appeared in:

Philosophy and the Computer

L. Burkholder (ed.)

Westview Press, 1992

8

Searching for Proofs (in Sentential Logic)

Wilfried Sieg and Richard Scheines

Introduction

The Carnegie Mellon Proof Tutor project was motivated by pedagogical concerns: we wanted to use a "mechanical" (i.e. computerized) tutor for teaching students

(1) *how to construct derivations in a natural deduction calculus,*

and

(2) *how to apply the acquired skills in non-formal argumentation.*

No available CAI system in logic provided support for these goals; neither did automated theorem provers, as they were largely based on machine-oriented resolution techniques not suitable for our purposes. So we started developing a proof search program that was to constitute the logical core of a "proof tutor." Indeed, we developed a novel calculus, the *intercalation calculus*, in terms of which the search is conducted.

This report focuses on the broad aspects of the project concerned with (1), and is long overdue.[1] Some of our plans for (2) are indicated in the Concluding Remarks (Section 4). In the first part of this report we sketch the background against which the distinctive features of our proof tutor stand out. That is followed, in the second part, by a discussion of the conceptual framework and the intercalation calculus; there we describe the search space and important metamathematical properties of the calculus. The third and last part is concerned with proof heuristics; i.e. motivated and efficient ways for traversing the search space.

1. Background

For our project it was crucial to have a "theorem proving system" that can provide advice to a student user; indeed, pertinent advice at any point in an attempt to solve a proof construction problem. To be adequate for this task a system must be able to find proofs, if they exist, and follow a strategy that *in its broad direction* is logically motivated, humanly understandable, and memorable. Thus, we have been developing an algorithm that does perform a **DIRECT, HEURISTICALLY GUIDED SEARCH** for derivations, as a first step, in just sentential logic. We argue implicitly against the view that there is a deep-seated conflict between a logical and a heuristic approach: given an appropriate formal frame, these approaches complement each other in a most satisfactory way. So let us describe such a frame for the representation of arguments.

1.1. *Natural Deduction*

If procedures that search for solutions to problems, e.g. proving mathematical theorems, are to have implications for (the theory of) human problem-solving, they should be "cognitively faithful." That was emphasized by Newell, Shaw, and Simon in their classical work "Empirical Explorations with the Logic Theory Machine–A case study in heuristics" (1957). Already in the twenties, David Hilbert had maintained that logical formalisms provide a framework for modelling cognitive processes that underly rigorous mathematical arguments. He claimed indeed more in the polemical discussions with Brouwer; let us quote from his 1927 paper "The Foundations of Mathematics" (Hilbert, 1927, p. 475):

> The formula game that Brouwer so deprecates has, besides its mathematical value, an important general philosophical significance. For this formula game is carried out according to certain definite rules, in which the *technique of our thinking* is expressed. These rules form a closed system that can be discovered and definitively stated. The fundamental idea of my proof theory is none other than to describe the activity of our understanding, to make a protocol of the rules according to which our thinking actually proceeds.... If any totality of observations and phenomena deserves to be made the object of serious and thorough investigation, it is this one.

To emphasize, the claims are: (1) logical rules express directly techniques of our thinking, and (2) derivations in a calculus based on them can serve as protocols of ways we actually think! If there is a plausible candidate for an early programmatic statement of tasks for cognitive science, this is one. But Hilbert's claim, it seems to us, was given some plausibility

&E

$$\frac{\phi \,\&\, \gamma}{\phi} \qquad \frac{\phi \,\&\, \gamma}{\gamma}$$

&I

$$\frac{\phi \qquad \gamma}{\phi \,\&\, \gamma}$$

∨E

$$\frac{\phi \vee \gamma \qquad \begin{matrix}[\phi]\\ \vdots \\ \eta\end{matrix} \qquad \begin{matrix}[\gamma]\\ \vdots \\ \eta\end{matrix}}{\eta}$$

∨I

$$\frac{\phi}{\phi \vee \gamma} \qquad \frac{\phi}{\gamma \vee \phi}$$

→E

$$\frac{\phi \qquad \phi \rightarrow \gamma}{\gamma}$$

→I

$$\frac{\begin{matrix}[\phi]\\ \vdots \\ \gamma\end{matrix}}{\phi \rightarrow \gamma}$$

~E

$$\frac{\begin{matrix}[\sim\phi]\\ \vdots \\ \varphi\end{matrix} \qquad \begin{matrix}[\sim\phi]\\ \vdots \\ \sim\varphi\end{matrix}}{\phi}$$

~I

$$\frac{\begin{matrix}[\phi]\\ \vdots \\ \varphi\end{matrix} \qquad \begin{matrix}[\phi]\\ \vdots \\ \sim\varphi\end{matrix}}{\sim\phi}$$

FIGURE 8.1. The natural deduction rules.

only by Gentzen's work on natural deduction calculi.[2] Rules in those calculi were fashioned explicitly after informal ways of reasoning; they were to reflect, according to Gentzen, "as accurately as possible the actual logical reasoning involved in mathematical proofs" (Gentzen, 1969, p. 74). They incorporate indeed strategies for the use and the introduction of logically complex formulas based on the understanding of their principal connectives. That they underly "cognitive processing" in ordinary propositional reasoning has been supported by recent psychological investigations of L. J. Rips (Rips, 1983, 1989).

The natural deduction rules for the sentential connectives &, ∨, →, and ~ are divided into elimination and introduction rules. The former specify how components of complex formulas can be used, the latter provide conditions under which complex formulas can be inferred from components. In the presentation of the rules we indicate that an assumption has been cancelled by enclosing it in brackets.

