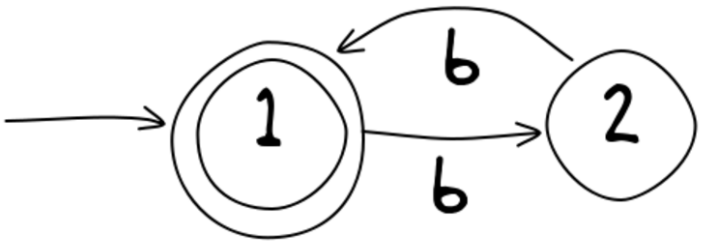
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE

PROJET D'ANALYSE LÉXICALE ET SYNTAXIQUE

AsciiMath to LaTeX



Diane LEBLANC-ALBAREL
Rand ASSWAD
Génie Mathématique

A l'attentien de :
M. Habib ABDULRAB

Table des matières

1 Introduction

Il existe beaucoup de langages et de logiciels d'édition de documents. Les plus utilisés sont ceux qui adhèrent au principe *What You See Is What You Get* **(WYSIWYG)** qui permettent à l'utilisateur d'apercevoir le docmuent lors de sa création. De tels outils ont été un facteur essentiel dans la démocratisation de l'ordinateur. Néanmoins, les logiciels WYSIWYG sont limités face à certains utilisateurs vis-à-vis de l'efficacité et de la qualité des résultats, notamment dans le domaine scientifique.

En effet, le langage LaTeX represente une alternative excellente pour l'édition des textes scientifiques ou ceux contenant du code informatique, ainsi que HTML pour l'édition des pages web. Néanmoins, l'utilisation de ces langages reste limité par soucis de simplicité et d'accéssiblité.

En revanche, il existe des logiciels/languages intermédiaires qui rendent LaTeX et HTML plus accéssibles. Nous nous sommes intéressés au langage **markdown** qui est simple à lire/écrire et peut être facilement interprété en LaTeX ou HTML par des programmes comme **pandoc**.

Markdown permet d'écrire des expressions mathématiques en format TeX, qui pourront être affichées correctement sur le document final. En revanche, les équations mathématiques en TeX sont rarement lisibles, ce qui est contradictoire avec l'objectif d'écrire les documents en markdown.

On a donc eu l'idée de développer un langage pour les expressions mathématiques où elles serons simples à lire et écrire. Nous avons tout de suite trouvé qu'un tel langage existe: **AsciiMath**. AsciiMath est un langage *markup* qui utilise dans sa syntaxe des symboles assez proches de leurs rendus respectifs (e.g. `oo` pour ∞).

Nous avons donc décidé de créer un interpréteur *AsciiMath en LaTeX*.

2 Grammaire

La première étape du dévelopement d'un tel outil est de définir la grammaire AsciiMath.

Soit la grammaire $G = (T, V, S, P)$ où:

- T: l'ensemble de symboles terminaux *(tokens)*
- V: l'ensemble de variable (symboles non-terminaux)
- S: l'axiome de la grammaire (la variable de départ)
- P: l'ensemble de règles de production

2.1 Les symboles terminaux

On groupe les symboles terminaux dans les catégories suivantes :

- **Constantes (C):** les lettres latins et grecques, les nombres, les noms des fonctions mathématiques, les symboles d'opérations mathématiques, etc.
- **Opérateurs unaires (U):** les opérateurs prenant un seul argument comme la racine carrée `sqrt`.
- **Opérateurs binaires (B):** les opérateurs prenant deux arguments comme la racine n$^{\text{ème}}$ `root` ou une fraction.
- **Délimiteurs droits (R):** parenthèse, accolade, crochet et chevron droits.
- **Délimiteurs gauches (L):** parenthèse, accolade, crochet et chevron gauches.

2.2 Les variables et l'axiome

Notre grammaire est définie par 3 variables:

- **Expression (e):** l'axiome de la grammaire
- Expression simple (s)
- Expression intermédiaire (i)

La grammaire peut s'écrire avec 2 variables mais elle est ambigüe.

