

FORMAL NOTATION BASED GRAMMARS AND META-LANGUAGE FOR COMPUTER PARSING OF MATHEMATICAL NOTATION

ZHAO Yanjie

Nagasaki Institute of Applied Science, Nagasaki, Japan

Abstract: Mathematical notation is quite different from both natural languages and programming languages. The studies of whole mathematical notation is lacking. The author studies the syntax and semantics of whole mathematical notation, and develops a linguistic formalization method, a set of formal notation based tree grammars, a knowledge based computer parsing system that is independent of the influence of human-computer interface, and a meta-language based approach for ambiguity avoidance and notation extension of mathematical expressions.

Keywords: mathematical notation, non-verbal, tree grammar, parsing, meta-language.

1. INTRODUCTION

Mathematical notation has evolved in about 400 years, and has become more and more important, powerful, and universal in use. Many efforts for input and parsing of mathematical expressions on computers have respectively been made in the fields of programming languages, pattern recognition, document processing, software specification, and human interface of mathematical computation systems. Many references have been

given in (Kajler and Soiffer, 1994) (Zhao, et al., 1994) (Zhao, 1996c). However, the grammars for whole mathematical notation and studies in the view of computational linguistics are still lacking. Many useful and powerful context-sensitive mathematical notations can not be parsed. The problems of semantic ambiguity and grammar extension have not been resolved.

To solve these problems, the author has contributed: **1)** establishment of a set of tree grammars (including both syntax and semantics) for modern mathematical notation (Zhao, et al., 1994) (Zhao, et al., 1996a) (Zhao, 1996c) **2)** definition of a set of formal notations to abstract the non-verbal components and concatenates of mathematical notation (Zhao, et al., 1994) (Zhao, et al., 1996a) (Zhao, 1996c), as well as its computer input by key and mouse (Sakurai, et al., 1996) (Zhao, 1996c), **3)** parsing of much more complicated context-sensitive notations (Zhao, et al., 1996a) (Zhao, 1996c), **4)** clarification of the necessities of more than one grammar to avoid the semantic ambiguities (Zhao, et al., 1996a) (Zhao, 1996c), and **5)** development of a meta-language to generate all the grammars for realizing grammar revision and extension (Zhao, et al., 1996a) (Zhao, et al., 1996b) (Zhao, 1996c). This paper will focus on the necessities of the formal notation based formalization, the independence of these tree grammars from the input methods of mathematical expressions, and a formal and complete description of the formal notation based grammars.

2. FORMALIZATION

Mathematical expressions can be recursively constructed from mathematical signs or morphemes. Such a morpheme is a minimal meaningful part of a mathematical expression, i.e., a numeral, the core part or index part of an object, operator, separator, or elliptical sign.

A mathematical expression (expr in short) is a morpheme, or a structure of morphemes. An expr occupies a rectangle of space denoted by a box. The notations of boxes and their concatenate lines and arrows shown in Figure 1 compose a **formal notation** that is denoted F_N . It is not a computer input image because of no WYSIWYG, but an input images abstraction or a well-formed or cognitively structured pattern that can be input by formula templates that are in advance built in human interface (Sakurai, et al., 1996) (Zhao, 1996c). F_N may become invisible after the input. A mathematical expression based on the F_N is called a **formal representation** (FR in short), which is defined in Figure 1.

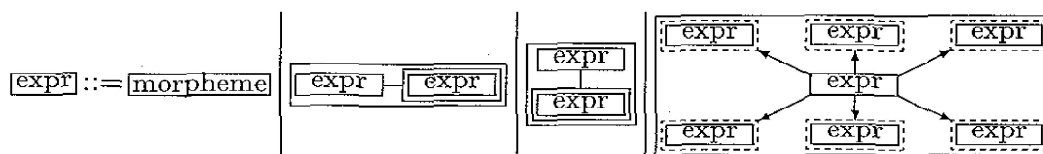


Figure 1: Definition of formal representation

In Figure 1, the line means horizontal or vertical concatenate; the arrow means one of six subordinate or adjectival concatenates; the duet box means the repetition of horizontal or vertical concatenates, or empty; the dashed box means an option.

