On the Logical Structure of Mathematical Notation

Dennis S. Arnon
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
415-494-4425; FAX: 415-494-4241
arnon@parc.xerox.com

Sandra A. Mamrak
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210 USA
614-292-2770; FAX: 614-292-9021
mamrak@cis.ohio-state.edu

Abstract

We show how the logical structures of a realistic class of mathematical formulae can be recovered from Plain TEX source representations, using the Centaur system, a tool for Language-Based Environments.

Introduction

A major current trend in structured document representation and processing is to distinguish the *logical* and *layout* structures of (the instances of) a given family of documents. Both ODA (Office Document Architecture) and SGML (Standard Generalized Markup Language) [3] offer tools, much akin to context-free grammars, for specifying either or both of these structures for a document class ("document type") of interest.

In general, we may say that document logical structure expresses the author's (and hopefully the reader's) organization of the material being presented, independently of how the words, formulae, and illustrations of the work are actually to be turned into marks on paper or screen. Document layout structure expresses how primitive "glyphs" (font characters, illustrations, images) are positioned and juxtaposed on display surfaces, and how a hierarchy of groupings of them (e.g., "paragraph blocks," "pages") can be identified. Both structures are usually thought of as trees, possibly with cross-links between nodes.

These general remarks specialize well to mathematical formulae, i.e., to mathematical notation. The author and reader of a technical document think about a formula in terms of its logical structure. Communication between them is achieved via a representation of the formula as a layout structure; this of course must be imaged (printed, displayed) for it to actually play its communicative role.

The logical structure of formulae is also the basis for computational applications, such as symbolic mathematical computation, that operate on their meanings, i.e., that manipulate (effective representations of) the objects *denoted* by the formulae. For example, a program to symbolically invert a matrix of polynomials would typically require a logical structure representation of the matrix, and not a layout representation. Beyond computation, the majority of information-retrieval applications one might imagine for a database of mathematical formulae (such as an online table of integrals) would use logical structure.

The high-quality mathematical typesetting that has been brought about by systems such as TEX has whetted the appetites of computational mathematicians for WYSIWYG symbolic computation, also sometimes called "direct manipulation," that provides the ability to interact directly with the pleasant-to-look-at (imaged) layout structures of formulae as they appear on the screen. The catch is that the manipulations that are desired require logical structures. And while it is now straightforward to generate layout from logical structure, going in the reverse direction is generally hard.

Building on these observations, signficant effort has been devoted recently to building WYSIWYG symbolic math systems in which logical structures of formulae are always held as the primary representation: Layout structures are generated when needed,

and links back to the logical structure are then maintained to enable desired subunits of logical structure to be inferred from (visible) selections of subunits of layout structure. (See, for example, [4] and [1].)

Nonetheless, there are numerous situations in which one starts without a logical structure representation of a formula of interest, and would like to obtain one. In this paper we shall suppose that we begin with a Plain TEX representation of a formula from a simple class of combinations of elementary functions and integrals. We then show that by using contemporary tools of Language-Based Environments we can do a reasonable job of recovering logical structure from TEX.

In the next section, we briefly discuss the Centaur system for Based Environments, which we have used. Then we specify the concrete syntax of TEX that we parse, and the abstract syntax (logical structure) we translate it into. We mention the restrictions we are forced to impose on the TEX syntax we can accept. Finally we show some examples. It should be clear that the logical structures we obtain are suitable, with minor transliteration, for input to such symbolic computation systems as Mathematica or Maple. We hope it will also be clear that we could also "unparse" our logical structures into SGML, EQN, or virtually any other concrete syntax for the logical structure of mathematical notation.

Centaur

Centaur [2] is a meta-tool for the generation of language-based environments. From a grammatical specification of a (context-free) language and executable specifications of its formal semantics, parsers, type checkers, and interpreters for it can be automatically generated. Centaur is written in Lisp and Prolog and usually runs under X-Windows.

In the next section we shall see a grammar for our class of formulae. Nonterminals in the concrete syntax rules are enclosed in angle brackets. Literal strings to match in the input stream are enclosed in double quotes. Underneath each rule is a specification of the portion of abstract syntax tree that gets built when that rule is recognized. The last part of the grammar defines the "signatures" of the abstract syntax tree, i.e., what arities they have and what "phyla" ("sorts," "types") their children must have. Finally the phyla themselves are given; each is simply a set of abstract syntax operators.

Grammar for a Class of Formulae

The appendix shows the grammar for each of the example expressions. Here are some properties to note:

1. Variables are restricted to single letters; integer constants are restricted to single digits.

2. Non-character integrands must be single chars or TEX subformulae, i.e., enclosed in braces. Also, the args of special functions (currently sin, cos, log, exp, prime) must be characters or subformulae. These requirements simplify the grammar and parsing.

