Logicographic Symbols: A New Feature in Theorema

Bruno Buchberger

RISC (Research Institute for Symbolic Computation)
A 4232 Schloss Hagenberg, Austria
Buchberger@RISC.Uni-Linz.ac.at

Abstract: Theorema is a mathematical software system emphasizing computer-supported theorem proving. Theorema is implemented in *Mathematica* and, thus, can be used on all platforms on which *Mathematica* is available.

In this paper and talk, we explain the new Theorema language feature of logicographic symbols. This feature allows to introduce arbitrary new symbols for mathematical functions and predicates so that the user is not confined to a fixed vocabulary of symbols. The implementation of logicographic symbols in Theorema heavily involves tools of the front-end of *Mathematica*.

Introduction

In this paper, we report on the new concept of logicographic symbols and its realization in the frame of the Theorema system, [Buchberger et. al. 1997], [Buchberger et. al. 2000]. This work is joint work with Koji Nakagawa: The concept of logicographic symbols is due to the author and was first introduced in [Buchberger 2000]. All the implementation of the concept in Theorema was done by Koji Nakagawa and is the main part of the forthcoming PhD thesis [Nakagawa 2001].

The development of formal mathematics is important for achieving a new level of rigor, reliability, manageability, automation and re-usability in mathematics and a smooth and computer-supported transition from mathematical knowledge to mathematical algorithms and, ultimately, to applications. Currently, formal mathematics is based on the usual linear and/or two-dimensional mathematical notation of (some version of) predicate logic starting from constants for functions and predicates. The vocabulary of constants consists of infinitely many textual identifiers and a fixed arsenal of mathematical symbols as, for example, "+", "/" etc.

Often, however, in order to understand the intuitive idea behind mathematical concepts, one uses graphics. These graphical illustrations of concepts and the constants (identifiers and/or symbols) used for denoting the concepts in formal texts, logically, are not connected: The constants live in the world of rigorous formal derivations whereas the graphical illustrations live in the world of informal heuristics. The graphical illustrations are helpful for understanding the concepts involved and their properties but do not have any logical bearing. The constants are the basic ingredients of the rigorous mathematical derivations but often do not convey well the underlying intuition.

In Theorema, a system for formal mathematics implemented in *Mathematica*, we now introduced the new language feature of "logicographic symbols". These are two-dimensional graphics that can be composed by the user of the Theorema language (using various design tools for generating graphics) in order to convey the meaning of a concept in an intuitively easy and appealing way. Formally, however, these graphics are then treated as new function or predicate constants of Theorema. For this, logicographic symbols have "slots" for the function or predicate arguments. Consequently, logicographic symbols can also be nested and can be combined arbitrarily with ordinary function and predicate constants. Formal texts using logicographic symbols are then both formally rigorous *and* intuitively appealing. In other words, logicographic symbols aim at bridging the gap between formal rigor and intuition in mathematics. Alternatively, one can view logicographic symbols simply as a means for providing the user with an infinite source of mathematical symbols of arbitrary graphical complexity to be used as function and predicate symbols. In this paper and the talk, we will explain how the Theorema user can introduce, interactively, any desired logicographic symbol and how these symbols can then be used for improving the readability and self-explanatory power of mathematical texts without reducing formal rigor. In this paper, we cannot go into any detail about the implementation of logicographic symbols, which is fully explained in [Nakagawa 2001] and, partly, also in [Nakagawa and Buchberger 2001].

Note that the concept of logicographic symbols goes far beyond what is possible even in advanced mathematical text processing systems like TEX, [Knuth 1986a]: Although it is possible in TEX to create arbitrary new symbols by the METAFONT tool, [Knuth 1986b], these symbols have no logical functionality, i.e. their syntax does not follow the rules of predicate logic and neither can we attach any (computational or logical) semantics to such new symbols. In contrast, in the Theorema system, formulae containing logicographic symbols can be executed and, more importantly, can also be processed by automated provers, solvers, and simplifiers. With the new feature of logicographic symbols, we now provide a means for the user to invent arbitrary new symbols that are part of the formal language of Theorema (a version of higher order predicate logic) and, at the same time, also convey the intuitive meaning of the functions and predicates denoted by these symbols. In this paper we describe the essence of the concept of logicographic symbols and their usage by giving just two typical examples.

