Two Tools for Mathematical Knowledge Management in Theorema

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: Theorema is a mathematical software system, programmed in *Mathematica*, that aims at computer-supporting all phases of mathematical activity. The manipulation of mathematical knowledge bases is a major issue in Theorema, which is discussed in other papers. In this paper, we describe two special techniques of Theorema that address two special subproblems of mathematical knowledge management:

- the technique of *logicographic symbols* addressing the problem of combining formal rigor and intuitive understanding of mathematical knowledge and
- the technique of *composing mathematical constants* from constituents addressing the problem of automated generation of variants of mathematical notions.

1. Introduction

Theorema is a software system, programmed in *Mathematica*, for computer-supporting the entire process of mathematical exploration, see [Buchberger 1996], [Buchberger et al. 1997] and [Buchberger et al. 2000]. The Theorema core library consists of various prover, solvers, and simplifiers for mathematical expressions in a uniform logical frame, for which we chose a version of predicate logic. Proving, solving, and simplifying within Theorema are always relative to an explicitly specified knowledge base, i.e. a set of mathematical statements.

The manipulation of mathematical knowledge bases is an essential feature in the conception of Theorema. Correspondingly, Theorema has a couple of language constructs for composing knowledge bases in a hierarchical way and for choosing well-defined portions of knowledge bases as an essential input argument to the provers, solvers, and simplifiers of Theorema. Also, for the practical usage of Theorema, suggestions are given how to organize the exploration of mathematical theories. In particular, it is suggested not to consider the generation of proofs for isolated theorems but, instead to explore all possible consequences of the introduction of new concepts in relation with the knowledge already derived for previously introduced notions, see [Buchberger 2000]. The general aspect of mathematical knowledge management in Theorema is discussed in a companion paper, see [Buchberger 2001].

In this paper, we concentrate on two special, but hopefully widely applicable, tools of Theorema that contribute to managing mathematical knowledge:

- the tool of logicographic symbols and
- the tool of composing mathematical constants from constituents.

These two tools address two particular problems of mathematical knowledge management:

- the problem of combining formal rigor and intuitive understanding of mathematical knowledge and
- the problem of automated generation of variants of mathematical notions.

Although the two problems are logically independent, we will see that the solution of the two problems proposed in this paper allows to combine the tools in a natural way.

The concept of logicographic symbols and of systematic creation of notion variants were introduced by the second author in various talks and technical reports starting from around 1999. The contribution of the first author consists in the implementation of the two techniques in the frame of the Theorema system and is documented in detail in the forthcoming thesis [Nakagawa 2001].

2. Tool of Logicographic Symbols

2.1. The Problem of Combining Formal Rigor and Intuitive Understanding

It is understood that computer-supported mathematical knowledge management can only be based on formal mathematics. By definition, this means that all logical derivations can only be based on formal rules rather than on the intuitive understanding of the concepts involved. In contrast, inventing and teaching mathematics is essentially based on the intuitive understanding of concepts in a complex, implicit context. In fact, the actual work of mathematicians (in research, teaching, and application) proceeds in alternating phases of

- conjecturing facts about mathematical concepts based on intuition in examples and
- formal verification of conjectures exclusively based on the application of formal inference rules.

Both, intuitive understanding and formal reasoning, and the intimate interaction between intuitive understanding and formal reasoning is essential for making progress in mathematics:

- Intuitive understanding (the "exciting phase" of mathematics) leads efficiently to "interesting" insight avoiding the complexity of systematic trial-and-error in large search spaces but includes the disadvantage and risk of drawing wrong conclusions for the general case from particular observations in the concrete examples.
- Formal reasoning (the "cool phase" of mathematics) gives reliable truth for the general case but often is hard to understand or, at least, boring.

As a matter of fact, one often observes that, in mathematical presentations (paper or talks) both the formal derivations *and* intuitive explanations in the form of examples (in particular graphical illustrations) are given in parallel because one feels that neither the formal derivations nor the intuitive illustrations alone are sufficient. For the general goal of computer-supported mathematical knowledge management it will therefore be of utmost importance to combine the rigor of formal reasoning with the possibility of making intuitive understanding easy.