The rules for & are absolutely straightforward (see Figure 8.1: &E and &I). The introduction rule for $\vee$ is similarly direct; the elimination rule corresponds to an argument by cases (Figure 8.1: $\vee$E and $\vee$I) . The elimination rule for $\rightarrow$ is the traditional rule of modus ponens; the introduction rule codifies the informal strategy of establishing a conditional by giving an argument from the antecedent to the consequent (Figure 8.1: $\rightarrow$E and $\rightarrow$I). The negation elimination rule is the characteristic rule of classical logic and is needed to prove, for example, the law of the excluded middle and Peirce's law; the introduction rule captures the form of indirect argument as used in the Pythagorean argument for the irrationality of $\sqrt{2}$ (Figure 8.1: ~E and ~I).

In the elimination rules we call the premise that contains the characteristic connective the *major premise*. Notice that the first negation rule implies "ex falso quodlibet," i.e.

$$\frac{\perp}{\phi};$$

where $\perp$ is any contradiction of the form φ & $\sim\varphi$. The precise metamathematical description of derivations in this calculus is a little cumbersome, as one has to keep track of the open assumptions. If a simple description of derivations is desired, e.g. for the proof of Gödel's Incompleteness Theorems, it is better to use axiomatic presentations; however, the tree representation reflects graphically the structure of arguments. For the tutor we chose a Fitch style representation; that has similar graphical advantages, but is easier to put on a screen and avoids the duplication of parts of proofs.

1.2. *Automated Proof Search*

Despite the "naturalness" of natural deduction calculi, the part of proof theory that deals with them has hardly influenced developments in automated theorem proving.[3] For that, a different tradition in proof theory has been important; a tradition that is founded on the work of Herbrand and that of Gentzen concerned with sequent calculi. The key word here is clearly *resolution*. From a purely logical point of view this is peculiar: it is after all the subformula property of special kinds of derivations[4] that makes resolution and related techniques possible, and normal derivations in natural deduction calculi have that very property (with a minor addition). A derivation is called *normal* if it does not contain an application of an I-rule whose conclusion is the major premise of an E-rule. As every derivation can be transformed into a normal one, normal derivations suffice to specify syntactically the logical consequences of assumptions. This theoretical fact, established by Prawitz already in 1965, can be exploited for automated proof search, not just automated theorem proving.

For some, however, natural deduction calculi are unsuited even for automated theorem proving. To point to one very recent example, Melvin Fitting writes in his book *First Order Logic and Automated Theorem Proving* (1990, p. 95):

> Hilbert systems are inappropriate for automated theorem proving. The same applies to natural deduction, since modus ponens is a rule in both.

If natural deduction calculi required unrestricted chaining as axiomatic Hilbert systems do, employed e.g. by the Logic Theory Machine, then they would indeed be inappropriate for theorem proving: there would not be any significant restriction on the search space. However, the presence of modus ponens can be a reason for considering natural deduction calculi as inappropriate only if one does not appreciate the normalization theorem and its corollary, asserting that normal derivations have the subformula property.

Before describing the framework in which our automated proof search proceeds, we want to make a few remarks about related work by, among others, Andrews and Pfenning. The former has been using (versions of) his theorem proving system TP for higher order logic in an educational setting (Andrews et. al., 1988). Students can give natural deduction derivations and are even allowed to work bottom-up and top-down. But the underlying prover is based on mating procedures, and does not provide advice. Pfenning developed an algorithm that uses a mating proof as the basis for a natural deduction argument (Pfenning, 1987). As the latter is by no means determined uniquely by the former, it is necessary to use strategic considerations of a similar sort as they are developed below; but they have not been pushed very far in Pfenning's dissertation. Analogous remarks apply to work on broad frameworks for the implementation of varieties of logical systems, in particular natural deduction systems, as reported e.g. by Paulson (1987) and Felty (1989). Tactics and tacticals for generating derivations are introduced, but there is no attempt to join them strategically in an automated search for proofs.

2. Conceptual Framework

The broad problem is this: How can one *derive a conclusion* φ *from assumptions* $\phi_1, \dots, \phi_n$? or, to put it more vividly, how can one *close —via* *logical rules —the gap between a conclusion* φ *and assumptions* $\phi_1, \ldots, \phi_n$? This question is at the heart of spanning the search space via the **INTERCALATION CALCULUS.**[5] The basic rules of that calculus are, locally, direct reformulations of those for Gentzen's natural deduction calculus; it is the preservation of inferential information and the restricted way in which the rules are used (to close the gap and thus) to build up derivations that is distinctive.

2.1. *Intercalating*

The idea is roughly this: one tries to bridge the gap between assumptions and conclusion by systematically intercalating formulas using the available rules of the natural deduction calculus. I.e., one pursues **ALL** possibilities of using elimination rules to come closer to the conclusion "from above" and inverted introduction rules to come closer to the assumptions "from below." Let us look at two easy examples where the gap is indicated by a question mark.

Example 1:

$$\begin{array}{c} P \to Q \\ P \\ ? \\ Q \end{array}$$

An application of the $\to$-elimination rule leads to:

$$\begin{array}{c} P \to Q \\ P \\ Q \\ ? \\ Q \end{array}$$

Clearly, the gap is closed now. From the "intercalation derivation" one reads off immediately the corresponding natural deduction derivation

$$\frac{P \qquad P \to Q}{Q}$$

This example allows only one straightforward way of attempting to close the gap, namely by using the elimination rule for the conditional. The next example gives us choices, as the conclusion is a complex formula.