As a credit to early forerunners, we would like to mention the work of Jason Harris, [Harris 1996], who, without having the general concept of logicographic symbols, designed a *Mathematica* package that allows to create slots for arguments at basically arbitrary positions when using symbols of the existing *Mathematica* vocabulary. However, for the implementation of our general concept of logicographic symbols, we needed implementation techniques that give more control on the design, shape, arity, and slot positions. The box structure of *Mathematica* expressions (with the two basic operations MakeExpression and MakeBoxes) turns out to be a very useful and sufficiently flexible platform on top of which our implementation can be built.

Example: Merge Sort

■ Theorema Formal Text

Here are a couple of formulae, in the "Theorema formal text language", that form part of the theory that describes the correctness of the merge-sort algorithm.

```
Algorithm["stmg", any[X],
                  ⎧X                ⇐ |X| ≤ 1
                  ⎪mg[              ⇐ otherwise
     stmg[X] :=   ⎨ 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]];
```

Words like '**Algorithm**' etc. are keywords. Texts in quotation are labels. For example, by **Algorithm["mg"]** one can refer to the definition of the function **mg**. The '**any[X, Y, a, b, x̄, ȳ]**' indicates that '**X**', '**Y**', '**a**', '**b**', '**x̄**', '**ȳ**' are free variables in the subsequent statement. Formulae like

```
mg[ X, ⟨⟩] := X
```

etc. form the actual mathematical statements (in the "Theorema expression language", a variant of higher-order predicate logic). The three statements in **Algorithm["mg"]** form the definition of the function **mg**. In the Theorema expression language, '**⟨⟩**', '**⟨x, X̄⟩**', '**x ⌣ X**', '**X ≍ Y**' stand for 'empty tuple', 'a tuple with the first element **x** and a finite sequence **X̄** of elements', 'tuple **X** with element **x** prepended', 'concatenation of **X** and **Y**', respectively.

With this explanation and the additional explanation that '**stmg**', '**mg**', '**istv**', '**ipm**', '**lsp**', '**rsp**' stand for 'sorted by merging', 'merged', 'is sorted version of', 'is permuted version of', 'left split', and 'right split', respectively, the meaning of the above Theorema formal text should be self-explanatory. For example, the definition of **stmg** describes the 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 now be formalized as follows:

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

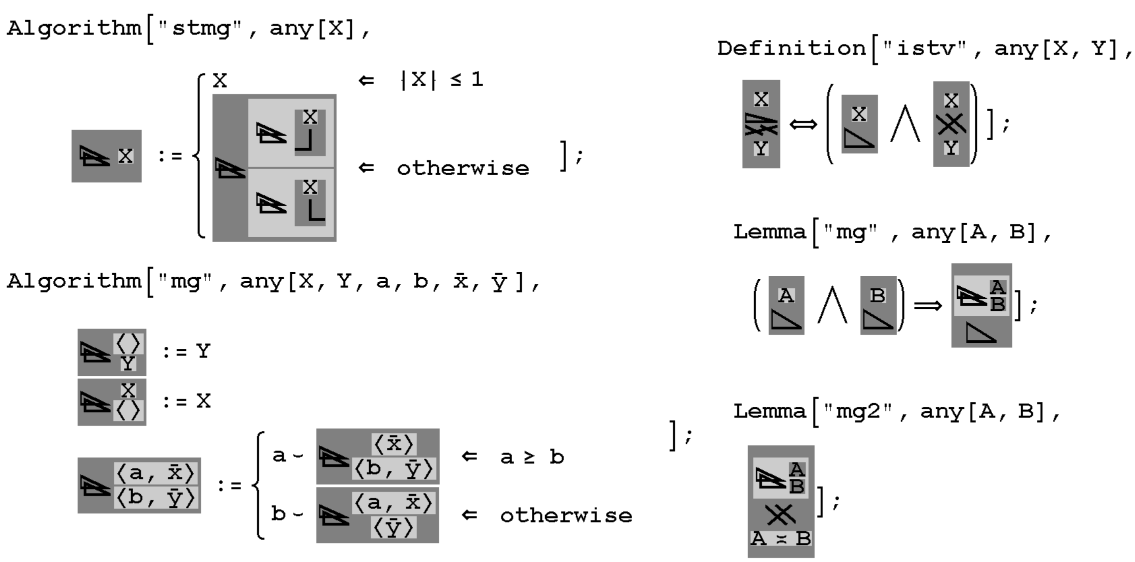
This proposition states that for any tuple '**A**', after application of the algorithm 'sorted by merging', the resulting tuple '**stmg[A]**' is a sorted version of '**A**'. It will be possible to prove this theorem automatically by one of the Theorema provers using the knowledge formalized above plus some additional lemmata on the ingredient notions which, in the proof below, will be referred to as (Lemma: extra).

