Proving through commutative diagrams

Yoshiki Kinoshita and Koichi Takahashi
Electrotechnical Laboratory, Tsukuba, 305 Japan

December 28, 1995

1 Introduction

In category theory, commutative diagrams are extensively used to help understand (informal) proof written in natural language and logical formulae. So, it seems natural to require the use of commutative diagrams in *formal* theorem proving in category theory too. In fact, the difficulty in understanding formal proof is one of the major obstacles in formal mathematics, so assistance in obtaining an intuitive idea of what is written would be of great value. Therefore, we are developing a user interface for proof assistant systems, based on commutative diagrams, and we report its functional specification.

In the course of developing our interface, we encountered a general user interface framework based on relations, rather than by simpler functions. This seems a new framework and we believe it is a contribution to visual logical systems, and we describe it in considerable depth.

What we wish is to represent formulae such as

$$\begin{array}{l}(\forall X)\mathrm{Obj}(X) \supset \\ \quad (\forall f)(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset \\ \quad\quad (\forall g)(\mathrm{source}(g) = X \wedge \mathrm{target}(g) = B) \supset \\ \quad\quad\quad (\exists h)(\mathrm{source}(h) = X \wedge \mathrm{target}(h) = A \times B) \wedge \\ \quad\quad\quad\quad (f = h\,;p_1) \wedge (g = h\,;p_2)\end{array} \tag{1}$$

by a diagram such as that in Figure 1. The problem of how to deal with very large formulae is now a major topic in formal proof development. Our aim is to propose a solution of this problem by providing a good user interface based on commutative diagrams to proof assistants.

While trying to model our commutative diagram interface, we encountered a difficulty which seems to occur when modeling many other user interfaces, too. Our model of user interface have two attributes; items which are the object of computation and items used to communicate with the user. We call the former "computed objects" and the latter "communication objects." The user gives a communication object to the system which then translates it to a *corresponding* computed object and manipulates (or computes) it. Or alternatively, the

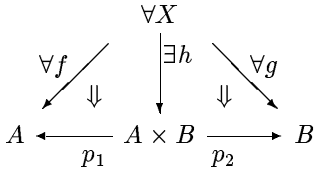


Figure 1: An example of diagram.

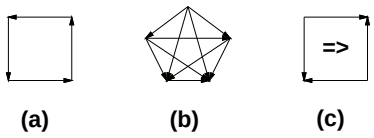


Figure 2: Non-examples of diagrams.

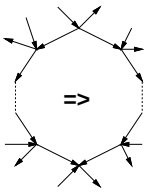


Figure 3: An exmaple of commutative cell.

system has a computed object as a result of some computation, translates it to a *corresponding* communication object, and then shows it to the user. In the case of our system, computed objects are logical formulae and communication objects are commutative diagrams. We have two "correspondences" here, and it would seem natural to search for *functions* to describe them. However, there are no such functions which are satisfactory. Instead, we have to live with *relations*, and the model must be more complicated. So, we propose what we call this a *relational model*, and we describe it in Section 4.

This paper is organised as follows. First, we give a formulation of commutative diagrams in Section 2, in terms of directed graphs with cells. Next, we describe the commands of the system. The commands are classified into draw commands and proof commands. We list up all draw commands, and then show how they work by example in Section 3. We then discuss the general framework of the user interface, and propose our own based on relations in Section 4. In Section 5, we explain the proof commands, which are also classified into "pasting" commands, to prove equations, and "logical" commands for each logical inference rule, to prove logical formulae. Finally, related works are reviewed in Section 6.

2 Commutative diagrams

Figure 1 shows a commutative diagram. But what exactly are commutative diagrams? In order to treat them by machines, we need their formal definition. We define commutative diagrams to be directed acyclic planar graphs (i.e., planar DAG) with labels attached to some nodes and edges, together with specified *commutative cells*. A directed graph consists of nodes and edges with the assignment of the source and target node to each edge. By "acyclic," we mean that there are no cycles. So Figure 2(a), for example, is not a commutative diagram. "Planar" means the graph can be embedded in the plane. Figure 2(b) has no such embeddings, so it is not a commutative diagram. Labels are attached to nodes and edges. Typical labels are variable names, possibly quantified by $\forall$ or $\exists$.