Example 2:

$$\begin{array}{c} P \,\&\, Q \\ ? \\ Q \,\&\, P \end{array}$$

If we do use the (inverted) &-introduction rule we are led to the configuration:

$$\frac{\begin{array}{c} P \,\&\, Q \\ ? \\ Q \end{array} \qquad \begin{array}{c} P \,\&\, Q \\ ? \\ P \end{array}}{Q \& P}$$

But now it is quite clear how to close the remaining gaps—by application of the &-elimination rule.

The idea underlying these simple examples is captured in the intercalation calculus. Its rules operate on triples of the form

$$\alpha;\beta?G.$$

α is a sequence of formulas, namely of the *available assumptions;* G is the *current conclusion* or *goal;* and β is a sequence of formulas obtained by &-elimination and $\to$-elimination from elements of α. Let us list the intercalating rules. The $\downarrow$-rules correspond to elimination rules, the $\uparrow$-rules to (inverted) introduction rules. We use the following conventions: if α and β are sequences, the concatenation of α and β is indicated by juxtapositing α and β; if α is a sequence of formulas and ϕ is a formula, the extension of α by ϕ is indicated by α,ϕ. We use $\phi \in \alpha$ to abbreviate that the formula ϕ is an element of the sequence α.

$\downarrow$&: $\alpha;\beta?G,\ \phi_1 \& \phi_2 \in \alpha\beta \Rightarrow \alpha;\beta,\phi_1?G$ OR $\alpha;\beta,\phi_2?G$

$\downarrow\vee$: $\alpha;\beta?G,\ \phi_1 \vee \phi_2 \in \alpha\beta \Rightarrow \alpha,\phi_1?G$ AND $\alpha,\phi_2?G$

$\downarrow\to$: $\alpha;\beta?G,\ \phi_1 \to \phi_2 \in \alpha\beta\ ,\ \phi_1 \in \alpha\beta \Rightarrow \alpha;\beta,\phi_2?G$

The question $\alpha;\beta?G$ is the *same question as* $\alpha^*;\beta^*?G$ just in case the sets of formulas in the sequences a;b and $\alpha^*;\beta^*$ are identical; if the first set is contained in the second, then the question $\alpha^*;\beta^*?G$ is *easier than* $\alpha;\beta?G$. The rules can be restricted to avoid obvious repetitions of questions and also the asking of easier questions; e.g.

r↓&$_i$: $\alpha;\beta?G,\ \phi_1 \& \phi_2 \in \alpha\beta,\ \phi_i \notin \alpha\beta \Rightarrow \alpha;\beta,\phi_i?G$, i =1 or 2

r↓∨: $\alpha;\beta?G,\ \phi_1 \vee \phi_2 \in \alpha\beta,\ \phi_1 \notin \alpha\beta,\ \phi_2 \notin \alpha\beta \Rightarrow \alpha,\phi_1?G$ **AND** $\alpha,\phi_2?G$

r↓→: $\alpha;\beta?G,\ \phi_1 \rightarrow \phi_2 \in \alpha\beta,\ \phi_1 \in \alpha\beta,\ \phi_2 \notin \alpha\beta \Rightarrow \alpha;\beta,\phi_2?G$

It is important to use these restricted rules when building up the search space.[6] Here we continue the presentation of the intercalation calculus by formulating the ↑-rules.

↑&: $\alpha;\beta?\phi_1 \& \phi_2 \Rightarrow \alpha;\beta?\phi_1$ AND $\alpha;\beta?\phi_2$

↑∨: $\alpha;\beta?\phi_1 \vee \phi_2 \Rightarrow \alpha;\beta?\phi_1$ OR $\alpha;\beta?\phi_2$

↑→: $\alpha;\beta?\phi_1 \rightarrow \phi_2 \Rightarrow \alpha,\phi_1;?\phi_2$

The rules for negation are split into three, where we consider $\perp$ as a placeholder for a pair of contradictory formulas:

$\perp_c$: $\alpha;\beta?\phi,\ \phi \neq \perp \Rightarrow \alpha,\sim\phi;?\perp$

$\perp_i$: $\alpha;\beta?\sim\phi \Rightarrow \alpha,\phi;?\perp$

$\perp_F$: $\alpha;\beta?\perp,\ \varphi \in \mathbf{F} \Rightarrow \alpha;\beta?\varphi$ **AND** $\alpha;\beta?\sim\varphi$.

In the last rule, **F** is the finite class of formulas consisting of all subformulas of elements in the sequence $\alpha\beta$. That **F** can be taken to be finite is clearly crucial for the finiteness of the search space. Smaller and yet sufficient classes are specified below; here we just remark that we always discount double negations: if $\sim\phi$ is in **F**, then we consider only the pair $\sim\phi$ and ϕ in the first two negation rules, not also $\sim\phi$ and $\sim\sim\phi$. A final technical note: it is for metamathematical simplicity that we suppressed in the rules which introduce new assumptions the sequence β of inferred formulas. Logically nothing is lost, as these formulas can be re-inferred; for an efficient implementation such a duplication of efforts has to be, and can be easily, avoided (see section 3.1).

