THE TRANSLATION OF FORMAL PROOFS INTO ENGLISH

by

Daniel Chester

Technical Report NL - 26

The Department of Computer Sciences
University of Texas at Austin

June 1975

Abstract

The argument of a logical proof written up in English can be represented by a formal proof in symbolic logic; this formal proof can also be taken to represent the deep structure of the written proof. This paper describes a program which generates a written proof from such a representation.

The Translation of Formal Proofs into English

Daniel Chester
University of Texas, Austin, Texas

Introduction

Collectively the proofs of theorems form one of the few categories of discourse for which there is a well-developed semantic theory. Formal representations for the structure of discourse in categories such as narration, exposition, procedure, exhortation, etc. are being attempted by Charniak [1], Longacre [2], Phillips [3], Schank [4], and others. But attempts to formalize the structure of logical argument go back as far as Aristotle [5,6] and have successfully culminated in the many formal proof systems which are now part of symbolic logic. With such formal systems available for representing the deep structure of written proofs, proof discourse is an excellent place to begin a study of discourse structure. In this paper we shall look at the relationship between formal proofs and written proofs, and at a program called EXPOUND which translates formal proofs into English.

Proofs

Formal proofs are defined in terms of formulas and inference rules. Typically a formal proof is defined to be a finite sequence of well-formed formulas such that every member of the sequence is an axiom or is inferred from earlier members by means of an inference rule. Defined this way, formal proofs are useful for studies in logic, but they do not closely resemble written proofs. Formal proofs show every step, while written proofs leave many out. Nevertheless, for every written proof a formal proof can be constructed which represents its logical content. (If this can't be done the proof isn't valid.) Such a formal proof is thus a likely representation for the written proof's deep structure.

Another principle difference between many formal and written proofs is that written proofs use the rule of conditionalization. They begin by making an assumption, then argue to a conclusion, and finally they assert the fact that the assumption implies the conclusion. In print the proof looks like

> Suppose P. . . . Therefore Q. Hence we have shown that P implies Q.

Formal deduction systems which do use the CD (conditionalization) rule are known as systems of natural deduction. See Fitch [7] and Quine [8] for examples.

Written proofs differ from typical formal proofs also because they often indicate what the logical relationships are that hold between statements to make the proof valid; formal proofs often do not have any such indication. Suppose that we want to show that the dog Toto is a lovable pet, and we already accept as facts the statements

> All pets are lovable or exotic.
> Toto is a pet.
> Toto is not exotic.

We might give a formal proof (using sentences in place of formulas) like this:

> All pets are lovable or exotic.
> If Toto is a pet then he is lovable or exotic.
> Toto is a pet.
> Toto is lovable or exotic.
> Toto is not exotic.
> Toto is lovable.
> Toto is a lovable pet.

This sequence, a mere listing of the individual steps in the proof, is not an acceptable essay; it reads more like a haphazerd list of facts. A better way to present this proof is

> Toto is a pet, and all pets are lovable or exotic, so Toto is too. But Toto is not exotic; consequently he is a lovable pet.

Note that the only lines of the original proof that are actually asserted are the statements which we already accept as facts. All the other assertions in this short essay are about how the lines of the proof are related. The last clause, for example, does not assert that Toto is a lovable pet, it says that this fact is implied by the statements mentioned earlier in the essay. A formal proof that is used to represent a written proof's deep structure should therefore belong to a natural system of deduction, and it should also explicitly show how the statements are deduced from their predecessors.

EXPOUND

The program EXPOUND, written in UT LISP, translates a formal proof into an English statement of a theorem and its proof. It performs this translation in four stages. In the first stage it makes a graph representing the inferential relationships between the lines of the proof. In the next two stages it uses this graph to make an outline of the text which it will generate. The program makes this outline by first grouping lines together into paragraphs, then putting these paragraphs in linear order and inserting introductory paragraphs where appropriate to explain how the other paragraphs are related. Finally it generates an English text by explaining how each line is obtained from the preceding lines in the outline.

The formal proof that EXPOUND expects is a sequence of triples such that each triple consists of a line identifier, a first-order predicate calculus formula and a list of atoms. We shall call the formula a <u>line</u> of the proof and we shall use the line identifier to refer to it. The list of atoms we shall call the <u>justification</u> for the line. This list consists of the name of the inference rule that was used to obtain the line, followed by the identifiers for the lines that the rule was applied to. If four triples in a proof are

L6	Fx	(PR)
L7	$\forall x\ (Fx \rightarrow Gx)$	(KN)
L8	$Fx \rightarrow Gx$	(UI L7)
L9	Gx	(TF L6 L8)

then the justification for line L6 shows that it is the result of the PR (premiss) rule, which states that any formula may be adopted as a premiss at any step in the proof. The justification for line L8 shows that it is a universal instantiation of line L7, a known fact, while the justification of L9 shows that it follows truth-functionally from lines L6 and L8. We shall call a line listed in the justification for a line a reason for the line, and we shall call the rule listed in the justification the rule for the line. Thus the rule for line L9 above is the TF rule and the reasons for L9 are L6 and L8. See Table 1 for a list of all the inference rules, their full names and an illustration of each; except for the KN rule, these rules are identical to those in [8]. An example of a proof is shown in Fig. 1. In the following sections we shall follow this example as EXPOUND translates it into English.