To define commutative cells (Figure 3), we need some auxiliary terminology. We define *cells* to be the minimal region surrounded by edges. Also, *paths* of a graph are defined as usual. We construct a *commutative cell* by placing a double

command	description
$\cdot$	put a node
$\rightarrow$	draw a edge
$A, \forall X, \exists h$	labeling to a node or edge
$\Rightarrow$	draw a double arrow

Figure 4: Draw commands.

arrow in a cell surrounded by just two paths. Figure 2(c) is not a commutative cell, because it is surrounded by four paths.

For example, Figure 1 is a diagram in this sense; it has two commutative cells.

3 Draw commands

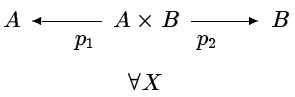
Figure 4 shows the list of draw commands. These commands are used to draw commutative diagrams. The commands are "to put a node," "to draw an edge from a node to another," "to put a label on a node or edge," and "to draw a double arrow in a cell, to make a commutative cell." Clearly, an arbitrary diagram can be drawn using a sequence of these commands.

Figure 5 shows how our example diagram from Figure 1 can be drawn. In (2), we put nodes "A," "$A \times B$," and "B," and draw edges "p_1" and "p_2." We then put another node and attach a label "$\forall X$" to it in (3). We then draw an edge from $\forall X$ to A, and attach a label "$\forall f$" to that edge in (4). Similarly we draw two edges "$\forall g$" in (5) and "$\exists h$" in (6). Since the left cell is surrounded by two paths, we put a double arrow in the cell, to make it a commutative cell in (7). The right cell is similarly made commutative in (8). Thus, we get the diagram as in Figure 1.

Before proceeding further, we give an intuitive meaning to the commutative diagrams. Suppose a node in the diagrams represents a set; so, X, A, B, and $A \times B$ are sets in our example. Also, suppose an edge represents a function from its source to its target, so, f is a function from set X to set A in Figure 1. A path is a composition of functions represented by the component edges. A commutative cell asserts the equality of functions represented by the surrounding paths. The diagram in Figure 1 says the composition of h and p_1 is equal to the function f. Meaning behind the diagram in Figure 1 thus is "for any set X, for any function f from X to A, for any function g from X to B, there exists a function h from X to $A \times B$ such that f is equal to the composition of h and p_1 and g is equal to the composition of h and p_2." The formula (1) is equivalent to this.

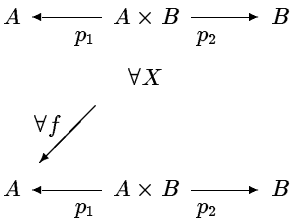
The draw commands create not only diagrams but also incrementally create formulae with a specified position, as depicted by the cursor $\triangle$ in Figure 5. We explain with the last drawing example how a formula in each step is generated in Figure 5.

(2): the diagram contains no predicate, so the corresponding formula is the truth $\top$ with the specified position left to itself. (3): "$(\forall X)$" is first inserted to the left of the cursor position, resulting the formula $(\forall X) \mathbin{\triangle} \top$, and then "$\mathrm{Obj}(X) \supset$" is inserted to the left of the cursor position. (4): likewise, "$(\forall f)$" is first inserted to the left of the cursor, then "$(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset$" is inserted. We put the precedence of the quantifier to be the weakest, so this formula is parsed as $(\forall X)\Big(\mathrm{Obj}(X) \supset (\forall f)\big(((\mathrm{source}(f) = X) \wedge (\mathrm{target}(f) = X)) \supset \top\big)\Big)$. The creation of formulae goes on in a similar fashion. Note that h is quantified by an existential quantifier, so the logical connective used is the

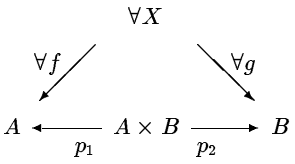


$\triangle\ \top$ (2)

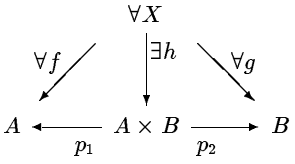
$(\forall X)\mathrm{Obj}(X) \supset\ \triangle\ \top$ (3)



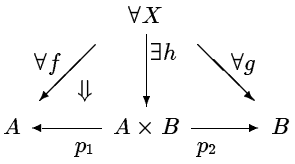
$(\forall X)\mathrm{Obj}(X) \supset$
$\quad(\forall f)(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset\ \triangle\ \top$ (4)