2.2. *Search Space*

Instead of presenting all the logical details, we just indicate the pertinent considerations. We choose to do so by discussing the *search tree* for the question ?(P∨~P); it is partially presented in Figure 8.2. We start out by

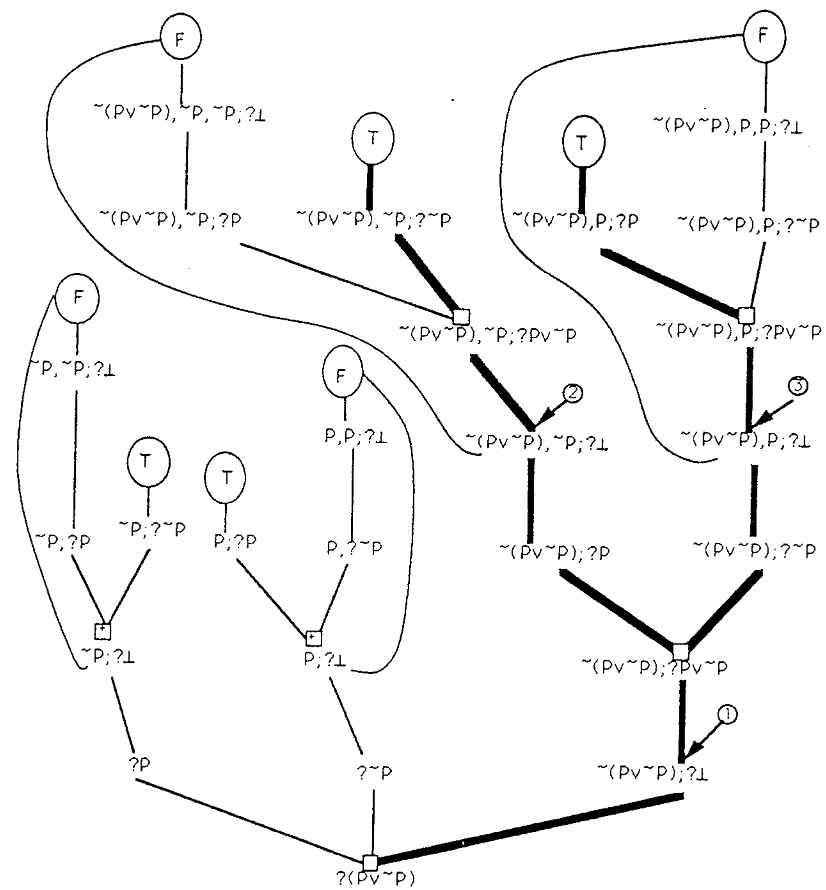


FIGURE 8.2. The search tree for the question ?(P∨~P).

applying three intercalation rules to obtain three new questions; namely, ?P **OR** ?~P **OR**, proceeding indirectly, ~(P∨~P);?⊥. That the branching in the tree is disjunctive is indicated by □. Let us pursue the leftmost branch in the tree: to answer ?P we have to use $\perp_c$ and, because of the restriction on the choice of contradictory pairs, we have only to ask ~P;?P **AND** ~P;?~P. ⊞ indicates that the branching is conjunctive here. In the first case only $\perp_c$ can be applied and leads to the "same" situation we just analyzed: using ~P as an assumption, ⊥ has to be proved. Thus we *close the branch* with a circled **F**, linking it to the "same" earlier question on this branch. In the second case the gap between assumptions and goal is obviously closed, so we top this branch with a circled **T**. The other parts of the tree are con-

$$\dfrac{\dfrac{\dfrac{\dfrac{\sim P}{P \vee \sim P} \qquad \cancel{\sim(P \vee \sim P)}}{P}}{P \vee \sim P} \qquad \cancel{\sim(P \vee \sim P)}}{(P \vee \sim P)}$$

FIGURE 8.3. Derivation from the "left" branch of the search tree.

structed in a similar manner. But the tree is not quite full: at the nodes that are distinguished by arrows the additional contradictory pair consisting of P and ~P has to be considered. At nodes 2 and 3 the resulting branches are almost immediately closed with a circled F; at node 1, in contrast, the resulting subtree is of interest and will be discussed below.

Every branch in a search tree constructed with the intercalation rules is finite and is being topped by either **T** or **F**. We can give rules that associate one of these values to a node N, given that its predecessors M_i have such values, and thus associate recursively a value with the initial question. This is done as follows: (i) if N has exactly one predecessor M, the value of N is that of M; (ii) if N has exactly two predecessors and the branching is conjunctive, the value of N is **T** if both predecessors have **T**, otherwise it is **F**; (iii) if N has two or more predecessors and the branching is disjunctive, the value of N is **F** if all predecessors have **F**, otherwise it is **T**. Using these rules it is quite easy to see that the basic question in our tree has the value **T**. Small subtrees will often lead already to this evaluation: in our example, from either of the branches with thicker vertices results the value **T**. These subtrees contain enough information for the extraction of derivations in a variety of styles of natural deduction. For our calculus we can easily obtain the corresponding derivations; namely, from the "left" (darkened) branch (see Figure 8.3).

The proof extracted from the "right" (darkened) branch is very similar; it is obtained by just interchanging the formulas ~P and P. Let us indicate the third proof that can be extracted when the branching at node 1 with P and ~P is taken into account. (The branchings at the remaining numbered nodes do not give additional proofs.) It is a combination of the two proofs just described (see Figure 8.4).

To summarize: given assumptions and a conclusion, we can build up systematically a finite search tree and thereby explore all (non-repetitive) possibilities of gap-closing via the intercalation rules. It can be shown, that

$$\dfrac{\dfrac{\dfrac{\cancel{P}}{P \vee \sim P} \qquad \cancel{\sim(P \vee \sim P)}}{P} \qquad \dfrac{\dfrac{\cancel{P}}{P \vee \sim P} \qquad \cancel{\sim(P \vee \sim P)}}{\sim P}}{(P \vee \sim P)}$$

FIGURE 8.4. The third proof.

if the original question evaluates to **T**, then a normal derivation can be extracted; otherwise, the full search tree allows us to define a semantic counterexample to the question. Thus we have a new kind of completeness proof suited to natural deduction calculi; it is similar to those for semantic tableaux or the sequent calculus. The completeness claim for the intercalation calculus can be formulated as follows:

> **Completeness Theorem.** The (full) search tree for the problem $\phi_1,\ldots,\phi_n$; ? φ either contains a subtree from which a normal derivation can be effectively constructed or it provides a counterexample to the problem.