Graph

The lines of a proof have a natural order to them, a partial order > which we can define by the following four conditions:

1. If line x is a reason for line y then x > y.

2. If line z is obtained by the CD rule using premiss x, line y is obtained by the CD rule using premiss w, x > y, and y > z, then x > w.

3. If line x is obtained by the EI rule, v is the variable having an unquantified occurrence in x but not in the reason for x, and y is another line which has an unquantified occurrence of v, then x > y.

4. If $x > y$ and $y > z$ then $x > z$.

These conditions describe basic facts about how the lines of a proof are usually presented. The first condition states that the reasons for a line are given before the conclusion drawn from them. The second asserts that when a proof is begun from some premiss x and a subproof cannot be completed without using a successor to x (a line y such that $x > y$) then the entire subproof is nested within the proof. This situation is illustrated by the five line proof in Fig. 2a. Lines w, x, y, and z are related as in the hypothesis of condition 2. If > were defined without condition 2, the partial ordering of Fig. 2a would look like Fig. 2b. With condition 2 the partial ordering is the natural one shown in Fig. 2c. In these graphs and in Figures 3 and 4 a node n is greater than a node n' ($n > n'$) if there is a downward path from n to n'. The third condition states that when a term c is introduced by an EI rule (informally this is the case when we say "Now there are things that . . .. Let c be such a thing.") then that introduction precedes all other references to c. The fourth condition is the transitive property for partial orderings.

Internally EXPOUND represents the partial ordering by means of a graph; in the case of the proof in Fig. 1, it produces the graph in Fig. 3. The edges (lines between nodes) of this graph show only the relationships of form x > z which are irreducible, that is, there is no node y such that x > y and y > z.

Most of the edges in this graph connect lines to their reasons. Line L13, for example, is connected to L5, L11, and L12 because it is inferred from these lines. It is also connected to L14 because it is a reason for L14. Line L14 is deduced by the CD rule from L13 and the premiss L10; but there is no edge from L10 to L14 because the relationship L10 > L14 can be determined by the transitive rule from the edges that are in the graph.

Three of the edges, however, are the result of condition 2 in the definition of the partial ordering. There is an edge indicating L4 > L6 because L6 is a premiss, it begins the subproof L6, L7, L8, L9, which ends with CD line L9, and L4 > L9 because of the chain L4, L5, L8, L9. Similarly there are edges indicating L3 > L4 and L3 > L10 because both L4 and L10 begin subproofs which are embedded in the subproof beginning with L3.

Outline

The next thing that EXPOUND does is to combine the nodes of the graph into sequences of lines which outline the paragraphs that will appear in the output. These sequences are the nodes of a new graph as shown in Fig. 4. The program combines the nodes by repeated application of three rules:

1. Add the lines at node x to those at node y if $x > y$, x is the only immediate predecessor to y and y is the only immediate successor to x.

2. Add the lines at node x to those at node y if y is an immediate successor to x and x is an eligible node of lowest rank. Node x is eligible iff it has y as its only immediate successor, the number of lines at x is less than some fixed number (5 in our examples), none of the lines are CD lines, and there are no immediate predecessors to y which should be added first to insure that subproofs will be unbroken sequences. The rank of a node is the number of its immediate predecessors.

3. When both rules 1 and 2 are applicable, apply rule 1 first, repeatedly if possible.

These rules assure us that the lines that go into one paragraph of the final text form an almost totally ordered sequence. The only deviations from total ordering occur just before lines derived from a set of lemmas with each lemma preceded by a proof having fewer than five lines and containing no CD lines. (The CD lines terminate subproofs which should be in seperate paragraphs, as is done in Fig. 4.)

At this stage the nodes, which correspond to paragraphs, may not yet be linearly ordered, so EXPOUND lists the nodes in an order compatible with the partial ordering (if x > y then x precedes y) and inserts introductory paragraph nodes to clarify the relationships between the other nodes. The program examines the total proof and each subproof seperately to determine where such nodes are needed. An introductory paragraph says nothing about the internal structure of nested subproofs; that explanation is left to the introductory paragraph within that subproof. An introductory paragraph node consists of a conclusion and a list of lemmas. At the top level the conclusion is the last line of the last node, i.e., the last line of the proof. In a subproof the conclusion is the last line before the CD line terminating it. The lemmas are lines which do not belong to subproofs of the current (sub)proof being examined and which are the last lines of nodes which have more than one immediate successor (except the first such node in a subproof) or whose immediate successor has more than one immediate predecessor (within the current subproof being examined), or whose immediate successor is part of a subproof at a deeper level. When the program creates an introductory paragraph node it inserts it either at the beginning of the entire proof, or, in a subproof, after the first node having more than one immediate successor. In the case of Fig. 4, EXPOUND inserts an introductory paragraph node consisting of conclusion L15 and lemmas L9 and L14 right after node w, making the final outline w, introductory paragraph, x, y, z.

The observant reader will note that lines L1 and L2 do not appear in Fig. 4. This is because they are the premisses which are still assumed when the last line L19 is deduced. They will therefore be mentioned in the statement of the theorem and do not need to be repeated in the body of the proof. For this reason EXPOUND removes L1 and L2 from the graph before outlining the proof and saves them until the proof is ready to be printed. EXPOUND then precedes the proof with a statement of the theorem, listing L1 and L2 as the hypotheses and L19 as the conclusion.