The textual or computer inner representation of F_N and FR is improved from its former one (Zhao, et al., 1994) (Zhao, 1996c) and is defined below:

```

<expression> ::= [ row, <expression list> ] |
                  [ column, <expression list> ] |
                  [ tree, <expression> {, [ sup, <expression> ] }
                                     {, [ upper, <expression> ] }
                                     {, [ pre_sup, <expression> ] }
                                     {, [ pre_sub, <expression> ] }
                                     {, [ lower, <expression> ] }
                                     {, [ sub, <expression> ] } } ] |
                  ' <morpheme> '

<expression list> ::= <expression> | <expression>, <expression list>

<morpheme> ::= <sign in Normal Code> | <sign in Normal Code> <morpheme>

```

Note: 1. The sign ' within <morpheme> has to be denoted as the sign ''.
 2. Here, meta-symbol {.} means option; | means alternation.

The Normal Code has been defined in (Zhao, 1996c). For example, $\int_a^b f(x)dx$ can be abstracted from its input image into the FR shown in Figure 2. Its textual FR is below:

```
[row,[tree,'\int',[sup,'b'],[sub,'a']], 'f', '\(', 'x', '\)', '{@rm d}', 'x']
```

FR contains the minimum kinds of the formalization structure, because except the three fundamental structures (row, column, and tree), there are not other structures that are input by the templates. Thus FR is independent of input methods and human-computer interfaces. Of course for a more efficient application, more structures (e.g., matrix, integral, etc) should be introduced. In this paper, the research focuses on the parsing implementability of mathematical notation based on F_N , and this implementability will become a foundation in the development of practical systems.

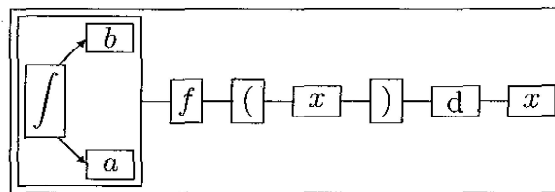


Figure 2: Example of formal representation

It is very important that computer graphical user interface permits us to input mathematical expressions based on F_N , because many difficulties can be solved or avoided during the input. The necessities of the F_N formalization can be summarized as follows:

1. FR can be used to express the universal set of F_N based mathematical expressions.
2. FR can be used to abstract a non-verbal symbol into its verbal one. For example, fraction strokes being different in length become same formal stroke; open-close symbols, such as |, (, etc, being different in height become same formal one respectively; square root symbols $\sqrt{\dots}$ being different in both length and style become same formal one like as [column, '\sqrt', ...]; etc.
3. FR is an abstract model for expressing the following **typesetting tacit agreements** explicitly and normally. No matter what methods one uses, such as pen and paper, printing, computer hand-written input, computer image scanning input, or computer key-mouse input, one has to obey the two-dimensional row, column, and tree structures on a base line. In addition, there are also some rules for line-breaking, page-breaking, etc, which need another study. All these things have become tacit agreements, called **typesetting tacit agreements**. FR is independent of input methods and typesetting details, such as the font produced by different makers, font size (e.g., the two x 's in x^x are the same), tiny differences in location and gap of symbols (e.g., between + and b in $a + b$ versus between - and b in $-b$), different space and skip, etc. These typesetting details contribute nothing to the parsing.
4. FR can be used to express all the following **determinable tacit agreements** and **indeterminable tacit agreements**. To omit symbols, some tacit agreements about priority and associative rule have been widely accepted. For the example of the priority, we can find that any *factor* is at the top priority. The second priority is a *term*. The third one is called *calculation* which includes most binary operations. In the case of the associative rule, we can find that some operations satisfy both left and right associative rules, some others only the left one, etc. These priorities are **determinable** or **definite** in a context, in spite of their different kinds (e.g., logical operators are prior to relation operators in one priority, and posterior to in another), which is called **determinable tacit agreements**. To omit much more symbols without causing ambiguities, other tacit agreements concerning priority, which is called **indeterminable tacit agreements**, have been accepted to a certain extent. For example, $\sum_{k=1}^{\infty} a^k \sin k\pi s + a$ often means $(\sum_{k=1}^{\infty} a^k \sin(k\pi s)) + a$; not means $(\sum_{k=1}^{\infty} a^k)(\sin k)\pi s + a$. However, some cases are difficult to determine or are indeterminable. For example, what does the expression $\sin x \sqrt{ax^3 + bx^2 + cx + d}$ means? $(\sin x) \sqrt{ax^3 + bx^2 + cx + d}$ or $\sin(x \sqrt{ax^3 + bx^2 + cx + d})$? This may be dependent on different conventions, contexts, fields, and personal habits. To these two kinds of tacit agreements, FR can be used to express them explicitly. For example, to parse the expression $a + b \times c$, we can simply input it as $a + [b \times c]$. Similarly, $\sin(n + 1)x$ can be input as $\sin[(n + 1)x]$ or $[\sin(n + 1)]x$. Of course, clear design