3. Multiplication is denoted by asterisk. This avoids three shift/reduce conflicts from yacc.

4. All integrals are represented by instances of a single abstract syntax operator. Formatting routines need to handle this appropriately (e.g., not print the "d" for a null (defaulted) variable of integration).

5. We assume that prime of an expression means derivative with respect to its main variable, and that there is some clear way to know what the main variable is (e.g., the expression has only one variable). It is the user's job to enclose the argument of prime in parentheses to prevent ambiguity. Similarly the args of sin and cos must be in parentheses.

6. Exponential function must be done as exp, not e to the x.

Examples

The following are examples of expressions accepted by our concrete grammar.

$$\int_{t+u*5}^{x} \frac{e^{-x^2}+e^{x^3}}{s*x^2+c^2*x}dx$$

$$\int \frac{e^{-x^2}+e^{x^3}}{s*x^2+c^2*x}dx$$

$$\int \frac{x}{(x-1)*(x+2)}dx$$

$$\int_{1}^{z} f*x*g*hdx$$

$$\int_{1}^{a}\int_{1}^{b}\int_{1}^{c} e^{x+y+z}dxdydz$$

$$\int_1 (a+b+c)*(e^{-x^2}+e^{x^3})dx$$

$$\int \frac{\exp -x^2 + \exp x^3}{\sin x^2 + \cos x^2} dx$$

$$\int \sin\log x dx$$

$$(\int \sin\log x dx)'$$

$$(\frac{-(x*\cos(\log(x)))}{2} + \frac{x*\sin(\log(x))}{2})'$$

$$(\frac{2*x^5}{5} - \sqrt{\frac{2*x^5*\log x}{5}} + \frac{x^5*\log x^2}{5})'$$

$$\int_{t+u*5}^{x} \frac{\int_{t+u*5}^{x} \frac{e^{-x^2}+e^{x^3}}{s*x^2+c^2*x} dx}{s*x^2+c^2*x} dx$$

Figure 1 shows the abstract syntax tree for the last expression.

```
integral(
  quotient(
    integral(
      quotient(
        sum(power(e, power(negate(x), 2)), power(e, power(x, 3))),
        sum(times(s, power(x, 2)), times(power(c, 2), x))), x,
      sum(t, times(u, 5)), x),
    sum(times(s, power(x, 2)), times(power(c, 2), x))), x,
  sum(t, times(u, 5)), x)
```

Figure 1: Abstract syntax tree

References

[1] D. Arnon, R. Beach, K. McIsaac, and C. Waldspurger. Caminoreal: an interactive mathematical notebook. In *Proc. Intl. Conf. on Electronic Publishing, Document Manipulation, and Typography*, pages 1–18. Cambridge University Press, April 20-22 1988. (J.C. van Vliet, ed.) ISBN 0-521-36294-6.

[2] P. Borras et. al. Centaur: the system. In *Proceedings of the SIGSOFT'88, Third Annual Symposium on Software Development Environments*, 1988. Boston, Massachusetts.

[3] *Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML)*, October 1986. ISO 8879-1986 (E).

[4] Neil Soiffer. The design of a user interface for computer algebra systems. Technical Report UCB/CSD 91/626, Computer Science Division (EECS), University of California, Berkeley, April 1991.

References

[1] D. Arnon, R. Beach, K. McIsaac, and C. Waldspurger. Caminoreal: An interactive mathematical notebook. In *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, pages 1–18. Cambridge University Press, April 20–22, 1988. (J.C. van Vliet, ed.) ISBN 0-521-36294-6.

[2] P. Borras et. al. Centaur: The system. In *Proceedings of the* SIGSOFT'88, *Third Annual Symposium on Software Development Environments*, 1988. Boston, Massachusetts.

[3] *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, October 1986. ISO 8879-1986 (E).

[4] Neil Soiffer. The design of a user interface for computer algebra systems. Technical Report UCB/CSD 91/626, Computer Science Division (EECS), University of California, Berkeley, April 1991.