Paragraphs

The program now has a detailed outline of the English text which it generates. This outline is a sequence of nodes which are of two types; regular, consisting of a sequence of lines from which the program generates a regular paragraph, and introductory, consisting of a conclusion line and some lemmas from which the program generates an introductory paragraph. In both kinds of paragraph EXPOUND makes statements about how the lines of the proof are related, i.e., this line is a tautological consequence of that one, to prove this line we must first prove such-and-such, etc. Also EXPOUND does not make a statement for every line in a node; it ignores some lines because the inferences deriving them are trivial and easily reconstructed by the reader.

For a regular paragraph EXPOUND examines each line and generates a statement based on the rule by which the line was deduced and on the previous statement generated. For instance, if a line x is a generalization of the previous line (by the UG or EG rule) the program generates "thus" (or a randomly chosen synonym) followed by a sentence asserting x. If x is an instantiation the situation is different. If x is a UI line it is ignored; it won't get mentioned until it is used to infer some other line. If x is an EI line the program generates a statement having a form like "Let Z denote such a . . . ." If line x is a TF consequence of a number of lines y_1, y_2, . . ., y_k, the program chooses randomly from several possible statement formats, with the range of choice depending on the circumstances. If x is deduced from $x \wedge y$ and this is the previous line then the program ignores it and procedes to the next line. If none of the y_i's is the previous line then EXPOUND generates a statement like "since y_1, and y_2, and . . ., and y_k, x" or "y_1, and y_2, and . . ., and y_k, so x" or "x because y_1, and y_2, and . . ., and y_k." The line x is replaced by a sentence asserting x and each y_i is replaced by a sentence asserting y_i, often preceded by an expression like "by hypothesis", "by assumption", or "we have shown that". The program also prefixes to the whole statement a synonym of "furthermore" if the previous line is also a TF line, or "now" or "but" if there is no relationship between x and the previous line. On the other hand, if x is related to the previous line, that is, if the previous line is one of the y_i's, say y_1, then EXPOUND generates a statement like "but y_2, and y_3, and . . ., and y_k, so x" or "this and the fact that y_2, and y_3, and . . ., and y_k imply that x" or "thus since y_2, and y_3, and . . ., and y_k, x". The lines x, y_2, . . ., y_k are replaced by sentences just as in the other case, and words like "thus", "since", and "so" may be replaced by synonyms. In every case the program generates sentences from the lines and connects them together using patterns determined by the justifications.

For an introductory paragraph EXPOUND uses one of several formats. If the conclusion is x and the lemmas are y_1, y_2, . . ., y_k, where $k > 1$, it usually generates a paragraph like

> We want to show that x. This is implied by the following statements. y_1. y_2. . . . y_k. These statements we shall now prove.

If all the y_i's have the form $p_i \rightarrow x$, however, it generates

> We want to show that x . This we shall do by considering the following k cases.

For other special cases EXPOUND generates similar paragraphs.

Sentences

We have now explained all the steps by which EXPOUND generates a text from a proof like Fig. 1 except how it generates a sentence to assert a formula. The grammar on which the sentence-building procedure is based is a simple case grammar. Each predicate has associated with it a verb string, a syntactic type, and the preposition, if any, associated with each argument. Sometimes a gender is also associated with the predicate so that EXPOUND can choose an appropriate pronoun when needed. This lexical information is supplied to EXPOUND in the form of a table such as the one illustrated by Table 2, which is the one that accompanies the proof in Fig. 1 as EXPOUND's input. The verb string is listed in up to four forms for convenience; these represent the positive active, negative active, positive passive, and negative passive forms. The syntactic type is either adjective, noun, phrase, or clause, and indicates which kind of word phrase to construct from the verb string when building a noun phrase description for some variable. The prepositions indicate the case for each argument. Using this information about how to translate the predicates, EXPOUND generates a sentence by examining the syntactic structure of the underlying formula and using that structure to guide its decisions about which subformulas generate noun phrases, which generate verb phrases, and how these phrases are connected together.

The way that EXPOUND combines sentences from subformulas is about what one expects when the formula is a conjunction, disjunction or implication: it translates $P \wedge Q$ into "P and Q", $P \vee Q$ into "either P or Q", and $P \rightarrow Q$ into either "if P then Q" or "Q whenever P". But it acts on quantified and atomic formulas in more complicated fashion.

Universally quantified formulas usually have the form $\forall x\ (P \rightarrow Q)$ with variable x occurring in both subformulas P and Q. After recording the fact that x is universally quantified, EXPOUND attempts to build from P a noun phrase describing x. If successful it then generates a sentence from Q; if not it generates a sentence from $P' \rightarrow Q$, where P' is what is left of P after the phrase-building process has terminated.

