Documents are not just for printing

T. V. Raman *
Center for Applied Mathematics
Cornell University
Ithaca NY 14853
Phone: (607)255-9202
E-mail: ⟨raman@cs.cornell.edu⟩

March 8, 1993

Abstract

Modern use has emphasized the written/printed nature of documents to the extent that today documents imply writing on paper. This trend is well reflected by document processing systems that can produce high quality paper output. The world of electronic documents currently abounds with a plethora of markup languages, document encodings and typesetting systems designed to facilitate the production of visual documents. In the process, the principal function of the document as a vehicle of information has been overlooked.

Our goal is to generate audio documents with little extra effort from the same electronic source used to produce visual documents. The principle focus of this paper is the high-level representations that we need in order to generate audio documents. Capturing two-dimensional document content, such as mathematical formulae, and rendering it in audio forms a major part of our work. An audio document generated from the electronic source used to typeset this printed copy is available.

*Supported by a summer Fellowship from the Xerox Design Research Institute. Supported by a Graduate Fellowship from the INTEL Foundation.

Contents

1 Introduction.

Audio documents have long meant books on cassette tape. These are recorded by trained readers and require both time and effort to produce. The advent of the reading machine resulted in the first computer-generated spoken readings of printed documents. However, the approach used, namely to scan printed text and convert it to speech ignores the issue of audio formatting. A printed document uses visual layout to convey document structure. Such cues are dropped when the scanned text is directly converted to speech. In addition, two-dimensional document content such as mathematical equations and tables are not handled by current day reading machines. Thus the true computer generated audio document continues to remain a thing of the future.

The electronic encoding of a document captures extra-textual information that can be exploited to generate audio renderings of the document. This was the principle motivation behind TEXTALK [Ram91, Ram92], a program that transformed LaTeX source to generate text similar to what a human reading the typeset document would say. The approach used by this program revealed several shortcomings and pointed the way ahead.

TEXTALK carried out simple string substitution on the linear representation of the document as embodied in the LaTeX [Lam86] source. This approach works only locally and is incapable of handling nested constructs. In addition, string substitution has severe limitations when it comes to handling examples that are context sensitive. Consider for instance the following naïve substitution rule for reading integrals.

$$\int_a^b f(x)\,dx$$

Here is the TeX source for the above expression.

```
$$\int_a^b f(x)\dx$$
```

A simple substitution rule that among other things replaces `_` with "from" and `^` with "to" would generate

```
integral from a to b f ( x ) dx
```

But this rule will cause trouble since subscripts do not always mean lower limits.

The above example is a bit contrived in that a more general rule would substitute `_` with "sub"; but it emphasizes the need for a high-level representation of the document content to generate reliable audio renderings. The rest of the paper considers representing mathematical content. This is because many of the representational issues for general document content are already being addressed by systems like SGML. [SGM86, Org90]

2 A structure based approach.

To overcome the shortcomings of string substitution, consider first parsing the linear encoding of the document into a high-level representation. Audio formatting can then be achieved by applying audio rendering rules to this high-level structure. This approach will also allow us to benefit from other ongoing work in the electronic documents community aimed at document recognition [AM91, Arn92, Arn91, BB90, MOB90, PR92], since we can obtain audio documents by applying audio formatting rules to the high-level structures obtained from sources as varied as document scanning systems and structure recognition from postscript documents.

2.1 Structure sensitive readings

Assume that we have parsed the mathematical content into a high-level form. In an early prototype implementation at PARC[1] in summer 1991, we assumed this to be a prefix form of the corresponding mathematical

[1] This was joint work with Dr. Dennis S. Arnon of Xerox PARC.

expression. In fact we may never be able to get such a high-level representation from written mathematics, since mathematical notation uses the same visual cues in different contexts to mean different things and is thus inherently ambiguous. In an ideal world, the electronic encoding of the mathematical expression would capture the full semantics, leaving it to a formatting program to generate the appropriate written notation. Instead, current-day document encodings directly capture the written notation, making it impossible to extract a prefix form of all mathematical expressions. In fact, it can be argued that the very nature of mathematical notation, which is characterized by frequent overloading of *visual layout* operators, precludes any document encoding that captures all the semantic content of a mathematical expression.