There are a number of direct and easy consequences (of the proof); namely,

> (1) we have a decision procedure for sentential logic, as every search tree is finite;
>
> (2) there are canonical ways of extracting derivations in various forms of natural deduction calculi;
>
> (3) the extracted derivations are normal and satisfy the subformula property.

More details concerning (3) will be discussed below. If we were not restricted by computational concerns, the *basic procedure* for answering a question of the form α;?G could be: generate the full search tree; if the value associated with the question is **F**, the answer is negative; if the value is **T**, then determine the "smallest" subtree of the full search tree that allows the **T** evaluation, and extract its derivation. But as it stands we want to find a subtree, that allows the **T** evaluation and provides us with a good proof as quickly as possible and hopefully without generating the full search tree. It is for this purpose that guiding heuristics are needed.

3. Heuristics for Search

There is no conflict between the use of heuristics to build up proper pieces of the search space to find a derivation quickly and the generation of, ultimately, the full space to guarantee completeness of the procedure! If there is no counterexample to the question we can close the gap between assumptions and conclusion by a finite sequence of intercalating steps: (i) from above via elimination rules, (ii) from below via inverted introduction rules, or (iii) via the rules for indirect argumentation. The central questions are: (1) can one reduce further the need for exploring paths in the search tree? and (2) which of the finitely many possibilities of proceeding should be selected before the others? As to the second question, we consider three classes of heuristic advice based on logical ideas; they are, clearly, related to (i) - (iii). But before presenting those, we refine the steps that have already been taken to cut down the search space; this is obviously relevant to (1).

3.1. Pruning

The build-up of the search tree guarantees, first of all, that the same question is not answered twice on a particular branch and, secondly, that extracted derivations are normal. The first feature helps to insure that no infinite paths are generated and is implemented, partly, by using the restricted forms of the intercalation rules and, partly, through the **F**-closure condition. Normality guarantees that derivations do not contain detours, as an introduction rule is never followed immediately by an elimination rule whose major premise is the conclusion of the introduction rule. This is a consequence of the fact that the $\downarrow$-rules (corresponding to elimination rules) are used only to close a gap from above, wheras the $\uparrow$-rules (corresponding to introduction rules) are only used to close a gap from below.

This separation of $\downarrow$-rules and $\uparrow$-rules has actually another significant consequence, as extracted derivations satisfy a stricter subformula property. Let us define the usual notion of *positive* and *negative* subformula of a given formula A: (1) A is a positive subformula of A; (2) if B & C or B $\vee$ C are positive [negative] subformulas of A, so are B and C; (3) if B $\rightarrow$ C or ~B are positive [negative] subformulas of A, then B is a negative [positive] and C a positive [negative] subformula of A. We say that a formula is a *strictly positive* subformula of A, just in case it can be shown to be a positive subformula without appealing to clause (3) in the above definition. It is not difficult to show that for extracted derivations from α to G the following holds: every formula is either a positive subformula of an assumption, a subformula of the conclusion, or (the negation of) a negative subformula of α,~G. This is a property of completed derivations. In stepping from one question to the next the syntactic connection is tighter and leads to a considerable further restriction on the choice of contradictory pairs: we have to consider only pairs ϕ and ~ϕ, such that ~ϕ is a strictly positive subformula of an available assumption.

The considerations in the last paragraph allow us to formulate the rule $\perp_{\mathbf{F}}$ for smaller classes **F** and thus reduce the number of branchings at certain nodes in the search tree. Now we make use of the already constructed part of the search tree to avoid answering a question that has been asked and answered before; indeed, that can be slightly generalized as we do not just focus on identical questions. That is done in three parts.

(A.1) We store globally all negative answers to questions of the form $\alpha;\beta$?G, and stop pursuing—on other branches—questions of the form $\alpha^*;\beta^*$?G, when the set of formulas in $\alpha\beta$ is a superset of those in $\alpha^*\beta^*$. Clearly, if G cannot be derived from $\alpha\beta$ then it cannot be derived from $\alpha^*\beta^*$. As it is not necessary to know how the negative answer was obtained, we discard the part of the tree leading to it.

(A.2) We store locally, i.e. in the current search tree that may lead to a derivation, all positive answers to questions of the form $\alpha;\beta$?G, and stop pursuing—on other branches—questions of the form $\alpha^*;\beta^*$?G, when the set of formulas in $\alpha\beta$ is a subset of those in $\alpha^*\beta^*$. Clearly, if G is already derivable from $\alpha\beta$ then it is derivable a fortiori from $\alpha^*\beta^*$; we are dealing with an easier question! (The positive information could also be stored globally, but we do not believe at the moment that that would speed up the search.)

(A.3) In parallel to the search tree we build up partial Fitch-derivations. This particular representation can be exploited to avoid re-obtaining positive answers by using a broader notion of "formula available on a branch." Roughly speaking, when asking the question $\alpha;\beta$?G at a particular node we consider as available not only the formulas in $\alpha\beta$, but all formulas on the branch leading to this node that are available in the corresponding partial Fitch-derivation. In the case of conjunction and indirect arguments this can be further extended, as we consider naturally the first derivation of one of the conjuncts, respectively one component of the contradictory pair as part of the partial Fitch-derivation.[7] (Here is a computational advantage of the Fitch-style representation over the tree representation; the latter would require duplication of subtrees.)