A noun phrase describing x is built from a formula P if P is either an atomic formula (a predicate with its arguments), a formula beginning with quantifiers, the negation of either an atomic formula or quantified formula, or the conjunction of the above kinds of formulas. The conjuncts are grouped together according to their syntactic types; atomic formulas (with one exception) are the same type as their predicate, while quantified formulas are either phrase or clause types (the choice depends on the last atomic formula occurring in them) and negated formulas or atomic formulas with a predicate of noun type and two or more arguments are clause types. Word strings are made from each formula and then strung together with adjectives coming first, then nouns, then (prepositional) phrases, and finally (dependent) clauses. Adjective and noun strings are generated by removing the first word (usually "is") or the first two words ("is" and a determiner) from the positive active form of a predicate's verb string. Phrases and clauses are generated by making sentences from formulas and then replacing the subjects by "who" or "which" (to make clauses) or by just deleting the subjects and their verbs (to make phrases). If there is more than one phrase or more than one clause, then "and" is inserted between them. The noun phrase describing x the first time it must be mentioned consists of the strung-together word strings preceded by a quantifier, either "every" or "some" as appropriate. Subsequent occurrences of x are indicated by a pronoun or "the N", where N is the last word on the list of nouns in the noun phrase. If that last word is the same as the last noun for some previous variable so that "the N" is potentially ambiguous, the noun phrase describing x is followed by ", say Z," where Z is an arbitrary symbol, and "Z" is used instead of "the N" for subsequent occurrences of x.

The sentence that EXPOUND generates from an atomic formula is simply the first argument followed by the predicate's verb string followed by prepositional phrases made from the remaining arguments. For example, if predicate G had three arguments x, y, and z, and its verb string were "gives", then z would have the preposition "to" associated with it. The formula Gx,y,z would then generate a sentence of the form "x gives y to z". The variables would of course be replaced by suitable noun phrases built from other formulas. The first occurrences of quantified variables in the argument list would be rearranged so that they appear in the same order in which they were quantified. If the first variable were moved from its initial position in the argument list then the passive form of the verb string, "is given", would be used in place of "gives". The preposition used with a variable that has been moved is the one corresponding to its original position in the argument list; e.g., if variables x, y, and z were quantified in the order y, z, x, the sentence generated from Gx,y,z would have the form "y is given to z by x".

Suppose the underlying formula is $\forall x\ ((Fx \wedge Gx) \rightarrow Hx)$ and the predicates are to be translated as shown in Table 2. Because the formula is the universal quantification of an implication, EXPOUND first builds a noun phrase to describe the variable x using the predicates in the antecedent formula $Fx \wedge Gx$. After noting that F and G are noun and clause syntactic types respectively, EXPOUND combines their verb strings to make the expression "worker who signed the contract". It then makes a sentence from the consequent Hx , giving rise to the output "every worker who signed the contract is in the union." If the underlying formula were $\forall x\ ((Fx \wedge P) \rightarrow Hx)$ and P were a formula that EXPOUND couldn't use to make a noun phrase, it would generate a sentence from the formula $P \rightarrow Hx$; this would produce an output like "if any worker . . . then he is in the union." (The quantifier "every" is changed to "any" when it is printed in the antecedent of such a conditional.)

The existential quantification of a conjunction is translated in similar fashion. All conjuncts but the last are used to make a noun phrase and a sentence is made from the last conjunct. The formula $\exists x (Fx \wedge Gx \wedge Hx)$, for example, would be translated "some worker who signed the contract is in the union." For some formulas the desired sentence may be generated from several conjuncts besides the last if they can't be used in building the noun phrase for the quantified variable. If no noun phrase is generated for the variable, EXPOUND makes one from the verb string for the predicate UNIVERSE. This means that the program would translate $\exists x Fx$ into "some person is a worker."

When a formula is a negation EXPOUND notes that fact and then procedes to make a sentence from the subformula which is negated. If the negation symbol precedes a quantifier, it affects the quantifying word in noun phrases. The formula $\neg \forall x (Fx \rightarrow Gx)$, for instance, is translated into "not every worker signed the contract." The formula

$\neg \exists x\, Fx$ becomes "no person is a worker." When an atomic formula is negated, the negative active (or negative passive) verb string is used. Thus $\exists x\, (Fx \wedge \neg Gx)$ is translated "some worker did not sign the contract."

Output

Altogether EXPOUND takes just seven seconds of computation on a CDC 6600 to accept Fig. 1 and Table 2 as input and perform all the operations discussed. During this time it makes a graph from the proof, condenses the graph down to a linear sequence of paragraph outlines, and then generates the statement of the theorem and its proof shown below.

Theorem:

Suppose that if every worker who signed the contract is in the union then some worker did not sign the contract. Suppose moreover that either every worker signed the contract, or every worker is in the union. Then if every worker in the union signed the contract then some worker who signed the contract is not in the union.

Proof:

Suppose that every worker in the union signed the contract. Suppose moreover that some worker did not sign the contract. Let w denote such a worker who did not sign the contract.

We want to show that a contradiction follows. This we shall do by considering the following 2 cases.

Suppose that every worker signed the contract. Now a contradiction follows, since by assumption w is a worker and he did not sign the contract, and if he is a worker then he signed the contract. Thus if every worker signed the contract then a contradiction follows.

Suppose that every worker is in the union. But a contradiction follows, as by assumption w is a worker and he did not sign the contract, and if he is a worker then he is in the union, and if he is a worker and he is in the union then he signed the contract. Thus if every worker is in the union then a contradiction follows.

Because by hypothesis either every worker signed the contract, or every worker is in the union, and we have shown that if every worker signed the contract then a contradiction follows, and if every worker is in the union then a contradiction follows, a contradiction follows. Thus if some worker did not sign the contract then a contradiction follows. This and the fact that by hypothesis if every worker who signed the contract is in the union then some worker did not sign the contract imply that not every worker who signed the contract is in the union. In other words some worker who signed the contract is not in the union. Therefore if every worker in the union signed the contract then some worker who signed the contract is not in the union.

