Verification of Mathematical Formulae Based on a Combination of Context-Free Grammar and Tree Grammar

Akio Fujiyoshi[1], Masakazu Suzuki[2], and Seiichi Uchida[3]

[1] Department of Computer and Information Sciences, Ibaraki University
fujiyosi@mx.ibaraki.ac.jp
[2] Faculty of Mathematics, Kyushu University
suzuki@math.kyushu-u.ac.jp
[3] Faculty of Information Science and Electrical Engineering, Kyushu University
uchida@is.kyushu-u.ac.jp

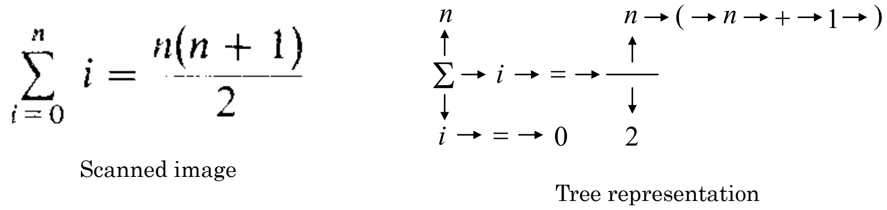
Abstract. This paper proposes the use of a formal grammar for the verification of mathematical formulae for a practical mathematical OCR system. Like a C compiler detecting syntax errors in a source file, we want to have a verification mechanism to find errors in the output of mathematical OCR. Linear monadic context-free tree grammar (LM-CFTG) was employed as a formal framework to define "well-formed" mathematical formulae. For the purpose of practical evaluation, a verification system for mathematical OCR was developed, and the effectiveness of the system was demonstrated by using the ground-truthed mathematical document database INFTY CDB-1.

1 Introduction

Grammatical analysis is useful for many types of verification problems. For example, a C compiler grammatically analyzes a source file and returns error messages with the location and type of errors. For mathematical OCR [1], it is natural to think that such grammatical analysis helps to detect misrecognitions of characters and structures in mathematical formulae. This paper proposes a mathematical-formulae verification method for a practical mathematical OCR system based on a combination of context-free grammar [2] and tree grammar [3].

Grammatical analysis can be classified into two levels: syntactic analysis and semantic analysis. This paper concentrates only on the syntactic analysis of mathematical formulae because we wanted to build a very fast verification system. Needless to say, semantic analysis is also very important for the improvement of mathematical OCR. However, we will leave this task for another time. We use the term "well-formed" to mean syntactic correctness. Since syntactic correctness doesn't necessarily mean semantic correctness, we can consider unsatisfiable formulae, e.g., "$1+2=5$", and tautological formulae, e.g., "$x=x$", as "well-formed" formulae if they are syntactically correct.

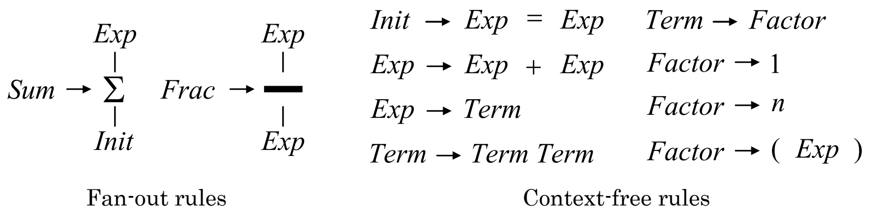
The final aim of this study is to completely define "well-formed" mathematical formulae. In other words, we want to have a grammar to verify any



Scanned image

Tree representation

Fig. 1. A result of structural analysis of a mathematical formula



Fan-out rules

Context-free rules

Fig. 2. Some rules of the grammar

mathematical formula that has appeared, or will appear, in a long-term build-up of mathematical documents. There were other grammatical approaches to the verification of mathematical formulae such as [4–6]. The proposed verification method will extend the coverage of those approaches.

In order to define "well-formed" mathematical formulae, we employed linear monadic context-free tree grammar (LM-CFTG) [3] as a formal framework. As shown in **Fig. 1**, a mathematical OCR system offers a tree representation of a mathematical formula from a scanned image. Therefore, we needed a grammar formalism to define a set of tree structures. An LM-CFTG defines a set of tree structures by arranging fan-out rules and context-free rules, where fan-out rules are used to describe the structural growth of a tree, and context-free rules are used to describe linear growth. For example, some fan-out rules and context-free rules of the grammar defining "well-formed" mathematical formulae are illustrated in **Fig. 2**.

The proposed verification method allows us to build a very fast verification system. Theoretically, the verification process of most mathematical formulae will be completed in linear time depending on the size of the input, though some exceptional mathematical formulae require cubic time. We need a very fast verification system because verification should be done for numerous recognition candidates to improve the reliability of mathematical OCR. We experimentally built a verification system and executed the system on the ground-truthed mathematical document database INFTY CDB-1 [7]. The verification of $21,967$ mathematical formulae (size: 48.1MB) was finished within 10 seconds by a PC (CPU: Pentium4 3.06GHz, RAM: 1GB).

The accomplishment of this very fast verification system mainly resulted from the following two features of the proposed verification method:

- Division of a mathematical formula into sub-formulae; and
- A grammar formalism with a fast recognition algorithm.

The idea of the division of a formula into sub-formulae is common to well-known algorithm design paradigms such as "Divide and Conquer" and "Dynamic Programming." The employment of LM-CFTG enables us to use not only parsing algorithms for LM-CFTG [8] but also well-established parsing techniques for context-free grammar (CFG) [2, 9].

Although the proposed verification method may be useful in general, this paper mainly discusses the implementation of a verification system created to be used with InftyReader [10]. The information about InftyReader and other supporting software can be found on the Infty Project website [11].