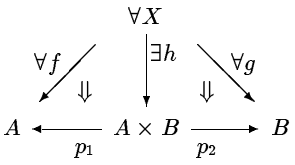
$(\forall X)\mathrm{Obj}(X) \supset$
$\quad(\forall f)(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset$
$\quad\quad(\forall g)(\mathrm{source}(g) = X \wedge \mathrm{target}(g) = B) \supset\ \triangle\ \top$ (5)



$(\forall X)\mathrm{Obj}(X) \supset$
$\quad(\forall f)(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset$
$\quad\quad(\forall g)(\mathrm{source}(g) = X \wedge \mathrm{target}(g) = B) \supset$
$\quad\quad\quad(\exists h)(\mathrm{source}(h) = X \wedge \mathrm{target}(h) = A \times B) \wedge\ \triangle\ \top$ (6)



$(\forall X)\mathrm{Obj}(X) \supset$
$\quad(\forall f)(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset$
$\quad\quad(\forall g)(\mathrm{source}(g) = X \wedge \mathrm{target}(g) = B) \supset$
$\quad\quad\quad(\exists h)(\mathrm{source}(h) = X \wedge \mathrm{target}(h) = A \times B) \wedge$
$\quad\quad\quad\quad(f = h\,;p_1) \wedge\ \triangle\ \top$ (7)



$(\forall X)\mathrm{Obj}(X) \supset$
$\quad(\forall f)(\mathrm{source}(f) = X \wedge \mathrm{target}(f) = A) \supset$
$\quad\quad(\forall g)(\mathrm{source}(g) = X \wedge \mathrm{target}(g) = B) \supset$
$\quad\quad\quad(\exists h)(\mathrm{source}(h) = X \wedge \mathrm{target}(h) = A \times B) \wedge$
$\quad\quad\quad\quad(f = h\,;p_1) \wedge (g = h\,;p_2) \wedge\ \triangle\ \top$ (8)

Figure 5: An example of drawing.

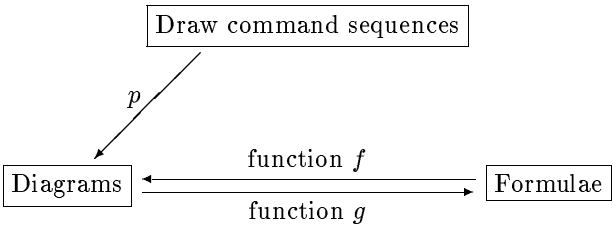


Figure 6: A functional model of user interface.

conjunction, instead of the implication. The draw command for the double arrow creates equations such as $f = h; p_1$ ($h; p_1$ denoting the composition of h and p_1), and we finally get the requisite formula.

4 Relational model of user interface

Our system displays a logical formula as a character sequence in a traditional way, or as a commutative diagram, as explained earlier. Now, given a commutative diagram typically displayed by the system, it is natural to expect that there is a unique formula represented, so that the user can determine which formula the diagram is showing. In other words, we can determine a function from diagrams to formulae. In terms of implementation, we can also reasonably expect a function from formulae to diagrams, a function which guides the implementation of the display algorithm.

This leads to what we call a *functional model of user interface*, as shown in Figure 6. We not only require the existence of functions f and g back and forth between diagrams and formulae, but also the canonical equivalence relations R between diagrams and S between formulae, as well as the following properties for all formulae ϕ and diagrams δ:

$$f(g(\delta))\ R\ \delta, \qquad \text{and} \qquad g(f(\phi))\ S\ \phi.$$

Here, g followed by f maps a diagram to an equivalent diagram, and f followed by g maps a formula to an equivalent formula. Also, we request a function p from draw command sequences to diagrams.

However, there is no successful definition of such functions! They can be very well defined, but the result (by our definition, at least) has been always unsatisfactory. There are clearly no reasonable functions from commutative diagrams to formulae (Figure 9), but there are functions from formulae to diagrams. However, all the functions that we could define map some formulae to a very cumbersome diagrams. But such over-complicated diagrams are pointless; diagrams are introduced in order to help, not hinder, understanding (Figure 8).