Two more examples of proofs and their translations are shown in the Appendix.

Conclusion

We have described a first approximation to the process by which people generate logical discourse from formal proofs. There are three principal ideas incorporated in this approximation. The lines of the proof are presented in a linear order where possible

so that each is deduced from the previously mentioned line, but where this is not possible, the reader is given signposts to warn him. These signposts consist mainly of the indentations showing the beginnings of paragraphs, introductory paragraphs which indicate the relationships between paragraphs, and, within paragraphs, by words like "now" and "furthermore". Also some lines get omitted or replaced by special constructions (as in the case of EI lines and contradictions) when this leads to greater efficiency in communication. Finally, the primary function of the text is to explain how the lines of the proof are deduced, i.e. to show the structure of the proof. This structural information is found mostly in the connective phrases like "thus", "by assumption", and "suppose", which are used to build complex sentences from the simple ones.

It is hoped that EXPOUND will evolve into a more sophisticated text generator as more complicated sentence constructions are incorporated into it. With longer proofs it will omit larger portions so that its output more closely resembles the writings of mathematicians. Perhaps it can be adapted to other forms of discourse involving natural orderings, like narrative or procedural discourse, based on chronological order, or descriptions making use of spatial order. Perhaps after we understand the structure of such ordered texts we will be better prepared to understand (by contrast) texts about sets of things which are not naturally ordered.

Appendix

The input for a second example of a proof is

Lexical information:

Predicate	Property	Value
N	arguments	x
	+active form	is a native of Ajo
	gender	M
	syntactic type	noun
H	arguments	x
	+active form	has a cephalic index in excess of 96
	gender	M
	syntactic type	clause
W	arguments	x
	+active form	is a woman
	gender	F
	syntactic type	noun

B	arguments	x
	+active form	has Pima blood
	gender	M
	syntactic type	clause
P	arguments	x, y
	+active form	is a parent
	+passive form	is parented
	preposition for x	by
	preposition for y	of
	gender	M
	syntactic type	noun
Universe	arguments	x
	+active form	is a person
	gender	M
	syntactic type	noun

Formal proof:

L1	$\forall x\ (Nx \rightarrow Hx)$	(PR)
L2	$\forall x\ ((Wx \wedge Hx) \rightarrow Bx)$	(PR)
L3	$\forall x\ \exists y\ (Wy \wedge Py,x)$	(KN)
L4	$\forall x\ \forall y\ ((Py,x \wedge By) \rightarrow Bx)$	(PR)

L5	$\forall y\ (Py,x \rightarrow Ny)$	(PR)
L6	$\exists y\ (Wy \wedge Py,x)$	(UI L3)
L7	$Wc \wedge Pc,x$	(EI L6)
L8	$Pc,x \rightarrow Nc$	(UI L5)
L9	$Nc \rightarrow Hc$	(UI L1)
L10	$(Wc \wedge Hc) \rightarrow Bc$	(UI L2)
L11	Bc	(TF L7 L8 L9 L10)
L12	$\forall y\ ((Py,x \wedge By) \rightarrow Bx)$	(UI L4)
L13	$(Pc,x \wedge Bc) \rightarrow Bx$	(UI L12)
L14	Bx	(TF L7 L11 L13)
L15	$(\forall y\ (Py,x \rightarrow Ny)) \rightarrow Bx$	(CD L5 L14)
L16	$\forall x\ ((\forall y\ (Py,x \rightarrow Ny)) \rightarrow Bx)$	(UG L15)

The output generated by EXPOUND is

Theorem:

Suppose that every native of Ajo has a cephalic index in excess of 96. Suppose furthermore that every woman who has a cephalic index in excess of 96 has Pima blood. Suppose that if any person is parented by any person, say w , and the person w has Pima blood then the person has Pima blood. Then if every person is a native of Ajo whenever he is a parent of any person, say p , then the person p has Pima blood.

Proof:

Suppose that every person who is a parent of a person x is a native of Ajo. Since some woman is a parent of x, let c denote such a woman who is a parent of him. But if she is a parent of him then she is a native of Ajo, and if she is a native of Ajo then she has a cephalic index in excess of 96, and if she is a woman and she has a cephalic index in excess of 96 then she has Pima blood, thus she has Pima blood. But by assumption she is a woman and she is a parent of x, and if she is a parent of him and she has Pima blood then x has Pima blood, so he has Pima blood. Thus if every person is a native of Ajo whenever he is a parent of any person, say z, then the person z has Pima blood.

The input for a third example of a proof is

Lexical information:

Predicate	Property	Value
H	arguments	x
	+active form	is in this house
	syntactic type	phrase

C	arguments	x
	+active form	is a cat
	syntactic type	noun
S	arguments	x
	+active form	is suitable for a pet
	-active form	is not suitable for a pet
	syntactic type	phrase
L	arguments	x
	+active form	loves to gaze at the moon
	syntactic type	clause
D	arguments	x, y
	+active form	detests
	syntactic type	clause
V	arguments	x, y
	+active form	avoids
	syntactic type	clause
N	arguments	x
	+active form	is carnivorous
	syntactic type	adjective

P	arguments	x
	+active form	prowls at night
	-active form	does not prowl at night
	syntactic type	clause
M	arguments	x
	+active form	kills mice
	syntactic type	clause
T	arguments	x, y
	+active form	takes
	-active form	does not take
	preposition for y	to
	syntactic type	clause
K	arguments	x
	+active form	is a kangaroo
	syntactic type	noun
Universe	arguments	x
	+active form	is an animal
	syntactic type	noun