2.2. The Concept of Logicographic Symbols

By the concept of logicographic symbols we offer a mechanism by which formal derivations *and* intuitive presentations are combined. The concept of logicographic symbols is general, i.e. independent of any particular logic or software system for mathematics. However, we took our Theorema software system as a test-bed for the implementation of the concept and its smooth integration into a general computer-supported system for mathematics, in particular, for mathematical knowledge management.

The main idea of logicographic symbols is to provide illustrations for mathematical concepts that have "slots" for arguments so that the illustrations, in addition to conveying intuitive meaning, are general. In other words, logicographic symbols are nothing else than the ordinary function and predicate constants of predicate logic with slots for argument terms (in this respect the logicographic symbols are "logico-") but in graphical, two-dimensional, user-definable notation rather than in linear identifier (string of characters) notation (in this respect the logicographic symbols are "graphical"). In fact, historically, most of the symbols of mathematics (for example, the symbol "<" for "less") are logicographic in the exact sense that they are both syntactically correct predicate logic constants with parameter slots as well as little pictures conveying the intuitive meaning of the concepts denoted by the constants. Our tool of logicographic symbols can thus be seen as a natural extension of what mathematicians did over the centuries.

2.3. Example: Correctness of Merge–Sort

Knowledge in Theorema is described by so called *Theorema formal text* [Buchberger et. al.1997]. Fig. 1 shows a part of the theory describing the correctness of the merge-sort algorithm in the Theorema formal text. 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 the statement afterwards. For example, in Fig. 1 by Algorithm["mg"] one can refer to the definition of the function 'mg'. The expression 'any[*variables*]' 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 example, this statement is the definition of the function 'mg'. In the Theorema basic notation, '⟨⟩', '⟨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 tuple X and tuple Y' respectively.

```
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 Part of Formalized Merge-Sort Theory

In fact the knowledge base in Fig. 1 is full of logico-graphic notation, e.g. '≥', '≍', '⟺', '∧' taken from the limited vocabulary of symbols available in *Mathematica*. Now we introduce new notation which should convey the meaning of the notions. Fig. 2 shows a possible choice of logicographic symbols for the merge-sort theory.

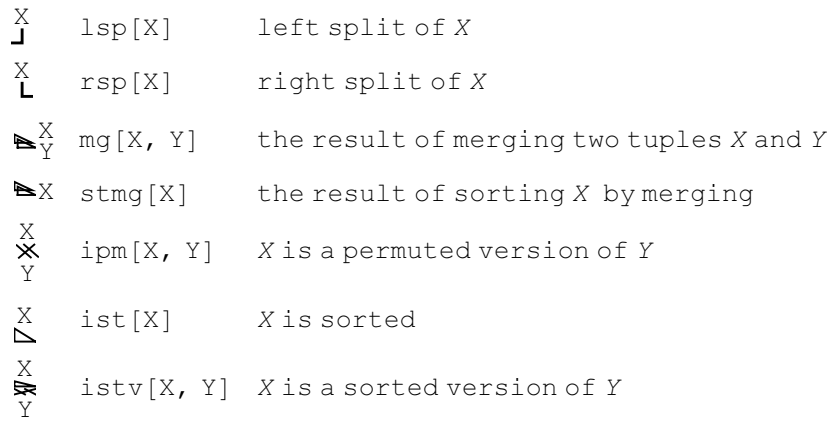


Fig. 2 Logicographic Symbols for Merge-Sort Theory

With these logicographic symbols, we can represent the knowledge base in the way shown in Fig. 3.

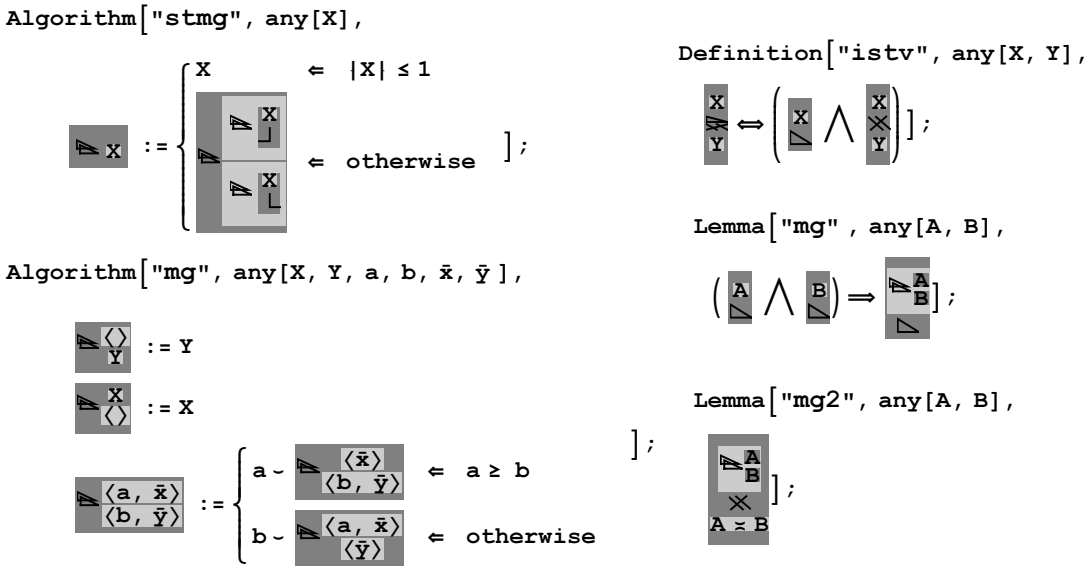
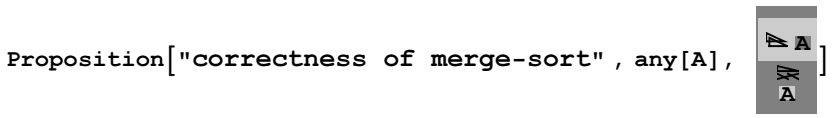


Fig 3. Formalized Merge - Sort Theory with Logicographic Symbols

The correctness of merge-sort can be expressed as follows:



In Theorema, the formulae in the knowledge base in Fig.3 and the correctness theorem can be manipulated in completely the same way as their counterparts in linear notation. As an illustration of using this notation, we will give the correctness proof of the merge-sort algorithm in the appendix.