This paper is organized as follows: In Section 2, the grammar defining "well-formed" mathematical formulae is explained; in Section 3, the outline of the proposed verification method is described; in Section 4, the results of the experiment are shown; in Section 5, LM-CFTG is introduced as a formal framework for the grammar defining "well-formed" mathematical formulae; and in Section 6, the conclusion is drawn and future work determined.

2 "Well-Formed" Mathematical Formulae

In order to define "well-formed" mathematical formulae, linear monadic context-free tree grammar (LM-CFTG) [3] was employed as a formal framework. The definition and the formal properties of LM-CFTG will be introduced in Section 5. To choose an appropriate grammar formalism, it was necessary for a grammar formalism to have sufficient descriptive power to process a diversity of mathematical formulae. In addition to descriptive power, we also required that a grammar formalism be accompanied by a very fast parser.

An LM-CFTG is defined by arranging *fan-out rules* and *context-free rules.* Fan-out rules are used to define possible structural configuration of mathematical formulae. We should arrange them for symbols which are possibly connected with adjunct symbols. Examples of those symbols are "capital sigma" for summation, "capital pi" for product, "radical sign" for square root, "long bar" for fraction, "integral sign" for definite integral, etc. Because any variable may have a subscript, we arranged a fan-out rule for all italic alphabet symbols. Context-free rules are used to define possible linear sequences of symbols of mathematical formulae. Context-free rules constitute a context-free grammar (CFG) [2], and thus we can use well-established parsing techniques for CFG [9].

We experimentally developed a grammar defining "well-formed" mathematical formulae. The grammar consists of 35 fan-out rules and 170 context-free rules. The number of rules will be increased with the refinement of the grammar. A representative sample of the grammar is illustrated in the appendix at the end of this paper.

Table 1. Grammatical categories

Category	Explanation and Example
Math	Acceptable mathematical formula “$u(a,b) = Int\ Frac$”
Range	Range of value of a variable “$1 \leq i \leq n$”
Init	Initialization of a variable “$i = 0$”, “$i = k$”
Exp	Acceptable expression “$2 + 3$”, “$n(n + 1)$”
ExpList	List of expressions connected with signs “$a < b < c < d$”, “$z = x + y$”
Subscript	Subscript of a variable “2”, “n”, “$1, 2$”, “$1, 2, 3, 4$”
Supscript	Supscript of a variable and expression “$\prime$”, “$\prime\prime$”, “2”, “n”, “$1, 2$”

On the development of the grammar, we tried to arrange context-free rules so that they constitute a deterministic context-free grammar (DCFG) [2] because we could take advantage of a linear-time parsing technique for DCFG [9]. Unfortunately, we needed to add some context-free rules, which break the condition of DCFG, and this is the reason why a verification process of some exceptional mathematical formulae requires cubic time. Most of those context-free rules are related to the vertical-bar symbol because the usage of vertical bar is too diverse: absolute value, divides, conditional probability, norm of a vector, etc.

Table 1 shows the grammatical categories defined by the context-free rules of the grammar.

3 Outline of the Verification Method

In this section, we describe the outline of the proposed verification method. We start with the input to the verification system, that is to say, the output of mathematical OCR.

The output of InftyReader is given in InftyCSV format. An example of an InftyCSV text expressing a mathematical formula is shown in **Table 2**. Each line corresponds to a symbol in the formula, where: “ID” is the number uniquely assigned to the symbol; “x_1, x_2, x_3, x_4” are the coordinates of the rectangular area; “Mode” is a flag showing if the symbol is a part of a mathematical formula; “Link” expresses the relationship with the parental symbol; “Parent” is the ID of the parental symbol; and “Code” is the internal character code of the symbol. The original image and the rectangular representation of the formula are shown at (1) and (2) in **Fig. 3**.

Table 2. An example of an InftyCSV text

ID	x_1	y_1	x_2	y_2	Mode	Link	Parent	Code
1,	1487,	708,	1535,	766,	1,	-1,	-1,	0x426C
2,	1542,	685,	1559,	758,	1,	0,	1,	0x1980
3,	1563,	704,	1603,	742,	1,	0,	2,	0x4161
4,	1610,	732,	1622,	753,	1,	0,	3,	0x142C
5,	1646,	683,	1679,	742,	1,	0,	4,	0x4162
6,	1686,	685,	1703,	758,	1,	0,	5,	0x1981
7,	1728,	708,	1780,	724,	1,	0,	6,	0x1D3D
8,	1801,	624,	1858,	812,	1,	0,	7,	0x33F0
9,	1853,	782,	1881,	810,	1,	2,	8,	0x4161
10,	1868,	622,	1891,	658,	1,	1,	8,	0x4162
11,	1909,	717,	2053,	722,	1,	0,	8,	0x33D1
12,	1945,	629,	1985,	689,	1,	5,	11,	0x4164
13,	1986,	650,	2016,	689,	1,	0,	12,	0x4163
14,	1911,	736,	1967,	800,	1,	6,	11,	0x0248
15,	1977,	740,	1994,	813,	1,	0,	14,	0x1980
16,	1999,	757,	2029,	796,	1,	0,	15,	0x4163
17,	2035,	740,	2051,	813,	1,	0,	16,	0x1981

3.1 Construction of a Tree Representation

First, the verification system converts an InftyCSV text into a linked list called a *tree representation.* A node of a linked list is illustrated in **Fig. 4**. By preparing nodes for all symbols and connecting them in accordance with "Link" and "Parent" information in an InftyCSV text, the tree representation of a mathematical formula is constructed. The InftyCSV text in **Table 2** is converted into the tree representation shown at (3) in **Fig. 3**.