Formal proof:

L1	$\forall x\ (Hx \rightarrow Cx)$	(PR)
L2	$\forall x\ (Lx \rightarrow Sx)$	(PR)
L3	$\forall x\ (Djoe,x \rightarrow Vjoe,x)$	(PR)
L4	$\forall x\ (Nx \rightarrow Px)$	(PR)
L5	$\forall x\ (Cx \rightarrow Mx)$	(PR)
L6	$\forall x\ (Tx,joe \rightarrow Hx)$	(PR)
L7	$\forall x\ (Kx \rightarrow \neg Sx)$	(PR)
L8	$\forall x\ (Mx \rightarrow Nx)$	(PR)
L9	$\forall x\ (\neg Tx,joe \rightarrow Djoe,x)$	(PR)
L10	$\forall x\ (Px \rightarrow Lx)$	(PR)
L11	Ty,joe	(PR)
L12	$Ty,joe \rightarrow Hy$	(UI L6)
L13	Hy	(TF L11 L12)
L14	$Hy \rightarrow Cy$	(UI L1)
L15	Cy	(TF L13 L14)
L16	$Cy \rightarrow My$	(UI L5)
L17	My	(TF L15 L16)
L18	$My \rightarrow Ny$	(UI L8)
L19	Ny	(TF L17 L18)
L20	$Ny \rightarrow Py$	(UI L4)
L21	Py	(TF L19 L20)
L22	$Ty,joe \rightarrow Py$	(CD L11 L21)
L23	$\forall x\ (Tx,joe \rightarrow Px)$	(UG L22)
L24	Kz	(PR)

L25	$Kz \rightarrow \neg Sz$	(UI L7)
L26	$\neg Sz$	(TF L24 L25)
L27	$Lz \rightarrow Sz$	(UI L2)
L28	$Pz \rightarrow Lz$	(UI L10)
L29	$\neg Pz$	(TF L28 L27 L26)
L30	$Tz,joe \rightarrow Pz$	(UI L23)
L31	$\neg Tz,joe$	(TF L30 L29)
L32	$\neg Tz,joe \rightarrow Djoe,z$	(UI L9)
L33	$Djoe,z \rightarrow Vjoe,z$	(UI L3)
L34	$Vjoe,z$	(TF L31 L32 L33)
L35	$Kz \rightarrow Vjoe,z$	(CD L24 L34)
L36	$\forall x\ (Kx \rightarrow Vjoe,x)$	(UG L35)

The output generated by EXPOUND is

Theorem:

Suppose that every animal in this house is a cat. Suppose moreover that every animal which loves to gaze at the moon is suitable for a pet. Suppose that Joe avoids every animal which he detests. Suppose that every carnivorous animal prowls at night. Suppose that every cat kills mice. Suppose that every animal which takes to Joe is in this house. Suppose that every kangaroo is not suitable for a pet. Suppose that every animal which kills mice is carnivorous. Suppose that Joe detests every animal which does not take to him. Suppose that every animal which prowls at night loves to gaze at the moon. Then Joe avoids every kangaroo.

Proof:

We want to show that Joe avoids every kangaroo. This follows from the fact that every animal which takes to Joe prowls at night, which we shall now prove.

Suppose that a animal y takes to Joe. Then it is in this house. Thus it is a cat. Consequently it kills mice. Consequently it is carnivorous. Consequently it prowls at night. Thus every animal which takes to Joe prowls at night.

Suppose that a animal z is a kangaroo. Then it is not suitable for a pet. But if it prowls at night then it loves to gaze at the moon, and if it loves to gaze at the moon then it is suitable for a pet, therefore it does not prowl at night.

Because if z takes to Joe then it prowls at night, and we have shown that it does not prowl at night, it does not take to Joe. This and the fact that if it does not take to him then he detests z , and if he detests it then he avoids it imply that he avoids it. Therefore he avoids every kangaroo.

Tables & Figures

Rule	Name	Informal meaning
PR	premiss	assume P
CD	conditionalization	after inference of Q from premiss P, infer $P \rightarrow Q$
KN	known fact	we know that P
TF	truth-functional inference	infer Y from X_1, X_2, ... because $(X_1 \wedge X_2 \wedge \ldots) \rightarrow Y$ is a tautology
CQ	converting quantifiers	from $\neg \forall x\, Fx$ infer $\exists x \neg Fx$
UI	universal instantiation	from $\forall x\, Fx$ infer Fy
UG	universal generalization	from Fy infer $\forall x\, Fx$
EI	existential instantiation	from $\exists x\, Fx$ infer Fy
EG	existential generalization	from Fy infer $\exists x\, Fx$

Table 1. Inference Rules.