2.2 A more feasible representation

This section considers a representation that appears more attainable from the written notation. Since written mathematics uses superscripts to cue both exponentiation as well as other concepts, all that it gives us in terms of information is whether a "box" appears as a subscript or a superscript to another "box". The contents of these boxes could themselves be arbitrarily complex expressions. Thus the structure we use to capture the mathematical content allow constructs to have subscripts and superscripts that are themselves complex sub-expressions without assigning any semantic interpretation to these. In general, an "object" in written mathematics can have subscripts and superscripts on either side and accent marks above and below the object. In this description, "object" is used loosely; in practice the object itself could be a mathematical expression. Thus the nodes in our tree representation of the formula can now have additional attributes rather than being simple objects. In addition, the representation should possibly capture the explicit use of different visual delimiters. Though we can generate delimiters given the precedence and associativity of operators appearing in a prefix representation of an expression, capturing visual delimiters explicitly can give us more information in situations where different types of visual delimiters have been used. We could model matching pairs of delimiters as unary operators. To make this statement more precise, we would represent $(x + y)$ by a tree whose root note is the paren operator and whose single child is the subtree corresponding to the expression being parenthesized. At this stage, the parenthesized expression is regarded simply as a list of boxes, and if more information is available about the contents of these, as in this case with the $+$ being a binary operator, this information is used to transform the list of boxes into the tree corresponding to the expression $x + y$ in prefix form.

The result of performing these steps is a tree-like representation of the formula, where subexpressions that are completely unambiguous are parsed into a prefix form while retaining all the visual information available about those pieces of the formula that cannot be interpreted unambiguously. This has the advantage of delaying the assignment of semantic interpretation in cases where insufficient domain specific information is available at the initial recognition step.

Here is a more rigorous definition of the tree-like structure described above. Expressions are represented by trees whose nodes can have the following attributes:

1. North: For overbars accents etc.

2. South: For underbars etc.

3. North-East: For right superscripts.

4. South-East: For right subscripts.

5. North-West: For left superscripts.

6. South-West: For left subscripts.

In addition to attributes, nodes of the tree also have content, and this content can be a simple variable or a sub-expression. The attributes themselves can have similar content. The structure is thus recursive.

The primary advantage with the above is that it appears attainable from written notation, as opposed to a pure prefix form. We now need to consider whether this representation is adequate for audio formatting the given expression.

Given a formula having no ambiguity, the above gives us the prefix representation of the expression. Adding in the missing pieces i.e., visual cues having multiple interpretations such as subscripts and superscripts, which any encoding of the electronic document will have to tag explicitly in order to achieve the visual layout, we find that the best we can now do is to capture this visual information, in the form of attributes as discussed above. Thus the representation loses no information as compared to the written notation with which we started.

Having such a structure will allow us to generate better audio renderings than if we were to use a representation that simply captures the geometric layout on paper. This is because with additional information, we will be able to decide on the order in which subexpressions are read, use variable substitution as in λ-calculus when reading out complex expressions, etc. This is the primary advantage in recognizing the bits of the expression that are completely unambiguous into a prefix form. Generating audio renderings for such structures now comes down to a proper choice of rendering rules that, in the absence of domain specific information, convey the written form of the expression.

Thus we can think of this as asking a naïve reader to read aloud a mathematical expression that he does not understand. The best such a reader can do in this situation is to explicitly describe the notation, i.e., say "superscript" etc.

To make this more explicit, given $D_x^1 u$, which would be written in TEX as `$D_x^1 u$`, we would build the following structure to represent this expression.

The root node of the tree that represents the inline formula has as its content the *juxtaposition* operator, (See Section 3.5 and line 5 of figure 3), and has two children, one representing D_x^1 and the other u. The node corresponding to u is a simple node with content u. The node representing D_x^1 has content D, (See line 10 of figure 3.) and two attributes, a `superscript` attribute with content 1 and a `subscript` attribute with content x.

See appendix A for more details of the representation generated by the recognizer. Figure 3 displays the CLOS object that represents this expression.

Given the above structure and no additional domain specific information, the reader would say:

cap d sub x super 1 u

using additional cues to indicate the end of the subscript and superscript. Since this example is simple, we could generate the same reading using string substitution. However, the advantages become immediately apparent if we consider expressions where the content of the nodes and attributes are themselves complex subexpressions.