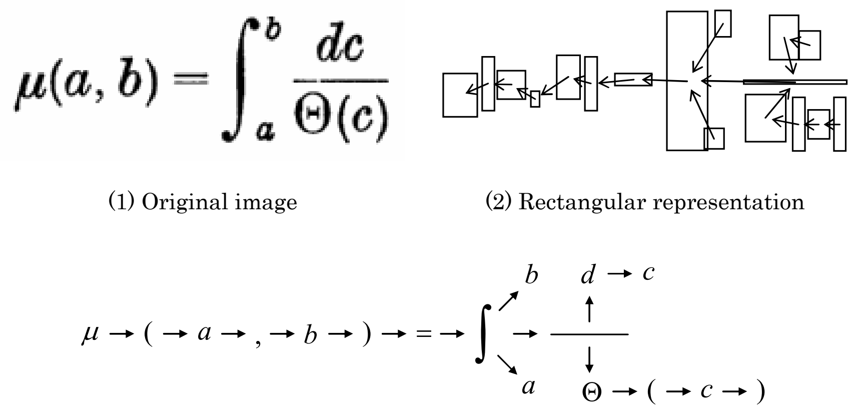
3.2 Division of a Mathematical Formula into Strings

Secondly, strings are extracted from a tree representation of a mathematical formula. Strings are obtained by concatenating symbols horizontally connected in a tree representation. From the tree representation shown in **Fig. 3**, the following five strings are extracted:

"u (a , b) $=$ *Int Frac*",
"b",
"a",
"d c", and
"Θ (c)".

3.3 Grammatical Analysis

Grammatical analysis is executed in two stages: *linear sequence analysis* and *structural inspection.* In the linear sequence analysis, a parser for a context-free



(1) Original image

(2) Rectangular representation

(3) Tree representation

Fig. 3. The mathematical formula

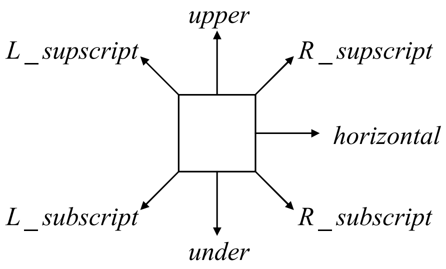


Fig. 4. A node

grammar (CFG) [2] is utilized, and, for each string extracted from a tree representation of a mathematical formula, the fitness to the grammatical categories is examined. In **Table 3**, the fitness to the grammatical categories for the strings is shown. The linear sequence analysis is the most time-consuming task in the proposed verification method, and may cost cubic time depending on the size of the input in the worst case, while the other tasks can be done in linear time.

After the linear sequence analysis, the structural inspection takes place. In the structural inspection, the connectivity of nodes is examined by searching for matching fan-out rules. The structural inspection process is illustrated in **Fig. 5**. The connection of the adjunct strings, "b" and "a", and the integral sign are inspected. The connection of the adjunct strings, "$d\ c$" and "$\Theta\ (\ c\ )$", and the long bar are also inspected.

The mathematical formula in the example was successfully verified as a "well-formed" mathematical formula.

Table 3. Fitness to the grammatical categories

Strings	Math	Range	Init	Exp	ExpList	Subscript	Supscript
$u\ (\ a\ ,\ b\ )\ =\ Int\ Frac$	yes	yes	yes	no	yes	no	no
b	yes	yes	no	yes	yes	yes	yes
a	yes	yes	no	yes	yes	yes	yes
$d\ c$	yes	yes	no	yes	yes	yes	yes
$\Theta\ (\ c\ )$	yes	yes	no	yes	yes	yes	yes

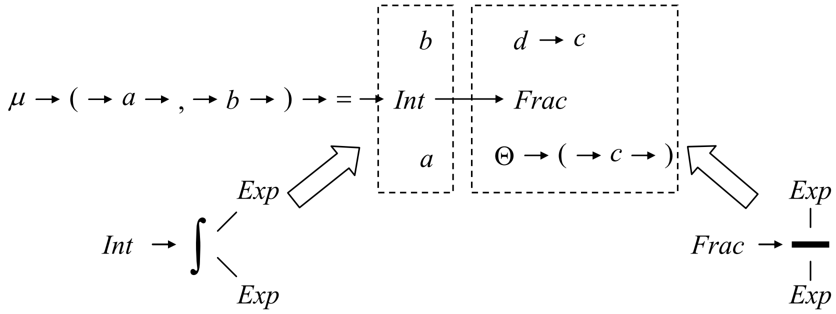


Fig. 5. Structural inspection

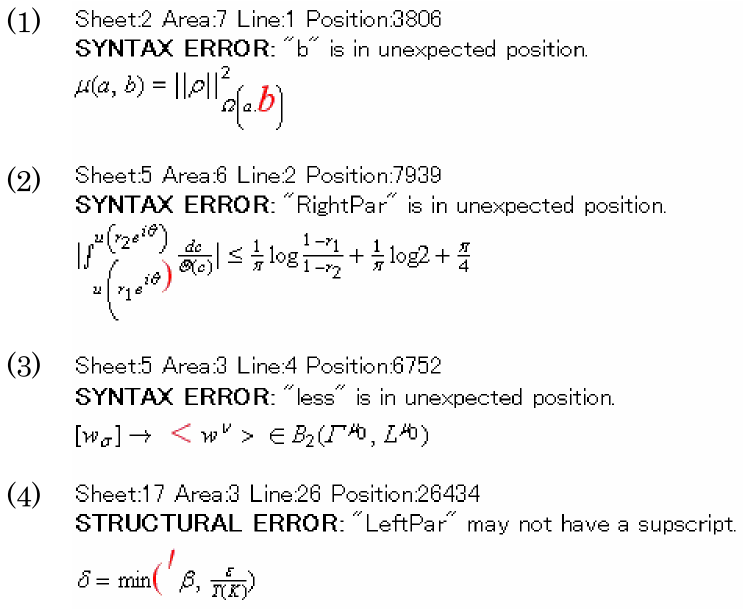
4 Experimental Results

We experimentally built a verification system in accordance with the proposed verification method. For implementation of the system, the program was written in C language, and GNU Bison [12], a parser generator for CFG, was utilized. For evaluation, we executed the system on the ground-truthed mathematical document database INFTY CDB-1 [7].