Theorema provides a couple of tools for designing, declaring, defining and using logicographic symbols. The details are described in [Nakagawa, Buchberger 2001] and [Nakagawa 2001].

3. Tool of Composing Mathematical Constants

3.1. The Problem of Automated Generation of Variants of Mathematical Notions

When a mathematical notion is introduced, typically, it is implicitly understood that many variants of the notions are also available for future use. In informal mathematics these variants are not explicitly defined because the formation of the variants follows certain patterns that are common sense. In formal mathematics, however, it would be necessary to give explicit definitions of all variants for making them available in future formal derivations. For example, when the notions of "relation", of "functionality in the first argument" and of "totality in the first argument" are defined it is also clear what we mean by a "total functional relation" (a "total function"). Also, if we have the notions of "less" and "equal", we also have the notion "less equal". As a third example, if the (binary) notion of "is limit of" is defined, it is clear what we mean by a "convergent" sequence (i.e. a sequence "having a limit").

In fact, the common rules behind the formation of variant notions are different in the three examples:

Example "total function": Formally, if we conceive "is–relation", "is–right–functional", and "is–left–total" as three ternary predicate constants, then "is–total–function" would be naturally defined by

$$\texttt{is-total-function[f, A, B]} :\Leftrightarrow \texttt{is-relation[f, A, B]} \wedge \texttt{is-right-functional[f, A, B]} \wedge \texttt{is-left-total[f, A, B]},$$

i.e. the common rule behind forming this variant notion is "conjunctive composition" of the ingredient notions.

Example "less equal": Formally, if we conceive "less" and "equal" as binary predicate constants, then "less–equal" would be naturally defined by

$$\texttt{less-equal[a, b]} :\Leftrightarrow \texttt{less[a, b]} \vee \texttt{equal[a, b]},$$

i.e. the common rule behind forming this variant notion is "disjunctive composition" of the ingredient notions.

Example "convergent": Formally, if we conceive "is–limit–of" as a binary predicate constant, then "is–convergent" would be naturally defined by

$$\texttt{is-convergent[f]} :\Leftrightarrow \exists_{\texttt{a}} \texttt{is-limit-of[f, a]}$$

i.e. the common rule behind forming this variant notion is "existential quantification" over one argument of the ingredient notion.

The problem is that, given a notion, creating all the variants explicitly (i.e. by explicit definitions) that are usually understood implicitly would be a complex task. By the tool of "variant formation" within Theorema we provide a systematic mechanism by which the user can specify which common rules he would like to be applicable to certain ingredient notions for providing variants of the notions in a systematic way. Our tool is very flexible so that we can specify all types of quantification and propositional composition. This saves enormous complexity in the practical formulation and exploration of theories. The Theorema tool of "variant formation" is best explained in an example.

3.2. Example: The Notion of Function in SetTheory

For example, if we consider the five elementary ternary predicates "is–relation", "is–left–functional", "is–left–total", "is–right–functional", and "is–right–total", with their corresponding five definitions:

