Presenting Proofs Using Logicographic Symbols*

Koji Nakagawa, Bruno Buchberger

Software Competence Center Hagenberg,
Hauptstrasse 99,
A-4232 Hagenberg, Austria
koji.nakagawa@scch.at

Research Institute for Symbolic Computation,
Johannes Kepler University,
A-4040 Linz, Austria
{nakagawa, buchberger}@risc.uni-linz.ac.at

Abstract. Mathematics has a rich tradition in creating symbols and notation that is soundly integrated into the syntax of the underlying formal language and, at the same time, conveys the intuition behind the concepts described by the symbols and notation. Continuing this idea, in the Theorema system, with the new feature of logicographic symbols, we now provide a means to invent arbitrary new symbols and notation.

In this paper we describe how logicographic symbols can be created, declared, and afterwards used as a part of the formal language of Theorema with an example. Also with logicographic symbols, formal proof text automatically generated by Theorema provers can become easy to read in a way that resembles telling a pictorial story about the mathematical concepts involved.

1 Introduction

In automated theorem provers, it is very important not only to produce formal proofs but also present proofs that can be easily understood by humans, see [1, 2]. One of the issues in facilitating human understanding of automatically generated proofs is the automated generation of natural language explanatory text as part of the proofs, see for example [3, 4]. Also, in the Theorema system[5, 6], an coherent mathematical environment for proving, solving, and computing implemented on top of *Mathematica*[7], we put a lot of emphasis on the natural language aspect of proof presentation. This is achieved by, first, using natural deduction calculi with special complex inference rules for the various special areas of mathematics and, second, by producing standardized natural language text blocks that are produced, in a post-processing step, with every application of these inference rules.

In this paper we describe a new and additional feature of the Theorema system, first proposed in [8], that should improve the readability of formal texts, in

* Work on this paper was supported by Project MathSoft of the Software Competence Center Hagenberg (Austria) and Project F1302 of the Austrian Science Foundation (FWF). We would like to thank Christopher Carlson and Theodore W. Gray of Wolfram Research Inc. for their aid and valuable advice on subtle details of the *Mathematica* front end.

particular proof texts, by allowing the user to introduce arbitrary new graphical symbols (for functions and predicates) in two-dimensional notation with slots for the parameters at arbitrary positions. We call these new symbols "logicographic" symbols: By the use of logicographic symbols, the formal structure of the language is not really extended, i.e. text including logicographic symbols can be viewed as just a notational variant of the Theorema language (a variant of predicate logic), i.e. logicographic symbols are "logical". On the other hand, these symbols are like little pictures (graphics) that support the intuition of the user while he is reading the formal text, i.e. logicographic symbols are "graphical".

In fact, the idea of logicographic symbols, in some way, is as old as mathematics. For example, the symbol '>' for "greater" in 'a > b' is a formal symbol (in infix notation) and, at the same time in a simplified and standardized way, conveys the intuitive / graphic idea that the object denoted by 'a' on the left-hand side of the symbol is bigger than the object denoted by 'b' on the right-hand side. However, so far, inventing, designing and introducing *new* "logicographic" symbols was not easily possible in mathematical systems, not even in mathematical text processing systems let alone formal systems like *Mathematica*. The new feature of logicographic symbols in Theorema is completely general and highly flexible so that it opens a new dimension of designing formal mathematical texts. We believe that this may have a significant influence on how mathematical research and teaching can be done in the future.

The results described in this paper are a part of the work carried out in the frame of the PhD thesis[9] of the first author under the direction of the second author.

2 Motivation

2.1 Theorema Formal Text

Knowledge in Theorema is described by so called *Theorema formal text*. The Fig. 1 shows a part of the theory describing the correctness of the merge-sort algorithm in the Theorema formal text language. Theorema formal text is of the form *hd*[*label*, any[*variables*], *statement*]. The identifier '*hd*' indicates the kind of the '*statement*', e.g. Definition, Lemma, Proposition, Theorem, Algorithm etc. The '*label*' is used to refer to the statement afterwards. For example, in Fig. 1 by Algorithm["mg"], one can refer to the definition of the function 'mg'. The 'any[*variables*]' expression indicates the free variables in the subsequent statement. The '*statement*' part is the essential formal statement (in the Theorema expression language, a variant of predicate logic).