Predicate	Property	Value
F	arguments	x
	+active form	is a worker
	-active form	is not a worker
	gender	M
	syntactic type	noun
G	arguments	x
	+active form	signed the contract
	-active form	did not sign the contract
	syntactic type	clause
H	arguments	x
	+active form	is in the union
	-active form	is not in the union
	syntactic type	phrase
Universe	arguments	x
	+active form	is a person
	gender	M
	syntactic type	noun

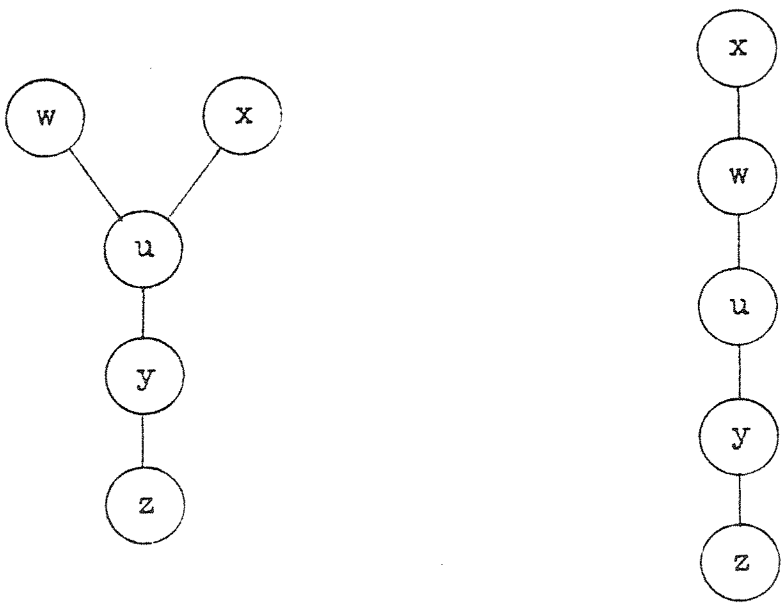
Table 2. Lexical Information.

L1	$\forall x \, ((Fx \wedge Gx) \rightarrow Hx) \rightarrow \exists x \, (Fx \wedge \neg Gx)$	(PR)
L2	$\forall x \, (Fx \rightarrow Gx) \vee \forall x \, (Fx \rightarrow Hx)$	(PR)
L3	$\forall x \, ((Fx \wedge Hx) \rightarrow Gx)$	(PR)
L4	$\exists x \, (Fx \wedge \neg Gx)$	(PR)
L5	$Fc \wedge \neg Gc$	(EI L4)
L6	$\forall x \, (Fx \rightarrow Gx)$	(PR)
L7	$Fc \rightarrow Gc$	(UI L6)
L8	$Gc \wedge \neg Gc$	(TF L5 L7)
L9	$\forall x \, (Fx \rightarrow Gx) \rightarrow (Gc \wedge \neg Gc)$	(CD L6 L8)
L10	$\forall x \, (Fx \rightarrow Hx)$	(PR)
L11	$Fc \rightarrow Hc$	(UI L10)
L12	$(Fc \wedge Hc) \rightarrow Gc$	(UI L3)
L13	$Gc \wedge \neg Gc$	(TF L5 L11 L12)
L14	$\forall x \, (Fx \rightarrow Hx) \rightarrow (Gc \wedge \neg Gc)$	(CD L10 L13)
L15	$Gc \wedge \neg Gc$	(TF L2 L9 L14)
L16	$\exists x \, (Fx \wedge \neg Gx) \rightarrow (Gc \wedge \neg Gc)$	(CD L4 L15)
L17	$\neg \forall x \, ((Fx \wedge Gx) \rightarrow Hx)$	(TF L1 L16)
L18	$\exists x \, (Fx \wedge Gx \wedge \neg Hx)$	(CQ L17)
L19	$\forall x \, ((Fx \wedge Hx) \rightarrow Gx) \rightarrow \exists x \, (Fx \wedge Gx \wedge \neg Hx)$	(CD L3 L18)

Figure 1. A proof.

x	P	(PR)
w	$P \rightarrow Q$	(PR)
u	Q	(TF x w)
y	$(P \rightarrow Q) \rightarrow Q$	(CD w u)
z	$P \rightarrow ((P \rightarrow Q) \rightarrow Q)$	(CD x y)

(a)



(b) (c)

Figure 2. Graphs representing a proof.