```
Definitions["relations", any[r, A, B],

 is-relation[r, A, B] ⇔ (r ⊆ A × B)
 is-left-total[r, A, B] ⇔ ∀a∈A ∃b∈B ⟨a, b⟩ ∈ r
 is-right-total[r, A, B] ⇔ ∀b∈B ∃a∈A ⟨a, b⟩ ∈ r
 is-left-functional[r, A, B] ⇔ ∀a1∈A,a2∈A ∀b∈B (⟨a1, b⟩ ∈ r ∧ ⟨a2, b⟩ ∈ r ⟹ a1 == a2)
 is-right-functional[r, A, B] ⇔ ∀b1∈B,b2∈B ∀a∈A (⟨a, b1⟩ ∈ r ∧ ⟨a, b2⟩ ∈ r ⟹ b1 == b2)
]
```

The notion of being a function can now be defined by the following formula:

```
is-function[f, A, B] : ⇔ is-relation[f, A, B] ∧ is-right-functional[f, A, B].
```

However if we continue this way for various variants, e.g. is-total-function, is-bijective-function, we would have to introduce 2^5-6(=26) new definitions. Instead, in Theorema we introduce a tool to combine existing function and predicate symbols by introducing a new construct '∧' (wedge) which is different from the logical construct '∧'(and). For example, with the construct we can have following expression:

```
(is-relation ∧ is-right-functional)[f, A, B].
```

Here, from the logic point of view, 'is–relation∧is–right–functional' is nothing else than a new predicate constant but the system understands the meaning systematically by appropriate inference rules in the Theorema provers, which have basically the same effect as adding the following definition to the knowledge base.

```
(is-relation ∧ is-right-functional)[f, A, B]
  : ⇔ is-relation[f, A, B] ∧ is-right-functional[f, A, B]
```

Similarly, we introduce a convention of omitting arguments: If we have the ternary notion '**is-relation**', we want to attach the following meaning automatically.

```
is-relation[A, B] : ⇔ ∃f is-relation[f, A, B]
```

However, here the problem is that it is ambiguous which argument should be omitted. In order to specify the argument to be omitted the special argument 'Null' is used. Then,

```
is-relation[Null, A, B] : ⇔ ∃f is-relation[f, A, B]
```

whereas

```
is-relation[f, Null, Null] : ⇔ ∃A,B is-relation[f, A, B].
```

At first sight, this solution does not seem to be elegant but by the combination with the logicographic symbols shown the technique allows very concise and intuitively appealing notation.

With this existential quantification, the variety of available variant notions increases further without any additional effort containing all the practically interesting and important variants of the basic notions. For example, the notion of two sets having equal cardinality can now be specified by the following expression:

```
(is-relation ∧ is-left-functional ∧ is-left-total
    ∧ is-right-functional ∧ is-right-total)[Null, A, B].
```

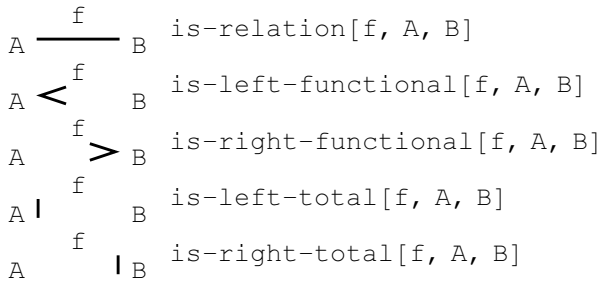
It is automatically understood by the system that this notion is defined as follows:

