# Hyperproof: Logical Reasoning with Diagrams

Jon Barwise and John Etchemendy

## Brief overview

*Hyperproof* is a Macintosh program designed to help students learn how to reason logically, using either sentences of first-order logic, or diagrams, or both.[1] An illustration of a typical *Hyperproof* screen is shown in Figure 1.

*Hyperproof* is a descendant of our earlier courseware *Tarski's World* [2]. *Tarski's World* uses diagrams to help teach the language of first-order logic, and has been very successful. *Hyperproof* extends the ideas embodied in *Tarski's World* into the realm of reasoning. It is intended to be part of a course aimed at teaching students to solve analytical reasoning puzzles and logic problems by whatever means of representation are most appropriate to the problem at hand. On our view, an important part of solving such a problem is figuring out what representational scheme to use, whether sentences, diagrams of some sort, a combination of both, or something else altogether.

Before we say more about what *Hyperproof* is, let's say a bit about what it *isn't*, just to forestall any misconceptions. *Hyperproof* is not an automatic theorem prover. Nor is it an AI system. It is rather a computational environment which provides a setting:

1. for teachers to pose interesting inference problems using pictures and first-order sentences,
2. for the student to give mathematically rigorous proofs, again using some combination of pictures and first-order sentences, and
3. for checking these proofs to make sure that they are correct and fulfill the original goal set by the problem.

While *Hyperproof* is not an automated reasoner or an AI system, we believe that it is of some interest to these fields. First, proofs in *Hyperproof* are typically an order of magnitude shorter and more natural than corresponding proofs in first-order logic which eschew diagrams. Understanding why this is so can only be of help in automating reasoning.

Second, for certain kinds of problems, the algorithms used for checking diagrammatic proofs can be turned on their head to discover one-step solutions that people cannot even follow, must less find. This suggests that these algorithms, or ones like them, could be put to good use in automating certain reasoning tasks.

Finally, *Hyperproof* has a much more liberal notion of what counts as an inference task than does traditional logic. For example, in this system, one is able to prove that a purported conclusion does not follow from the premises, by constructing a diagram which depicts a situation in which the information in the premises obtains but the conclusion fails. Thus we can pose questions neutrally: does the conclusion follow from the premises or not? One and the same set of techniques and tools can be used for providing the solution, whether it is positive or negative. We think that much ordinary reasoning is like this: visualizing a situation where the premises hold but some undesirable possible consequence fails, and then working toward realizing that situation.

## Historical introduction

Over the past twenty years, some workers in AI have explored uses of diagrams in automating reasoning. By contrast, logicians have had little truck with diagrams and other forms of non-linguistic representation.[2] The traditional attitude is all too well captured by the following quotation:

> [The diagram] is only an heuristic to prompt certain trains of inference; ... it is dispensable as a proof-theoretic device; indeed, ... it has no proper place in the proof as such. For the proof is a syntactic object consisting only of sentences arranged in a finite and inspectable array. (Neil Tennant, in [6])

[1] While the ideas for *Hyperproof* are ours, the program itself is the creation of Gerard Allwein, Mark Greaves, and Mike Lenz with help from Alan Bush, Doug Felt, and Chris Phoenix. We express our appreciation of and admiration for their work here. The term "Hyperproof" is suggested by the notion of hypertext, where information is presented in a host of non-linear, often non-textual means.

[2] There have been a few notable exceptions, of course, like Venn, Euler, and Peirce.

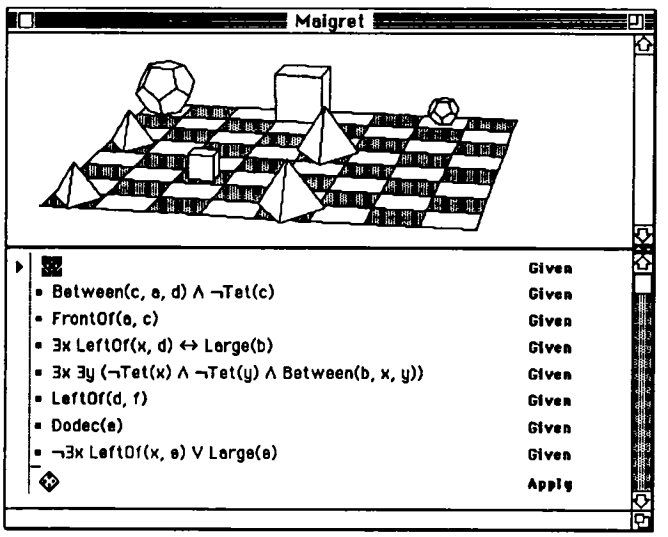


Figure 1: Problem: Which objects are named a, b, ...?

