Cahiers GUTenberg

UN COMPILATEUR D'EXPRESSIONS MATHÉMATIQUES GÉNÉRANT DU MATHML

Benjamin Jennes, Raphaël Marée

Cahiers GUTenberg, nº 33-34 (1999), p. 183-190.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1999___33-34_183_0>

Un compilateur d'expressions mathématiques générant du MathML

Benjamin JENNES *

Rue des Charmes 14, B-4650 Herve, Belgique.
<jennesb@stud.montefiore.ulg.ac.be>

Raphaël MARÉE*

Rue du Vieux Chaffour 6, B-4460 Grâce-Hollogne, Belgique.
<maree@stud.montefiore.ulg.ac.be>

Résumé. Nous nous intéressons au problème de la publication d'expressions mathématiques pour le web et nous présentons une solution qui recourt aux techniques de compilation et au langage MathML. Après une description générale de l'approche utilisée, nous dépeignons les étapes de la compilation à partir d'un exemple et enfin nous discutons des avantages et limitations du programme `maje` que nous avons réalisé.

Abstract. *In this article we look at the problem of publishing mathematical expressions on the Web. We present a solution based upon the use of compiling techniques and the MathML language. After a general description of the approach, we describe the different stages of the compilation with the help of an example. We conclude with a discussion of the advantages and limitations of `maje`, the program we have implemented.*

Mot-clés : XML, HTML, MathML, `maje`.

1. Introduction

1.1. Expressions mathématiques et MathML

MathML (*Mathematical Markup Language*) est une application XML qui décrit les expressions mathématiques et qui permet de les publier sur le web, à la manière du HTML pour le texte. La définition de ce langage est définie par le *W3C Math Group* et est disponible sur le site du Consortium W3 à l'URL `http://www.w3.org/TR/REC-MathML`.

* Étudiants en première licence en informatique à l'Université de Liège (Belgique).

MathML présente de nombreux avantages par rapport aux moyens actuels pour publier des documents scientifiques sur le web. On peut citer la possibilité de réutilisation des informations mathématiques décrites en MathML dans d'autres applications, l'utilisation facile du copier/coller, la légèreté du code (par rapport aux images générées pour les formules par le convertisseur LaTeX2HTML par exemple). Les vertus de ce langage sont énoncées à l'URL `http://www.w3.org/TR/REC-MathML/chapter1.html`.

1.2. Principe et intérêt de la compilation

Un compilateur traduit un texte initial exprimé dans un langage, le *texte source*, en un texte exprimé dans un autre langage, le *texte cible*. Ce dernier conserve le sens initial du *texte source*. Le compilateur, qui est un programme, est lui aussi écrit dans un certain langage. La compilation est divisée en plusieurs phases. L'analyse lexicale consiste à récupérer les symboles terminaux à partir de la suite de caractères qui constituent le *texte source*. L'analyse syntaxique vérifie la syntaxe du *texte source* à partir de la définition de la grammaire du *langage source*. La génération du code produit la traduction du *texte source* dans le *langage cible*.

La compilation ne permet pas uniquement de traduire du code écrit dans un langage de haut niveau vers un langage machine compréhensible par un équipement donné. En effet, notre programme montre que la compilation peut être utilisée afin de développer des mini-langages concis pour générer des codes utiles.

2. Présentation générale de notre approche

Notre compilateur, `maje`, permet la publication d'expressions mathématiques sur le web. Le code de notre programme est écrit en C. Il a été réalisé dans le cadre du cours « Compilateurs et systèmes d'écriture des compilateurs » donné par P.-A. DE MARNEFFE à l'Université de Liège (Belgique).[1]

2.1. Langage source

Le *langage source* permet — à la date de rédaction de cet article — l'écriture d'expressions arithmétiques élémentaires (multiplication, addition, division, puissance, racine carrée) ainsi que l'écriture de quelques opérateurs d'analyse mathématique (intégrales, sommes et produits). Notre but est d'obtenir une représentation claire et précise des expressions mathématiques mais pas de vérifier la justesse de celles-ci.

1. Service d'Informatique (ingénierie du logiciel et algorithmique), Institut d'Électricité Montefiore (B28) Sart Tilman, B-4000 Liège, Belgique. <PA.deMarneffe@ulg.ac.be>

Ainsi, l'écriture d'une division par zéro, de la racine carrée d'un nombre négatif, ... sont permises puisqu'on s'intéresse uniquement à la présentation de ces formules.

