Understanding System ». In *Proceedings of the 7th International Conference on Pattern Recognition (ICPR)*, volume 1, pages 612–614. IEEE Computer Society, August 1984.

[Knu68] Donald E. Knuth. « Semantics of Context-Free Languages ». *Journal of Mathematic and System Theory*, 2 :127–146, 1968.

[Knu79] Donald E. Knuth. « Mathematical Typography ». In *Bulletin of the American Mathematical Society*, volume 1, March 1979.

[Knu93] Donald Ervin Knuth. *The Tex Book*. Addison-Wesley Publishing Company, 1993.

[KS98] Norbert Kajler and Neil Soiffer. « A Survey of User Interfaces for Computer Algebra Systems ». *Journal of Symbolic Computation*, 25 :127–160, February 1998.

[Lam94] Leslie Lamport. *LaTeX : a document preparation system*. Addison-Wesley Publishing Company, 2 edition, 1994.

[Lav95] Stéphane Lavirotte. « Interface pour le Calcul Formel ». Technical Report, Université de Nice Sophia-Antipolis et INRIA Sophia-Antipolis, 1995.
http ://www.inria.fr/cafe/Stephane.Lavirotte/Publications/dea.ps.gz.

[Mar71] William A. Martin. « Computer Input/Output of Mathematical Expressions ». In *Proceedings of The Second Symposium on Symbolic and Algebraic Manipulation*, pages 78–87, 1971.

[MPW+90] Gale Martin, James Pittman, Kent Wittenburg, Richard Cohen, and Tom Parish. Computing without Keyboards. In *Byte*. McGraw-Hill, July 1990.

[NKMS93] Masaki Nakagawa, Naoki Kato, Kimiyoshi Machii, and Toshio Souya. « Principles of Pen Interface Design for Creative Work ». In *Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR)*, pages 718–721, Tsukuba, Japon, October 1993. IEEE Computer Society Press.

[Pre97] Adobe Press, editor. *Adobe FrameMaker 5.5 Classroom in a Book*. Adobe Press, October 1997.

[Res99] TCI Software Research. « Scientific Workplace », 1999.
http ://www.tcisoft.com/.

[Rus87] Michaël Rusinowitch. « *Démonstration Automatique par des Techniques de Réécriture* ». PhD Thesis, Université de Nancy, November 1987.

[Tom89] Masaru Tomita. « Parsing 2-Dimensional Language ». In M. Tomita, editor, *First International Workshop on Parsing Technologies (IWPT)*, pages 277–289, Pittsburgh, August 1989.

[Tom98] Karl Tombre. Analysis of Engineering Drawing : State of the Art and Challenges. In K. Tombre and A. K. Chhabra, editors, *Lecture Notes in Computer Science*, volume 1389, pages 257–264. Springer-Verlag, April 1998.

[Zwi96] Daniel Zwillinger, editor. *Standard Mathematical Tables and Formulae*. CRC Press, 30 edition, 1996.

Index

Index des termes français utilisés dans ce mémoire

ambiguïté
    notation, 25
    placement symboles, 24, 121
    relations spatiales, 24
    symbole, 23
analyse
    ascendante, 93
    contextuelle, 23
    de documents, 10, 18
    géométrique, 69, 71–76
    lexicale, 67–68
    structurelle, 93
    syntaxique, 27, 43–47
attribut, 57, 66, 68, 119
    hérité, 82
    synthétisé, 82, 85, 94, 121
Axiom, 141

base
    de connaissances, 47
    de formules, 4, 17, 141
bruit, 21, 30, 102

caractère
    manuscrit, 37
    reconnaissance, 20, 22
    typographié, 35
chaîne de Markov, 112
chaînes de Markov, 22, 38
connexe
    composante, 104, 105
    graphe, 71
ContruireGrapheInitial, 75, 77
Cuneiform, 100, 103

définition
    grammaire, 80–81
    grammaire de graphes, 85
    graphe, 70
    notation mathématique, 19, 52

édition
    bidimensionnelle, 15
    de formules, 4, 12
    manuscrite, 16
    syntaxe linéaire, 12
Emath, 15

filtrage de termes, 85
Fine Reader, 103
Frame Maker, 14

grammaire
    attribuée, 81
    de graphes, 5, 83, 86
    formelle, 80
Guile, 137, 139

image
    analyse, 63–66
    réalignement, 31
    traitement, 28–33
interface graphique, 4, 16, 136
Irma, 6, 114, 118, 145