The verification of 21,967 mathematical formulae in INFTY CDB-1 (size: 48.1MB) was finished within 10 seconds by a PC (CPU: Pentium4 3.06GHz; RAM: 1GB). The speed of the proposed verification method was experimentally confirmed. Theoretically, a verification process of most mathematical formulae will be finished in linear time depending on the size of the input, though some exceptional mathematical formulae require cubic time.

The verification system produces verification results in XHTML format with MathML inclusions and displays error messages on a web browser such as Mozilla Firefox [13]. An error message identifies the position and type of suspicious mathematical-formula error as enlarged and colored red.

(1), (2), (3) and (4) in **Fig. 6** are error messages produced by the verification system. Original images corresponding to the error messages are shown in **Fig. 7**. As for (1), the verification system successfully detected the misrecognition of a comma before the letter 'b'. The comma was misrecognized as a period. With regard to (2), the verification system detected a faulty correspondence of



(1) Sheet:2 Area:7 Line:1 Position:3806
SYNTAX ERROR: ˝b˝ is in unexpected position.
$\mu(a, b) = ||\rho||^{2}_{\Omega(a.b)}$

(2) Sheet:5 Area:6 Line:2 Position:7939
SYNTAX ERROR: ˝RightPar˝ is in unexpected position.
$|\int_{u(r_1 e^{i\theta}))}^{u(r_2 e^{i\theta})} \frac{dc}{\Theta(c)}| \leq \frac{1}{\pi}\log\frac{1-r_1}{1-r_2} + \frac{1}{\pi}\log 2 + \frac{\pi}{4}$

(3) Sheet:5 Area:3 Line:4 Position:6752
SYNTAX ERROR: ˝less˝ is in unexpected position.
$[w_\sigma] \to < w^V > \in B_2(\Gamma^{\mu_0}, L^{\mu_0})$

(4) Sheet:17 Area:3 Line:26 Position:26434
STRUCTURAL ERROR: ˝LeftPar˝ may not have a supscript.
$\delta = \min(^{\prime} \beta, \frac{\varepsilon}{\Gamma(K)})$

Fig. 6. Error messages produced by the verification system

parentheses. Looking at the original image, we noticed that this was an error from the original document. Concerning (3), the verification system successfully detected the misrecognition of the angle bracket. The angle bracket was misrecognized as a less-than sign. And about (4), a structural error was detected since a left parenthesis may not have a superscript. A portion of the left parenthesis was misrecognized as a prime symbol.

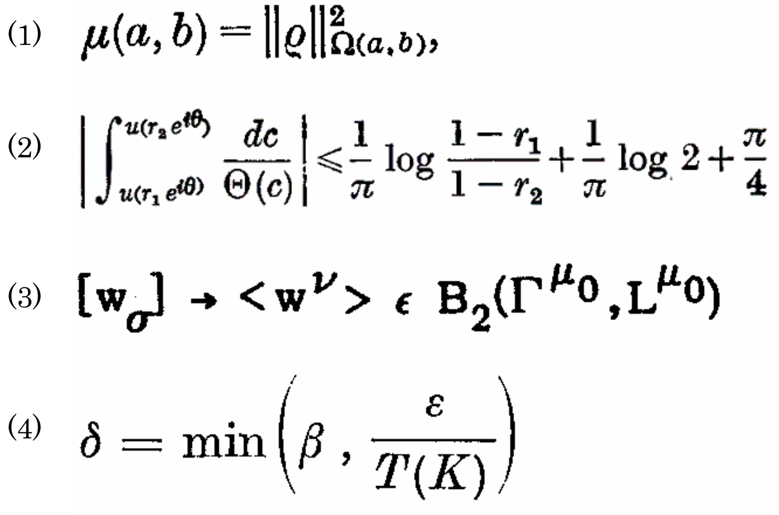
5 Formal Framework

In this section, we introduce the formal definitions of tree and linear monadic context-free tree grammar (LM-CFTG). LM-CFTG was employed as a formal framework to define "well-formed" mathematical formulae. We also introduce known results for LM-CFTG.

5.1 Tree

A *ranked alphabet* is a finite set of symbols in which each symbol is associated with a natural number, called the *arity* of a symbol. Let Σ be a ranked alphabet. For $n \geq 0$, let $\Sigma_n = \{a \in \Sigma \mid \text{the arity of } a \text{ is } n\}$. A ranked alphabet is *monadic* if the arity of each its element is at most 1.

The set of *trees* over Σ, denoted by T_Σ, is the smallest set of strings over elements of Σ, parentheses and commas defined inductively as follows:



(1) $\mu(a,b) = \|\varrho\|^2_{\Omega(a,b)},$

(2) $\left|\int_{u(r_1 e^{i\theta})}^{u(r_2 e^{i\theta})} \frac{dc}{\Theta(c)}\right| \leqslant \frac{1}{\pi} \log \frac{1-r_1}{1-r_2} + \frac{1}{\pi} \log 2 + \frac{\pi}{4}$

(3) $[\mathbf{w}_\sigma] \to <\mathbf{w}^\nu> \epsilon\ \mathbf{B}_2(\Gamma^{\mu}0, \mathbf{L}^{\mu}0)$

(4) $\delta = \min\left(\beta\ ,\ \frac{\varepsilon}{T(K)}\right)$

Fig. 7. Original images

(1) $\Sigma_0 \subseteq T_\Sigma$, and
(2) if $a \in \Sigma_n$ for some $n \geq 1$, and $t_1, t_2, \ldots, t_n \in T_\Sigma$, then $a(t_1, t_2, \ldots, t_n) \in T_\Sigma$.