```
(is-relation ∧ is-left-functional ∧ is-left-total
    ∧ is-right-functional ∧ is-right-total)[Null, A, B]
 : ⇔∃f (is-relation[f, A, B] ∧ is-left-functional[f, A, B] ∧ is-left-total[f, A, B] ∧
      is-right-functional[f, A, B] ∧ is-right-total[f, A, B]).
```

All details of implementing the tools for the automatic generation of all variants of notions along the above lines are described in [Nakagawa 2001].

4. Attractive Combination of Two Tools

Logically, the technique of logicographic symbols and the technique of automated variant notions creation are independent. However, in fact, the combination of the two techniques is particularly attractive. For example, a possible choice of logicographic symbols for the theory of relations and functions in set theory is



Now it is natural to associate the logicographic representation '$_{A}\overset{f}{\longmapsto}_{B}$' with the expression

```
(is-relation ∧ is-left-functional ∧
   is-left-total ∧ is-right-functional ∧ is-right-total)[f, A, B]
```

The representation '$\longmapsto$' is automatically composed by the logicographic symbols of 'is–relation', 'is–left–functional', 'is–left–total', 'is–right–functional', and 'is–right–total', namely '$\longrightarrow$', '$<$', '$>$', '$\mid$', and '$\mid$'.

Also, in logicographic symbols, omitting arguments as notation for quantification is represented by just omitting slots. For example, in the previous example, the logicographic representation of

```
(is-relation ∧ is-left-functional ∧
   is-left-total ∧ is-right-functional ∧ is-right-total)[Null, A, B]
```

becomes '$_{A}\longmapsto_{B}$'. And evaluating '$_{A}\longmapsto_{B}$' then, internally, is the expression above.

In order to exploit this natural correspondence between the propositional combination of predicate logic constants and the graphical composition of logicographic symbols and the correspondence between implicit quantification and omission of slots in logicographic symbols we provide a couple of tools in Theorema whose details are described in [Nakagawa 2001].

By these tools, the formal definitions of all possible 247(=$(2^5\text{-}1)\cdot 2^3$) variants that result from the above five basic notions by conjunction and existential quantification are available "on demand" in the system without storing these definitions explicitly in advance. In other words, these definitions will be created "lazily" (and processed by corresponding inference rules) at the moment when they are needed. For example, in a proof of the theorem

$$_{A}\longmapsto_{B} \wedge {}_{B}\longmapsto_{C} \Rightarrow {}_{A}\longmapsto_{C}$$

only at the moment when the assumption '$_{A}\longmapsto_{B}$' is used the corresponding definition