This shows that the functional model does not work for our case. Instead, we propose a *relational model of user interface*, shown in Figure 7. Here, there is a relation, rather than two functions, between formulae and diagrams. In addition to function p, we also have a function q from draw command sequences to formulae. In the functional model, q was available as the composition of p followed by g, but in the relational model, we need to give q explicitly. We also need canonical equivalences R between diagrams and S between formulae, as in the functional model. For all draw command sequences σ and σ', we postlate the following property:

$$q(\sigma)\ S\ q(\sigma') \supset p(\sigma)\ R\ p(\sigma'). \tag{9}$$

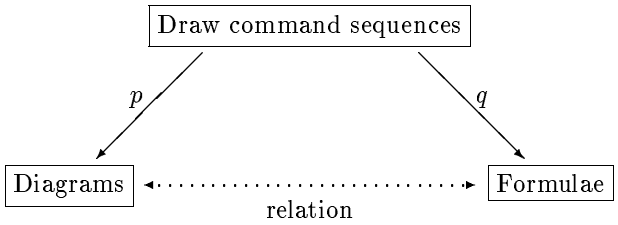


Figure 7: A relational model of user interface.

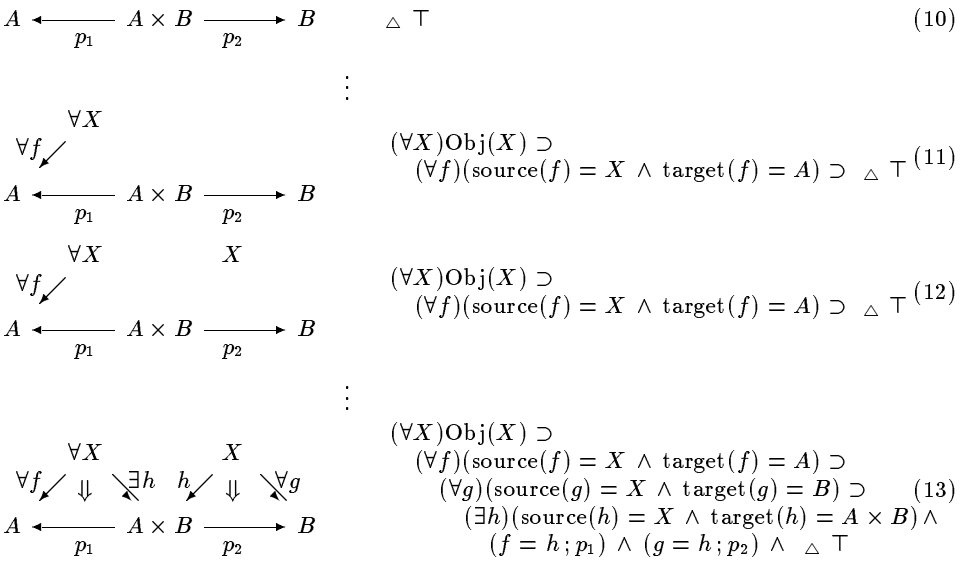


Figure 8: A formula does not determine diagrams.

There are only finitely many command sequences mapped to a given commutative diagram. Also, given a logical formula, there are only finitely many command sequences mapped to it. Moreover, those formulae are equivalent in many cases. For example, by changing $(\forall f)$ and $(\forall g)$ draw commands, the resulting formula is equivalent. So, we conjecture that in fact there is a limited, tractable number of logical formulae corresponding to a given commutative diagram.

Since a similar difficulty in finding functions arises in the design of other user interface [7], we believe our relational model is worth generalizing to some class of user interfaces. Interestingly, property (9) recalls us logical relations, the central notion in the theory of parametricity of programming languages, which might support the use of our relational model.

5 Proof commands

There are two types of proof commands: pasting commands for proving equations and logical inference commands for proving compound formulae.

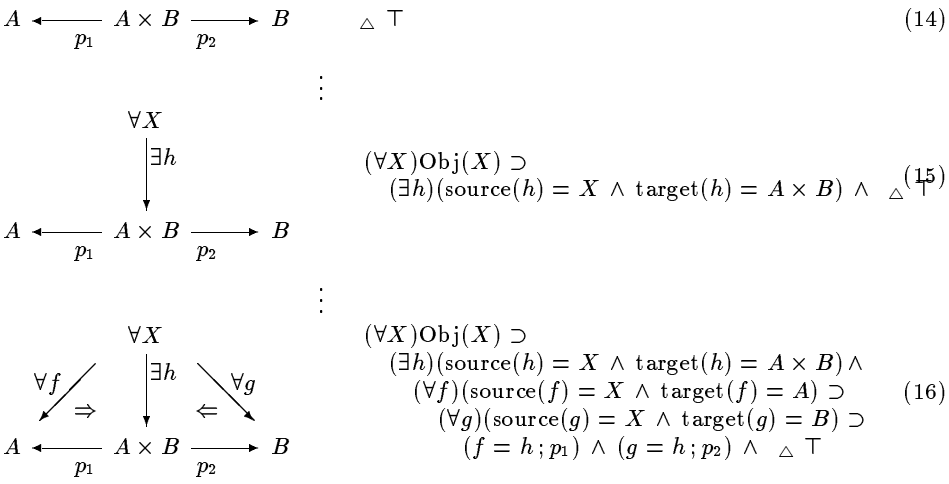


Figure 9: A diagram does not determine formulae.