Let x be a variable. $T_\Sigma(x)$ is defined as $T_{\Sigma \cup \{x\}}$ taking the rank of x to be 0. For $t, u \in T_\Sigma(x)$, $t[u]$ is defined as the result of substituting u for the occurrences of the variable x in t. A tree $t \in T_\Sigma(x)$ is *linear* if x occurs exactly once in t.

5.2 Linear Monadic Context-Free Tree Grammar

An *linear monadic context-free tree grammar* (LM-CFTG) is a four-tuple $\mathcal{G} = (N, \Sigma, P, S)$, where:

- N is a monadic ranked alphabet of *nonterminals*,
- Σ is a ranked alphabet of *terminals*, disjoint with N,
- $S \in N_0$ is the *initial nonterminal*, and
- P is a finite set of *production rules* of one of the following forms:

$$(1)\ \ A \to u$$

with $A \in N_0$ and $u \in T_{N \cup \Sigma}$, or

$$(2)\ \ A(x) \to u$$

with $A \in N_1$ and a linear tree $u \in T_{N \cup \Sigma}(x)$.

For an LM-CFTG $\mathcal{G}$, the *one-step derivation* $\underset{\mathcal{G}}{\Rightarrow}$ is the relation over $T_{N \cup \Sigma}(x)$ such that, for $t \in T_{N \cup \Sigma}(x)$, (1) if $A \to u$ is in P and $t = t'[A]$ for some linear

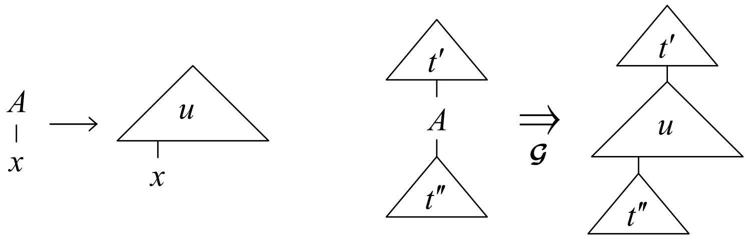


Fig. 8. One-step derivation

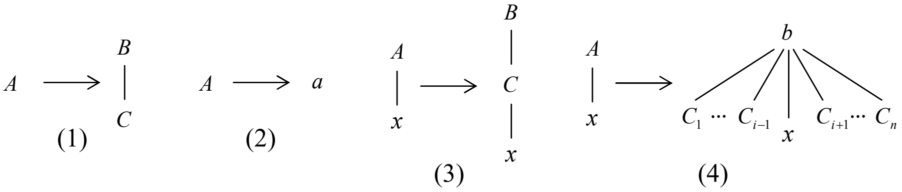


Fig. 9. Chomsky-like normal form

tree $t' \in T_{N\cup\Sigma}(x)$, then $t \underset{\mathcal{G}}{\Rightarrow} t'[u]$, and (2) if $A(x) \to u$ is in P and $t = t'[A(t'')]$ for some linear trees $t', t'' \in T_{N\cup\Sigma}(x)$, then $t \underset{\mathcal{G}}{\Rightarrow} t'[u[t'']]$. See **Fig. 8**.

Let $\overset{*}{\underset{\mathcal{G}}{\Rightarrow}}$ denote the reflexive transitive closure of $\underset{\mathcal{G}}{\Rightarrow}$. The *tree language generated by* $\mathcal{G}$ is the set $L(\mathcal{G}) = \{t \in T_\Sigma \mid S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} t\}$.

5.3 Known Results for LM-CFTG

First, we introduce normal forms for LM-CFTG. The reason fan-out rules and context-free rules are sufficient to define an LM-CFTG is based on **Theorem 1**.

Theorem 1. (Fujiyoshi [14]) [Chomsky-like normal form] Any LM-CFTG can be transformed into an equivalent one whose rules are in one of the following forms:

(1) $A \to B(C)$ with $A, C \in N_0$ and $B \in N_1$,
(2) $A \to a$ with $A \in N_0$ and $a \in \Sigma_0$,
(3) $A(x) \to B(C(x))$ with $A, B, C \in N_1$, or
(4) $A(x) \to b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_n)$ with $A \in N_1$, $n \geq 1$, $b \in \Sigma_n$, $1 \leq i \leq n$ and $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_n \in N_0$.

See **Fig. 9**.

Theorem 2. (Fujiyoshi [14]) [Greibach-like normal form] Any LM-CFTG can be transformed into an equivalent one whose rules are in one of the following forms:

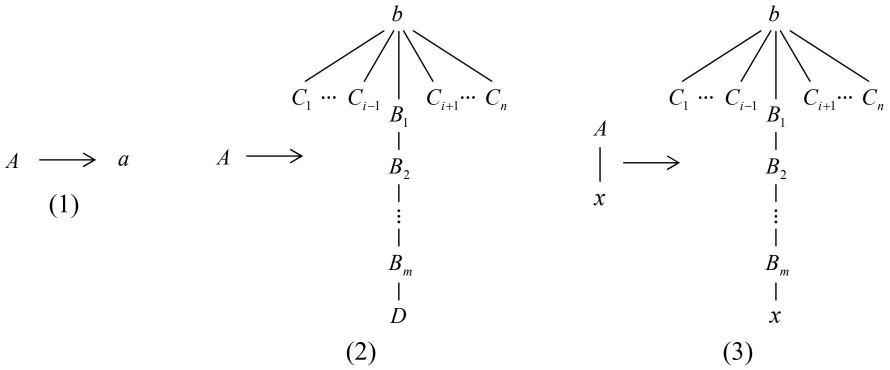


Fig. 10. Greibach-like normal form

(1) $A \to a$ with $A \in N_0$ and $a \in \Sigma_0$,
(2) $A \to b(C_1, \ldots, C_{i-1}, u, C_{i+1}, \ldots, C_n)$ with $A \in N_0$, $n \geq 1$, $b \in \Sigma_n$, $1 \leq i \leq n$, $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_n \in N_0$ and $u \in T_N$, or
(3) $A(x) \to b(C_1, \ldots, C_{i-1}, u, C_{i+1}, \ldots, C_n)$ with $A \in N_1$, $n \geq 1$, $b \in \Sigma_n$, $1 \leq i \leq n$, $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_n \in N_0$, and $u \in T_{N_1}(x)$.