In the Theorema built-in notation, '$\langle\rangle$', '$\langle x, \overline{X}\rangle$',' $x \smile X$', '$X \asymp Y$' stand for 'empty tuple', 'a tuple with the first element x and a finite sequence $\overline{X}$ of elements', 'tuple X with element x prepended', 'the concatenation of tuple X and tuple Y' respectively. With the additional explanation in Fig. 2 (for the moment ignore the leftmost column), the meaning of the above Theorema formal text should be self-explanatory. For example, the definition of 'stmg' describes the

```
Algorithm["stmg", any[X],
    stmg[X] := { X                   ⇐ |X| ≤ 1
               { mg[                 ⇐ otherwise
               {   stmg[lsp[X]],
               {   stmg[rsp[X]]]
];

Algorithm["mg", any[X, Y, a, b, x̄, ȳ],
    mg[⟨⟩, Y] := Y
    mg[X, ⟨⟩] := X
    mg[⟨a, x̄⟩, ⟨b, ȳ⟩]
      := { a ⌣ mg[⟨x̄⟩, ⟨b, ȳ⟩]   ⇐ a ≥ b
         { b ⌣ mg[⟨a, x̄⟩, ⟨ȳ⟩]   ⇐ otherwise
];

Definition["istv", any[X, Y],
  istv[X, Y] ⟺ (ist[X] ∧ ipm[X, Y])];

Lemma["mg" , any[A, B],
  (ist[A] ∧ ist[B]) ⟹ ist[mg[A, B]]];

Lemma["mg2", any[A, B],
  ipm[mg[A, B], A ≍ B]];
```

Fig. 1. Formalized Merge-Sort Theory

algorithm of merge-sort: If the length of the argument tuple 'X' is less than or equal to 1, then the result is 'X'. Otherwise, 'X' splits into 'lsp[X]' and 'rsp[X]', then each of these tuples is sorted by a recursive call of 'stmg' and, finally, the two sorted parts are merged by 'mg'.

With the definitions above, the correctness of merge-sort can be formalized as follows:

```
Proposition["Correctness of Merge Sort",any[A],istv[stmg[A],A]]
```

This proposition states that for any tuple 'A', after application of the algorithm 'sorted by merging', the resulting tuple 'stmg[A]' is a sorted version of 'A'. It will be possible to prove this theorem automatically by one of the Theorema provers.

2.2 Logicographic Symbols

Of course, one could be happy with the above formal text in Fig. 1 from a strictly formal point of view. However, it is difficult to grasp the intuition behind in the formal way. So we will now demonstrate how, by the introduction of new "logicographic" symbols in two-dimensional notation, the above formulae become easier to understand.

Fig. 2 shows a possible choice of logicographic symbols for the merge-sort theory. Of course the user has complete freedom in designing new symbols for the various notions. With these logicographic symbols, the knowledge base of Fig. 1 can now be written in the way shown in Fig. 3. The expressions are represented in a nested 2-dimensional syntax with dark gray and light gray coloring for indicating the syntactical structure. (The users can change the coloring by writing an appropriate *Mathematica* function).

Note that the Theorema formal texts in Fig. 3 can be evaluated exactly like the expressions in Fig. 1, i.e. they still represent formal expressions of the Theorema language but, at the same time, they convey the intuition behind the formulae in a, hopefully, appealing way.

	lsp[X]	left split of X
	rsp[X]	right split of X
	mg[X,Y]	the result of merging two tuples X and Y
	stmg[X]	the result of sorting X by merging
	ipm[X,Y]	X is a permuted version of Y
	ist[X]	X is a sorted tuple
	istv[X,Y]	X is a sorted version of Y

Fig. 2. Logicographic Symbols for the Merge-Sort Theory

3 Working with Logicographic Symbols

3.1 Declaring Logicographic Symbols

In order to tell the system the way to display certain expressions, the 'LogicographicNotation' declaration is used. For example, for the function symbol 'mg':

```
LogicographicNotation["merge",any[X,Y],
  mg[X,Y](X " and " Y "merged")⇌  X Y];
```