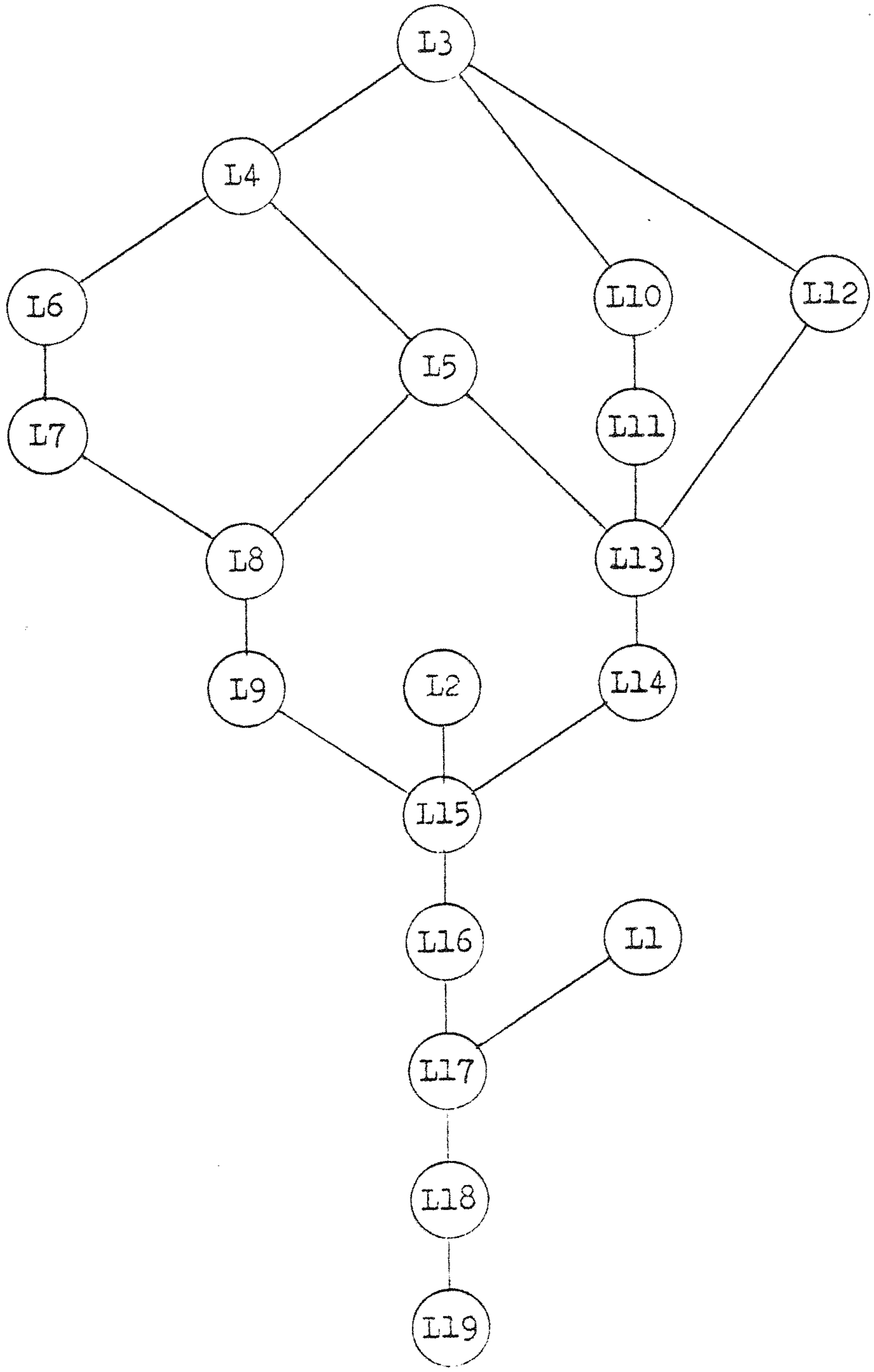


Figure 3. First graph for input proof.

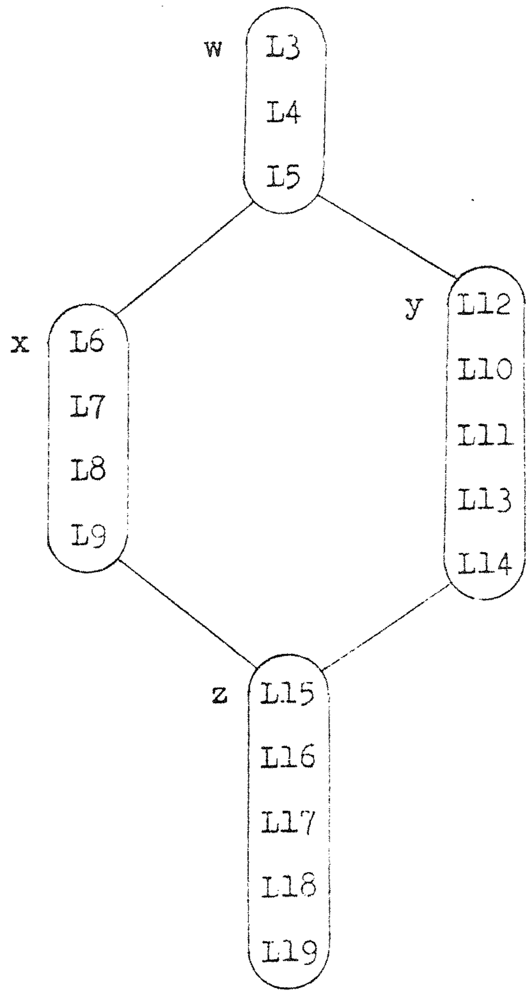


Figure 4. Final graph showing lines grouped into paragraphs.

References

1. Charniak, Eugene. Toward a model of children's story comprehension. AI-TR-266. M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass., 1972.

2. Longacre, Robert E. Discourse, Paragraph, and Sentence Structure in Selected Philippine Languages. The Summer Institute of Linguistics, Santa Ana, Ca., 1968, vol. 1.

3. Phillips, Brian. Topic Analysis. Presented at the 1973 International Conference on Computational Linguistics, Pisa, Italy, Aug. 27-Sept. 1, 1973.

4. Schank, Roger C. Understanding paragraphs. Centro di Documintazione della Fondazione Dalle Molle pir gli studi linguistici e di comunicazione internazionale, Villa Barbariga, Italy, 1974.

5. Aristotle. On Interpretation.

6. ———. Prior Analytics.

7. Fitch, Frederic Brenton. Symbolic Logic, an Introduction. The Ronald Press Company, N.Y., 1952.

8. Quine, Willard Van Orman. Methods of Logic, Revised ed. Henry Holt and Company, Inc. N.Y., 1959.