See **Fig. 10**. Since N is monadic, all trees in T_N and $T_{N_1}(x)$ may be written as $B_1(B_2(\cdots(B_m(D))\cdots))$ and $B_1(B_2(\cdots(B_m(x))\cdots))$, respectively, for some $m \geq 0$, $B_1, B_2, \ldots, B_m \in N_1$ and $D \in N_0$. Note that m may be 0.

LM-CFTG is related to the tree adjoining grammar (TAG) [15–17], one of the most famous and well-studied mildly context-sensitive grammar formalisms. The definition of "weakly equivalent" is found in [3].

Theorem 3. (Fujiyoshi & Kasai [3]) LM-CFTG is weakly equivalent to TAG.

There exists an effective recognition algorithm for LM-CFTG.

Theorem 4. (Fujiyoshi [8]) There exists a recognition algorithm for LM-CFTG that runs in $O(n^4)$ time, where n is the number of nodes of an input tree.

It is known that a recognition algorithm that runs in $O(n^3)$ time can be obtained with some modifications to the $O(n^4)$-time algorithm in [8].

6 Conclusion and Future Work

We have proposed a verification method of mathematical formulae for a practical mathematical OCR system based on a combination of context-free grammar and tree grammar. Though we have recognized the usefulness of the proposed verification method by experimental results, we know the necessity of the improvement of the grammar defining "well-formed" mathematical formulae. Moreover, in order to avoid the ambiguity of the grammar, the inclusion of semantic analysis needs to be considered.

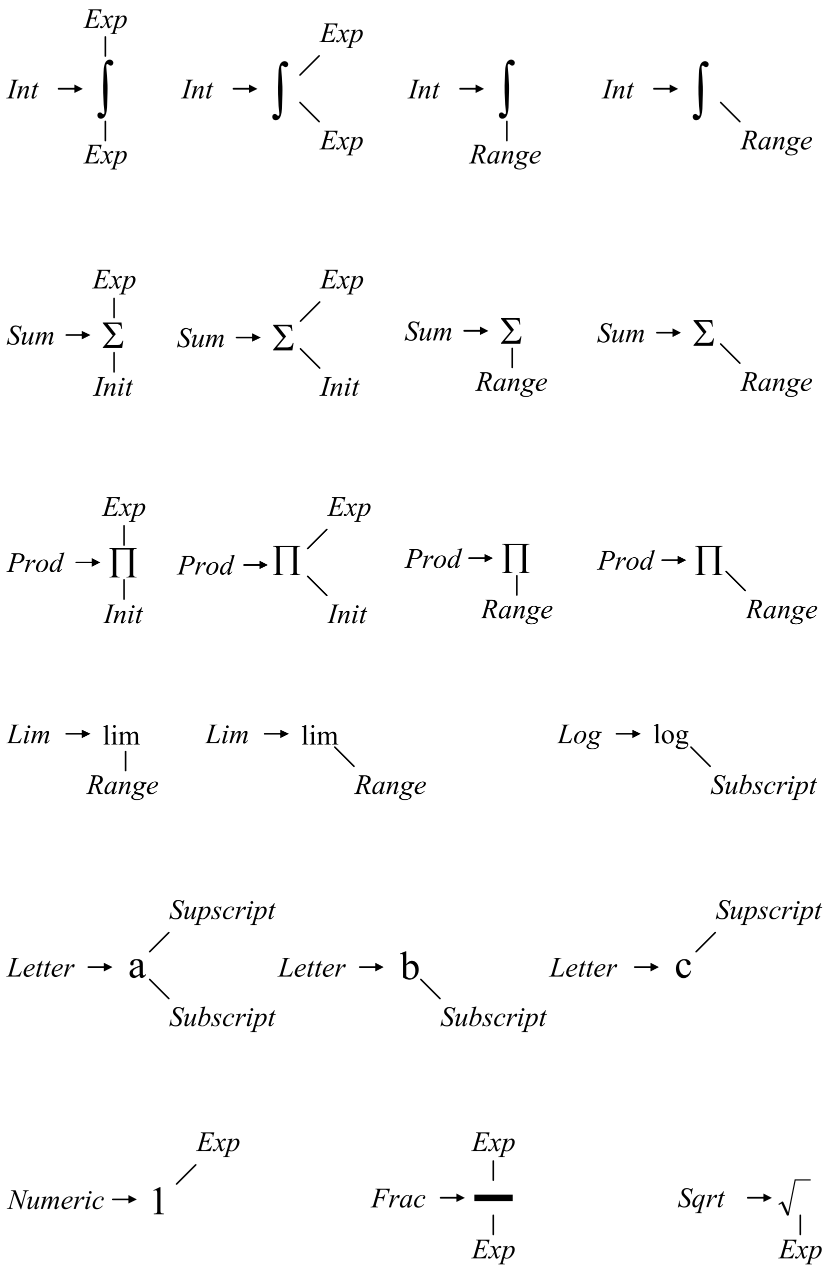
In the future, we plan to internalize a verification system within the recognition engine of a mathematical OCR system. Because the flexibility of the grammar is important, we want to allow users to manipulate the grammar. Therefore, we will reflect on ways users can update the grammar by themselves.