Given more information, e.g., D denotes the derivative, we could interpret the subscript and superscript in context by applying the rule, the cardinal-number(*superscript*) derivative with respect to the *subscript* of *the object that follows*, where cardinal-number(1) = first, cardinal-number(2) = second, etc. and now generate

First Derivative with respect to x of u

We have a whole range of possible readings that fall between the two extremes indicated above. The advantage with generating an *audio layout* of the written notation in the absence of domain specific information is that a knowledgable listener, like a knowledgable reader who sees the printed form, will be able to assign the right semantic interpretation to what he hears. (See appendix C.1 for another example of audio layout.)

2.3 Summary

To summarize, let M denote the actual math object and W its printed representation. The mapping $M \rightarrow W$ is not one-one. Different M get mapped to the same W. Let S denote the spoken representation. We can try to get S from either M or W. Since the mapping from the mathematical object to the printed form is not one-one, we need additional information in order to recover M given W. Hence if we try to get S from W directly we again need additional information in order to generate spoken renderings similar to what we would get if we were starting with M. In an ideal world, the electronic encoding of the document would capture M directly, thus giving us all the information we needed. Current electronic encodings capture W directly. Since W is ambiguous and does not give complete semantic information, we need to generate a spoken description of W that will allow a listener who has the necessary domain specific information to correctly interpret the spoken form. Thus we can regard S_w as a base-level spoken form, or *audio layout*, for M. Using S_w, we would read $D_x u$ as "cap d sub x u". Adding domain specific information will allow us to generate readings that are closer to what we would generate if we were starting from M, i.e., in this case `(derivative u x)`.

3 Recognizing written mathematics.

This section considers recognizing written mathematics and building up a representation as defined in the previous section. When we consider recognizing mathematics in electronic documents, we run into several problems caused by the inherent ambiguity of written mathematical notation. Mathematical notation overloads many of the *visual layout* operators to achieve conciseness. Thus subscripting and superscripting are used to mean different things in different contexts. What is more, often the information present on paper is incomplete and needs to be correctly interpreted by the reader. Thus the notation $\sum_{i=1}^{n} A_i$ is inherently incomplete. Moreover the same expression can be written as $\sum_{1\leq i\leq n} A_i$. An ideal encoding for the summation operator would be $\mathrm{SUM}(i, r, e)$ where i is the index of summation, r is the range, and e is the expression being summed.

In order to recognize the summation operator encoded in its various forms and generate audio renderings of the resulting expression we need to build up a representation that captures all the information present in the electronic encoding. The representation we build needs to have the following characteristics:

1. Capture how the expression is written.

2. Capture as far as possible the meaning of the expression.

The first of the above is required to generate readings that reflect how the expression is written. The second is essential to generate renderings where the reading order is not exclusively determined by the linearized form in which the expression appears in the electronic document.

3.1 Building up representations from written notation

We begin recognizing such mathematical operators by systematically building up a representation that captures the written notation. The first step is to recognize each of the operators that occur in the expression. For instance the summation operator can appear with a possible subscript and/or superscript. We first recognize this and build up the summation operator with attributes subscript and superscript, both of which can themselves be math expressions. At this stage, we do not interpret the subscript and superscript, but leave this for later. So the summation operator along with its attributes is now just a unary operator, and in order to recognize the rest of the expression, we need to assign it the correct precedence among all the operators. Doing this will allow us to construct a tree like representation where the expression being summed over appears as a child to the annotated summation operator.

Other *large* operators, like the integral and set union, that normally occur in exactly the same position as the summation operator can be handled similarly. Thus when recognizing

$$\bigcap_{X_i \in X} A \cap X_i = \emptyset$$

we would recognize the $\bigcap_{X_i \in A}$ as a *large* operator that is being applied to the subexpression $A \cap X_i$. The above expression also shows that the large operators like summation have a lower precedence than the arithmetical operators but have a higher precedence than the relational operators.

3.2 Sorting operators by precedence

The following section attempts to sort the various operators according to their precedence. Though this is relatively straight-forward when it comes to sorting the standard arithmetic and logical operators, things are made a little different by the presence of the large operators described above as well as the *arrow operators* that occur in typeset mathematics. Another operator that is normally not considered when organizing operators according to their precedence are the quantifiers, as in

$$\forall x, y{:}\, Set.\ \exists p{:}\, Set.\ (x \in p \ \&\ y \in p)$$

The following enumerates the various types of mathematical objects, in order of their precedence.