One aim of our work, as explained in [1], is to demonstrate that this dogma is quite misguided. We believe that many of the problems people have putting their knowledge of logic to work, whether in machines or in their own lives, stems from the logocentricity which has pervaded its study for the past hundred years.

We approach logic from an informational perspective. Wherever there is structure, there is information. But in order for agents (animals, people, or computers) to traffic in information, the information must, in some way or other, be presented to or represented by the agent. Typically, a given representation, or family of representations, will represent certain information explicitly, while other information will be implicit in the information explicitly represented. Inference, as we understand the term, is the task of extracting information implicit in some explicitly presented information.

This informational perspective is part of the modern, semantic approach to logic associated with names like Gödel, Tarski, Robinson, and the like. On this view, a purported rule of inference is valid or not depending on whether it in fact guarantees that the information represented by the conclusion is implicit in the information represented by the premises. But when one takes this informational perspective seriously, the logician's disdain for nontextual representations seems like an oversight, a case of dogma that desperately needs reexamination. The most casual survey of the ways people actually represent information shows an enormous variety of representational devices that go beyond simple text.

In carrying out a reasoning task, part of the solution lies in figuring out how to represent the problem. In problem solving, well begun really is half done. Indeed, this is probably an understatement. Figuring out how to represent the information at hand is often the most important part of the solution. What we seek to teach our students, then, is to use the most appropriate form of representation for the reasoning task at hand. As long as the purported proof really does clearly demonstrate that the information represented by the conclusion is implicit in the information represented by the premises, the purported proof is valid.

Why are logicians so suspicious of diagrams and other forms of non-textual representation? The answer goes back to the tradition in geometry, where diagrams were viewed with suspicion by the ancients. Certain mistaken proofs were seen to result from being led astray by a misleading diagram that accompanied it. So, the tradition went, the diagram should, in theory, be eliminable from the proof. The textual part of the proof should stand on its own two feet.

This argument is itself a nonsequitur. If we threw out every form of reasoning that could be misapplied by the careless, we would have very little left. Mathematical induction, for example, would go. No, the correct response is not to throw out methods of proof that have been misapplied in the past, but rather to give a careful analysis of such methods with the aim of understanding exactly when they are valid and when they are not.

A nice case study along these lines has been carried out by Sun-Joo Shin in [3] and reported in [4]. In introductory logic, many people teach the method of Venn diagrams. But often they also tell their students that Venn diagrams are only a heuristic aid to giving proofs. A "real" proof has to be given in first-order logic. Shin shows that this is a mistake. She gives a

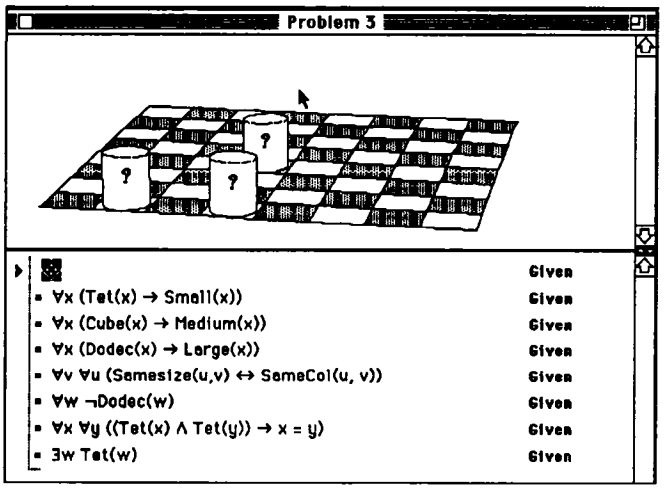


Figure 2: Another *Hyperproof* Problem

rigorous analysis of Venn diagrams as a formal system, with its own syntax, semantics, and notion of logical consequence. She shows that the usual rules of inference are sound with respect to this notion of logical consequence. Furthermore, she shows that, with some trivial additional rules that had been overlooked, one can give a completeness proof for the deductive system.

An even earlier study is given by John Sowa in [5]. In the early chapters of that book, Sowa gives a careful formal treatment of Peirce's existential graphs, including their syntax, semantics, and proof theory. This is a formalism that is expressively equivalent to first-order logic, but it is given in purely diagrammatic terms.