2.1.1. Définition du langage source: sa grammaire

Une grammaire contient un ensemble fini de symboles terminaux, l'*alphabet*, des *classes syntaxiques*, des *productions* (qui indiquent comment une classe syntaxique peut être transformée en une suite de classes syntaxiques ou de symboles terminaux) et la *variable distinguée* (le symbole de départ, `<maje>` ici). La grammaire de notre *langage source* est décrite ci-dessous selon la notation BNF.

```
<maje> ::= { <suiteexpr> }
<suiteexpr> ::= <suiteexpr> ; <expr> | <expr>
<expr> ::=  <exprsimple> | <exprsimple> <oprelation> <exprsimple>
<exprsimple> ::= <terme> | <exprsimple> <opadditif> <terme>
<terme> ::= <facteur> | <terme> <opmultiplicatif> <facteur>
<facteur> ::= id | nombre | <appelprimitive> | ( <expr> )
<oprelation> ::= = | <> | < | <= | >= | >
<opadditif> ::= + | -
<opmultiplicatif> ::= * | /
<appelprimitive> ::= <integrale> | <somme> | <produit> | <fraction> |
                     <racine> | <puissance>
<integrale> ::= integral [ <expr> , <bornes> ]
<somme> ::= sum [ <expr> , <bornes> ]
<produit> ::= prod [ <expr> , <bornes> ]
<fraction> ::= frac \ <expr> , <expr> \
<puissance> ::= exp ^ <expr> , <expr> ^
<racine> ::= sqrt ( <expr> )
<bornes> ::= id : <expr> .. <expr>
```

Note : Un nombre est un chiffre ou une suite de chiffres, un `id` est un caractère alphabétique.

Nous verrons plus tard les étapes de la compilation d'un exemple dont nous pouvons déjà donner le *texte source* en respectant la grammaire définie ci-dessus, « La somme de i pour i allant de 0 à n » s'écrit : `{ sum [ i, i: 0..n ] }`.

2.2. Langage cible

Le *langage cible* est MathML. Ce langage comprend deux types de balises : les balises de présentation et les balises sémantiques. pour des raisons de disponibilité de logiciels reconnaissant MathML, `maje` exprime les expressions mathématiques selon les balises de présentation.

En effet, les programmes de balayage hypertexte actuels comme Netscape Navigator et Microsoft Internet Explorer ne permettent pas encore — à la date de rédaction de cet article — l'affichage d'expressions écrites en MathML. La version finale des spécifications de MathML 1.0 a été écrite en avril 1998, mais les demandes de la communauté scientifique semblent être moins importantes aux yeux des éditeurs de logiciels de balayage hypertexte que les demandes du « grand public ». On peut le comprendre. En attendant que ces applications reconnaissent MathML — nous invitons ici le lecteur à exprimer cette demande auprès des éditeurs des programmes de balayage hypertexte — on peut se rendre compte du résultat à l'aide du programme de balayage Amaya du W3C[2].

3. Exemples de compilation

3.1. Étapes de la compilation à partir d'un exemple simple

Afin d'illustrer les étapes générales de notre compilateur[3], nous présentons un exemple simple de compilation. Nous supposons vouloir générer le code MathML correspondant à la *candidate-phrase* « La somme de i pour i allant de 0 à n » qui se trouve dans le fichier `exemple.txt`. Elle est écrite en respectant la syntaxe de la grammaire du *langage source* vue précédemment, c'est-à-dire : `{ sum [ i, i: 0..n ] }`

Le programme `maje` peut être piloté à l'aide d'options spécifiées dans la ligne de commande. Nous nous contentons ici d'afficher le code MathML résultant de la compilation à l'écran, on tape donc la commande : `maje -f exemple.txt`.

3.1.1. Analyse lexicale

L'analyse lexicale consiste à examiner la séquence de caractères du *texte source* afin de déterminer les caractères que l'on peut regrouper comme étant des symboles terminaux. Dans notre cas, il s'agit de la suite : `{`, `sum`, `[`, `i`, `,`, `i`, `:`, `0`, `..`, `n`, `]` et `}`.

3.1.2. Analyse syntaxique

L'analyse syntaxique consiste à déterminer si une *candidate-phrase* est grammaticalement correcte et à exprimer cette *candidate-phrase* en une structure interne. La

2. Moins connu que Netscape par exemple, la version 1.4a de ce logiciel affiche les balises MathML de présentation uniquement.