References

1. Chan, K.F., Yeung, D.Y.: Mathematical expression recognition: a survey. Int. J. Document Analysis and Recoginition **3**(1) (2000) 3–15
2. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison Wesley, Reading, Massachusetts (1979)
3. Fujiyoshi, A., Kasai, T.: Spinal-formed context-free tree grammars. Theory of Computing Systems **33**(1) (2000) 59–83
4. Anderson, R.: Syntax-directed recognition of hand-printed two-dimensional mathematics. In: Interactive Systems for Experimental Applied Mathematics. Academic Press (1968) 436–459
5. Chou, P.A.: Recognition of equations using a two-dimensional stochastic context-free grammar. In: Proc. SPIE. Volume 1199. (1989) 852–863
6. Kanahori, T., Sexton, A., Sorge, V., Suzuki, M.: Capturing abstract matrices from paper. In: Proceedings of the 5th International Conference on Mathematical Knowledge Management (MKM 2006), LNCS 4108 (2006) 124–138
7. Suzuki, M., Uchida, S., Nomura, A.: A ground-truthed mathematical character and symbol image database. In: Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR 2005). Volume 2. (2005) 675–679
8. Fujiyoshi, A.: Application of the CKY algorithm to recognition of tree structures for linear, monadic context-free tree grammars. IEICE Trans. Inf. & Syst. **E90-D**(2) (2007) 388–394
9. Sikkel, K., Nijholt, A.: Parsing of Contex-Free Languages. In: Handbook of Formal Languages. Volume 2. Springer-Verlag, Berlin (1997) 61–100
10. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., Kanahori, T.: Infty - an integrated OCR system for mathematical documents. In: Proceedings of ACM Symposium on Document Engineering 2003. (2003) 95–104
11. Infty Project, `http://www.inftyproject.org/en/`
12. Donnelly, C., Stallman, R.: Bison: The yacc-compatible parser generator. Available on: `http://www.gnu.org/software/bison/manual/` (2006)
13. Mozilla Firefox, `http://www.mozilla.com/firefox/`
14. Fujiyoshi, A.: Analogical conception of chomsky normal form and greibach normal form for linear, monadic context-free tree grammars. IEICE Trans. Inf. & Syst. **E89-D**(12) (2006) 2933–2938
15. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree adjunct grammars. J. Computer & System Sciences **10**(1) (1975) 136–163
16. Joshi, A.K., Schabes, Y.: Tree-adjoining grammars. In: Handbook of Formal Languages. Volume 3. Springer-Verlag, Berlin (1997) 69–124
17. Abeillé, A., Rambow, O., eds.: Tree adjoining grammars: formalisms, linguistic analysis and processing. CSLI Publications, Stanford, California (2000)

Appendix: Representative Sample of the Grammar Defining "Well-Formed" Mathematical Formulae

Fan-Out Rules:



Context-Free Rules:

$$\begin{array}{lll} \mathit{Math} & \rightarrow & \mathit{ExpList} \\ & \rightarrow & \mathit{Sign}\ \mathit{ExpList} \end{array}$$

$$\begin{array}{lll} \mathit{ExpList} & \rightarrow & \mathit{Exp} \\ & \rightarrow & \mathit{ExpList}\ \mathit{Sign}\ \mathit{Exp} \end{array}$$

$$\begin{array}{lll} \mathit{Exp} & \rightarrow & \mathit{Term} \\ & \rightarrow & \mathit{UnaryOp}\ \mathit{Term} \end{array}$$

$$\begin{array}{lll} \mathit{Term} & \rightarrow & \mathit{Factor} \\ & \rightarrow & \mathit{Term}\ \mathit{Factor} \\ & \rightarrow & \mathit{Term}\ \mathit{BinaryOp}\ \mathit{Factor} \\ & \rightarrow & `\infty\text{'} \end{array}$$

$$\begin{array}{lll} \mathit{Range} & \rightarrow & \mathit{Exp} \\ & \rightarrow & \mathit{Exp}\ \mathit{Sign}\ \mathit{Exp} \\ & \rightarrow & \mathit{Exp}\ \mathit{Sign}\ \mathit{Factor}\ \mathit{Sign}\ \mathit{Exp} \end{array}$$

$$\begin{array}{lll} \mathit{Init} & \rightarrow & \mathit{Exp}\ `=\text{'}\ \mathit{Exp} \end{array}$$

$$\begin{array}{lll} \mathit{SubScript} & \rightarrow & \mathit{Exp} \\ & \rightarrow & \mathit{Sign} \\ & \rightarrow & \mathit{SubScript}\ `,\text{'}\ \mathit{Exp} \\ & \rightarrow & \mathit{SubScript}\ `,\text{'}\ \text{Sign} \\ & \rightarrow & \mathit{Exp}\ `=\text{'}\ \mathit{Exp} \end{array}$$

$$\begin{array}{lll} \mathit{SupScript} & \rightarrow & \mathit{Exp} \\ & \rightarrow & \mathit{Sign} \\ & \rightarrow & `\prime\text{'}\ \ \text{//prime} \\ & \rightarrow & `\prime\prime\text{'}\ \ \text{//doubleprime} \\ & \rightarrow & `\prime\prime\prime\text{'}\ \ \text{//tripleprime} \end{array}$$