The importance of these results is this. They show that there is no principled distinction between inference formalisms that use text and those that use diagrams. One can have rigorous, logically sound (and complete) formal systems based on diagrams. And while Venn diagrams may seem very simple, they are probably used more often in real life than first-order formalisms.

## *Hyperproof*

*Hyperproof* is a working program that we are now using in teaching elementary logic courses. We will demonstrate the program at the meeting. Here we would like to draw attention to one very important part of *Hyperproof*.

One of the differences between diagrams and text is suggested by the old Chinese saying "A picture is worth ten thousand words." The truth behind this is that diagrams and pictures are extremely good a presenting a wealth of specific, conjunctive information. It is much harder to use them to present indefinite information, negative information, or disjunctive information. For these, sentences are often better. That is why we favor a heterogeneous system, one that allows us to use both.

*Hyperproof* has all the usual rules of first-order logic, plus a number of rules for dealing with diagrams and the interplay between diagrams and sentences. In this short abstract, we discuss two of these rules, the rules of **Observe** and **Cases Exhausted** (and a special case of the latter, **Apply**). The rule **Observe** allows us to extract sentential information from diagrammatic, while the latter allows us to go the other way around.

In *Hyperproof* every diagram represents partial information about some class of blocks worlds, the kind of partial information that can be modelled by a partial first-order structure. **Observe** allows the user to infer any sentence which is definitely true in the partial structure that corresponds to the diagram—that is, true according to the truth evaluation schema embodied in the (weak) Kleene three-valued logic. The three values are *true*, *false*, and *unknown*.

The truth behind the old cliche referred to earlier is evident from the uses of this rule. For any given diagram there are literally an infinite number of sentences that can be observed to hold on the basis of the diagram. And typically, if one wanted to completely specify a given diagram with sentences, it would take an enormous number. Indeed, it is not always possible to do it at all, since there are elementary equivalent diagrams (i.e., diagrams satisfying the same first-order sentences) that represent distinct possibilities.

But the most important features of the *Hyperproof* system are the techniques for transfering information from sentences into a range of diagrams. Thus, we are typically give a diagram $D$ and some sentences $S_1, \ldots, S_n$ describing the target world. Based on one of these sentences, say $S$, we see that there is a range of possibilities, each of which can be embodied by fleshing out the diagram $D$ in some way. Thus we create some new diagrams $D_1, \ldots, D_k$, each an amplification

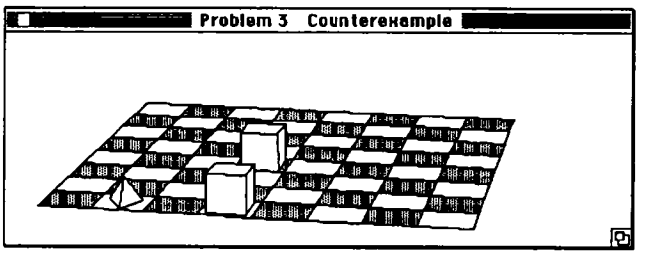


Figure 3: *Hyperproof*'s solution

of $D$ (in that it carries more information than $D$), and say that these exhaust all the possibilities, given the sentence $S$. This step is valid if any way the world could be, consistent with $D$ and $S$, is also consistent with at least one of these new diagrams. This is the rule of **Cases Exhaustive**.

A special instance of **Cases Exhaustive** is where $k = 1$, that is, where there is just one new diagram. Applying the rule in this case means that the information added to $D$ in the new diagram $D_1$ is implicit in the sentence $S$. We call this special instance **Apply**, since you can think of it as applying the information in $S$ to add new information to the diagram.

Rather than describe the general **Cases Exhaustive** algorithm, let us discuss the special case of **Apply**. The interested reader can readily think through the extension to the more general rule. So we imagine ourselves in the following situation. The user has a diagram $D$ and a sentence $S$. She wants to apply $S$ to amplify the diagram to $D'$.

Ideally, what one would like would be an algorithm which did the following: If this **Apply** is legitimate, then say so. If not, then produce a counterexample, by displaying a diagram $D^*$ that amplifies $D$ in a way where $S$ is true but which is incompatible with $D'$.

This specification is not in general realizable in a computationally feasible manner. The reason is that it can happen that such a diagram would have to settle too many issues, issues which themselves can be resolved in a variety of ways. Thus one gets into a huge search space.

