ISSN 0105-8517

GIPSY
– a Grammar Based Interactive Document Processing System

Jan Holdam
Claus Nørgaard

DAIMI PB – 221
February 1987

AARHUS UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
Ny Munkegade 116 – DK 8000 Aarhus C – DENMARK
Telephone: + 45 6 12 83 55 *Telex: 64767 aausci dk*

Gipsy
- a Grammar Based Interactive Document Processing System

Jan Holdam, Computer Science Department
Aarhus University, Denmark
Claus Nørgaard, Institute of Electronic Systems
Aalborg University Centre, Denmark

Abstract

Gipsy is an interactive document processing system based on syntax-directed editing, where a document is viewed as an abstract syntax tree derived over a document grammar. The outer structure of the document is described by means of a formatting rule for each production in the document grammar. These rules are written in a language based on a box and glue concept. Two kinds of users exist, grammar designers and ordinary users. To allow the user to control the formatting the grammar designer can use inherited attributes when writing the formatting rules. The values of these variables can be set by the user.

The article was written using the LaTeX document preparation system running under Unix on a Sun-3, and printed on a QMS-PS 800 laserprinter. The figures in section 7 were made with the Gipsy prototype implemented on an ICL Perq 1, and plotted on a Hewlett-Packard Laserjet.

1 Introduction

A current trend for WYSIWYG document processing systems is to provide the same typographic quality as batch formatting systems together with the more friendly user interface usually provided on graphical workstations. This paper describes our work in that direction. Inspired by syntax-directed editing used in systems like [6], the text formatting system TeX [4], and interactive systems like [2] and [5] we have designed a model for interactive document processing, and a system based on that model. The system is called Gipsy. A prototype of the system has been implemented.

The work was done as a master's study project at Aarhus University during the years 1983-1985. The primary aims were to handle documents consisting of "normal text" divided into chapters, sections, subsections etc., as well as mathematical formulae.

2 Syntax-Directed Document Processing

Syntax-directed editing has been used for quite a long time as a tool for editing and manipulating programs. Gipsy is an attempt to do something similar in the area of document processing. The user working with a syntax-directed editor manipulates abstract syntax trees derived over a context-free grammar. Context-free grammars are widely used to describe the syntax of programming languages, and as described below they can also be used to describe the structure of documents. For an introduction to context-free grammars and abstract syntax trees see [1].

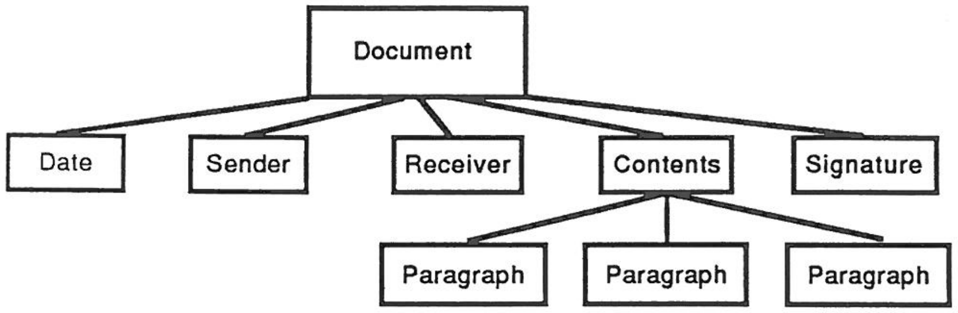
The important aspects of the inner structure of a document (e.g., chapter, section, paragraph) basically form a hierarchy and may as such be viewed as an abstract syntax tree derived over a document grammar. We could for example have the following productions in a document grammar:

`(letter)`[1]	DOCUMENT	::=	BUSINESSLETTER
`(a-letter)`	BUSINESSLETTER	::=	DATE SENDER RECEIVER
			CONTENTS SIGNATURE
`(paragraphs)`	CONTENTS	::=	(PARAGRAPH)*

[1]This designation is the *name* of the production.

describing that a document can be a business letter, and that a business letter consists of a date, a sender, a receiver, some contents and a signature. The contents are a possibly empty list of paragraphs.

A document derived over the example grammar above could have the following structure:



The abstract syntax tree describing the document is formatted to an outer structure presented to the user by means of a formatting rule for each production in the document grammar. A formatting rule describes how a node in the abstract syntax tree derived by that production is to be formatted. The formatting rules are expressed in a language based on a box and glue concept. A fundamental choice in the design of Gipsy is that the formatting of an internal node in the abstract syntax tree must result in exactly one box, which of course may consist of a number of inner boxes. Alternatively, a node could be formatted into several boxes and/or glue, but that would impede incremental formatting.

