

Computer Processing of Nemeth Braille Math Notation¹

H. Guo, G. Gupta, J. Mendez, C. Weaver, A. Karshmer, S. Geiger

*The MAVIS Group
New Mexico State University
Las Cruces, NM 88003*

1 Introduction

1.1 The Problem: Converting Nemeth Math to \LaTeX

Nemeth Braille Code is a Braille-based notation to enable visually impaired scholars and students to read and write Mathematics. It was designed by Dr. Abraham Nemeth in the late 1940s to allow visually impaired individuals to convey mathematical and technical information. While Nemeth Braille code is well suited for humans, it is not easy to process with a computer. To make Nemeth Math code computer processable, we should be able to *parse* Nemeth code text for grammatical correctness (i.e., check that it conforms to the rules of syntax laid out in the Nemeth code specification [5]), and once parsed, transform it into a form so that it can be processed further. To the best of our knowledge there are no parsers available that will parse Nemeth code to check its grammatical correctness. The principal reason for this is that the Nemeth code notation falls in the class of languages known as *context sensitive languages* [7]. Context sensitivity means that the grammatical correctness of a subexpression written in Nemeth code depends on the context in the expression it occurs in. This is similar to the case of English, where, for example, to check the grammatical correctness of the sentence one has to make sure that a singular subject noun must have a singular verb. So *The man eats an apple* is grammatically correct, while *The man eat an apple* is not. Another negative point about the Nemeth code is that it is not an *LALR(k)* [7] language. This means that if we were to write grammar rules for Nemeth code, then given a Nemeth code expression that we are trying to parse, we cannot tell by looking ahead k symbols in the expression, which grammar rule should we apply to parse the expression. Thus, traditional parsing and compiling technology [7], that is normally used to process computer languages, cannot be used. In fact, traditional parsing and compiler technology has been developed only for *LALR(1)* languages. It should be noted that properties of the Nemeth code—especially context sensitivity—are precisely the reason why Nemeth code is so suitable for a visually impaired person. Since a blind person reads by touching the Braille code, it is important that he/she be aware of the context of the current symbol or expression he is currently looking at, at any given time. Thus, building-in the context into the language is quite helpful for the blind reader, but it make computer processing quite difficult as the language becomes context sensitive and non-LALR(k). Also, at the time the Nemeth notation was designed (late 40s and early 50s), the computer was just being invented. Thus, computer readability or computer processability was not one of the design criteria for Nemeth Math code. Additionally, very little was known about structure of languages then (the seminal work of Chomsky on structure of languages appeared only in the early 50s).

In this paper we present a technique based on *denotational semantics* and *logic programming* that allows us to build a parser and transform Nemeth code into an intermediate form that can then be used for generating equivalent \LaTeX code or MathML code, or whatever one wishes. Our system has been developed as part of the MAVIS project (Mathematics Accessible to Visually Impaired Students) at New Mexico State University. The goal that lead to this research was to design two systems, one to automatically translate \LaTeX to Nemeth Braille code and the other to automatically translate Nemeth code to \LaTeX in order to facilitate the communication between a blind student and a sighted instructor. Thus, we envisage the following scenario: an instructor prepares homework using \LaTeX , which is automatically converted to Nemeth code, using the \LaTeX to Nemeth code translator, so a blind student can read it. The blind student prepares his/her answer in Nemeth code, which is then automatically translated to \LaTeX so the instructor can grade it.

Building the \LaTeX to Nemeth code translator was a relatively easy task, since \LaTeX 's mathematical expressions are written using a simple, context-free notation. However, building a Nemeth code to \LaTeX translator is difficult due to the problems of context sensitivity and non-LALR(k) properties of Nemeth

¹This work has been supported by NSF HRD 9800209 and INT 9904063.

Math code. However, by using a combination of innovative semantics-based techniques we have been able to develop such a translator for translating Nemeth code to \LaTeX . As part of the MAVIS project a complete translator has been developed. The translator we have developed as part of the MAVIS project is, to the best of our knowledge, the first of its kind ever developed.