In the declaration above, the entire drawing with two "slots" for the two possible arguments 'X', 'Y' constitutes the new symbol for 'mg'. The label "merge" can be used to refer to this logicographic declaration afterwards by LogicographicNotation["merge"]. The expression 'any[X, Y]' means that 'X, Y' are treated as variables in the declaration. The annotation '(X " and " Y "merged")' indicates a suggestion for human readers, how to *read* texts containing these symbols. Alternatively, this annotation can be used for making proofs automatically pronounced by a voice synthesis system. This feature will be described in the forthcoming PhD thesis[9] by the first author.

The 'LogicographicNotation' can be nested. Namely, one can compose large notation from several smaller constituents. For example, one could compose logicographic notation with label "Merge Sort", by enumerating various constituent notation in the following way:

```
LogicographicNotation["Merge Sort",
  LogicographicNotation["perm"]
  LogicographicNotation["sorted"]    ];
  LogicographicNotation["split"]
```

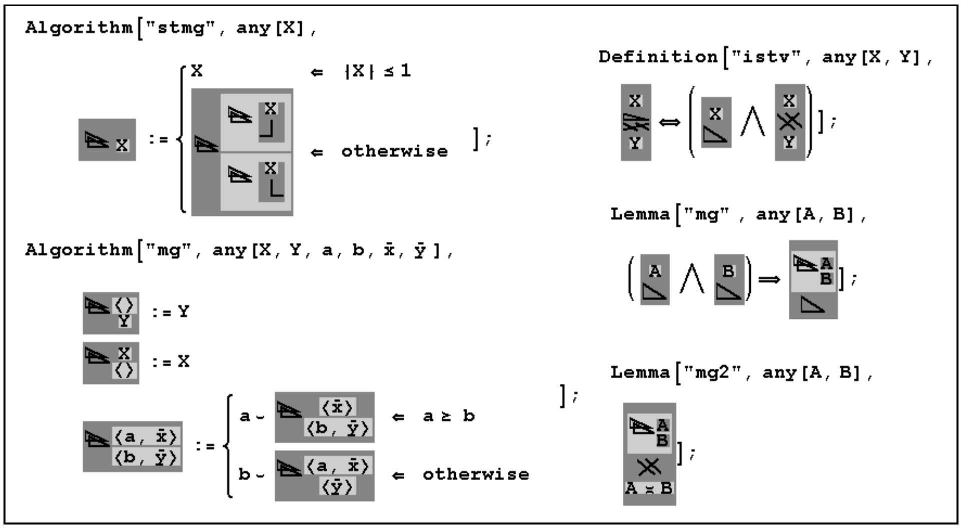


Fig. 3. Formalized Merge-Sort Theory with Logicographic Symbols

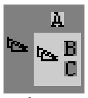
3.2 Using Logicographic Symbols

Since logicographic symbols can be evaluated and they are just a different way of writing the corresponding Theorema (function or predicate) constants, they can be used in all contexts in which function and predicate constants appear in Theorema (e.g. in Definition[...], Prove[...], etc.) Namely, when formulae containing logicographic symbols are evaluated, this causes the same effect as executing the formula with all logicographic symbols replaced by their internal constants. As we saw in Fig. 3, we can compose arbitrary knowledgebases using logicographic symbols.

Executing 'LogicographicNotation' declarations does not yet activate their usage. In order to make these declarations effective, one must apply the function 'UseNotation' to the notation object obtained by 'LogicographicNotation[*label*]'. For example, in order to activate the logicographic symbol of 'mg', the following command must be executed:

```
UseNotation[LogicographicNotation["merge"]]
```

Once the logicographic symbol is activated by 'UseNotation', expressions in ordinary Theorema syntax are displayed with the corresponding logicographic symbols. For example, entering 'mg[A,mg[B,C]]' now produces the expression appearing right to this text.