1. Relational operators. e.g., $=$, $<$.
2. Arrow operators. e.g., $\longrightarrow$.
3. Large operators. e.g., $\int$.
4. Infix logical operators. Among these $\wedge$ has higher precedence than $\vee$.
5. Infix arithmetical operators. Among these $+$ has lower precedence than $*$.
6. Parenthesized subexpressions.
7. Function applications. e.g., sin. These are unary, right associative and, except in some cases (See Section 3.3), have the highest precedence.

Among the operators listed above, the arrow operators may need finer classification, since some arrow operators get used as relationals while others are used to denote logical implication, function mapping, limits etc.

3.3 Function application

The previous subsection listed the precedence of function application as being the highest and right associative. At least in computer science this is true. However, in written math notation the use of juxtaposition to denote multiplication leads to a notational convenience which allows authors to violate the fact that function application has highest precedence. Thus, in the expression $\sin 2\pi x$, the trigonometric ratio is being applied to $2\pi x$, whereas assigning highest precedence to function application would result in the expression being interpreted as $(\sin 2)\pi x$. In fact, current convention in written mathematics appears to dictate that the precedence for function application is higher than that of addition and subtraction but lower than juxtaposition used to denote multiplication. Thus, in the expression $\sin(a + b) = \sin a \cos b + \cos a \sin b$, the segment $\sin a \cos b$ is interpreted as the product $\sin a * \cos b$ rather than $\sin(a * \cos b)$ as a result of the function application binding the tightest. To cite another example, in the identity $\sin 2x = 2 \sin x \cos x$, the left hand side relies on the lower precedence of the sin, while the right hand side relies on the right associativity of the function application. (see appendix $B.7$ for more examples of similar ambiguities handled by our recognizer.)

3.4 Encoding document content unambiguously

Visual math notation is inherently ambiguous and gets most of its expressiveness by freely overloading standard visual layout operators. It would be too restrictive to insist that all of a mathematical object's semantics appear explicitly in the markup, since this would make inventing new notation in a text developing new concepts cumbersome if not impossible. So to an extent we will never be able to attach semantic meaning to every object occurring in the document; but we insist that a document encoding should not use the same markup to encode objects having the same visual layout but different meanings. This will allow a recognizer to process the objects occurring in the document according to their semantic interpretation. To make the above statement concrete, consider the following examples. The visual layout of stacking a mathematical object above another separated by a horizontal rule could be used in several contexts.

1. To denote a fraction.
$$\frac{1+\sqrt{5}}{2}$$

2. In stating an inductive proof.
$$\frac{\frac{}{P(0)} \quad \frac{x{:}N,\ P(x)}{P(S(x))}}{\forall x{:}\,N.\,P(x)}$$

3. In stating an inference rule, where an inference is written as $\frac{\text{premise}}{\text{conclusion}}$ as in
$$\frac{X=Y,\ Y=Z}{X=Z}$$

An author could encode any of the above in a document preparation system like LaTeX using the same macro[2]. If the document uses `\frac` to encode only one of the above cases, a recognizer can unambiguously interpret occurrences of this encoding. Using the encoding `\frac{object-1}{object-2}` in all of the cases enumerated above within the same document makes it impossible to disambiguate between the various interpretations. When an author wishes to use the same layout to mean different things, the different occurrences should be marked up distinctly. This would not be difficult to do, for instance in LaTeX, the author would extend the markup by defining three new macros:

1. `\newcommand{\fraction}[2]{\frac{#1}{#2}}`.
2. `\newcommand{\induction_clause}[2]{\frac{#1}{#2}}`.
3. `\newcommand{\inference}[2]{\frac{#1}{#2}}`.

3.5 Adjoining additional operators.

In the absence of such unambiguous markup, the important thing is to realize that the same written notation can mean more than one thing and to allow for this in our representation by explicitly capturing the visual notation rather than one of its several possible semantic interpretations. To an extent this has already been achieved by the representation described so far, which treats subscripts and superscripts as visual attributes rather than interpreting the superscript as an exponent. In addition we need to introduce some extra operators in our representation that do not correspond to any single mathematical operator, but are purely a result of the overloading of notation in written mathematics.