Up to now we sidestepped, in a sense, the difficult problems either by

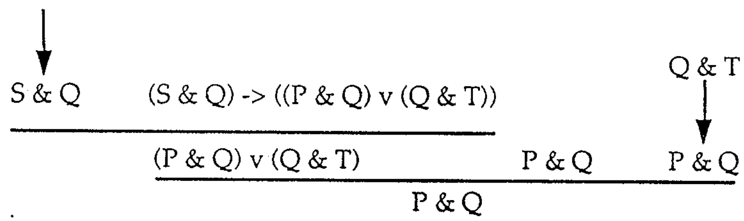


FIGURE 8.5.

avoiding to ask questions (through the restricted formulation of rules and the narrower choice of contradictory pairs) or by exploiting already obtained answers. But how do we obtain, intelligibly and efficiently, answers that allow us to close the gap between assumptions and conclusion?

3.2. *Extracting and Inverting*[8]

If we consider just the ↑-rules, they seem to help to bridge the gap. They may lead to derivations longer than necessary, but in general—when no indirect argument is required—they go in the right direction. The reason is that answers to the newly raised questions provide immediately an answer to the original question. ↓-rules, in contrast, may go off in completely irrelevant directions when applied "mechanically." After all, among the assumptions may be formulas that are not appealed to in any normal derivation of the conclusion. Here it is necessary to ensure the directedness of applications, so that they lead to formulas "closer" to the conclusion. For that purpose we introduce an additional, complex rule, the *Extraction Strategy:* try to obtain the goal via a sequence of elimination rules when the goal is a strictly positive subformula of available formulas. This will lead in general to new problems, as the minor premises of the elimination rules have to be derived. But, as in the case of the ↑-rules, if all the subproblems are solved, the original question has been answered. Instead of describing this in utter generality let us show by an example what is involved. Consider the problem:

$$(S \,\&\, Q) \rightarrow ((P \,\&\, Q) \vee (Q \,\&\, T)), \sim T, P \,\&\, S, \sim T \rightarrow Q;\ ?\ P \,\&\, Q$$

As P & Q is a strictly positive subformula of the first assumption, we try to extract P & Q from it. We have immediately the configuration in Figure 8.5. So the problem is reduced to finding from the remaining assumptions a proof of

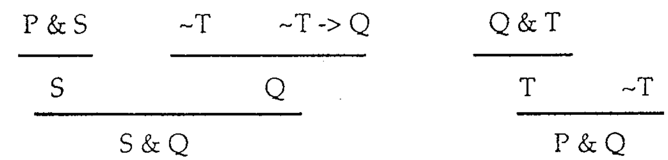


FIGURE 8.6.

(1) S&Q

and of

(2) P&Q

using also the temporary assumption Q & T. But that is easy to do as seen by the derivations in Figure 8.6.

We hope the idea behind the extraction strategy is clarified by this example. We chose it, however, also to indicate the real problem. How are we to make a choice between inversion and extraction? In this particular example, taking the inversion step first leads to a shorter and better derivation, namely the one in Figure 8.7. What we can do (also when the goal is not a strictly positive subformula of an assumption) is to pursue the *Inversion Strategy:* apply inverted introduction rules for & and →, in stages, until these rules cannot be applied or the new goals are strictly positive subformulas of available formulas.

3.3. *Choosing*

The above considerations point to a general moral: the choice of the "next step" has to be informed by the purely syntactic context consisting of available formulas and the goal. We use that context to determine a ranking of the rules or strategies by means of which the goal can be prima facie obtained. Several factors play a role in determining this ranking; let

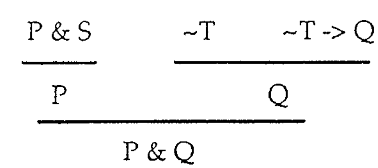


FIGURE 8.7.

us formulate relevant questions for the extraction strategy:

(a↓) Is the goal G a strictly positive subformula of an available formula?

(b↓) How deeply is G embedded, in case (a↓) has an affirmative answer or, indeed, several affirmative answers?

(c↓) What are the main connectives of the formulas in which G is embedded?

Similar questions can be asked for the inversion strategy:

(a↑) Can the conclusion be built up out of other formulas?

(b↑) Are these other formulas strictly positive subformulas of available formulas?

(c↑) In case (b↑) has an affirmative answer, (b↓) and (c↓) apply.

We assign numerical scores depending on the answers to these questions and rank the rules and, thus, strategies accordingly.[9] In the example just discussed, this ranking indeed favors the second derivation. The point is that we take into account obviously significant contextual features whose determination is local.

Up to now we have hardly addressed the rules for negation. We turn to this next. Once we have decided to "go indirect" and pursue the refutation strategy, we have to select a formula $\sim\phi$ and prove both it and its unnegated matrix. It is here that an additional ranking comes in, namely the ranking of contradictory pairs of formulas. Since indirect proof works when the assumption ψ of an indirect proof leads to absurdity, we favor those contradictories that have an obvious connection to ψ. That is, we rank highly those contradictories that are positive subformulas of ψ or contain ψ as a positive subformula. Then the procedures used to determine the earlier ranking are exploited: it is after all largely a matter of trying to determine which pair of contradictory formulas is easiest to prove.

The overall strategy of selecting the question following $\alpha;\beta?G$ is very roughly described now. (For a corresponding flow-diagram see Appendix 1.) The first distinction is made according to the form of G. If G is a conjunction or conditional we order the inversion-extraction possibilities and pursue the one with the lowest score; in case these possibilities do not lead to a positive answer, we pursue the refutation strategy. If G is a disjunction, negation, or an atomic formula, we make one step towards an

indirect argument using G itself as the new goal and then proceed as before; in case this does succeed, we check whether the assumption ~G was used in the proof at all and construct, in case it was not, a direct argument. If we apply this procedure of building the search tree piecewise to our problem ?(P v ~P), the part of the tree that is being traversed at all is the "left" (thickened) branch in the earlier tree (see Figure 8.8).