The logicographic presentation can be used for displaying formal proofs. Note that the above logicographic symbols may appear with various different argument terms at different places within the text. One can control which expressions should be displayed with logicographic symbols by specifying the option

'Notation' in the option 'ShowOptions' of the Theorema 'Prove' command. For example,

```
Prove[Proposition["Correctness of Merge Sort"],
  using -> Theory["Merge Sort"], by -> CourseOfValueProver,
  ShowOptions->{Notation->LogicographicNotation["Merge Sort"]}]
```

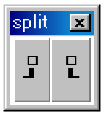
In the appendix, a sketch of this proof is displayed, which should demonstrate the positive effect of logicographic symbols on making proofs easier to understand.

3.3 Typing Logicographic Symbols

Of course, for the practical usage of logicographic symbols, it is very important that typing such symbols becomes easy for the user. In *Mathematica*, symbols that are not available on the keyboard can either be input by using *input aliases* or *palettes*. We extended these facilities by making it possible to input arbitrary logicographic symbols (declared by the user as explained above) by either input aliases or palettes.

The *input aliases* facility is a shortcut of typing some special symbols. Extending this facility of *Mathematica*, for example, typed text 'ESC mg ESC' changes into the corresponding logicographic symbol. (Here ESC means the escape key.)

A *palette* is a window containing buttons, and if one presses one of the buttons in the palette, the expression written on the button appears in the current cursor position. For example, for 'left split' and 'right split', the palette in the right of this text is produced, and from this time on, if one presses one of the buttons in the palette, the corresponding logicographic symbols, together with argument slots, appears in the current cursor position.



3.4 Creating New Logicographic Symbols

In the Theorema system we will provide various ways by which a user can easily design and create new logicographic symbols of arbitrary complexity using the graphical tools of Mathematica, photos, hand-made drawings etc. We implemented and experimented with a couple of prototypes for this purpose. A detailed description of the software-technology behind these approaches will be given in [9].

4 Implementation of Logicographic Symbols

The implementation of logicographic symbols needs the facility to convert an representation with logicographic symbols into the corresponding internal expression and an internal expression into the corresponding representation with logicographic symbols. This implementation feature is based on the *Mathematica* functions 'MakeExpression' and 'MakeBoxes'.

In *Mathematica* all two-dimensional graphical representations are stored in *boxes* format that keeps information on the graphical appearance. From an expression in the boxes format, 'MakeExpression' *makes* the corresponding internal expression and conversely from an internal expression 'MakeBoxes' *makes* the corresponding expression in the boxes format. For example, the representation '$\int xdx$' is stored in the boxes format as 'RowBox["$\int$", RowBox["x", RowBox["d", "x"]]]' which can be transformed into the corresponding internal expression 'Integrate[x,x]' by 'MakeBoxes'. Conversely, by 'MakeExpression' the internal expression 'Integrate[x,x]' can be transformed into the corresponding boxes format expression. For more details, see [7].

Activating logicographic symbols appropriately modifies these 'MakeExpression' and 'MakeBoxes' functions such that the system does also the treatment of slots, coloring etc.

5 Remarks

5.1 Related Work

In 1879, Frege introduced 2-dimensional notation for logical formulae [10]. Frege's notation, however, shows only the syntactical structure of formula, whereas logicographic symbols try to convey the intuitive semantics behind the logical constants.

In the TEX system[11] one can create new characters by the METAFONT[12] system and compose them by using macros in a very flexible way. However, TEX is a typesetting system and it is for printing. In contrast, in the Theorema system, formulae containing logicographic symbols can be evaluated and, more importantly, can be also used in the process of proving, solving, and computing. The logicographic symbols, together with explicitly specified slots for arguments, strictly respect the predicate logic syntax for terms and atomic formulae.

As a forerunner, Jason Harris already implemented a system similar to ours called 'notation package' (`http://library.wolfram.com/packages/notation`), which comes with Mathematica. By the notation package, one can create new notation having specified slots for arguments by using the existing Mathematica characters. However, for the implementation of our general concept of logicographic symbols, which needs an unlimited arsenal of possible symbols for capturing intuitive semantics, we needed implementation techniques that give more control on the design, shape, arity, and slot positions.