It should be noted that the translator from Nemeth to Braille is part of a larger system, which also includes an audio equation browser for the \LaTeX output [1]. Essentially, the \LaTeX expression generated is graphically displayed, and the audio equation browser also reads it out using sound. The user has the ability to press keys and go deeper into the subexpressions, and to re-hear subexpressions and expressions. The audio browser is still under development.

Our research makes several contributions: It has produced a system that solves a problem that was considered to be unsolvable [6]. It also develops a general methodology that can be used for processing other similar codes for marking up mathematics (e.g., those used in Europe).

2 A Semantics based Approach to Translation

The translation filter that we have developed for translating Nemeth Math Braille expressions to \LaTeX is based on a *semantics-based* approach. In this approach, the semantics of Nemeth Math Braille expressions is specified in terms of \LaTeX parse trees using a *denotational semantics* based approach. Moreover, this semantics is specified using a *logic programming* notation, rendering it executable on a computer. This executable specification automatically yields the needed translator.

Logic programming is a new programming paradigm that has emerged in the last 25 years. The best known representative of this new class of programming languages is *Prolog*, which originated from ideas of Kowalski [8]. Programming in Prolog differs from conventional programming both stylistically and computationally, as it uses logic to declaratively state problems and deduction to solve them.

In the *logic programming* philosophy, an algorithm consists of two disjoint components, the logic and the control [9]. The logic is the statement of *what* the problem is that has to be solved. The control is the statement of *how* it is to be solved. The programmer should only specify the logic component of an algorithm. The control should be executed solely by the logic programming system. Since the denotational semantics based approach only needs the syntax and semantics specifications, Logic Programming is well suited to realize these specifications regardless of controls.

Logic Programming languages, specifically *Prolog*, have turned out to be ideal as prototyping and application development languages. *Logic Programming* helps to specify the syntax and semantics easily, and develops an executable application at the same time. Given that the syntax of a language L_1 , it can be expressed as a *Definite Clause Grammar (DCG)*. This executable DCG specification can yield a syntax parser for L_1 . Semantics specification can also be easily implemented by appropriate valuation predicates, which are essentially maps from syntactic structure of L_1 to its semantic values, the sentence with language L_2 .

Denotational Semantics [3] is a well-established methodology for design and analysis of computer languages. In the denotational approach, the semantics or meaning of a computer language/notation is specified in terms of mathematical objects (such as sets and functions). A denotational specification has three components: *syntax specification* which maps sentences of the language to parse trees; *semantic algebras* that specify these mathematical objects, and *valuation functions* that map parse trees to semantic algebras. It turns out [4] that the syntax specification, the semantic algebra, and the valuation functions can be specified using logic programming [2]. This logic programming specification is also executable. To obtain a translator from a language \mathcal{L}_1 to \mathcal{L}_2 we choose the semantic algebra consisting of parse trees of \mathcal{L}_2 . The logic programming specification then automatically turns into a translator which can be executed to translate sentences of language \mathcal{L}_1 to sentences of language \mathcal{L}_2 .

In the MAVIS project the language \mathcal{L}_1 is Nemeth Math code, and the language \mathcal{L}_2 is \LaTeX . Syntax and Semantics of Nemeth Math code is given in terms of parse trees of \LaTeX using the denotational approach. These specifications are written using logic programming, and thus when executed, produce a translator from \mathcal{L}_1 to \mathcal{L}_2 . The syntax and semantic specification covers the whole Nemeth Math code, except for a few symbols that are found in Nemeth Math code, but do not have a counterpart in \LaTeX . There is error recovery built in for some of the constructs, if the user types Nemeth code that is not well-formed. Examples of such error recovery includes missing grouping indicator (e.g., missing end-fraction symbol, missing parenthesis, etc.).