The memorable and very crude guiding strategy[9] is then this: try to extract the conclusion; if that is not possible invert, in case the conclusion is a conjunction or a conditional, and refute, in case the conclusion is an atomic formula, a negation, or a disjunction. When pursuing this strategy one has to keep in mind these imperatives: (1) avoid pursuing avenues that have been pursued; (2) take into account the context and possibly

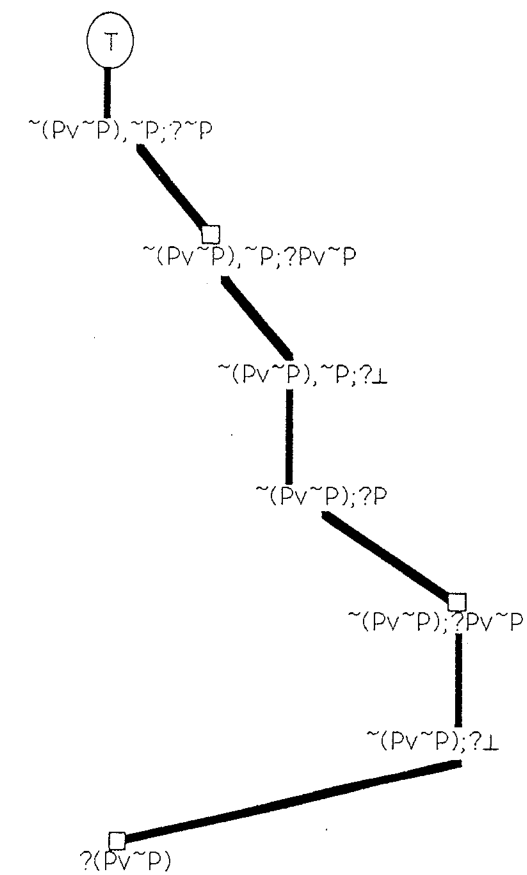


FIGURE 8.8.

change the order of extraction and inversion; (3) turn refutations, if possible, into direct arguments.

4. Concluding Remarks

We think it is logically significant that fast automated proof search is possible. However, for our project it is more important that the tutor based on the search algorithm seems to be pedagogically effective in teaching students strategies for problem solving. We have used the tutor within a totally computerized introduction to logic, a version of the VALID program developed by Patrick Suppes and collaborators at Stanford University. Students who took the course within the tutor environment (i.e., having the possibility of working forward and backward) surpassed students who were allowed to either work just forward or just backward significantly in their ability to solve difficult problems.[11]

We plan to extend the search algorithm to predicate logic and then to elementary parts of set theory. There are clearly non trivial difficulties to be overcome, but they are not insurmountable. Apart from logical and mathematical problems, we will continue to address the psychological and pedagogical issues surrounding informal argumentation. To do this we plan to supplement the logical part of the tutor by a linguistic module that translates between (relatively regimented) parts of English, as used in elementary set theory, and the appropriate, definitionally expanded formal language. Students should be able to give informal arguments that are controlled—using the linguistic module—by the checker that is trivially contained in the search algorithm; and the latter should be powerful enough to provide intelligible and subject-specific assistance.

Notes

1. The project has been pursued by us since 1986 together with Jonathan Pressler and Chris Walton. The very basic ideas go actually back to 1985, when Sieg and Preston Covey discussed the underlying issues, and when the first steps were taken with Leslie Burkholder and Jonathan Miller.

2. Gentzen was a student of Hilbert's. Note also that the axiomatic calculi discussed by Hilbert were organized in such a way that the distinctive role of each logical connective was brought out—in analogy to the organization of the axioms of geometry in Hilbert's "Grundlagen der Geometrie". Most of the axioms correspond directly to rules in the Gentzen calculus; see pp. 465-466 of Hilbert, 1927.

3. For a survey of natural deduction theorem proving, see Bledsoe, 1977.

4. Derivations in Herbrand's calculus and derivations in the sequent calculus without cut have the *subformula property*: they contain only subformulas of their endformula, respectively endsequent. Both calculi enjoy the completeness property.

5. This calculus was proposed by Sieg in August 1987 to capture the essence of the problem formulated above; the basic metamathematical properties, also concerning predicate logic, were established then. The detailed proofs of these results will be reported in a separate publication.

6. That is described in the next subsection; compare also section 3.1.

7. In these two cases we are facing a conjunctive branching and address canonically the "left" problem first. (In principle, one should even here make a contextually informed choice.)

8. I.e. here: extracting of formulas from assumptions via elimination rules.

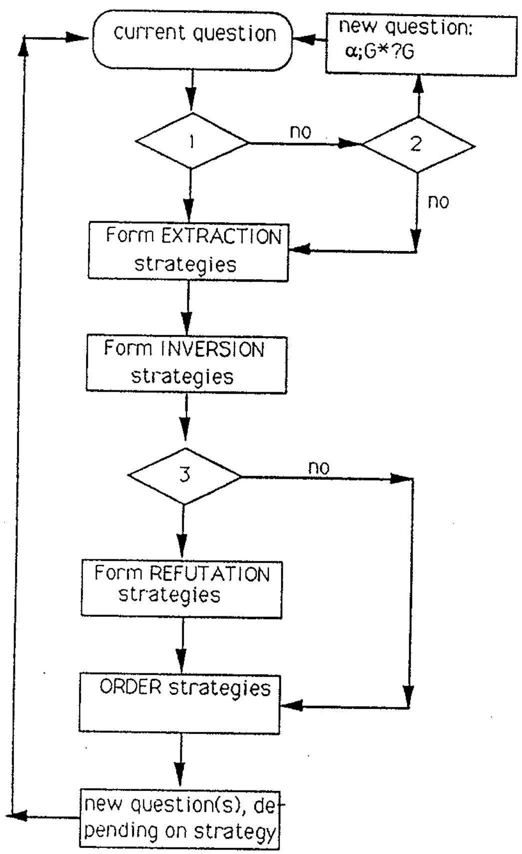
9. We give preference to lower scores. Here and below, if there is a tie, i.e. at least two possibilities attain the same score, pick one.