5.2 Importance of Design

The design of logicographic symbols is very important, because carelessly designed logicographic symbols might mislead us. This phenomenon happens, for example, in a situation that logicographic symbols are so concrete that the graphics do not capture all cases. Note that the logicographic symbols shown in this paper, however, do not fall into this case, because they do not lose their generality at all, e.g. like '$a > b$'. On the other hand, although too concreate graphics are not general enough, they often give us intuition in an appealing way.

We are currently undertaking extensive case studies for inventing and designing logicographic symbols in various areas of mathematics. By this, we hope that we will be able to find reasonable didactic guide lines that may help the user of our system in designing logicographic symbols. Also, we are working on a logicographic symbols library from which the users can create their own symbols more easily, e.g. by combining them[13].

5.3 Future Work

Besides in Theorema, texts with logicographic symbols can be shown in other systems, e.g. in LaTeX with METAFONT or HTML with Java applets. Making such converters from Theorema to other systems is possible future work.

It will be an important practical future task to carry out systematic didactical experiments on the influence of using logicographic symbols for the presentation of formal mathematical texts. While we did not yet carry out such experiments using our implementation of logicographic symbols in Theorema, we should, however, mention the fact that the idea for the logicographic symbols technique evolved in the many years in which the second author was conducting his regular "Thinking, Speaking, Writing" course for PhD students. In the frame of this course, particular attention is given to practical proof training in predicate logic, and logicographic symbols were extensively used. The experience is that, in fact, logicographic symbols can drastically improve the ease of understanding formal derivations in comparison to presenting the same text with just ordinary identifiers.

6 Conclusion

Future systems for formal mathematics must combine various functionalities within the same logic and software technological frame:

- traditional functionality of math software systems like *Mathematica*,
- automated theorem proving tools,
- tools for knowledge management and attractive presentation of formal text.

Theorema is designed to integrate all these functionalities coherently. The facility of logicographic symbols is a contribution to the third functionality. Logicographic symbols convey graphical intuitive ideas behind the concepts used and, at the same time, they are completely formal as a part of the underlying formal language. We think that, with logicographic symbols, we can reach a new level of clarity in formal mathematics. We implemented and are currently trying out logicographic symbols in the frame of our Theorema system. However, of course, the concept could be integrated into any other system for formal mathematics.

Philosophically, it is interesting to note that, in historically old languages, keeping syntactical structure and meaning as closely together as possible was felt to be quite important. For example, in the Chinese and Japanese Kanji system, characters convey the graphical-intuitive meaning of the concepts they denote. (This is true, at least, if we consider the historical evolution of these characters. In every-day reading practice, though, it seems that native Chinese and Japanese readers do not any more view the semantics into the characters.) Also, in the ancient Vedic literature, the composition of words was conscientiously done following the composition of the corresponding semantics. For the purification of formal languages, notably in the past century, a clear distinction between syntax and semantics was, however, of utmost of importance. The approach of logicographic symbols, seen in this context, is an attempt at reconciling the two directions in the history of language development: By the technique of logicographic symbols, we conserve the distinction between syntax and semantics for formal purity and mathematical correctness of proving as a thinking discipline and, at the same time, we conserve the close connection between syntax and semantics for ease of understanding.