■ Logicographic Symbols

Of course, one could be happy with the above formal text from a strictly formal point of view. However, it is difficult to grasp the intuition behind the formulae when presented in this "textual" form, i.e. with all constants just being identifiers and with the ordinary linear notation. We now introduce logicographic symbols for some of the constants occurring above. The user has complete freedom in designing new symbols for the various notions. The following may be a possible choice:

`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 sorted
`istv[X, Y]`	`X` is a sorted version of `Y`

With these logicographic symbols, the above knowledge base can be now be written in the way shown below. 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).



We believe that, with a good choice of logicographic symbols, the above mathematical formulae should be basically self-explanatory. At the same time, and this is the essence of the concept of logicographic symbols, the above text with logicographic symbols, is a completely formal text within Theorema, i.e. the above expressions an be evaluated (and used in proofs) exactly like the expressions in the conventional notation.

■ Declaring, Typing, and Creating Logicographic Symbols

In order to inform the system about new logicographic symbols, the '**`LogicographicNotation`**' declaration is used. In the example of the '**`mg`**' symbol, this declaration looks like this:

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

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 definition afterwards by **`LogicographicNotation["merge"]`**. The expression '**`any[X, Y]`**' means that '**`X`**', '**`Y`**' are treated as free variables in the declaration. The annotation '**`(X "and" Y "merged")`**' indicates a suggestion for reading the formula. This annotation can be considered as a suggestion, for human readers, how to "read" texts containing these symbols. (In a later version, this annotation will also be processable in the sense that it will automatically produce "spoken" versions of the formal text, see [Nakagawa 2001].)

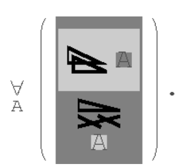
Of course, for the practical usage of logicographic symbols, it is very important that typing such symbols becomes easy for the user. In *Mathematica*, symbols that are not available on the keyboard can either be input by using input aliases or palettes. We extended these facilities by making it possible to input arbitrary logicographic symbols (declared by the user as explained above) by either input aliases or palettes. For example, typed text 'ESC *mg* ESC' changes into the corresponding logicographic symbol. (Here ESC indicates the escape key.)

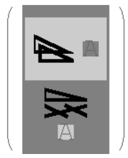
In order to give the users the possibility to invent arbitrary new symbols, a facility to create new characters is needed. One way for doing this is to use a *Mathematica* inline cell with GraphicsData of images or postscript. Note that, in this way, we can even use picture images for creating new mathematical symbols. We are currently also experimenting with using touchscreens or digital notepads etc. for drawing new symbols and importing them into *Mathematica*. The slots of logicographic symbols for the arguments can be created by just including the corresponding variable names occurring in the logicographic declaration as a text into the logicographic symbol created.

■ Proof of the Correctness of Merge-Sort

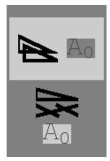
Now we present part of the proof of the correctness of the merge-sort algorithm, which in fact can be generated automatically by Theorema. If the appropriate option is set in the prover call, the proof generated uses the logicographic symbols declared above:

Prove:



We use course of value induction on A. Let A_0 be arbitrary but fixed and assume (ind-hyp) $\underset{|A|<|A_0|}{\forall_{A}}$  ,

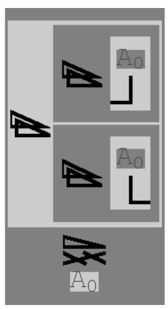
and show

(G)  .