Assumptions: $f = \mathrm{id}\,; f$, $\mathrm{id} = g\,; h$, $h\,; f = \mathrm{id}$, $g\,; \mathrm{id} = g$.
Goal: $f = g$.
Proof: $f = \mathrm{id}\,; f = g\,; h\,; f = g\,; \mathrm{id} = g$.

Figure 10: An example of an informal proof.

5.1 Pasting command—proving equations

There are commands to prove equations between paths of diagrams by *pasting* the cells. As an example, assume we wish to prove $f = g$ from four assumptions $f = \mathrm{id}\,; f$, $g\,; \mathrm{id} = g$ and $h\,; f = \mathrm{id}$. An informal proof is shown in Figure 10. We show how to prove this using pasting commands. The proof goes on in a "goal-oriented" manner. There is always a goal formula (usually only one) and some assumptions. Our assumptions and goals are shown in Figure 11. The goal diagrams are pasted using some assumptions. In Figure 12, we explain how the pasting command works by example. We paste the diagram (1) in the assumption to the goal diagram. Edge f matches, so we paste it along edge f. Pasted cells are coloured; uncoloured cells become the new goal. This means that we can now prove $id; f = g$ instead of $f = g$. Similarly, the diagrams (2) and (3) are pasted. Finally, the goal cell is the same as in diagram (4) in the assumption. In this case, we can paste the whole diagram. The goal diagram is now completely coloured, so it has been proved. The pasting command is used to prove the diagram by pasting.

We now describe the exact condition under which the pasting commands can be applied. If the assumption diagram exactly matches the goal diagram,

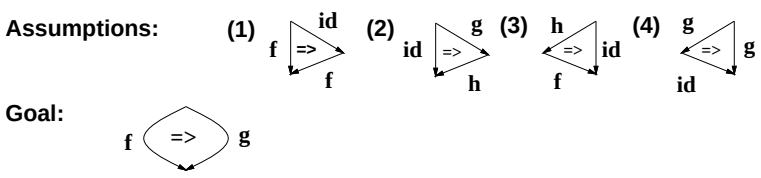


Figure 11: A representation using diagrams.

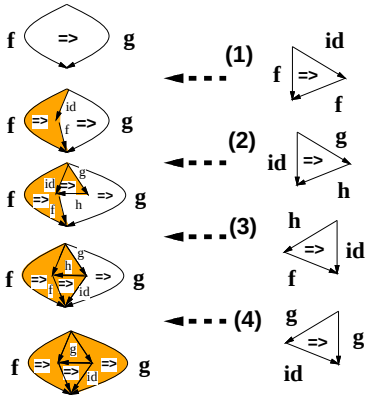


Figure 12: An example of pasting.

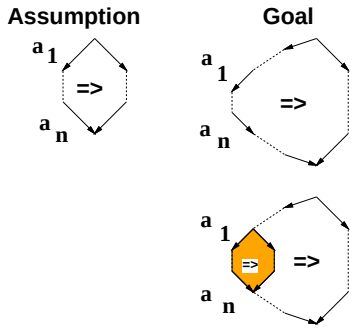


Figure 13: Condition of pasting.

the whole diagram is then pasted. In general, when the whole left path of the assumption diagram matches a part of the left path of the goal diagram, we can paste. In this case, the assumption diagram is pasted in the goal diagram like in Figure 13. The pasted part is coloured, and the uncoloured part becomes a new goal commutative cell which is surrounded by two paths. As in Figure 14, if a part of left path of diagram matches the goal, we can not paste it. Our example proof is written within the realm of proof trees shown in Figure 15. It is clear that there are many steps and that it is more cumbersome than the proof by pasting!