3. Une description plus complète de la compilation est donnée à `http://www.stud.montefiore.ulg.ac.be/~maree/info/compil/index.html`

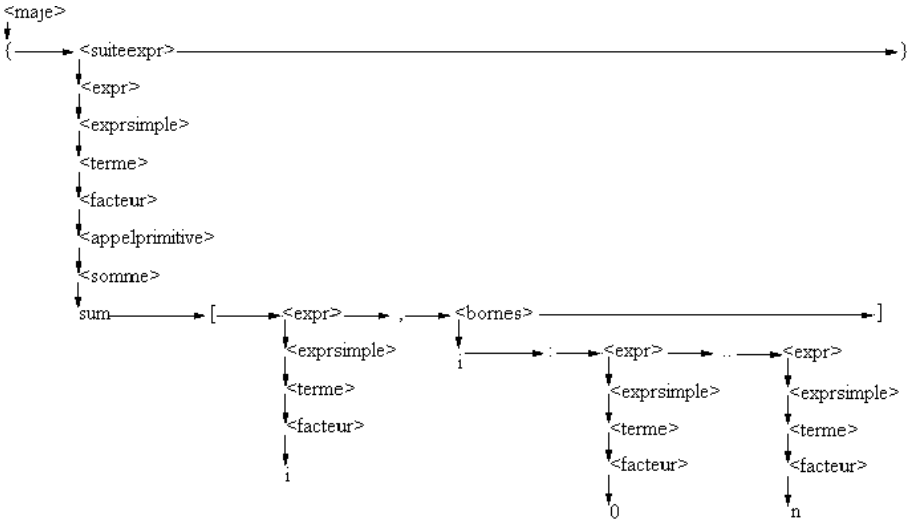

FIGURE 1 – *Représentation de l'arbre syntaxique.*

représentation interne de la structure d'une phrase est une forme d'arbre, l'*arbre syntaxique*. Les feuilles de l'arbre correspondent aux symboles terminaux de la grammaire, les nœuds internes aux classes syntaxiques. L'analyseur ascendant que nous avons réalisé construit l'arbre syntaxique à partir des feuilles de l'arbre vers la racine. Après vérification de la syntaxe, on obtient l'*arbre syntaxique* de la figure 1.

3.1.3. Génération du code

Le *texte cible* est généré en parcourant l'*arbre d'analyse* de manière récursive.

Le code MathML généré est embelli dans le sens où il est produit avec des retraits de ligne. Ainsi on distingue facilement dans le *texte cible* les expressions qui composaient le *texte source* ainsi que la structure du document MathML.

On obtient le code MathML suivant:

```
<math>
<mrow>
  <munderover>
    <mo> &Sum; </mo>
    <mrow>
      <mo> i </mo>
      <mo> = </mo>
      <mn> 0 </mn>
```

```
    </mrow>
    <mi> n </mi>
  </munderover>
  <mi> i </mi>
</mrow>
</math>
```

3.2. Un exemple plus complexe

Cet exemple permet d'apprécier la compacité et la lisibilité d'expressions relativement complexes exprimées dans le langage source. Nous indiquons le nombre de symboles dans le texte MathML généré par la compilation comparé à celui dans le texte source. Nous ne reproduisons pas le texte MathML généré parce qu'il est relativement long mais nous montrons le résultat affiché par le programme Amaya.

```
{ sum [ k , k : 0 .. n ] = frac \ n * ( n + 1 ) , 2 \ ;
  integral [ frac \ exp ^ x , 2 ^ , 1 + x \ , x : 0 .. 2 ] <> sqrt (x +
  exp ^ y , 3 ^ ) ;
  prod [ u , u : 1 .. j ] - 1 }
```

Le texte MathML généré est composé de 151 symboles, le texte exprimé dans le langage source en contenait 73. Une capture d'une fenêtre du programme Amaya permet de visualiser le résultat (voir la figure 2).

4. Discussion des avantages et limitations

4.1. Avantages

Les bienfaits de MathML pour la publication d'expressions mathématiques sur le web sont évidents et ont été énoncés précédemment.

Les exemples de compilation que nous avons donnés illustrent la concision de notre langage par rapport à la verbosité de MathML. Cet avantage de notre langage par rapport à MathML s'accentue lorsque l'on désire publier des expressions plus longues et plus complexes[4]. Aussi, nous pensons qu'une personne désirant publier des expressions mathématiques sur le web apprendra plus facilement notre langage, assez intuitif, que MathML.