A good example of the above is the use of juxtaposition in written mathematics to mean multiplication. This is notationally ambiguous, and representing abc as the product of a, b, and c would be unsuitable since

[2] In fact, that is how they are encoded in this document!

later on we will not be able to distinguish between products that were denoted unambiguously in the text and products that were assumed to be products based on one possible interpretation of the visual notation. Thus we would not be able to correctly handle $f(g)$, if we were to learn later that the author of the document uses f to denote a function.

To make the above more concrete, let W denote a piece of written math notation, and let $\{M_i\}$ denote all of its possible mathematical meanings. If we choose any specific $M_j \in \{M_i\}$ as the representation of W, our representation will not distinguish occurrences of M_j that appear explicitly in the text from occurrences of W. In order to avoid such pitfalls, we need to introduce an operator corresponding to W in our representation and leave the interpretation of W for later. Thus for each piece of mathematical notation W that is potentially ambiguous we introduce an operator M_w in our representation. If no further information is available at the rendering step, we can now at least generate S_w from M_w, leaving it to the listener to interpret the rendering in context. The first example of such an operator is the *juxtaposition* operator, which is assigned the same precedence as multiplication. Other possible operators of this type would include the TEX`\atop`, which is used to build up a stack of mathematical objects. When used in a subscript to a summation as in the following example:

$$\sum_{{\scriptstyle 1\le i\le p \atop \scriptstyle 1\le j\le q} \atop \scriptstyle 1\le k\le r} a_{ij}b_{jk}c_{ki}=1$$

which would be marked up as:

```
$$\sum_{{\scriptstyle1\le i\le p\\
\atop\scriptstyle1\le j\le q} \\
\atop\scriptstyle1\le k\le r}\\
a_{ij}b_{jk}c_{ki}=1$$
```

`\atop` could be interpreted as a conjunction since this is what the written notation means. However representing `\atop` as a conjunction would be incorrect. In the phrase, ${p \atop q}$, which would be marked up as `$p\atop q$`, `\atop` clearly does not denote a conjunction.

Introducing such operators with their own precedence allows us to reliably recognize such constructs appearing in the typesetting source, because any typesetting program that provides such operators as `\atop` also necessarily assigns them a precedence. For example, TEX treats `\atop` as an infix binary operator having the least precedence. To add a TEX-specific comment, TEX treats the `\over` operator similarly, i.e., as n infix binary operator that has least precedence, which is why careless use of `\over` can have disastrous effects like typesetting the body of an entire equation as the numerator etc.

4 Some results.

We have implemented the ideas described in this paper and our recognizer now works with mathematical expressions present in (LA)TEX documents. All of the standard TEX mathematical operators are recognized. The recognizer is extensible in the sense that the user can specify additional mathematical operators.

The ideas discussed in Section 3.5 for handling juxtaposition in written mathematical notation have also been successfully implemented. (See figure3 for the result of recognizing the juxtaposition operator.) We produce different readings for the same object based on the currently active reading style. Reading styles vary from being purely layout oriented, which produce "audio layout", to being subject specific. The richness of the high-level structures we generate enables us to produce context sensitive readings of complex mathematical expressions.

Here are a few examples from our current implementation.

Consider the following expressions:

$$x_1^k + x_2^k + x_3^k + \cdots + x_n^k = 0$$

$$x^{k_1} + x^{k_2} + x^{k_3} + \cdots + x^{k_n} = 0$$

Simple string substitution applied to the TEX encoding would result in the two expressions being read alike. Using the representation described in this paper, our audio formatter generates distinct audio renderings for the above expressions.

Using the internal representation described in this paper, reading rules can be written to produce audio renderings that vary from being layout oriented to being fully descriptive. Layout oriented readings convey little or no semantic information, leaving it to the reader to infer the meaning. For instance, a layout oriented reading of an integral renders the lower and upper limits using different audio states. Thus the reading conveys the contents of the subscript and superscript unambiguously, leaving it to the listener to interpret them as the limits of integration. (See appendix C.1 for an example.) A descriptive reading on the other hand interprets these in context, and also tries to infer the variable of integration. (See appendix C.2 for an example.)

Our internal representation by being general allows us to write reading rules that can interpret the written notation in context. The example discussed in appendix A is a case in point. The audio formatter produces specialized readings where it can correctly interpret the meaning of the subscript and superscript as the variable of differentiation and order of the derivative respectively, while allowing the user to easily turn off this feature, or alternatively specify other semantic interpretations.