It should be noted that the translation filter generated by *Logic Programming* is totally based on the

syntax and semantics specifications of the source language. A programmer does not need to worry about how the filter actually does the translation, all he needs to do is to provide the syntactic and semantic specification in terms of logic programming. The actual translation will be automatically obtained by running the specification on a logic programming system. Thus, *Logic programming* makes the specification executable. This also provides the possibility to prove the correctness of the filter [10]. In our approach syntax specification is based on *Definite Clause Grammar* which can be used to check the syntax, as well as generate the parse trees. Semantics specification based on valuation predicates can be proved to be correct by mathematical induction because the semantics specification is defined recursively .

Obviously, there is an intrinsic relationship between language translation and Logic Programming. Historically speaking, language processing is one of the two roots from which *Logic Programming* grew, while automatic theorem proving is the other.

3 Nemeth Braille Math Notation

The Nemeth Braille Code (heretofore referred to as the Nemeth code) for Mathematics and Science Notation has been prepared to provide a system of symbols which will allow technical literature to be prepared and read in braille [5]. A wealth of technical materials are written using Nemeth code to meet the requirements of blind students at all levels of educational pursuits. Braille Code consists of an arrangement of one to six dots in a space two dots wide and three dots high. A blind person can read the Braille based paper by touching the raised dots embossed in paper. One of the greatest challenges to the visually impaired student in science and mathematics disciplines is the reading and writing of complex mathematical equations or have convenient access to information based tools such as the world wide web [11]. Given a technical document written in Nemeth code, a blind student can read and understand the technical literature very well. Also, they can use Nemeth code to write technical documents. Thus, the Nemeth Code enables a blind person to communicate with his teacher, his colleagues, and his associates, in the realm of Science, Mathematics, Engineering and Technology.

While the Nemeth Math code allows a blind person to write technical document, they can only be read by other blind people, but not by sighted people, who are not Braille Nemeth literate. A solution for bridging the communication gap is to build an automatic translation filter between Nemeth and \LaTeX . An obvious problem we encounter then is the following: how do we make a computer process the Nemeth Braille code? A mapping chart about ASCII-Braille Cell Correspondence is available to help the computer to understand the Nemeth Codes. Based on this chart, one can convert the Nemeth Codes to the corresponding ASCII text, and then feed this ASCII text to the computer. This is anyway done by most typing devices available for typing Braille (e.g., Braillelite [5]). After processing the ASCII text, the computer can translate the results back into the Nemeth Codes and print them on an embosser. The translation between ASCII and Braille Cell is implemented by special hardware devices (Braille typewriter and Embossers).

4 \LaTeX

Generally, there are two standards applied to choose the target language:

- When translating a program from one language to another, it must be guaranteed that the target program is semantically equivalent to or the source one. That is to say, the semantics which the target language can represent should be greater than or at least equivalent to that of the source language.
- The target language selected should be popularly used in practice so that the translation filter can maximize its utility.

\LaTeX is a document preparation system for high-quality typesetting. It is most popularly used for medium-to-large technical or scientific documents, and has been adopted by many authors and publishers who generate technical books and papers. Based on the above two standards, we chose \LaTeX as the target notation of the translation filter from Nemeth because \LaTeX has the following advantages:

- Most Sighted mathematics instructors write mathematical literature using \LaTeX as a markup language.
- \LaTeX provides a complete list of symbols and methods to present mathematical expressions, even very complex mathematical formulas.

- \LaTeX is a very well-structured language. Its Context-Free grammar can be easily written, which means that \LaTeX is well suited to be processed by computer.

5 Difficulties in Translating Nemeth to \LaTeX

The Nemeth Braille Code for Mathematics was designed during the early 50s and its designer did not even anticipate that one day it may be processed by a computer. The Nemeth notation was primarily designed to transcribe printed mathematics in Braille in a way that relative spatial placement of symbols is preserved [5]. The notation was also designed without any regard of computer processibility. Not very surprising, since the study of syntactic structure of languages was just beginning.