of a grammar and elaborate introduction of its grammatical categories can also realize the description of these tacit agreements in principle. For example, to parse an expression like $a + b \times c$, we can establish rules like as $\langle \text{calculi} \rangle ::= \langle \text{calculi} \rangle + \langle \text{term} \rangle$ and $\langle \text{term} \rangle ::= \langle \text{term} \rangle \times \langle \text{factor} \rangle \dots$ However the indeterminable situations are not so explicit. Nevertheless, in representation of these tacit agreements, the more boxes we use, the less grammar rules and categories we need; and vice versa.

5. FR can also be used to distinguish some inherent ambiguities. For a simple example, what does the f_i^3 mean? $(f_i)^3$, $(f^3)_i$, or (f_i^3) ? Without a context-sensitive restriction or a clear formalization, we can not obtain any answer.
6. FR can be used as the meta-notation of a meta-language to describe grammars of mathematical notation, because Backus-Naur Form (BNF in short) is inconvenient and even impossible to define the grammars of mathematical notation. BNF can not explicitly express the column and tree concatenates, some tacit agreements, and non-verbal symbols, if additional meta-symbols, which are equivalent to F_N , are not introduced.

Therefore, in comparison to the input and parsing of natural languages and programming languages, the formalization is not only necessary, but also as important as the grammar. A certain formalization with a certain grammar description determines a parsing. The formalization is not only a special method to aid the input and the parsing. F_N is inevitable and becomes a base-component of the grammars.

3. PARSING

The essential meaning of a mathematical expression can be described by functions with their codomains. They form a static denotational semantics of mathematical notation, which is called **meta-representation** (MR in short) and is defined below:

$$\langle \text{MR} \rangle ::= f(\langle \text{MR} \rangle, \dots, \langle \text{MR} \rangle), d \mid n, d$$

where f means a function name; d means a domain or codomain name; n means an individual name.

MP is independent of programming languages and algorithms, and includes only the name of functions, but does not give the function definitions. Up to the present, the author has implemented a prototype to translate MR into front-end languages of computer algebra systems (*Mathematica* and *Maple*) for further interpretation or simulation on computers. The parsing here means a translation from FR into its MR based on the grammars.

All mathematical symbols can be classified into two kinds: 1) **fixed mathematical symbols**, i.e., fixed individuals, operators and auxiliary symbols, 2) **changeable mathematical symbols**, i.e., variables, self-defined functions, self-defined function calls, and polynomials (Zhao, et al., 1994). The meanings and usages of existing fixed mathematical

symbols have been formed historically and have been accepted by the whole world. They are invariant and do not need declarations to say what they are. However, changeable mathematical symbols need declarations to say what they are and what domains they belong to, and they may have different meanings and usages in different contexts.