Refer to appendix B for a collection of different mathematical expressions handled by our recognizer. This collection has been chosen to highlight different features of the recognizer and audio formatter. Several cases of ambiguous visual notation alluded to in the text are shown. A cassette tape containing audio renderings of these expressions can be obtained upon request. An audio rendering of this paper generated from the same (LA)TEX file as the printed version is also available.

5 Acknowledgments.

I would like to thank David Gries, Bruce Donald, Dennis Arnon, and Jim Davis for their advice and encouragement. The Cornell Computer Science Robotics and Vision Laboratory provided support during the initial phase of this project. Intel Corporation supported this work with a graduate fellowship for the academic year 1992–93. And Xerox Corporation provided summer support in 1991 and 1992 and an equipment grant.

References

[AM91] Dennis S. Arnon and S. Mamrak. On the logical structure of mathematical notation. *Proceedings of the TEX Users Group*, 12:479–484, July 1991.

[Arn91] Dennis Arnon. *DocTypes: A Methodology for Managing Structured Documents of Multiple Types*, April 1991. Draft Manuscript.

[Arn92] Dennis S. Arnon. Model-directed conversions of lateX documents. *Proceedings of the TEX Users Group*, July 1992. To be published.

[BB90] Allen L. Brown and Howard Blair. A logic grammar foundation for document representation and document layout. In *Proc. Intl. Conf. on Electronic Publishing, Document Manipulation, and Typography*, pages 47–64. Cambridge University Press, September 1990. R. Furuta, ed. ISBN 0-521-40246-8.

[Knu84] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[Lam86] Leslie Lamport. LATEX*: A Document Preparation System.* Addison-Wesley Pub. Co., Reading Mass., 1986.

[MOB90] S. A. Mamrak, C. S. O'Connell, and J. A. Barnes. The integrated chameleon architecture: A software toolset to support data translation. Technical Report OSU-CISRC-11/90-TR37, Department of Computer and Information Science, The Ohio State University, November 1990.

[Org90] International Standards Organization. *Information Technology: SGML Support Facilities: Techniques for Using SGML*, 1990. Draft.

[PR92] Gilbert B. Porter and Emil V. Rainero. Document reconstruction: A system for recovering document structure for layout. *Electronic Publishing*, 1992.

[Ram91] T. V. Raman. TEXTALK. *TUGboat*, 12:178, March 1991.

[Ram92] T. V. Raman. An audio view of (LA)TEX documents. *Proceedings of the TEX Users Group*, 13:372–379, July 1992.

[SGM86] *Information Processing: Text and Office Systems: Standard Generalized Markup Language SGML*, October 1986. ISO 8879-1986 E.

Appendix

A Representations generated by current implementation.

This section shows the representations generated by the recognizer for the expression shown in figure 1. Figure 2 shows the (LA)TEX source that is the input to the recognizer, and figure 3 displays the CLOS representation returned by the recognizer.

In figure 3, the CLOS object is shown with its slot names and slot values. All document objects are linked to their parent as well as their next and previous siblings. The notation `#N#` where $N \in \{1, 2, 3, \cdots\}$ is used in the printed representation to avoid circularities while displaying the object. In figure 3, `#1#` represents the document object containing the expression.

The `weight` slot holds a measure of the complexity of math objects. The `pattern` slot is used by the recognizer to mark special patterns in the input. These special patterns are then used by reading rules to generate specialized readings. For the example shown in figure 1, the subexpression D_x^1 is recognized as a derivative, and this is marked in the corresponding `pattern` slot of the object that represents D. (See line 24 of figure 3.) The special pattern matching rules that help the recognizer identify D_x^1 as a derivative handle the occurence D i.e., with no attributes as an ordinary variable. Thus the D in the distance formula (see appendix B.14 is correctly handled.