2.3 Les règles

- e $\rightarrow$ ie + ε
- i $\rightarrow$ s + s_s + s^s + s_s^s + i/i
- s $\rightarrow$ C + LeR + Us + Bss

2.4 Notation Backus-Naur

- C ::= lettres latines | lettres grecques | nombres | etc
- U ::= sqrt | abs | floor | ceil | dot | etc
- B ::= frac | root | overset | etc
- L ::= (| [| { | (: | {: | «
- R ::=) |] | } | :) | :} | »
- <s> ::= C | L <e> R | U <s> | B <s> <s>
- <i> ::= <s> | <s> _ <s> | <s> ^ <s> | <s> _ <s> ^ <s> | <i> / <i>
- <e> ::= <i> <e> | <i>

3 Implémentation

3.1 Choix du langage

Une implémentation d'un interpréteur dépend principalement de deux outils:

- **Lexer (Tokenizer):** Un programme qui reconnait les symboles terminaux dans un texte donné. Le code définit les symboles de la grammaire.
- **Parser:** Un programme qui analyse le syntaxe d'un texte donné. Le code définit les règles de la grammaire.

On parle souvent de couple *Lexer/Parser*, les couples d'analyse syntaxique les plus établis sont lex/yacc et leurs équivalents libres flex/bison.

Nous ne sommes lancé sur l'implémentation sur flex/bison, nous avons implémenté plusieurs grammaires simples comme celle d'une calculatrice.

Or, pour notre grammaire qui est relativement compliquée, nous n'avons pas obtenue les résultats attendus, nous avons donc décidé d'utiliser **ANTLR4** qui est très efficace surtout pour débugger la grammaire.

3.2 Grammaire ANTLR

Le syntax ANTLR est très simple et intuitif. Les symboles terminaux commencent par une lettre majuscule et les variables commencent par une lettre miniscule.

Voici le fichier grammaire Antlr4.

```
cat grammar/AM.g4
```

```
grammar AM;
import AMTokens;

/* labeled tokens */

OVER    : '/' ;
SUB     : '_' ;
SUPER   : '^' ;

/* grammar rules */

e   : i e?              # append
    ;

i   : s                 # simple
    | s SUB s           # sub
    | s SUPER s         # super
    | s SUB s SUPER s   # subSup
    | i OVER i          # frac
    ;
```

```
s   : C                 # constParse
    | L e R             # parens
    | U s               # unary
    | B s s             # binary
    ;
```

3.3 Structure du programme

Voici l'arborescence du repértoire.

```
.
├── grammar
│   ├── AM.g4
│   ├── AMTokens.g4
│   ├── AMTokens.tokens
│   └── dictionary.csv
├── lib
│   └── antlr-4.7.2-complete.jar
├── src
│   └── AsciiMath
│       ├── antlr
│       │   ├── AMBaseVisitor.java
│       │   ├── AM.interp
│       │   ├── AMLexer.interp
│       │   ├── AMLexer.java
│       │   ├── AMLexer.tokens
│       │   ├── AMParser.java
│       │   ├── AM.tokens
│       │   └── AMVisitor.java
│       ├── tex
│       │   ├── Dictionary.java
│       │   └── Visitor.java
│       ├── IO.java
│       └── Translator.java
├── am2t
├── Makefile
└── README.md

6 directories, 20 files
```

La librairie antlr4 (fichier `lib/antlr-4.x.x-complete.jar`) permet de générer le paquet java **AsciiMath.antlr** à partir de notre grammaire

3.4 Traducteur

Nous avons donc créé le paquet java **AsciiMath.tex** qui traduit en LaTeX.

La libraire antlr4 fournit une interface d'arbre syntaxique `ParseTree`, on implémente donc un *visiteur* de l'arbre qui interprète dans chaque nœud les symboles reconnus. Il s'agit de la classe `Visitor` qui est une sous-classe de `AMBaseVisitor` générée par antlr.

Afin de traduire les symboles reconnus, nous avons écris un *dictionnaire* en format CSV indiquant la traduction d'un symbole AsciiMath en LaTeX. Nous avons défini une classe `Dictionary` qui sert d'une interface de lecture du fichier fournis, en l'extrayant dans un `HashMap`.

3.5 Interface

Les classes `IO` et `Translator` servent d'interface d'utilisation du programme. Elle permettent de traduire un fichier AsciiMath en LaTeX et de générer une image d'arbre syntaxique du texte analysé.