According to this classification, the fixed mathematical symbols determine the structure pattern of mathematical expressions. Therefore, all the grammatical knowledge about both the syntax and the semantics of mathematical notation can be expressed in a *frame and rewriting rule* knowledge representation below according to the relation between the structure and its meanings of a mathematical expression (Zhao, et al., 1994).

A grammar is mainly composed of a **rule base** and its **background knowledge**. The **rule base** is constructed by a series of rules. A rule has the following construction (where a pair of brackets [and] as meta-symbols denote an optional component):

```
rule{
  structure{ <category> → <structure_pattern> [ action ... ] }
  meanings{
    function{ textual function of meta-representation }
    [
      domains{ <codomain> ← <domain_pattern> [ action ... ];
               <codomain> ← <domain_pattern> [ action ... ];
               ⋮
            }
    ]
  }
  [
    meanings{ ... }
  ]
  ⋮
}
```

Where, <structure_pattern> and every <domain_pattern> in a rule have the same formalized structure. Every *action* is a procedure to perform a certain context-sensitive processing (being similar to the actions of *augmented transition network grammars*). The whole of **structure** part corresponds to the syntax. The $\rightarrow$ means "can have the form". The $\leftarrow$ means "is mapped from". The **function** part together with one of <codomain>'s composes a MR. An example of such a rule for $\int_{\square} \square d\square$ is written in a meta-language and is shown below (omitting all the boxes in this example):

```
rule{
  structure{
    < factor > →  $\int_{\text{<calculation2>}}^{\text{<calculation1>}}$  < term > select{d,d} < atom >

    declare_symbol{
      symbol{< atom >}
    }
  }
}
```

```

    derived_domain{<calculation2>, <calculation1>}
    scope{< term > } } }
meanings{
  function{integral(< term >, < atom >, < calculation2 >, < calculation1 >)}
  domains{

    R ← ∫RR R select{d, d} void
    C ← ∫CC C select{d, d} void } }
}

```

Where the function of the MR in **function** is in advance defined as below:

integral(<expression>, <integral_element>, <lower_bound>, <upper_bound>)

The **background knowledge** contains two parts: 1) a series of rules for describing the grammar of domain notations appeared in declarations and definitions, and 2) *domain inclusion relations* to dramatically decrease the number of domain rules. A typical *domain inclusion relation* has been built and shown in Figure 3.

In Figure 3, every arrow means $\supset$, and **U** means universal set. The "finite sets", such as $\mathbf{Z}_n$, and the "real intervals", such as $[a, b]$, can also be used. However it is very difficult to describe the relation between various finite sets and between various real intervals in a static knowledge representation, which needs further research in future.

Many compound domains, such as $\mathbf{R} \times \mathbf{R}$, $\mathbf{R}^{n \times m}$, $\mathbf{Q}[x]$, etc, and their inclusion relations are processed by *actions*. These *actions* declare the whole of compound domain notation, and point out which part is the fundamental domain in its notation. According to Figure 3, their inclusion relations can be determined. For example, $\mathbf{N}^{n \times m} \subset \mathbf{R}^{n \times m}$ because $\mathbf{N}$ and $\mathbf{R}$ are fundamental domains in $\square^{n \times m}$, and $\mathbf{N} \subset \mathbf{R}$. Similarly, $\mathbf{Q}[x] \subset \mathbf{R}[x]$ because $\mathbf{Q}$ and $\mathbf{R}$ are fundamental domains in $\square[x]$, and $\mathbf{Q} \subset \mathbf{R}$, etc.