Figure 1: Sample expression

$$d_x^1 u$$

Figure 2: Input to recognizer:

```
$D_x^1u$.
```

Figure 3: Output of recognizer:

```
(#2=[[INLINE-MATH
 PARENT: #1#NEXT:NIL PREVIOUS: NIL
 CONTENTS: (#3=[[JUXTAPOSITION
            PARENT: #2#NEXT: NIL PREVIOUS: NIL
            CONTENTS: "juxtaposition"
            CHILDREN:(#4=[[ORDINARY
                      PARENT: #3#
                      NEXT: #5=[[ORDINARY
                      PARENT: #3#NEXT: NIL PREVIOUS: #4#
                      CONTENTS: "u"
                      CHILDREN: NIL PATTERN: NIL WEIGHT: 1
                      ATTRIBUTES: NIL ]]
                      PREVIOUS: NIL
                      CONTENTS: "D"
                      CHILDREN: NIL PATTERN: derivative WEIGHT: 3
                      ATTRIBUTES: ([[SUBSCRIPT
                                   PARENT: #4#NEXT: NIL PREVIOUS: NIL
                                   NAME: SUBSCRIPT
                                   VALUE: [[ORDINARY
                                   PARENT: #4#NEXT: NIL PREVIOUS: NIL
                                   CONTENTS: "x"
                                   CHILDREN: NIL PATTERN: NIL WEIGHT: 1
                                   ATTRIBUTES: NIL ]]
                                   PATTERN: NIL ]]
                                   [[SUPERSCRIPT
                                   PARENT: #4#NEXT: NIL PREVIOUS: NIL
                                   NAME: SUPERSCRIPT
                                   VALUE: [[MATH-NUMBER
                                   PARENT: #4#NEXT: NIL PREVIOUS: NIL
                                   CONTENTS: "1"
                                   CHILDREN: NIL PATTERN: NIL WEIGHT: 1
                                   ATTRIBUTES: NIL ]]
                                   PATTERN: NIL ]])
                      ]]
                      #5# )
            PATTERN:  nil WEIGHT: 5 ATTRIBUTES: NIL N- ARGS: 2
            ]])
 ]])
```

B A collection of math expressions.

B.1 simple fractions and expressions.

$$a+b+c+d$$

$$a+\frac{b}{c}+d$$

$$\frac{a+b}{c+d}$$

$$\frac{a}{b}+c+d$$

$$\frac{a}{b+c+d}$$

$$a+\frac{b+c}{d+e}+x$$

$$a+bc+d$$

$$(a+b)(c+d)$$

B.2 superscripts and subscripts.

$$x_1^k+x_2^k+x_3^k+\cdots+x_n^k=0$$

$$x^{k_1}+x^{k_2}+x^{k_3}+\cdots+x^{k_n}=0$$

$$x_{k^1}+x_{k^2}+x_{k^3}+\cdots+x_{k^n}=0$$

$$x^{k^1}+x^{k^2}+x^{k^3}+\cdots+x^{k^n}=0$$

$$x_{k_1}+x_{k_2}+x_{k_3}+\cdots+x_{k_n}=0$$

$$x+_n y+_n z$$

B.3 Knuth's examples of fractions and exponents.

Note: The examples in this subsection have been taken from the TEX Book [Knu84].

$$\frac{x+y^2}{k+1}$$

$$\frac{x+y^2}{k}+1$$

$$x+\frac{y^2}{k+1}$$

$$x+\frac{y^2}{k}+1$$

$$x+y^{\frac{2}{k+1}}$$

$$x^{2^y} \neq x^{2y}$$

$$x^{2y} = x^{2y}$$

B.4 A continued fraction.

$$1+\frac{x}{1+\frac{x}{1+\frac{x}{1+\frac{x}{1+\frac{x}{1+\cdots}}}}}$$

B.5 Simple School algebra.

$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

$$a^3 + b^3 = (a+b)(a^2 - ab + b^2)$$

Given $ax^2 + bx + c = 0$, we have

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

B.6 square roots.

$$\frac{1+\sqrt{5}}{2} = \phi$$

$$\frac{\sqrt{\pi}}{2} \neq \sqrt{\frac{\pi}{2}}$$

$$\sqrt{1+\sqrt{2+\sqrt{2+\sqrt{2+\cdots}}}}$$

B.7 Trigonometric identities.

$$\sin^2 x + \cos^2 x = 1$$

$$\sin x^2 + \cos x^2 \neq 1$$

$$\sin^{-1} x \neq \sin x^{-1}$$

$$\sin(a+b) = \sin a \cos b + \cos a \sin b$$

$$\cos(x+y) = \cos x \cos y - \sin x \sin y$$

$$\sin 2x = 2 \sin x \cos x$$

$$\cos 2x = \cos^2 x - \sin^2 x$$

B.8 Logarithms.