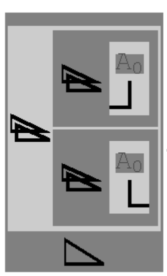
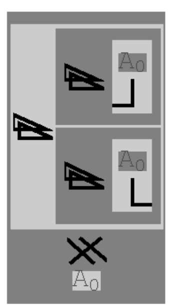
We prove (G) by case distinction using (Algorithm: stmg).

Case $|A_0| \leq 1$: ... (*this easy part of the proof is omitted in this paper*) ...

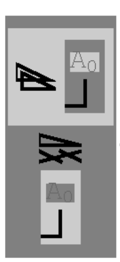
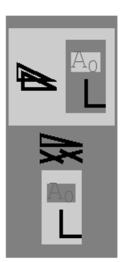
Case $|A_0| \not\leq 1$: In this case we have to prove

 .

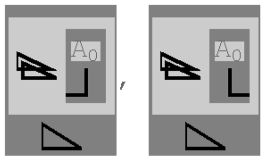
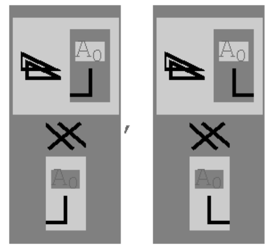
By (Definition: istv), we have to prove

(G1)  , (G2)  .

By (ind-hyp), we know

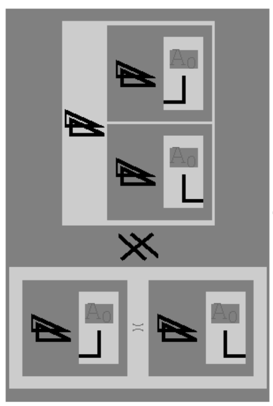
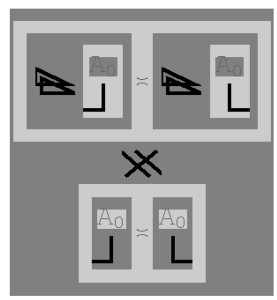
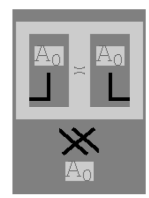
(K)  ,  .

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

(K1) , (K2) .

We prove (G1): ... (*this easy part of the proof is omitted in this paper*) ...

We prove (G2): We know (C1) by (Lemma: mg2), (C2) by (Lemma: extra), (C3) by (Lemma: extra):

(C1) , (C2) , (C3) .

Hence, by (C1), (C2), (C3), and (Lemma: extra), (G2) is proved.

We believe that the use of logicographic symbols in such a formal proof significantly supports readability. In fact, a proof becomes a little game with pictures which, however, is completely formal and general.

Example: Relations and Functions in Set Theory

■ Composition of Logicographic Symbols

With the example of logicographic notation for relations and functions in set theory we want to illustrate an advanced feature of the logicographic concept, namely the composition of new logicographic symbols from given ones in such a way that the meaning of a composed symbol is just the conjunction of its constituents. As a well known example, consider the logical connective '$\Longleftrightarrow$' which, of course, should be understood as the composition of '$\Longrightarrow$' and '$\Longleftarrow$' and, correspondingly has the combined meaning of the two constituent arrows. In the concept of logicographic symbols, we implement this natural technique in its full generality. In fact, this technique (in a not completely systematic way) is frequently used in pictorial writing systems like the Chinese and Japanese Kanji system.

For implementing this general technique of symbol composition we, first, introduce a new construct into the Theorema expression language, namely '$\curlywedge$'. The construct '$\curlywedge$' is different from the symbol '$\wedge$', which is ordinary conjunction. Namely, for example, the expression

$$\texttt{(rel} \curlywedge \texttt{tot)[A, r, B],}$$

by an appropriate inference rule, can be transformed into

$$\texttt{rel[A, r, B]} \wedge \texttt{tot[A, r, B],}$$

i.e. '$\curlywedge$' is a construct that combines predicate constants (whereas '$\wedge$' combines formulae).

Now, in the frame of our logicographic notation, when the system encounters a formulae that contains '$\curlywedge$' and logicographic symbols are declared for the individual constants combined by '$\curlywedge$', the composition of the corresponding logicographic symbols is produced at the appropriate position in the formula. Conversely, if the system encounters input with a logicographic symbol that is the composition of elementary logicographic symbols, then the system, internally, produces the corresponding formula using the '$\curlywedge$'.