The Nemeth Braille Code for Mathematics is still under modification. From the first publication of Nemeth Code, it has been updated and refined several times to assure the faithful transference from ink print to braille. Problems in interpretation and clarity are still encountered sometimes when the Code is put into actual use. Still the representation of some of the mathematical expressions in Nemeth Code is very complicated and cumbersome, when it comes to automatic processing using a computer.

The first step in implementing a translation filter from Nemeth to \LaTeX is to specify the syntax of Nemeth. In order to design the syntactic specification of a programming language, usually we use the BNF (Backus-Naur Form) notation, which is powerful enough to express context-free grammar [12, 7]. However, the Nemeth Notation contains a lot of rules that put it in the class of formal languages called context-sensitive (non-context-free) languages. This makes the grammar of Nemeth very complex, so that its syntactic specification turns out to be difficult. There are at least three features of the Nemeth code that exhibit context-sensitive behavior:

Context Sensitive Consider the encoding of superscripts and subscripts. A sign which is elevated relative to the base line of mathematical expression is called a superscript; one which is depressed relative to the base line is called a subscript. Superscripts or subscripts may carry superscripts or subscripts of their own, which form the hierarchy of superscripts and subscripts. In Nemeth Code, a base-line indicator identifies the symbols which follow it as representing signs on the base line; a level indicator identifies the symbols which follow it as representing superscript or subscript, and different order superscript or subscript are identified by different level indicators [5]. For instance, the mathematical expression $x^a + b$ is represented by the ASCII Nemeth code as $x^{\wedge}a + b$. However, another mathematical expression $y^{x^{a+b}}$ is represented by ASCII Nemeth code as $y^{\wedge}x^{\wedge\wedge}a^{\wedge} + b$, where \wedge is superscript indicator, $\wedge\wedge$ means second order superscript indicator, and '' is the base line indicator. It is easy to find out that the same sub-expression x^{a+b} is represented as $x^{\wedge\wedge}a^{\wedge} + b$ rather than $x^{\wedge}a^{\wedge} + b$ because of the context environment of superscripts. Since the information about the context environment is also encoded in the Nemeth Code, this context information has to be analyzed during the parsing procedure so that the correct syntax structure will be generated.

Space Sensitive Blank space has an important role in the rules of Nemeth Notation. In some cases spaces are required to represent specific expressions or symbols, while in other cases spaces are forbidden, otherwise the expressions may cause ambiguity. Consider the spacing with symbols of operation in Nemeth Notation. one and only one space must be left on either side of an operation symbol under the circumstances listed below [5]:

- On the both sides of a comparison symbol, for example: $x = y$ is represented by ASCII Nemeth Notation code as $x .k y$ instead of $x.ky$.
- After a function name or its abbreviation, for example: $\sin x$ is represented by ASCII Nemeth Notation code as $\sin x$.

On the other hand, a space must not be left on either side of a symbol of operation in some other situation [5]:

- $a + b$ is represented by ASCII Nemeth Notation as $a+b$ instead of $a + b$ or $a+ b$.
- $12 \div 3$ is represented by ASCII Nemeth Notation as $?12./3$ instead of $12 ./ 3$ or $12 ./3$.

Spatial Arrangements In Nemeth Notation, there are spatial arrangements for addition, subtraction, multiplication, division, determinants and matrices, which will cover more than one lines input. Furthermore, These spatial arrangements have strict format requirement. Consider determinants or matrices, each entry must be left-justified (moved as far to the left as possible) in the column to which it belongs, and top-justified (moved as far up as possible) in the row to which it applies [5]. This spatial information is hard to faithfully represent as part of a grammar.

6 Denotational Semantics Based Approach

The *denotational semantics* based approach consists of specifying both syntax and semantics of the Nemeth code. The syntax specification specifies the syntactic structures of the mathematical expressions (expressed in ASCII Nemeth Code) according to its grammar, while the semantics specification defines semantic valuation mappings from the syntactic structures to corresponding semantic values, namely, the corresponding \LaTeX expression. The implementation of the translation filter from Nemeth to \LaTeX exactly follows the theory of denotational semantics.