Nous fournissons un `Makefile` permettant de compiler le code facilement ainsi que le script `am2t` permettant d'executer le programme.

```
./am2t --help
```

```
usage: am2t [-h] [-i INPUT] [-o OUTPUT] [-no] [-t TREEFILE] [-nt]

AsciiMath to LaTeX convertor tool

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input AsciiMath file
  -o OUTPUT, --output OUTPUT
                        Output file name (default: input.katex)
  -no, --nooutput       Suppress katex file output
  -t TREEFILE, --treefile TREEFILE
                        Output parse tree file name (default: input.png)
  -nt, --notree         Suppress parse tree image output
```

4 Résultats

```
echo "e^x = lim_(n->oo) (1+x/n)^n" > test/exp.txt
./am2t -i test/exp.txt -o test/exp.katex -t test/exp.png
cat test/exp.katex
```

```
e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n
```

$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

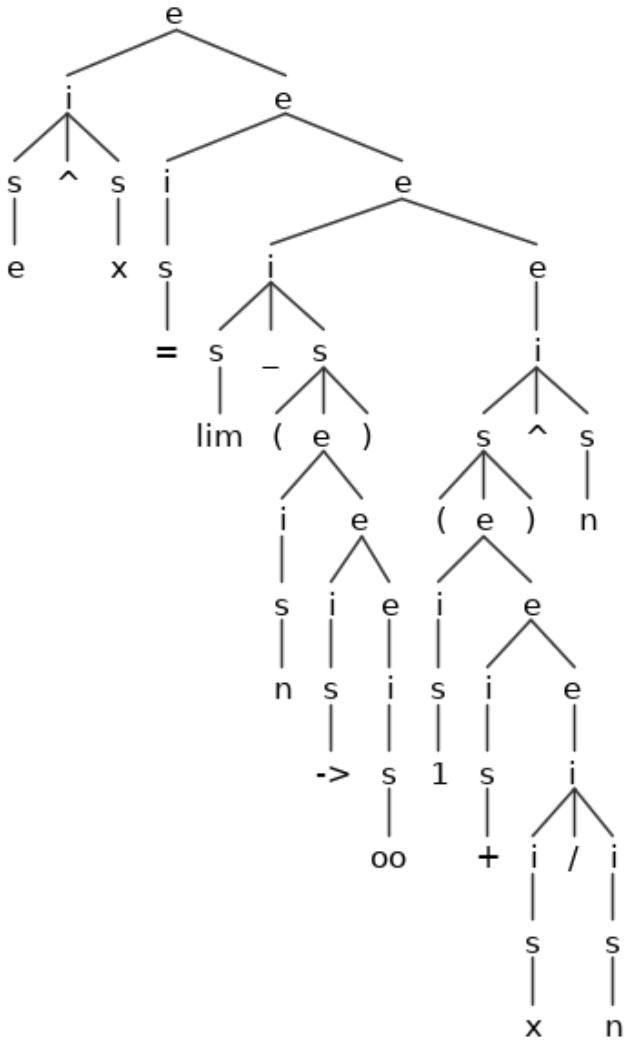


FIGURE 1 – Arbre syntaxique obtenue pour `exp.txt`

Essayons avec un autre exemple:

```
echo "ln(x) = int_1^x 1/t dt" > test/ln.txt
./am2t -i test/ln.txt
cat test/ln.katex
```

```
\ln \left(x\right) = \int_1^x \frac{1}{t} \mathrm{d}t
```