A formatting rule for the `letter` production in the example grammar above could be:

vbox <*hbox to* **pagewidth** <SENDER *fill* DATE>
skip 3 mm
RECEIVER
skip 3 mm
CONTENTS
skip 3 mm
hbox <*skip* 50 mm SIGNATURE>>

When the abstract syntax tree is formatted, the outer structure of a node in the tree is composed of the visual elements found in the formatting rule for the production in use. Among these visual elements, which are positioned relatively to a base point of the actual node, are typically boxes for the sons of the node as well as some textual constants and amounts of glue. The sons of a node are described by the nonterminals on the right side of the production in the grammar. When a node is formatted, glue for the node is set and dimensions of boxes are computed. Formatting is performed bottom-up, which means that before the formatting of a node can be completed all of its subtrees have to be formatted.

The outer structure of the document is shown on the screen or printed on paper. Displaying the document is performed by traversing the tree top-down. This means that the absolute position of a visual element is computed using the absolute position of the base point of the parent node.

Grammars define the framework for a given kind of document. As Gipsy is driven by the grammar, a user edits a document in terms of the concepts defined in that framework (e.g. BUSINESSLETTER and SENDER). To avoid "ordinary" users to be concerned with the definition of grammars and specification of formatting instructions, another kind of user who is an especially skilled user sets up the grammatical framework. This kind of user is called a grammar designer.

3 Attributes

In order to provide the user with a flexible way of changing the formatting, inherited attributes can be used for different purposes in the grammar framework. Two kinds of attributes exist, built-in and those defined by the grammar designer. The built-in attributes are used by the system when formatting the leaves of the tree. Typical examples of built-in attributes are **font** (describes the typeface used), **fontsize** and **hsize** (the width of the paragraph). The attributes defined by the grammar designer are used to control the formatting, e.g., to describe sizes of boxes and glue, or they might be used as elements in documents. Examples of attributes controlling the formatting are **pagewidth** and **figuresize**. Examples of attributes as elements in documents are **date**, which will insert a box containing today's date at a specified position in the document, and **chapterdesignation**, containing "Chapter" when you write a document in English, and "Kapitel" when the document is in Danish. The attributes are inherited which means that their values flow from the root of the tree towards the leaves. A value assigned to an attribute in a node has scope in the subtree with that node as root, but might be overwritten in subtrees of this subtree.

Assignments of values to attributes can be specified by the grammar designer and by the user. An assignment in the grammar is connected to a production and is used in every subtree with that production in its root. The grammar designer could thus specify the size of the font used for plain text to be 14 pt, and the size of the font used for chapter headings to be 18 pt by assigning these values to the attribute **fontsize** in the productions deriving respectively plain text and chapter headings.

The user may specify assignments to attributes for the document edited. These assignments apply to the currently selected node and overrides a possibly grammar specified assignment to the same attribute for the same node. In this way the user can control the appearance of the document. The width of all plain text in the document can be changed by selecting the root symbol and changing the **hsize**, and it can be done for a single paragraph as well. The font of a single word could be changed by selecting it and changing the value of **font**, and if the grammar designer has decided that the font used in chapter headings is specified by the value of the attribute **chapterheadingfont**, the user could change the appearance of all chapter headings by changing the value of this attribute at the root symbol.

4 Leaves

As we want the structure of a mathematical formula to be derived over a grammar, and as we want inline mathematics in paragraphs to be possible, paragraphs cannot be used as leaves in the tree. Instead, "word" has been chosen as the "textual leaf", although it leads to some problems with respect to hyphenation, as a word that is divided between two lines has to be split into two word nodes with a hyphen node in between. A word node has a textual content, which is specified by the user by typing. Words always have to be used in lists, so when the user hits the spacebar Gipsy will expand this word list and insert a new word node. In this way normal typing of text at the keyboard results in plain text in the document. A word list is a list of components where one of the possible derivations is word. Word lists may be formatted as paragraphs or as lines.

Other leaves in the tree are attributes used as document components (see above) and textual constants specified in the grammar. An example of a textual constant is the name of the company sending a business letter.