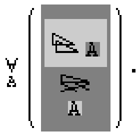
References

1. R. Bornat and B. Sufrin. Animating formal proof at the surface: The Jape proof calculator. *The Computer Journal*, 42(3):177–192, 1999.
2. Y. Bertot and L. Théry. A generic approach to building user interfaces for theorem provers. *Journal of Symbolic Computation*, 25(2):161–194, February 1998.
3. X. Huang and A. Fiedler. Proof presentation as an application of NLG. In *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI), Nagoya, Japan*, 1997.
4. Y. Coscoy, G. Kahn, and L. Théry. Extracting text from proofs. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proc. of the Second International Conference on Typed Lambda Calculi and Applications, Edinburgh, UK*, volume 902 of *LNCS*, pages 109–123, 1995.
5. B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuţa, and D. Văsaru. A survey on the Theorema project. In Wolfgang W. Küchlin, editor, *Proc. of ISSAC'97 (the 1997 International Symposium on Symbolic and Algebraic Computation), July 21–23, 1997, Maui, Hawaii*, pages 384–391. ACM Press, 1997.
6. B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Văsaru, and W. Windsteiger. The Theorema project: a progress report. In M. Kerber and M. Kohlhase, editors, *Proc. of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, St. Andrews, Scotland, August 6-7*, pages 100–115, Universität des Saarlandes, Germany, 2000.
7. S. Wolfram. *The Mathematica book*. Cambridge University Press and Wolfram Research, Inc., fourth edition, 1999. `http://www.wolfram.com`.
8. B. Buchberger. Logicographic symbols: some examples of their use in formal proofs, February 2000. Manuscript, RISC (Research Institute for Symbolic Computation), Johannes Kepler University, Linz, Austria.
9. K. Nakagawa. *Advanced input / output features in the Theorema system*. Ongoing PhD thesis, RISC (Research Institute for Symbolic Computation), Johannes Kepler University, Linz, Austria, 2001.

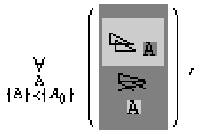
10. G. Frege. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. In Jean van Heijenoort, editor, *From Frege to Gödel: a Source Book in Mathematical Logic, 1879–1931*, pages 1–82. Harvard University Press, 1967. Original in German in 1879.
11. D. E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, 1986.
12. D. E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, 1986.
13. B. Buchberger. Logicographic symbols: A new feature in Theorema. In *Proc. of IMS 2001 (the 4th International Mathematica Symposium), Chiba, Japan*, 2001.

A Proof of the Correctness of Merge-Sort

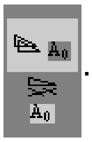
Prove:



We use course of value induction on A. Let now A_0 be arbitrary but fixed and assume

(ind-hyp) 

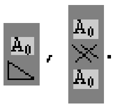
and show

(G) 

We prove (G) by case distinction using (Algorithm: stmg).

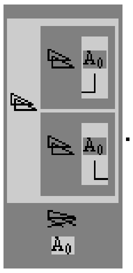
Case $|A_0| \leq 1$: We have to prove

By (Definition: istv), we have to prove

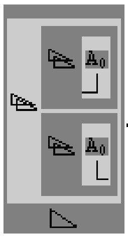
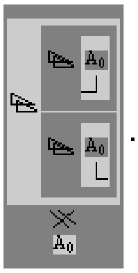


These are true, because (properties of sorted tuples) and (reflexivity of perm).

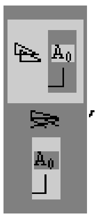
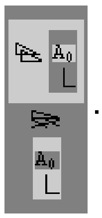
Case $|A_0| \nleq 1$: We have to prove



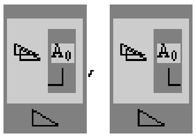
By (Definition: istv), we have to prove

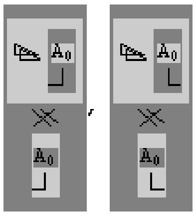
(G1) , (G2) 

By (ind-hyp), we know

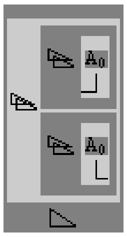
(K) , 

And hence, by (Definition: istv) we also know

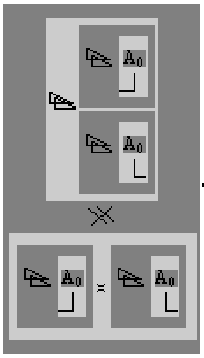
(K1)  ,

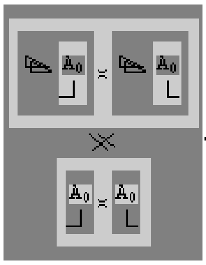
(K2)  .

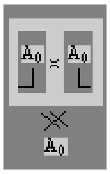
We prove (G1): By (Lemma: mg) and (K1),

 .

We prove (G2): We know (C1) by (Lemma:mg2), (C2) by (properties of permutation), (C3) by (properties of splitting),

(C1)  ,

(C2)  ,

(C3)  .

Hence, by (C1), (C2), (C3) and (transitivity of permutation), (G2) is proved.