$$\begin{array}{lll} \mathit{Sign} & \rightarrow & `=\text{'} \\ & \rightarrow & `\neq\text{'} \\ & \rightarrow & `<\text{'} \\ & \rightarrow & `\leq\text{'} \\ & \rightarrow & `>\text{'} \\ & \rightarrow & `\geq\text{'} \\ & \rightarrow & `\in\text{'} \\ & \rightarrow & `\ni\text{'} \\ & \rightarrow & `\not\ni\text{'} \\ & \rightarrow & `\subset\text{'} \\ & \rightarrow & `\not\subset\text{'} \\ & \rightarrow & `\supset\text{'} \\ & \rightarrow & `\not\supset\text{'} \\ & \rightarrow & `\equiv\text{'} \\ & \rightarrow & `\cong\text{'} \\ & \rightarrow & `\sim\text{'} \\ & \rightarrow & `\rightarrow\text{'} \\ & \rightarrow & `\Rightarrow\text{'} \\ & \rightarrow & `\leftrightarrow\text{'} \\ & \rightarrow & `\Leftrightarrow\text{'} \\ & \rightarrow & `\mapsto\text{'} \\ & \rightarrow & `,\text{'}\ \ \text{//comma} \\ & \rightarrow & `;\text{'}\ \ \text{//semicolon} \\ & \rightarrow & `|\text{'}\ \ \text{//vert} \end{array}$$

$$\begin{array}{lll} \mathit{UnaryOp} & \rightarrow & `+\text{'} \\ & \rightarrow & `-\text{'} \\ & \rightarrow & `\pm\text{'} \\ & \rightarrow & `\mp\text{'} \\ & \rightarrow & `\forall\text{'} \\ & \rightarrow & `\exists\text{'} \end{array}$$

$$\begin{array}{lll} \mathit{BinaryOp} & \rightarrow & `+\text{'} \\ & \rightarrow & `-\text{'} \\ & \rightarrow & `\times\text{'} \\ & \rightarrow & `/\text{'} \\ & \rightarrow & `\cap\text{'} \\ & \rightarrow & `\cup\text{'} \\ & \rightarrow & `\cdot\text{'} \\ & \rightarrow & `\bullet\text{'} \\ & \rightarrow & `:\text{'}\ \ \text{//colon} \end{array}$$

$$\begin{aligned}
\mathit{Factor} &\rightarrow \mathit{Variable}\\
&\rightarrow \mathit{Number}\\
&\rightarrow \emptyset\\
&\rightarrow *\\
&\rightarrow \triangle\\
&\rightarrow \nabla\\
&\rightarrow \aleph\\
&\rightarrow \text{‘(’ ExpList ‘)’}\\
&\rightarrow \text{‘[’ ExpList ‘]’}\\
&\rightarrow \text{‘\{’ ExpList ‘\}’}\\
&\rightarrow \text{‘(’ ExpList ‘]’}\\
&\rightarrow \text{‘[’ ExpList ‘)’}\\
&\rightarrow \text{‘$\langle$’ ExpList ‘$\rangle$’}\\
&\rightarrow \mathit{Frac}\\
&\rightarrow \mathit{Sqrt}\\
&\rightarrow \text{‘$|$’ Term ‘$|$’}\\
&\rightarrow \mathit{TrigOp}\ \mathit{Factor}\\
&\rightarrow \mathit{SumOp}\ \mathit{Factor}\\
&\rightarrow \mathit{FuncOp}\ \mathit{Factor}
\end{aligned}$$

$$\begin{aligned}
\mathit{TrigOp} &\rightarrow \mathit{Sin}\\
&\rightarrow \mathit{Cos}\\
&\rightarrow \mathit{Tan}
\end{aligned}$$

$$\begin{aligned}
\mathit{SumOp} &\rightarrow \mathit{Int}\\
&\rightarrow \mathit{Sum}\\
&\rightarrow \mathit{Prod}\\
&\rightarrow \mathit{Bigcap}\\
&\rightarrow \mathit{Bigcup}
\end{aligned}$$

$$\begin{aligned}
\mathit{FuncOp} &\rightarrow \mathit{Lim}\\
&\rightarrow \mathit{Log}\\
&\rightarrow \mathit{Min}\\
&\rightarrow \mathit{Max}
\end{aligned}$$

$$\mathit{Variable} \rightarrow \mathit{Letter}$$

$$\begin{aligned}
\mathit{Letter} &\rightarrow \text{‘a’}\\
&\rightarrow \text{‘b’}\\
&\ \ \vdots\\
&\rightarrow \text{‘z’}
\end{aligned}$$

$$\begin{aligned}
\mathit{Number} &\rightarrow \mathit{Integer}\\
&\rightarrow \mathit{Integer}\ \text{‘.’ Integer}\\
&\rightarrow \text{‘.’ Integer}
\end{aligned}$$

$$\begin{aligned}
\mathit{Integer} &\rightarrow \mathit{Numeric}\\
&\rightarrow \mathit{Integer}\ \mathit{Numeric}
\end{aligned}$$

$$\begin{aligned}
\mathit{Numeric} &\rightarrow \text{‘0’}\\
&\rightarrow \text{‘1’}\\
&\ \ \vdots\\
&\rightarrow \text{‘9’}
\end{aligned}$$

$$\begin{aligned}
\mathit{Sin} &\rightarrow \text{‘sin’}\\
\mathit{Cos} &\rightarrow \text{‘cos’}\\
\mathit{Tan} &\rightarrow \text{‘tan’}
\end{aligned}$$

$$\begin{aligned}
\mathit{Int} &\rightarrow \text{‘$\int$’}\\
\mathit{Sum} &\rightarrow \text{‘$\sum$’}\\
\mathit{Prod} &\rightarrow \text{‘$\prod$’}\\
\mathit{Bigcap} &\rightarrow \text{‘$\bigcap$’}\\
\mathit{Bigcup} &\rightarrow \text{‘$\bigcup$’}
\end{aligned}$$

$$\begin{aligned}
\mathit{Lim} &\rightarrow \text{‘lim’}\\
\mathit{Log} &\rightarrow \text{‘log’}\\
\mathit{Min} &\rightarrow \text{‘min’}\\
\mathit{Max} &\rightarrow \text{‘max’}
\end{aligned}$$