5 Incremental Reformatting and Screen Updating

When a change to the abstract syntax tree is made or when the value of an attribute is changed, the abstract syntax tree must be reformatted and redisplayed. This has to be done "quickly" so that the user will not feel annoyed and get disturbed in the primary goal of making a document. This updating can be done with acceptable speed because of the choice of having one box per node.

Reformatting a node typically involves formatting the subtree hanging in the affected node, the affected node, and the nodes from the affected node and upwards in the tree until either the root node of the document or a node whose box does not change by the formatting is met. Because an abstract syntax tree for a document typically is flat only a few nodes in the tree are involved in the reformatting, and in that way speed is ensured.

Incremental screen updating can be performed by a method similar to the method used in PEN [2].

6 The Formatting Language

The formatting language is based on a box and glue concept like the one used in TEX [4]. Unlike a TEX box, a Gipsy box has both a horizontal and a vertical baseline, the latter one to allow easier vertical composition.

The formatting language has constructions specifying that a box should be composed of a number of boxes and some glue to form a new box, either horizontally or vertically. A box may be a box for a son which is specified by naming the corresponding nonterminal on the right side of the production, or it may be a terminal box containing a textual constant or the value of an attribute. Furthermore, a box may contain a paragraph or a line formed over a list on the right side, a list whose major amount of components would be words.

A box can be specified to be of natural size, constant size or to have the size described by the value of an attribute. Its horizontal and vertical baselines can be adjusted. Glue is specified by three components (natural size, stretch and shrink) and each component has either a constant value or takes the value of an attribute.

7 How to Use Gipsy

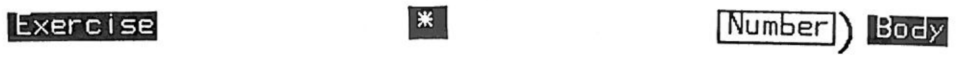
This section contains an example illustrating how Gipsy could be used by an ordinary user. In the example a grammar for making mathematical exercises is used and it is shown how to make an exercise by using this grammar. The mathematical part of the grammar describes the same types of mathematical formulae as can be made in the EDIMATH system [5]. The part of the grammar used in the example is given in the appendix. The complete mathematical grammar can be found in [3].

User Commands

When the user prepares a document the abstract syntax tree is manipulated through the screen representation of the outer structure. Among the commands the user may apply are structure editing facilities such as selection of a node by pointing at its screen representation with the cursor, derivation of a production by selecting it in a menu, and expansion of a list also by a menu command. Textual editing facilities (insertion and backspacing) are performed using the keyboard. There is also a copy, cut and paste facility using a clipboard. The user can change the formatting of the document by assigning new values to attributes at different places in the tree.

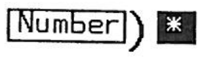
An Example of Use

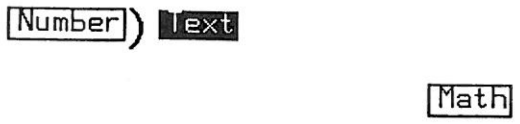
To prepare an exercise in calculus, the exercise grammar can be used in the following way. The nonterminal EXERCISE is derived using the production `problems`, and a template for a problem is shown by expanding the PROBLEM list and then use the `one-problem` production. * is the *list symbol* and denotes an unexpanded list.



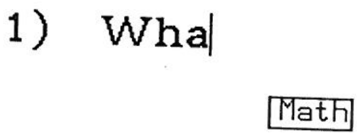
Exercise   *   Number) Body

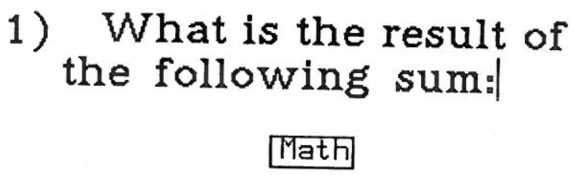
The body is derived to a list of entities, and the list is expanded so that it contains one textual and one mathematical entity.



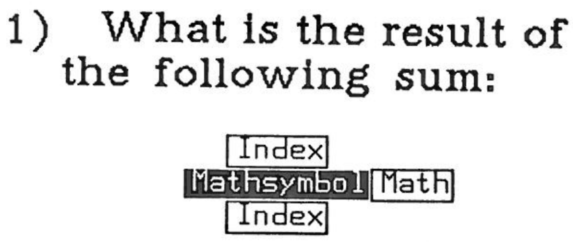


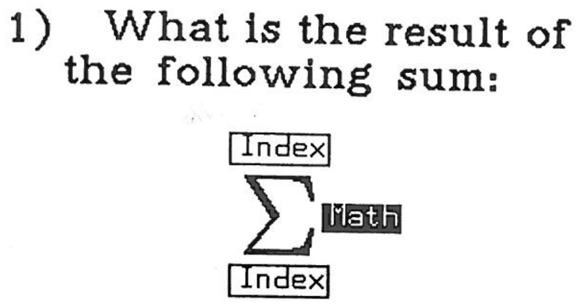
The NUMBER is derived, and the typing of text starts and is eventually completed.