All the grammatical knowledge mentioned above has been built into a knowledge base. The parser based on the knowledge-base performs a top-down lexical and syntactic analysis, and then, a bottom-up domain reduction and semantic interpretation. We use this parsing method, because mathematical expressions often include some unique structures

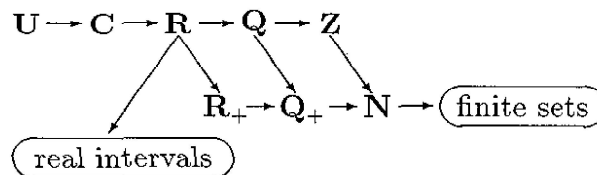


Figure 3: A typical domain inclusion relation

being attached with bound variables, such as the x within $\int_a^b f(x)dx$ and its scope is $f(x)$, which are implicitly declared beyond their scopes. Other parsing methods will be studied in future. This parsing program is summarized as follows:

Data Structure is a *declaration registration table* to register any declared or defined symbol or notation, i.e., to memorize its notations, class (variable, function, polynomial, parameter, or indeterminate symbol), domain, codomain, parameterized domain, scopes, assignment state, and its corresponding denotation in MR.

Algorithm is composed of

The input: FR, the output: MR and error message, and the method: as below.

1. If the FR has been declared, return its declared MR.
2. Match the FR with the `structure_pattern` of a rule.
If they are matched,
 if there are *actions*, execute them.
Else, goto 5.
3. Recursively parse all the sub-FRs through all categories in the `structure_pattern`.
If one of them is failed, goto 5.
Obtain all the MRs of the sub-FRs.
4. Match the domains of these MRs with every `domain_pattern` of a **meanings**.
If they are matched,
 if there are *actions*, execute them;
 return the MR: a function and a codomain.
Else,
 if there is a next **meanings**, get it and goto 4.
5. If there is not a succeed rule in the knowledge-base,
 return with error message "fail to parse".
Else, get the next rule from the knowledge-base, goto 2.

To same expression, the formalization method is not unique, and the grammar is also not unique. According to the tacit agreements, the formalization methods and their corresponding grammars can be divided into three kinds, and each includes a different parsing method. All of them and the necessities of more than one grammar can be found in (Zhao, et al., 1996a) (Zhao, 1996c).

4. FORMAL DEFINITION OF GRAMMARS

The whole set of grammars can briefly and formally be defined. At first, Σ is a set of (computer encoded) mathematical signs. For a fixed n , $\Sigma^+ = \bigcup_{k=1}^n \Sigma^k$ is a tree Kleene-closure of the Σ based on F_N . Thus $FR = \Sigma^+$. Any grammar (both syntax and semantics) or parsing system for mathematical notation based on F_N is denoted G_{FN} , and $L(G_{FN}) \subset FR$. Hence $[,]$ as meta-symbols means option below),

$$G_{FN} = (F_Z, \Sigma, V, \Sigma_c, A, P, S, F_{unc}, Dom, R, f_t, D_{IR}, D_{RT})$$