Java Script, 138

Klm, 140
Klone, 137, 139

label, 141
langage
    de script, 130, 134, 139
    système, 130
LaTeX, 13, 56, 108

machine virtuelle, 132
Maple, 3, 13, 116, 145
Mathematica, 3, 116, 145
MiseAJourGraphe, 95
modèle objet, 133

numérisation, 28
    seuil, 29

OCR, 20, 104
    comparatif, 100
    résultats, 65
OCRchie, 100, 103, 104
OFR, 5, 53, 55, 61, 99, 129, 145
OmniPage, 2, 20, 100, 103
opérateur, 118
    arrangement spatial, 19
    grammaire d', 86
    implicite, 24
    précédence, 20, 25
OpenMath, 6, 57, 141–142

palette de modèles, 14
Perl, 138, 139
Python, 137, 139

réécriture, 46, 81, 83, 85
Recognita, 103
reconnaissance
    caractère
      manuscrit, 37
      typographié, 35
    structurelle, 39–50
    symbole, 22
Reduce, 116

scanner, 5, 21
ScanWorX, 100
Scientific Workplace, 14
segmentation, 22, 35
SetIntegerValue, 140
setq, 141
STk, 136, 139
substitution, 85
symbole
    ambiguïté, 23
    manuscrit, 37
    petit, 21, 113
    placement, 20, 121
    reconnaissance, 5, 20, 22, 28, 35
    style, 20, 22
    taille, 20, 22, 66
    typographié, 35
    variété, 22

Tcl/Tk, 137, 139
TextBridge, 2, 20, 100, 103
Typage, 133
TypeReader, 100

Visual Basic, 138

Xocr, 100, 103

Index

Index des termes anglais utilisés dans ce mémoire

Axiom, 141

base
    de connaissances, 47
byte-compile, 132

content dictionaries, 142
Cuneiform, 100, 103

Emath, 15

Fine Reader, 103
Frame Maker, 14
Free Software Foundation, 137

garbage collector, 134
Generic Window Manager, 139
graphical toolkit, 136
Guile, 137, 139

HMM, 22, 38, 112

Irma, 6, 114, 118, 145

Java Script, 138

keystroke-level model, 15
Klm, 140
Klone, 137, 139

LaTeX, 13, 56, 108

Maple, 3, 13, 116, 145
Mathematica, 3, 116, 145

nearest-neighbor method, 31
noise reduction, 30

OCR, 36, 104
OCRchie, 100, 103, 104
OFR, 5, 53, 55, 61, 99, 129, 145
OmniPage, 2, 20, 100, 103
OpenMath, 6, 57, 141–142

pattern matching, 85
Perl, 138, 139
profiling, 135
projection profile method, 31
Python, 137, 139

Recognita, 103
Reduce, 116

ScanWorX, 100
Scientific Workplace, 14
skew orientation, 31
STk, 136, 139

Tcl/Tk, 137, 139
TextBridge, 2, 20, 100, 103
thresholding, 29
TypeReader, 100

Visual Basic, 138

widget, 140

Xocr, 100, 103

Résumé

Reconnaissance structurelle de formules mathématiques typographiées et manuscrites

Le sujet de ce mémoire est l'étude et la réalisation d'un composant pour la reconnaissance structurelle des formules mathématiques typographiées et manuscrites. Ces travaux s'inscrivent dans une thématique plus large : l'analyse et la reconnaissance de documents.

La problématique générale que nous avons considérée peut se résumer de la manière suivante ; il s'agit d'identifier la structure, ou arbre de syntaxe abstraite, d'une formule à partir des données graphiques et géométriques (les symboles composant la notation et leur position). L'architecture logicielle retenue permet d'adapter très facilement le composant, baptisé *OFR* (*Reconnaissance Optique de Formules*), aux logiciels fournissant les symboles, ainsi qu'aux diverses notations mathématiques identifiées.

Pour effectuer cette reconnaissance structurelle, nous avons eu recours à une modélisation à base de graphes. Elle permet une abstraction des données receuillies et une transformation de ces informations par la définition d'une grammaire de graphes contextuelle attribuée, spécialement adaptée aux opérateurs mathématiques.

En nous appuyant sur des protocoles de communication d'objets mathématiques, comme OpenMath, nous pouvons envisager l'utilisation de l'interface développée autour d'*OFR* comme une alternative à la saisie des formules mathématiques.

Mots-clés: formules mathématiques, reconnaissance struturelle, grammaire de graphes.

Abstract

Structural Recognition of printed and handwritten mathematical formulae

This thesis describes the study and implementation of a component for structural recognition of handwritten or typesetted mathematical formulae. This work is related to document analysis and recognition fields of research.

Our aim could be resumed as : how to recognize the mathematical notation, i.e. the abstract syntax tree of a formula, just with graphical and geometrical informations (symbols and their position). The design of our software, *OFR* (*Optical Formula Recognition*) enables us to reuse the same prototype with different OCR systems and to adapt the recognition process to a large variety of mathematical notations.

To achive the structural recognition, we used a graph modelisation. This allows us to have all data in a synthetized format. We use an attributed contextual graph grammar for parsing graph, especially developped for mathematical operators.

Thanks to formulae exchange protocols like OpenMath, the interface developped with the help of *OFR* may be used as a front end for writing mathematical ans send them to a computer algebra system like Mathematica.

Keywords: mathematical formulae, structural recognition, graph grammar.