10. The experiments that we carried out and are carrying out were discussed in our contribution to the Fifth Conference on Computers and Philosophy, held at Stanford University, August 1990; for a brief description of experiments concerning only the interface, see Appendix 3.

11. It should be possible to use diagrams as steps in arguments via their proper linguistic representation.

Appendix 1: Flowchart

Flowchart to determine the choice of the next question, when it has been determined that the branch with question α;β?G as top node has to be expanded.



At 1 we ask: are we in an indirect argument with respect to G ? If not, we ask at 2: is G a negation, a disjunction, or an atomic formula? If yes, we let in the latter two cases G* be ~G; in the first case G* is the unnegated matrix of G. Finally, at 3 we determine whether the set of inversion and extraction strategies is empty or not.

Appendix 2: Proof Examples

We give PROOF TUTOR proofs of (1) Peirce's Law that was not provable by the Logic Theory Machine, and (2) a problem from Pelletier's list characterized as "not solvable by unit resolution nor by 'pure' input resolution" (Pelletier, 1986). The naming of rules is different from the one used above, but self-explanatory; ∨-elimination is here also formulated in a different way, namely as allowing the inference from (ϕ ∨ ψ) and ~ϕ toψ, and from (ϕ ∨ ψ) and ~ψ to ϕ.

1	* (p→q)→p	assumption
2	* * ¬p	assumption
3	* * * p	assumption
4	* * * * ¬q	assumption
5	* * * q	¬_elim_1,4,2,3
6	* * p→q	→_intro_3,5
7	* * p	→_elim_r,1,6
8	* p	¬_elim,2,2,7
9	((p→q)→p)→p	→_intro,1,8

1	* ((p∨q)&(¬p∨q))&(p∨¬q)	assumption
2	* * ¬p∨¬q	assumption
3	* * * ¬q	assumption
4	* * * * ¬p	assumption
5	* * * * (p∨q)&(¬p∨q)	&_elim_l,1
6	* * * * p∨q	→_elim_1,5
7	* * * * q	∨_elim_r,6,4
8	* * * ¬¬p	¬_intro,4,3,7
9	* * * (p∨q)&(¬p∨q)	&_elim_l,1
10	* * * (¬p∨q)	&_elim_r,9
11	* * * q	∨_elim_r,10,8
12	* * ¬¬q	¬_intro,3,3,11
13	* * ¬p	∨_elim_l,2,12
14	* * (p∨q)&(¬p∨q)	&_elim_l,1
15	* * p∨q	&_elim_l,14
16	* * q	∨_elim_r,15,13
17	* * p∨¬q	&_elim_r,1
18	* * ¬q	∨_elim_r,17,13
19	* ¬(¬p∨¬q)	¬_intro,2,18,16
20	((((p∨q)&(¬p∨q))&(p∨¬q))→¬(p∨¬q)	→_intro,1,19

Appendix 3: Experiments

Computerized proof checkers have proliferated, but little experimental work has been done to assess their effectiveness as learning environments, or the effectiveness of various of their features. In the fall of 1989 we conducted experiments in which three proof construction environments were compared in the context of a course on introductory logic taught entirely on-line by VALID. In each course unit, VALID introduces students to concepts in logic and then requires them to complete a series of proof construction exercises before beginning the next unit. Since VALID is fully uniform and impervious to who sits before it, and since it handles virtually the entire teaching duties for the course, it presents a perfect platform upon which to perform controlled experiments.

We gave all our VALID students a pretest for "logical aptitude" (designed by the Education Testing Service). We used the results of the test to split the class into three matched groups. Each group used VALID to learn logical concepts, but used a separate version of PT (the PROOF TUTOR) to complete all of the sentential proof construction problems in VALID's curriculum. The first group used a version of PT that simulated standard proof checking programs, i.e., the student was only allowed to work forwards from the premises toward the conclusion, and the proofs were represented as columns of lines with a dependency field. Call this group Forwards-only. The second group used a version of PT in which students had a sophisticated graphical display representing their search for a proof and their paritally completed proof. However, they were only allowed to work backwards from the conclusion toward the premises. Call this group Back-only. The third group had the ability to work forwards or backwards and had the sophisticated display. No group received intelligent help from PT. Thus all groups used identical computer environments save the differences described above. Everyone took the same online midterm with the version of PT that they had used throughout. There were eight problems on the test of the same sort they had faced in the regular course. Two problems were quite easy, three of medium difficulty, and three fairly hard. Below we list the results.

Group	Pretest	Midterm			
		Total	Easy	Med.	Hard
PT-full (11)	66.06	80.63	95.4	81.8	70.0
Back-only (9)	76.27	76.37	100.0	92.7	44.4
Forwards-only (8)	68.33	68.75	87.5	79.1	45.8

All results are group means, expressed as percentages. The sample size of each group is in parentheses. The pretest means are not identical due to students who dropped the course after the groups were created.

It is clear that the group that worked in a standard proof checking environment (forwards only) did the worst. On the easy and medium problems, they did the worst, but were within the general range of the other two groups. The difference on the hard problems is dramatic, however. It seems, the full version of PT significantly improved the students' ability to solve problems that weren't either immediately obvious or almost so.