1. F_Z is the **formalization** (F_N, H_c, F_M) , where
 F_N is the **formal notation**.
 H_c is a finite set of hierarchy codes. $\forall h_c \in H_c$, h_c denotes the location of a box within F_N .
 $F_M = \{ \text{strong, weak, free} \}$ for pointing out the different formalization methods of G_{FN} (Zhao, et al., 1996a).
2. Σ is a finite set of mathematical signs.
3. V is a finite set of syntactic categories.
4. Σ_c is a finite set of computer characters. Σ_c^+ is a textual Kleene-closure of the Σ_c .
5. A is a finite set of *actions* or procedures (being similar to those of *augmented transition network*) to perform context-sensitive processing: registering a declaration, replacing a brief expression with its normal one (e.g., replacing $a < b \leq c$ with $a < b \wedge b \leq c$), realizing operator definition, etc.
 $\forall a \in A$, a has the form: $p[(q_1, \dots, q_n)]$, where p is a reserved name, $q_1, \dots, q_n \in V \cup A \cup \Sigma^+ \cup (V \cup A \cup \Sigma)^+$.
6. P is a finite set of productions. $\forall p \in P$, p has one of the following forms:
 - 1) $v \rightarrow \alpha$
 - 2) $v \rightarrow \beta[, a_1, \dots, a_n]$
 - 3) $v_1 \Rightarrow \beta \Rightarrow v_2$ only for the first parsing of the two-parsing of *free formalization*, see (Zhao, et al., 1996a) (Zhao, 1996c),
 where $\rightarrow$ means "can have the form"; the first $\Rightarrow$ means "can be well-formed as"; the second $\Rightarrow$ means "be parsed continually at"; $v, v_1, v_2 \in V$, $\alpha \in \Sigma^+$, $\beta \in (V \cup \Sigma \cup A)^+$, $a_1, \dots, a_n \in A$.
7. S is the start syntactic category of P , $S \in V$.
 (F_Z, Σ, V, P, S) independently form a tree syntax.
8. F_{unc} is a finite set of function calls written in the words from Σ_c^+ .
 $\forall g \in F_{unc}$, g has one of the following forms:
 - 1) $[.]n$
 - 2) $[.]f(p_1, \dots, p_n)$
 where $n, f \in \Sigma_c^+$, $p_1, \dots, p_n \in V \cup \Sigma_c^+ \cup A \cup F_{unc}$. F_{unc} (see G_{MD} later) is defined in MP in advance.
9. D_{om} is a finite set of mappings of mathematical fundamental domains. $\forall h \in D_{om}$, h has the form: $\gamma[\leftarrow \beta[, a_1, \dots, a_n]]$, where $\leftarrow$ means "is mapped from", $\gamma, \beta \in (\Sigma \cup A)^+$, $a_1, \dots, a_n \in A$.
10. R is a finite set of grammatic rules for describing both the top-down lexical and syntactic analysis, and the bottom-up domain reduction and semantic interpretation.
 $\forall r \in R$, r has the form: $p[, a_1, \dots, a_n][, g[, d_1, \dots, d_m]]$, where $p \in P$, $g \in F_{unc}$, $a_1, \dots, a_n \in A$, $d_1, \dots, d_m \in D_{om}$; within $p, g, a_1, \dots, a_n$, different occurrences of the same syntactic category are denoted by distinct tails of a positive integer number.

11. f_t is a transform from a variable name or function symbol in Σ^+ to its corresponding textual name in Σ_c^+ , i.e., $f_t : \Sigma^+ \rightarrow \Sigma_c^+$.
12. D_{IR} is a finite set of relations $rt(R_D)$, where rt means reflexive and transitive closure, and R_D is an inclusion relation of fundamental domains, i.e., $R_D = \{(d_1, d_2) | d_1, d_2 \in \Sigma^+ \cup \Sigma_c^+, d_1 \subset d_2\}$.
13. D_{RT} is a *declaration registration table* that is implemented by a bi-direction list: $(B_m, \dots, B_1, s, D_1, \dots, D_n)$, where B_i is for an implicitly declared bound variable; D_j is for an explicitly declared object; s is a boundary stone between the both, and also a start point of both. During the parsing, B_1 is the shallowest bound variable; B_m is the deepest one; D_1 is the first declared; D_n is the final one; the registration order is from s to the both ends; the retrieval or matching order is the left-to-right. Either B_i or D_j is a list: $(\alpha, C, \beta, \gamma, S_c, A_s, n)$, where $\alpha \in (V \cup \Sigma \cup A)^+$ for denoting the notation of the declared object; $C \in \{\text{variable, function, polynomial, parameter, indeterminate}\}$ for its class; $\beta \in \Sigma^+ \cup \Sigma_c^+$ for its domain; $\gamma \in \Sigma^+ \cup \Sigma_c^+$ for its codomain if any; $S_c \in \{(\text{from}, h_c), (\text{from_to}, h_c, h_c), (\text{within}, h_c, \dots)\}$ for its scope, where $h_c \in H_c$; $A_s \in \{0, 1\}$ for its value assignment state; $n \in \Sigma_c^+$ for its denotational textual name.