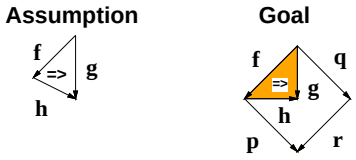


Figure 14: A non-example of pasting.

$$\dfrac{f = \mathrm{id}\,;f \qquad \dfrac{\mathrm{id} = g\,;h \qquad \dfrac{h\,;f = \mathrm{id}}{g\,;h\,;f = g\,;\mathrm{id}} \quad g\,;\mathrm{id} = g}{\mathrm{id}\,;f = g\,;h\,;f} \qquad g\,;h\,;f = g}{\mathrm{id}\,;f = g}}{f = g}$$

Figure 15: Proof tree.

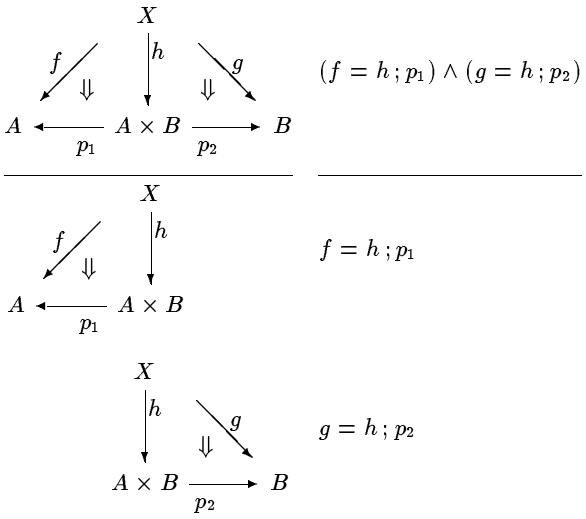


Figure 16: An example of and-decomposition command.

5.2 Proof commands for the logical connectives

There are many proof commands for the logical connectives, but we only explain the and-decomposition command.

We can apply the and-decomposition command when the goal formula has the form $A \wedge B$. To prove $A \wedge B$, we must prove both A and B. The and-decomposition command divides $A \wedge B$ into A and B. It also simultaneously divides the diagram (for example, Figure 16). The formula is divided to two formulae, and the diagram is divided into two. The divided diagram is a part of the original diagram.

6 Related works

Shin gave a formal system for Venn diagrams, VENN, whose primitive objects are diagrammatic, not linguistic [4]. He also gave a semantic analysis for VENN. Cooper argued for a particular treatment of generalised quantifiers [3]. Barwise and Etchemendy developed a visual logical system Tarski's World which treats first-order logic [1, 2]. Yamamoto et al. showed the formalisation of planar graphs which seems to be applicable to our system [5]. Kinoshita and Takahashi gave a theoretical background of pasting commands [6]. They used an algebraic structure arising from the pasting style proofs.

7 Conclusion

We introduced the functional specification of our interface using a commutative diagram for proving predicates in category theory. In the design process, we encountered the notion of relational models for user interfaces, making this worth of further investigation.

Acknowledgments

We thank Masami Hagiya, Mitsuhiro Yamamoto, and Shin-ya Nishizaki for their comments, especially letting us know of their work on formalisation of planar graphs. Satoru Tomura, and Hidemoto Nakada for useful comments, especially about implementation. We also thank Yutaka Sato for comments about our work in the early stages.

References

[1] J.Barwise and J. Etchemendy. Hyperproof: Logical Reasoning with Diagrams, *AAAI Spring Sympo. Series, Reasoning with Diagrammatic Representations*, pp.80–84(1992).

[2] J.Barwise and J. Etchemendy. The Language of First-Order Logic (including the Macintosh program, Tarski's World), *CLSI Lecture Notes*, No.23(1990). *AAAI Spring Sympo. Series, Reasoning with Diagrammatic Representations*, pp.80–84(1992).

[3] R. Cooper. Generalized Quantifiers and Resource Situations, *Situation Theory and its Applications Vol.3*, *CLSI Lecture Notes* No.37, pp.191–211(1993).

[4] S. Shin. A Situation-Theoretic Account of Valid Reasoning with Venn Diagrams, *Situation Theory and its Applications Vol.2*, *CLSI Lecture Notes* No.26 pp.581–605(1991).

[5] M. Yamamoto, S. Nishizaki, M. Hagiya, and Y. Toda. Formalization of Planar Graphs, *8th International Workshop on Higher-Order Logic Theorem Proving System and Its Applications*, Volume 971 of *LNCS*, pp.369–384(1995).

[6] Yoshiki Kinoshita, Koichi Takahashi. Gropoid of equational proofs, ETL Technical Report, TR95-17 (1995).

[7] J. Tanaka. Personal communication, December 1995.