In *Hyperproof* we solve this problem in two ways, relying on partial information. First, we only search for counterexamples among those diagrams that settle just issues raised in $D'$.[3] If we can find a such a diagram in which $S$ is true, then we will have found a counterexample. But we do not insist that the sentence $S$ come out as true in $D^*$, only that it *not* come out as false. If it comes out as neither true nor false in a diagram $D^*$ which is incompatible with $D'$ and which settles all issues raised in $D'$, we call this a possible counterexample, and give it to the user as a possibility that needs to be considered explicitly. In this way, the search space is greatly reduced and the rule is sound, in that it never ratifies an invalid use of **Apply**. The algorithm is not complete, in that it will not be able to recognize some valid uses of **Apply**, but in such cases it returns a possible counterexample for the user to consider.

This rule, and the more general version of **Cases Exhaustive**, give proofs in *Hyperproof* quite a different character than we expected. It is very difficult to find problems where the best solution relies on the standard sentential rules of inference. It is typically much more efficient to break into several diagrammable cases and then work on these diagrams.

## The role of the computer

As we said earlier, *Hyperproof* is not an AI program. It is, rather, a platform for teaching students some basic principles of valid reasoning. The role of the computer is two-fold:

- It is an environment for creating proofs: it provides a convenient environment for generating visually perspicuous representations of the situations the students are asked to reason about.
- It is a proof checker: it checks the students work to see if each step is correct, and if the proof as a whole satisfies the goal of the problem.

This said, though, it must be said that the program does have some smarts. It is able to perform some reasoning tasks that outstrip our original intentions for it. While we do not want to make any AI claims for our program, we would be gratified if the features that make it smarter than we intended were of use in AI.

Most noticeable of these are the rules of **Apply** and **Cases Exhaustive**. By repeated uses of these rules, one can basically have the computer figure out the cases for a case analysis. All one does is start it off

[3] For example, if $D'$ settles the size of a single block $b$, and that is all, then the only issue we consider is the size of that block. All other amplifications of $D$ are ignored.

with one case to activate the issues you want considered in your case analysis.

Using this trick, the user can prod the program to perform feats of inference that are very difficult for the user herself to follow. For example, it may produce a set of exhaustive cases which she cannot readily see to be exhaustive. For our pedagogical purposes, this poses a bit of a problem, since we want to teach the student that proofs are composed out of self-evident steps. But if you think of using *Hyperproof*-like technology for automated or computer-aided inference in other settings, this feature could be a big win.

We close with a simple example of this phenomenon. Figure 2 shows a sample problem given in *Hyperproof*. The goal of the problem is to determine the sizes and shapes of the objects depicted. (In *Hyperproof*, we assume that there are only three sizes of objects, small, medium and large, and three shapes, cubes, tetrahedra, and dodecahedra, so the sizes and shapes of the objects depicted are, in fact, all determined by the sentences in the given.) This is not an entirely trivial chore. However, using a trick, and *Hyperproof*, she can make short work of it. If she makes a random guess at the sizes and shapes, and asks the program to verify that her guess follows from the given information, *Hyperproof* quickly comes up with the counterexample shown in Figure 3. This counterexample is, of course, the correct answer to the problem.[4]

This is a fairly simple example, but the program can solve much more complicated problems in the same way. For example, try to figure out which objects depicted in Figure 1 have which of the names used in the sentences displayed. The trick just described makes short work of it.

## References


1. Jon Barwise and John Etchemendy, "Valid inference and visual representation," in *Visualization in Mathematics*, ed by Zimmerman and Cunningham, Mathematical Association of America, 1990
2. Jon Barwise and John Etchemendy, *Tarski's World 3.0*, CSLI Lecture Notes, University of Chicago Press 1991
3. Sun-Joo Shin, *Vaild Reasoning and Visual Representation*, Dissertation, Stanford University, 1991
4. Sun-Joo Shin, "An Information-Theoretic Analysis of Valid Reasoning with Venn Diagrams," in *Situation theory and its applications, Part 2*, ed by Barwise et. al., CSLI Lecture Notes, University of Chicago Press, 1991
5. John Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, 1984
6. Neil Tennant, "The Withering Away of Formal Semantics," *Mind and Language*, Vol. 1, No. 4, 1986, pp. 302–318.


[4] Interestingly, it takes a *lot* longer for the program to determine that the counterexample is the only solution than that it is a solution.


Jon Barwise
Department of Computer Science
Indiana University
Bloomington, IN 47405

E-mail: Barwise@cs.indiana.edu

John Etchemendy
Center for the Study of Language and Information
Ventura Hall
Stanford University
Stanford, CA 94305

E-mail: Etch@csli.stanford.edu