Any grammar for MR is denoted as G_{MD} , and $L(G_{MD}) = MR$. Hence,

$$G_{MD} = (\Sigma_c, V_M, P_M, S_M)$$

1. Σ_c is same as that of G_{FN} .
2. V_M is a finite set of syntactic categories for MR parsing.
3. P_M is a finite set of productions.
 $\forall p \in P_M, p$ has the form $v \rightarrow \alpha$. where $v \in V_M, \alpha \in (\Sigma_c \cup V_M)^+, \alpha$ has one of the following forms:
 - 1) $.n$
 - 2) $.f(\alpha, \dots, \alpha)$
 - 3) n, β
 - 4) $f(\alpha, \dots, \alpha), \beta$
 where $n, f \in \Sigma_c^+, \beta \in (\Sigma_c \cup V_M)^+$ for the textual name of mathematical domains.
4. S_M is the start syntactic category of P_M .
 $S_M \in V_M$.

For example, the MR of $\int_a^b f(x)dx$ is below:

integral($f(x, \langle \text{domain of } x \rangle$), $\langle \text{codomain of } f \rangle$, $x, \langle \text{domain of } x \rangle$,
 $a, \langle \text{domain of } a \rangle, b, \langle \text{domain of } b \rangle$), $\langle \text{codomain} \rangle$

5. META-LANGUAGE

A meta-language that includes 12 sentences and 38 actions (Zhao, et al., 1996a) (Zhao, 1996c) has been developed to describe all the grammars. One example of the rule for integral has been given in section 3. Through a grammar generator, a grammar written in the meta-language can be translated into a grammatical knowledge-base. The parser can translate a FR into a MR based on the grammatical knowledge-base. The meta-language can also be used to create and modify the grammars. All the contents about notation extension can be found in (Zhao, et al., 1996b) (Zhao, 1996c).

6. PROTOTYPE

Prototypes of the key-mouse WYSIWYG input, grammar generating, FR parsing, and translating from MR to the program of computer algebra systems have been developed. Three grammars have been established, and each includes about 200 notations or rules to cover elementary algebra, elementary function, linear algebra, polynomial algebra, set algebra, logic algebra, calculus, equation solving, and operator self-definition.

REFERENCES

- Kajler, N. and N. Soiffer (1994). A Survey of User Interfaces for Computer Algebra Systems. To appear in *Journal of Symbolic Computation*. Its PostScript preprint is available at <http://www.ensmp.fr/~kajler/>.
- Sakurai, T., Y. Zhao, H. Sugiura, and T. Torii (1996). A User Interface for Natural Mathematical Notations (in Japanese). *Transactions of the Japan Society for Industrial and Applied Mathematics*, **Vol.6, No.1, Mar.**, pp.147-157.
- Zhao, Y., H. Sugiura, T. Torii, and T. Sakurai (1994). A Knowledge-Based Method for Mathematical Notations Understanding. *Transactions of Information Processing Society of Japan*, **Vol.35, No.11, Nov.**, pp.2366-2381.
- Zhao, Y., T. Sakurai, H. Sugiura, and T. Torii (1996a). A Methodology of Parsing Mathematical Notation for Mathematical Computation. In: *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, Jul.24-26, 1996, ETH Zurich, Switzerland (Y.N.Lakshman (Ed.)), pp.292-300. ACM.
- Zhao, Y., T. Sakurai, H. Sugiura, and T. Torii (1996b). Notation Extension Methods in Mathematical Notation Parsing for Mathematical Computation. In: *Proceedings of Asian Symposium on Computer Mathematics*, Aug.20-22, 1996, Kobe Univ. Japan (H.Kobayashi (Ed.)), pp.81-91. Scientists.
- Zhao, Y. (1996c). *Knowledge-Based Parsing Method of Mathematical Notation for Improving Mathematical Computation Environment*. PhD Thesis, Nagoya University.