```
A |<—>| B  :⇔
∃f (is-relation[f, A, B] ∧ is-left-functional[f, A, B] ∧ is-left-total[f, A, B] ∧
   is-right-functional[f, A, B] ∧ is-right-total[f, A, B] )
```

will be created and used.

5. Conclusion

Summarizing, by the combined technique of logicographic symbols and systematic notions variant creation we believe that the gap between formal rigor and intuitive understanding, in many situations, can be closed in systematic way. In other words, these two techniques help to reconcile the distinction between syntax and semantics, which of course was of utmost importance for developing mathematics to its present level but, on the other hand, is sometimes felt to be an obstacle for the acceptance of mathematics as a general frame for expressing truth in real-world context. Interestingly enough, in ancient systems of philosophy and language, as for example the Chinese/Japanese kanji system or the Vedic literature, the distinction between syntax and semantics was much less pronounced. The development of mathematics (and in particular logic) can also be seen as starting, some thousands of years ago, from a situation in which syntax and semantics was basically identical and developing through a phase in which the distinction between syntax and semantics was seen to be of crucial importance arriving now at a situation where we can close the gap between syntax and semantics in a conscientious way. This will have a decent impact on the feasibility and acceptability of mathematical knowledge management systems.

6. References

[Buchberger 1996] B. Buchberger. Symbolic Computation: Computer Algebra and Logic. Frontiers of Combining Systems, F.Baader and K.U.Schulz (eds.), Kluwer 1996, pp.193-220.

[Buchberger 2000] B. Buchberger. Theory Exploration with Theorema. Proceedings of the 2nd International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2000), Dept. of Computer Science, Univ. of the West Timisoara, Romania, Oct. 4-6, 2000, pp. 1-16.

[Buchberger 2001] B. Buchberger. Mathematical Knowledge Management Using Theorema. Proceedings of Workshop on Mathematical Knowledge Management, Hagenberg, Sep. 24-26, 2001.

[Buchberger et al. 1997] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A Survey on the Theorema Project. Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation), Maui, Hawaii, Jul. 21-23, 1997, W. Kuechlin (ed.), ACM Press 1997.

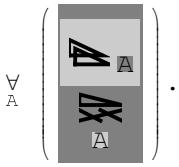
[Buchberger et al. 2000] B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema Project: a Progress Report. M. Kerber and M. Kohlhase (eds.), Proceedings of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, St. Andrews, Scotland, Aug. 6-7, 2000.

[Nakagawa, Buchberger 2001] K. Nakagawa and B. Buchberger. Presenting Proofs Using Logicographic Symbols. Proceedings of the Workshop on "Proof Transformation and Presentation" (A. Fiedler, H. Horacek eds.) at the IJAR-2001, Siena, Jun. 18, 11 pages.
Also available as a SCCH technical report (http://www.scch.at/research/publications/522/index.html).

[Nakagawa 2001] K. Nakagawa. Advanced input / output features in the Theorema system. RISC (Research Institute for Symbolic Computation), Johannes Kepler University, Linz, Austria, 2001. Forthcoming PhD thesis.

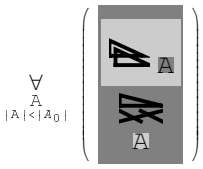
A. Apppendix: Proof of the Correctness of Merge–Sort

Prove:



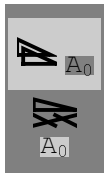
We use course of value induction on A. Let now A_0 be arbitrary but fixed tuple and assume

(ind-hyp)



and show

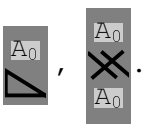
(G)



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

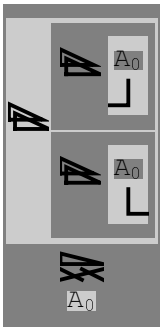
Case $|A_0| \leq 1$: We have to prove

By (Definition: istv), we have to prove



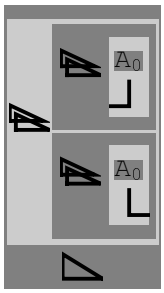
These are true, because (properties of sorted tuples) and (reflexivity of perm).

Case $|A_0| \not\leq 1$: We have to prove

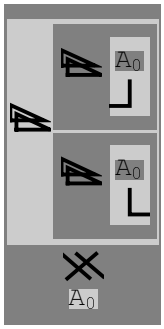


By (Definition: istv), we have to prove

(G1)

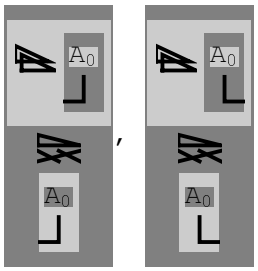


(G2)



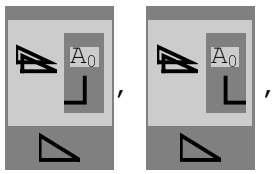
By (ind-hyp), we know

(K)

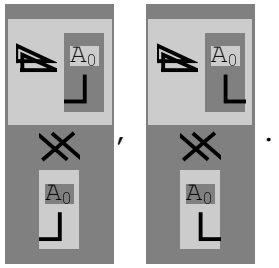


And hence, by (Definition: istv) we also know

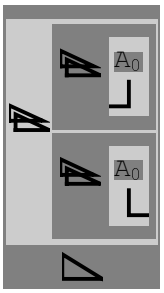
(K1)



(K2)

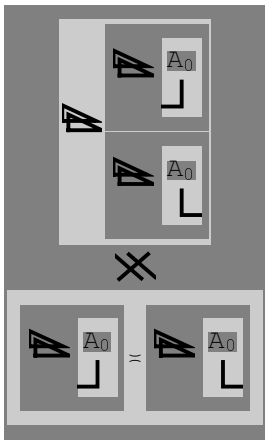


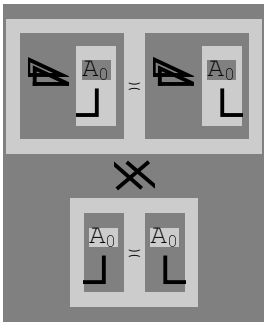
We prove (G1): By (Lemma: mg) and (K1),

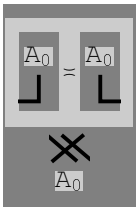


.

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 permutation), (G2) is proved.