■ Example of Elementary Predicates Expressed by Logicographic Symbols

Let us consider, as an example, the following five definitions of elementary notions in set theory:

$$
\begin{array}{l}
\texttt{Definitions}\Big[\texttt{"relations", any[A, r, B],} \\
\quad \texttt{isrel[A, r, B]} \Leftrightarrow (\texttt{r} \subseteq \texttt{A} \times \texttt{B}) \\
\quad \texttt{isltot[A, r, B]} \Leftrightarrow \underset{a\in A}{\forall}\ \underset{b\in B}{\exists}\ \langle \texttt{a, b}\rangle \in \texttt{r} \\
\quad \texttt{isrtot[A, r, B]} \Leftrightarrow \underset{b\in B}{\forall}\ \underset{a\in A}{\exists}\ \langle \texttt{a, b}\rangle \in \texttt{r} \\
\quad \texttt{islfun[A, r, B]} \Leftrightarrow \underset{b1\in B, b2\in B}{\forall}\ \underset{a\in A}{\forall}\ (\langle \texttt{a, b1}\rangle \in \texttt{r} \wedge \langle \texttt{a, b2}\rangle \in \texttt{r} \Rightarrow \texttt{b1} = \texttt{b2}) \\
\quad \texttt{isrfun[A, r, B]} \Leftrightarrow \underset{a1\in A, a2\in A}{\forall}\ \underset{b\in B}{\forall}\ (\langle \texttt{a1, b}\rangle \in \texttt{r} \wedge \langle \texttt{a2, b}\rangle \in \texttt{r} \Rightarrow \texttt{a1} = \texttt{a2}) \quad \Big]
\end{array}
$$

Let us now declare logicographic symbols for these notions:

$$
\begin{array}{l}
\texttt{LogicographicNotation}\Big[\texttt{"relations", any[A, r, B],} \\
\quad \texttt{isrel[A, r, B] (r "is a relation between " A "and" B)} \leftrightharpoons A \overset{r}{\text{———}} B \\
\quad \texttt{isltot[A, r, B] (r "is left total on " A "and" B)} \leftrightharpoons A\,| \overset{r}{\quad\quad} B \\
\quad \texttt{isrtot[A, r, B] (r "is right total on " A "and" B)} \leftrightharpoons A \overset{r}{\quad\quad} |\,B \\
\quad \texttt{islfun[A, r, B] (r "is left functional on " A "and" B)} \leftrightharpoons A < \overset{r}{\quad\quad} B \\
\quad \texttt{isrfun[A, r, B] (r "is right functional on " A "and" B)} \leftrightharpoons A \overset{r}{\quad\quad} > B \quad \Big]
\end{array}
$$

■ Example of Composing Predicates

What we now want is that, after this definition and this declaration, the user can write a formula like

$$
\begin{array}{l}
\texttt{Proposition}\Big[\texttt{"transitivity",} \\
\quad \left(A \overset{r}{\Longleftarrow} B \wedge B \overset{s}{\Longleftarrow} C\right) \Rightarrow A \overset{r \circ s}{\Longleftarrow} C \Big]
\end{array}
$$

and that this will be understood internally by the system as

```
Proposition["transitivity",
  ((isrel[A, r, B] ∧ islfun[A, r, B]) ∧
    (isrel[B, s, C] ∧ islfun[B, s, C])) ⟹
  (isrel[A, r∘s, C] ∧ islfun[A, r∘s, C])]
```

With our extended logicographic symbols tool this is achieved automatically by, first, decomposing the above logicographic symbols generating

```
Proposition["transitivity",
  ((isrel⋏islfun)[A, r, B] ∧ (isrel⋏islfun)[B, s, C]) ⟹
  (isrel⋏islfun)[A, r∘s, C]]
```

and then using the general inference rule for '⋏'.

In other words, by giving the five declarations and definitions above, the user has now actually 2^5, i.e. 32 different notions available!

For making this work, one actually has to enter the above declaration with the keyword 'CompposibleLogicographicDeclaration' so that the system can make sure that all the symbols that should be composable have a standard size so that just overlapping them generates a reasonably composed symbol.