$$\log^2 x \neq 2 \log x$$

$$\log x^2 = 2 \log x$$

$$\frac{\log x}{\log a} = \log_a x$$

$$\log_{a^2} x = \frac{1}{\log_x a^2} = \frac{1}{2 \log_x a} = \frac{\log_a x}{2}$$

B.9 Series.

$$1 + x + x^2 + x^3 + x^4 + \cdots + x^{n-1} + \cdots = \frac{1}{1-x}$$

$$1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots + \frac{x^n}{n} + \cdots$$

$$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} \pm \cdots = \log(1+x)$$

$$\gamma = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} - \log n$$

$$\log(1+x) - \log(1-x) = \log \frac{1+x}{1-x} = \sum_{i=1}^{\infty} \frac{x^{2i-1}}{2i-1}$$

B.10 Integrals.

$$\int \frac{dx}{x} = \log x$$

$$\int_1^a \int_1^b \int_1^c e^{x+y+z} \, dx \, dy \, dz$$

$$\int_1^\infty e^{x^2-x-1} \, dx$$

$$\int_1^\infty e^{x^{2-x}-1} \, dx$$

$$\int_0^1 \int_0^{\sqrt{1-y^2}} 1 \, dx \, dy = \int_0^{\pi/2} \int_0^1 r \, dr \, d\theta$$

$$s = \int_a \int_b f \, dx \, dy + 1$$

B.11 Summations.

$$\sum_{i=1}^n a_i = 1$$

$$\sum_{1 \leq i \leq n} a_i = 1$$

$$\sum_{i=1}^n a_i + b_i = 1$$

B.12 Limits.

$$\lim_{x \to \infty} \int_0^x e^{-y^2} \, dy = \frac{\sqrt{\pi}}{2}$$

$$\lim_{x \to 0} \frac{\sin x}{x} = 1$$

B.13 Cross referenced equations.

$$\cosh x = \frac{e^x + e^{-x}}{2} \tag{1}$$

$$\sinh x = \frac{e^x - e^{-x}}{2} \tag{2}$$

Squaring 1 and 2 and computing their difference gives

$$\cosh^2 x - \sinh^2 x = 1$$

B.14 Distance formula.

Given $x = (x_1, x_2), y = (y_1, y_2)$ the distance between the two points is given by:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

This is the distance formula.

B.15 Quantified expression.

$$\forall x \in X : \exists y \in Y : x = y$$

B.16 Faa de Bruno's formula

Let $D_x^k u$ represent the kth derivative of a function u with respect to x. The chain rule states that $D_x^1 w = D_u^1 w D_x^1 u$. If we apply this to second derivatives, we find $D_x^2 w = D_u^2 w (D_x^1 u)^2 + D_u^1 w D_x^2 u$. Show that the *general formula* is

$$D_x^n w = \sum_{0 \le j \le n} \sum_{\substack{k_1 + k_2 + \cdots + k_n = j \\ k_1 + 2k_2 + \cdots + nk_n = n \\ k_1, k_2, \ldots, k_n \ge 0}} D_u^j w \frac{n! (D_x^1 u)^{k_1} \cdots (D_x^n u)^{k_n}}{k_1! (1!)^{k_1} \cdots k_n! (n!)^{k_n}}$$

C Different audio renderings of integrals.

Consider the integrals shown in appendix $B.10$. The recognizer returns a high-level representation in which the subscript and superscript to the integral symbol are represented as attributes. The math objects to which the integral is being applied are represented as the children of the integral operator.

C.1 Audio layout

A simple way of reading integrals is to generate an *audio layout* for the expression. By audio layout, we mean an audio rendering that unambiguously conveys the layout of the expression. Thus the audio layout for the following expression

$$\int_1^2 f \, dx$$

would be:

integral $_1$ ^{2} f d x

In the above, the limits are shown in different typestyles to reflect the audio rendering which uses different audio states to read the subscript and superscript. When listening to such a reading, a listener who is familiar with the integral operator can interpret the superscript and subscript in context.

C.2 Descriptive reading

The layout oriented reading shown above is concise. Alternatively the listener may wish to have the expression interpreted in context. This is done by the descriptive reading shown below. A descriptive reading of the above is:

integral from x equals 1 to 2 f d x