With *Logic Programming (LP)*, the syntax of the ASCII Nemeth can be specified by Definite Clause Grammars (DCG) even though the Nemeth Code belongs to context sensitive formal language. DCG provides some nice methods such as argument passing, which will be discussed in next chapter, to obtain a parser to handle context sensitive grammars. For instance, a parser for the language $\{a^n b^n c^n | n \geq 0\}$ can be easily built using a DCG grammar described as the figure 1 even though its language is proved to be not context free [13]. Given a grammar written as a DCG, the Prolog interpreter interprets this DCG specification as a logic program which serves as a parser for that grammar. Also, the backtracking capabilities of logic programming help a great deal. Nemeth Math grammar is not LALR(1). In fact, arbitrary number of look aheads may be required to find out which grammar rule will apply in a given situation. Given a logic programming based system, if the rule is guessed wrongly, then there is always the possibility that when the error is discovered we can backtrack and apply another rule.

```

abc --> a(V), b(V), c(V).

a([]) --> [].
a([a | V]) --> [a], a(V).

b([]) --> [].
b([a | V]) --> [b], b(V).

c([]) --> [].
c([a | V]) --> [c], c(V).

```

Figure 1: The DCG for parsing $\{a^n b^n c^n | n \geq 0\}$

In the approach we take, we build a context free grammar (but not LALR(1)) for Nemeth code. The context sensitive aspects are handled in the semantics phase, much in the same manner as in parsing of imperative languages where context sensitive features (e.g., number of actual parameters match the number of formal parameters, or a variable is declared before it is used) are also handled in the semantic phase. However, this grammar is not LALR(1), therefore, the traditional approach could not have been used, and a logic programming approach based on DCGs is needed.

Semantics specification can be implemented by appropriate valuation predicates, which essentially map the syntax structure (parse-trees) of Nemeth Code to its semantic value expressed in terms of syntactic structures (parse-trees) of \LaTeX . The valuation predicates, defined recursively based on the syntax structure of Nemeth Code, not only specify the semantics of the Nemeth code, but also form an executable application, that can be executed using a logic programming system.

Therefore, *Logic Programming* and *Denotational Semantics* make the translation filter from the ASCII Nemeth Code to \LaTeX both feasible and easy to implement. All that we need to do is to specify the syntactic and semantic specification, and then execute them using a logic programming system.

References

- [1] A. Karshmer, G. Gupta, S. Geiger, C. Weaver. A Framework for Translation of Braille Nemeth Math to \LaTeX : The MAVIS Project. In *Proc. ACM Conference on Assistive Technologies*, ACM Press, pp. 136-143, Mar. 1998.
- [2] L. Sterling & S. Shapiro. *The Art of Prolog*. MIT Press, '94.
- [3] D. Schmidt. *Denotational Semantics: a Methodology for Language Development*. W.C. Brown Publishers, 1986.
- [4] G. Gupta. Horn Logic Denotations and Their Applications. In *The Logic Programming Paradigm: A 25 year perspective*. Springer Verlag. pp. 127-160. May 1999.
- [5] A. Nemeth. *The Nemeth Braille Code for Mathematics and Science Notation 1972 Revision* (Frankfort KY: American Printing House for the Blind, 1972)
- [6] K. Miesenberger, B. Stöger. Personal Communication.
- [7] A. Aho, J.D. Ullman, R. Sethi. *Compilers: Principles, Techniques, and Tools*. Addison Wesley. 1986.
- [8] R.A. Kowalski. Predicate logic as a programming language. In *IFIP 74*, pages 569 – 574, 1974.
- [9] John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
- [10] Flemming Nielson and Hanne Riis Nielson. Two-level semantics and code generation. *Theoretical Computer Science*, 56:59–133, 1988.
- [11] A. Karshmer, G. Gupta, S. Geiger, and C. Weaver. Reading and writing mathematics: The mavis project. *Behavior and Information Technology*, 18(1):2–10, 1998.
- [12] Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1976.
- [13] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley Longman, 1990.