$$\ln\left(x\right) = \int_1^x \frac{1}{t} \mathrm{d}t$$

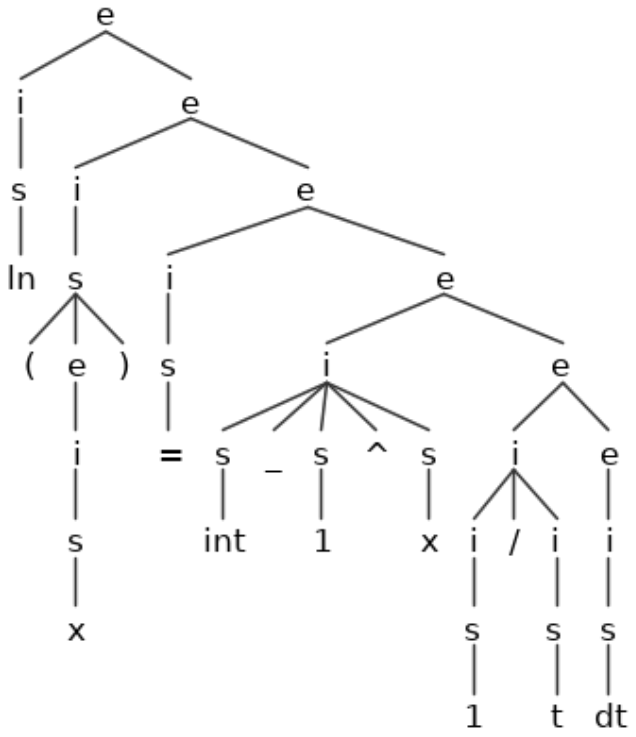


FIGURE 2 – Arbre syntaxique obtenue pour `ln.txt`

Essayons avec un autre exemple:

```
cat test/myth.txt
```

```
sum_(n in NN) n != -1/12
```

```
./am2t -i test/myth.txt
cat test/myth.katex
```

```
\sum_{n \in \mathbb{N}} n \ne - \frac{1}{12}
```

$$\sum_{n \in \mathbb{N}} n \ne - \frac{1}{12}$$

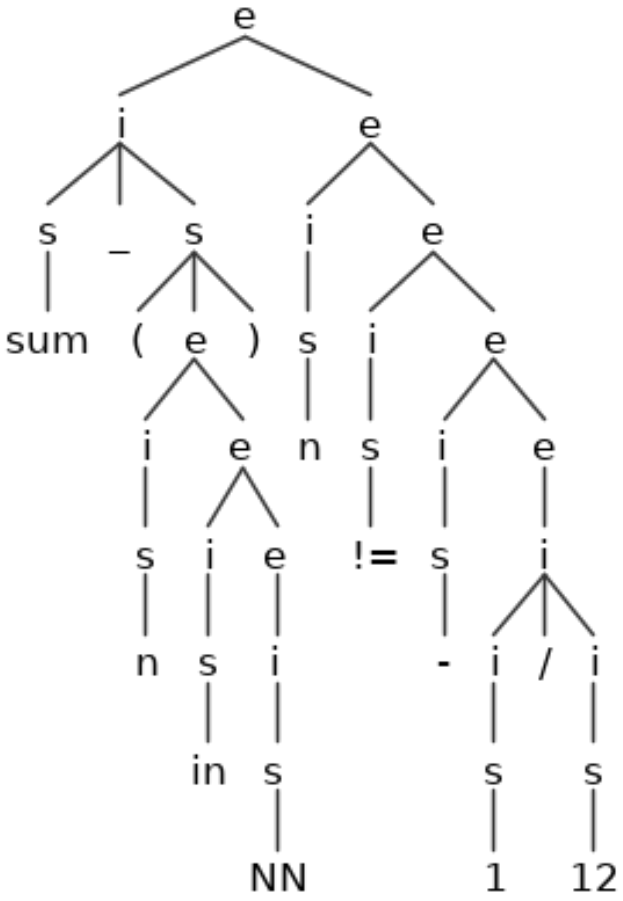


FIGURE 3 – Arbre syntaxique obtenue pour `myth.txt`

Prenons maintenant un exemple plus complexe

```
cat test/poisson.txt
```

```
I = int_RR e^(-x^2) dx =>
I^2 = int_(RR^2) e^(-x^2-y^2) dx dy
= int_0^(2pi) int_0^oo r e^(-r^2) dr d theta
= [theta]_0^(2pi) * [(-e^(-r^2))/2]_0^oo
= 2pi [0 + e^0/2] = pi
=> I = sqrt(pi)
```

Voici ce que ça donne en LaTeX.

```
./am2t -i test/poisson.txt
cat test/poisson.katex
```

```
I = \int_\mathbb{R} e^{- x^2} \mathrm{d}x \Rightarrow I^2 = \int_{\mathbb{R}^2} e^{- x^2 - y^2} \mathr
```

$$I = \int_{\mathbb{R}} e^{-x^2}\mathrm{d}x \Rightarrow I^2 = \int_{\mathbb{R}^2} e^{-x^2-y^2}\mathrm{d}x\mathrm{d}y = \int_0^{2\pi}\int_0^{\infty} re^{-r^2}drd\theta = [\theta]_0^{2\pi}\cdot\left[\frac{-e^{-r^2}}{2}\right]_0^{\infty} = 2\pi\left[0+\frac{e^0}{2}\right] = \pi \Rightarrow I = \sqrt{\pi}$$