4. De tels exemples peuvent être trouvés à `http://www.stud.montefiore.ulg.ac.be/~maree/info/compil/index.html`

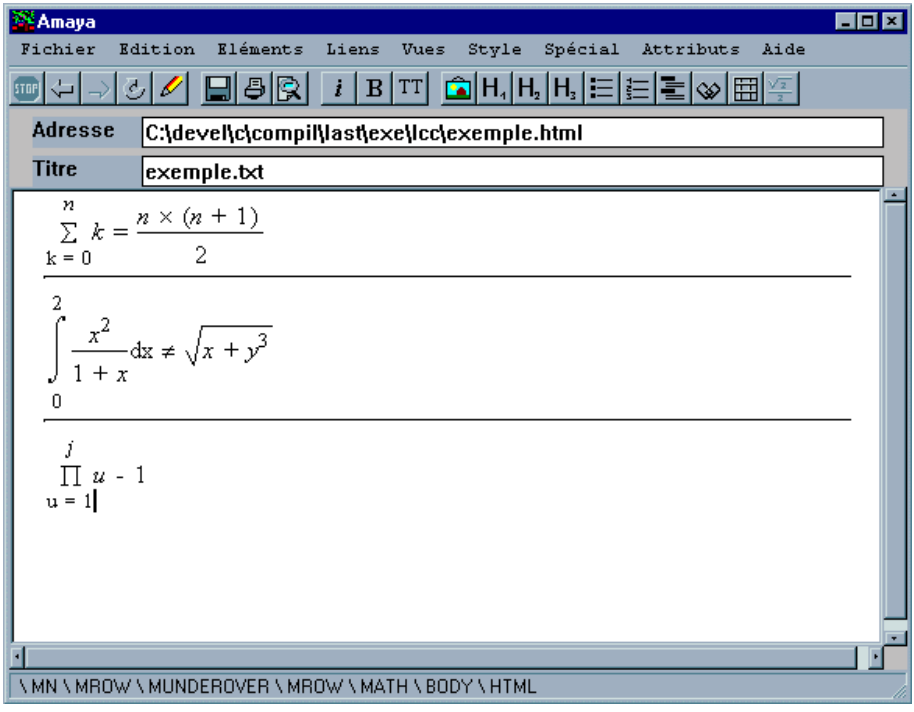

FIGURE 2 – *Résultat du second exemple affiché par Amaya*

4.2. Limitations

Les possibilités offertes par le *langage source* sont actuellement relativement limitées. Nous envisageons d'étendre — durant nos loisirs — la grammaire en permettant l'écriture d'un plus grand nombre d'opérateurs et d'éléments mathématiques (matrices, symboles de l'alphabet grec, ...). `maje` a été conçu dans cette optique d'extensibilité.

5. Disponibilités

Le compilateur `maje` est disponible sur le serveur web de l'Institut Montéfiore de l'Université de Liège à l'URL :

`http://www.stud.montefiore.ulg.ac.be/~maree/info/compil/index.html`

Il se présente sous la forme d'un exécutable pour Dos et Linux. D'autres versions sont envisageables. Le lecteur y trouvera également le présent article, le rapport détaillé accompagnant notre travail initial ainsi que divers exemples de compilation.

Le programme de balayage Amaya du W3C est disponible gratuitement pour de nombreux équipements à `http://www.w3.org/Amaya/`.

6. Remerciements

Nous tenons à remercier P.-A. DE MARNEFFE, promoteur du travail initial, et Raphaël FLEURY pour leurs commentaires et suggestions sur une version préliminaire de cet article.

Bibliographie

[1] P. Ion et R. Miner, *Mathematical Markup Language (MathML) 1.0 Specification*, W3C, 7 Avril 1998.
`http://www.w3.org/TR/REC-MathML/`.

[2] P.-A. de Marneffe, *Compilateurs et systèmes d'écriture des compilateurs*, Université de Liège, 1998.

[3] P.-A. de Marneffe, *Introduction à l'algorithmique 1*, Université de Liège, version 2.81, Septembre 1995.

[4] L. Ammeraal, *Algorithmes et structures de données en langage C* , InterÉditions/Masson, 1996.

[5] H. Zarb, *Compiler Writing Techniques* , University of Malta, 1996.
`http://www.cs.um.edu.mt/~hzarb/CSM201/main.html`.

[6] J. Ferber, *Cours de compilation* , Université de Montpellier II, 1997.
`http://www.lirmm.fr/~ferber/Compilation/compil.htm`.

[7] B. Jennes et R. Marée, rapport du travail de *Compilateurs et systèmes d'écriture des compilateurs*, Université de Liège, 1999.