1) Wha

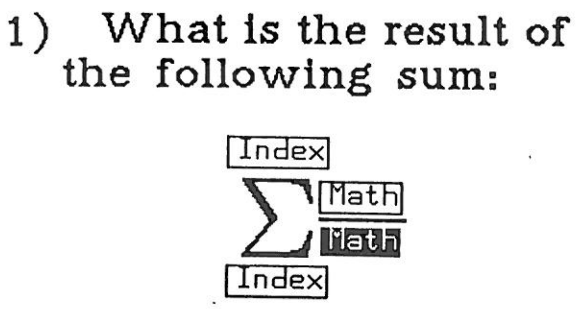

1) What is the result of the following sum:

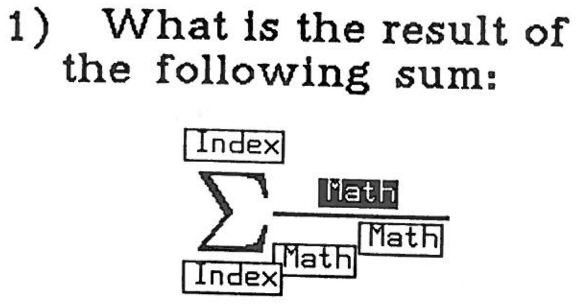
The MATH nonterminal is then derived to a "triple" by selecting the production in a menu, and the triple is in turn derived to a sum.


1) What is the result of the following sum:

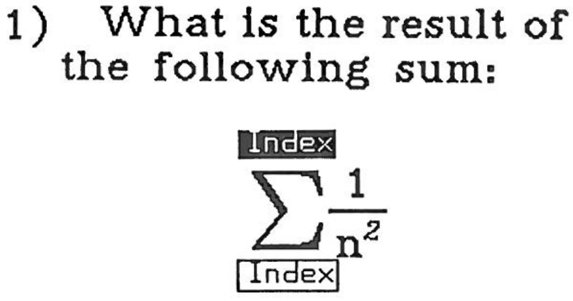

1) What is the result of the following sum:

The summand is derived to a fraction, the denominator of which is turned into a MATH with a superscript.

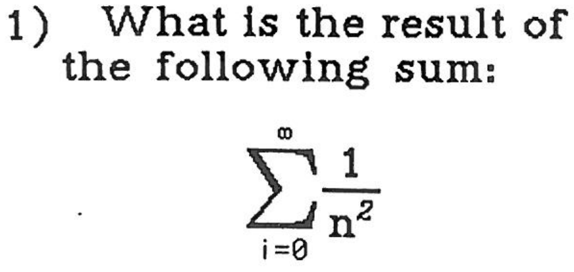

1) What is the result of the following sum:


1) What is the result of the following sum:

The fraction is completed, the indices are derived and the final result is:

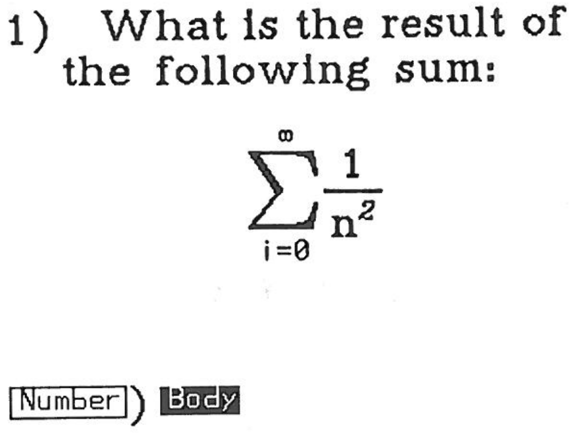

1) What is the result of the following sum:

$$\sum_{\boxed{\text{Index}}}^{\boxed{\text{Index}}} \frac{1}{n^2}$$


1) What is the result of the following sum:

$$\sum_{i=0}^{\infty} \frac{1}{n^2}$$

And then, the PROBLEM-list might be expanded with another problem to be constructed:



1) What is the result of the following sum:

$$\sum_{i=0}^{\infty} \frac{1}{n^2}$$

8 Conclusion

A prototype of Gipsy has been implemented on an ICL Perq 1. During the work with the prototype some minor changes to the original design turned out to be necessary. There were no constructions to adjust the baselines of a box in the formatting language used in the prototype, and to extend the functionality of the inherited attributes it ought to be possible to let their values depend on values computed higher in the tree.