■ Existentialization of Formulae

Together with the logicographic mechanism, we also allow in Theorema the mechanism that certain arguments in an n-ary predicate can be left out with the implicit meaning that the corresponding variable is existentially quantified. For example, again after having introduced the declarations in the previous section, the following formula

$$\texttt{Proposition["transitivity",}\ (A \longleftarrow B \wedge B \longleftarrow C) \Rightarrow A \longleftarrow C]$$

will generate the following internal formula

$$\texttt{Proposition}\Big[\texttt{"transitivity",}\ \Big(\underset{r1}{\exists}\ (\texttt{isrel}\curlywedge\texttt{islfun})[\texttt{A, r1, B}] \bigwedge \underset{r2}{\exists}\ (\texttt{isrel}\curlywedge\texttt{islfun})[\texttt{B, r2, C}]\Big) \Rightarrow \underset{r3}{\exists}\ (\texttt{isrel}\curlywedge\texttt{islfun})[\texttt{A, r3, C}]\Big]$$

With this additional mechanism, by a few declarations and definitions, one can now generate huge classes of logicographic symbols with varying arity and corresponding predicates with both exact formal and intuitive meaning at the same time. For example, the meaning all of the following should now be clear:

$$A \ \overset{r}{\longmapsfrom\!\!\!\!\longmapsto} \ B$$

$$A \ \overset{r}{\longleftrightarrow} \ B$$

$$\overset{r}{\longrightarrow}$$

$$A\mathrm{x}B \ \overset{r \circ s}{\longmapsfrom\!\!\!\!\longmapsto}$$

Conclusion

A picture can only illustrate a predicate or function for concrete argument values. A textual identifier used as a predicate or function constant, together with argument terms containing variables, can describe a general situation - for example as part of a proof. A picture together with slots for argument terms, i.e. a logicographic symbol, can illustrate the intuition behind a concept *and* can describe general situations. We believe that this combined expressive power of logicographic symbols opens new levels of both intelligibility and formality in the representation of mathematics. This may have significant consequences for future math research and teaching and, maybe more importantly, for the computer-supported management of mathematical knowledge.

Tracing back the history of human language one sees that, in very ancient times, form ("syntax") and meaning ("semantics") of language was as close together as can be. Gestures are what they mean, exclamations just reflect the status of the body, pictures are homomorphic images of reality. In ancient Vedic literature, for example, the vibration of sounds ("syntax") and the reality ("vibration of matter") they describe were considered to be identical. Over the centuries, by the expansion of rational thinking, the distinction between syntax and semantics became pronounced and, in fact, was the major tool for the purification of scientific language. This had the positive effect of, ultimately, making language processable. However, there is also a negative side to this separation of syntactical form and meaning: What is formally processable (i.e. processable by considering only the syntactical form), is sometimes hard to "see" (i.e. to understand on the level of meaning) and vice versa. Seen in this broader context, our technique of logicographic symbols is meant to be a natural approach for the reconciliation between syntax and semantics.

References

[Buchberger et. al. 1997] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vãsaru. 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.

[Buchberger et. al. 2000] B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vãsaru and W. Windsteiger. The Theorema project: a progress report. In M. Kerber and M. Kohlhase, editors, *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.

[Buchberger 2000] B. Buchberger. Logicographic symbols: some examples of their use in formal proofs. Manuscript, RISC (Research Institute for Symbolic Computation), Johannes Kepler University, Linz, Austria, February 2000.

[Nakagawa 2001] 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.

[Nakagawa and Buchberger 2001] K. Nakagawa, and B. Buchberger. Presenting Proofs Using Logicographic Symbols. In *Proc. of the Workshop on Proof Transformation and Presentation and Proof Complexities* (PTP-01), Siena, Italy, June 19, 2001.

[Knuth 1986a] D. E. Knuth. *The T_EXbook.* Computer & Typesetting, Addison-Wesley, 1986.

[Knuth 1986b] D. E. Knuth. *The METAFONTbook.* Computer & Typesetting, Addison-Wesley, 1986.

[Harris 1996] J. Harris. Mathematica Notation Package. Accessible from the AddOn Help Browser `AddOn->Extras->Utilities->Notation Package`. Also http://library.wolfram.com/packages/notation/.