Appendix

```
definition of texMath is
rules
<markedTexExpr> ::= "$$" <texExpr> "$$" ;
<texExpr>

<markedTexExpr> ::= "\(" <texExpr> "\)" ;
<texExpr>

<markedTexExpr> ::= "$" <texExpr> "$" ;
<texExpr>

<texExpr> ::= "\int" <null_limit> <null_limit> <integrand> <null_name> ;
integral(<integrand>, <null_name>, <null_limit>.1, <null_limit>.2)

<texExpr> ::= "\int" <null_limit> <null_limit><integrand> "d" <name> ;
integral(<integrand>, <name>, <null_limit>.1, <null_limit>.2)

<texExpr> ::= "\int" "_" <lowerLimit><null_limit><integrand> <null_name> ;
integral(<integrand>, <null_name>, <lowerLimit>, <null_limit>)

<texExpr> ::= "\int" "_" <lowerLimit> <null_limit> <integrand> "d" <name> ;
integral(<integrand>, <name>, <lowerLimit>, <null_limit>)

<texExpr> ::= "\int" <null_limit> "^" <upperLimit><integrand> <null_name>;
integral(<integrand>, <null_name>, <null_limit>, <upperLimit>)

<texExpr> ::= "\int" <null_limit> "^" <upperLimit><integrand> "d" <name>;
integral(<integrand>, <name>, <null_limit>, <upperLimit>)

<texExpr> ::= "\int" "_" <lowerLimit> "^" <upperLimit><integrand>  <null_name>;
integral(<integrand>, <null_name>, <lowerLimit>, <upperLimit>)

<texExpr> ::= "\int" "_" <lowerLimit> "^" <upperLimit><integrand> "d" <name> ;
integral(<integrand>, <name>, <lowerLimit>, <upperLimit>)

<integrand> ::= <bracedTexExprOrChar> ;
<bracedTexExprOrChar>

<lowerLimit> ::= <bracedTexExprOrChar> ;
<bracedTexExprOrChar>

<upperLimit> ::= <bracedTexExprOrChar> ;
<bracedTexExprOrChar>

<texExpr> ::= "\frac" <numerator><denominator> ;
quotient(<numerator>, <denominator>)

<numerator> ::= <bracedTexExprOrChar> ;
<bracedTexExprOrChar>

<denominator> ::= <bracedTexExprOrChar> ;
<bracedTexExprOrChar>
```

```
<texExpr> ::= <texFactor> ;
<texFactor>

<texExpr> ::= <texExpr> "+" <texFactor> ;
sum( <texExpr>, <texFactor>)

<texExpr> ::= <texExpr> "-" <texFactor> ;
difference( <texExpr>, <texFactor> )

<texFactor> ::= <texPower> ;
<texPower>

<texFactor> ::= <texFactor> "*" <texPower> ;
times(<texFactor>, <texPower>)

<texFactor> ::= <texFactor> "\over" <texPower> ;
quotient(<texFactor>, <texPower>)

<texPower> ::= <texUnary> ;
<texUnary>

<texPower> ::= <texPower> "^" <texUnary> ;
power(<texPower>, <texUnary>)

<texUnary> ::= <texTerm> ;
<texTerm>

<texUnary> ::= "-" <texTerm> ;
negate(<texTerm>)

<texTerm> ::= <bracedTexExprOrChar> ;
<bracedTexExprOrChar>

<texTerm> ::= "(" <texExpr> ")" ;
<texExpr>

<texTerm> ::= "\sin" <bracedTexExprOrChar>  ;
sin(<bracedTexExprOrChar>)

<texTerm> ::= "\cos" <bracedTexExprOrChar>  ;
cos(<bracedTexExprOrChar>)

<texTerm> ::= "\log" <bracedTexExprOrChar>  ;
log(<bracedTexExprOrChar>)

<texTerm> ::= "\exp" <bracedTexExprOrChar>  ;
exp(<bracedTexExprOrChar>)

<texTerm> ::= "\sqrt" <bracedTexExprOrChar>  ;
sqrt(<bracedTexExprOrChar>)

<texTerm> ::= <bracedTexExprOrChar> "\prime"  ;
derivative(<bracedTexExprOrChar>)
```

```
<texTerm> ::= <bracedTexExprOrChar> "'"  ;
derivative(<bracedTexExprOrChar>)

<bracedTexExprOrChar> ::= <char> ;
<char>

<bracedTexExprOrChar> ::= <bracedTexExpr> ;
<bracedTexExpr>

<bracedTexExpr> ::= "{" <texExpr> "}" ;
<texExpr>

<char> ::= <name> ;
<name>

<char> ::= <digit> ;
<digit>

<name> ::= %LETTER ;
name-atom (%LETTER)

<digit> ::= %DIGIT ;
digit-atom (%DIGIT)

<null_name> ::=  ;
null_inst ()

<null_limit> ::=  ;
null_inst ()


abstract syntax
integral -> EXP NAME EXP EXP;
quotient -> EXP EXP ;
power -> EXP EXP ;
sum -> EXP EXP ;
difference -> EXP EXP ;
times -> EXP EXP ;
negate -> EXP ;
sin -> EXP ;
cos -> EXP ;
log -> EXP ;
exp -> EXP ;
sqrt -> EXP ;
derivative -> EXP ;
name -> implemented as IDENTIFIER ;
digit -> implemented as STRING ;
null_inst -> implemented as SINGLETON ;

EXP ::= integral quotient power sum difference times negate sin cos log
exp sqrt derivative NAME digit null_inst ;
NAME ::= name ;

end definition
```