The prototype has shown that the idea of making a syntax-directed interactive document processing system based on grammars and inherited attributes with formatting rules connected to the productions in the grammar is reasonable. The ordinary user does not have to bother about formatting instructions, as the formatting rules are hidden in the grammar. Instead, the user may manipulate the document and its formatting through concepts especially suited for that kind of documents, concepts that are defined by the grammar designer. The grammatical framework available for the grammar designer supports a modular description of the document structure. The prototype has also shown that the choice of letting each node in the abstract syntax tree form exactly one box makes incremental formatting possible and that the typographic quality can be as good as in batch systems.

References

[1] Alfred V. Aho, Jeffrey D. Ullman: *Principles of Compiler Design.* Addison-

Wesley, Reading, Massachusetts, 1978.

[2] Todd Allen, Robert Nix, Alan Perlis: *PEN: A Hierarchical Document Editor.* ACM Sigplan Notices, Vol. 16, No. 6, June 1981.

[3] Jan Holdam, Claus Nørgaard: *Gipsy - Et system til syntaksdirigeret interaktiv dokumentbehandling.* Master's Thesis, Computer Science Department, Aarhus University, Denmark, June 1985 (in Danish).

[4] Donald E. Knuth: *The TEXBook.* Addison-Wesley, Reading, Massachusetts, 1984.

[5] Vincent Quint: *EDIMATH - An Interactive System for Editing Mathematical Documents.* In: P. Degano, E. Sandewall (eds.): *Integrated Interactive Computer Systems*, ECICS, North-Holland, 1983.

[6] Tim Teitelbaum, Thomas Reps: *The Cornell Program Synthesizer: A Syntax-Directed Programming Environment.* Communications of the ACM, Vol. 24, No. 9, September 1981.

Appendix: An Example Grammar

This is the part of the grammar for mathematical exercises that is used in section 7. The formatting rules shown are somewhat simplified in that the placement of the vertical and horizontal baselines are not treated.

A mathematical exercise consists of a number of problems:

`(problems)`	EXERCISE ::= (PROBLEM)*
	vlist between **problemskip** < PROBLEM >
	problemskip ← 40.[2] **hsize** ← 260.
	problemwidth ← 290.

Each problem is made up of a number and a body:

`(one problem)`	PROBLEM ::= NUMBER BODY
	hbox to **problemwidth** <NUMBER ')' *hskip* 5 BODY>

The body consists of a sequence of mathematical entities and text entities:

`(entity-list)`	BODY ::= (ENTITY)*
	hbox to **hsize** <*vlist between* **entityskip** < ENTITY >>
	entityskip ← 20.
`(a-math)`	ENTITY ::= MATH
	hbox to **hsize** <*fill* MATH *fill*>
`(some-text)`	ENTITY ::= TEXT
	hbox <TEXT>

The text part is specified by the following productions:

`(paragraph)`	TEXT ::= (WORD)*
	paragraph
`(word)`	WORD ::= *text*
	text

The mathematical part is specified by:

`(triple)`	MATH ::= MATHSYMBOL INDEX INDEX MATH
	hbox <*vbox* <*hbox* < INDEX$_1$ >
	hbox <MATHSYMBOL>
	hbox < INDEX$_2$ >>

[2] **attribute** ← ... denotes assignment to an attribute.

```
                MATH>
(superscript) MATH      ::= MATH MATH
              hbox < MATH_1 vbox < MATH_2 skip superscriptskip>>
(fraction)    MATH      ::= MATH '/' MATH
              vbox <hbox < MATH_1 >
                    hbox <rule^3 width max(width(MATH_1), width(MATH_2))>
                    hbox < MATH_2 >>

(math-text)   MATH      ::= TEXT
              hbox <TEXT>
```

[3] *rule* is a construction for drawing a line of a specified length.

This paper was presented at the PROTEXT III Conference in Dublin, Ireland, 22-24 October 1986. It will appear in the conference proceedings published by Boole Press Ltd.