DOCUMENT RESUME

ED 059 894 SE 013 378

AUTHOR	Kreisel, Georg
TITLE	Five Notes on the Application of Proof Theory to Computer Science.
INSTITUTION	Stanford Univ., Calif. Inst. for Mathematical Studies in Social Science.
REPORT NO	TR-182
PUB DATE	Dec 71
NOTE	52p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	*Artificial Intelligence; *Computer Science; Deductive Methods; *Logic; Logical Thinking; Mathematics; Philosophy

ABSTRACT

The primary aim of these five technical papers is to indicate aspects of proof theory which may be of use in the study of non-numerical computing. The three main papers are entitled: "Checking of Computer Programs;" "Consistency Proofs and Programs for Translators;" and "Experiments with Computers on the Complexity of Non-numerical computations." The author shows that many theorems on computability in traditional metamathematics are of little use to the computer scientist because they do not lead to feasible algorithms. He also suggests alternative approaches to proof theory which would be of greater applicability. (MM)

N-AI
EM

FIVE NOTES ON THE APPLICATION OF PROOF THEORY TO COMPUTER SCIENCE

BY

GEORG KREISEL

TECHNICAL REPORT NO. 182

DECEMBER 10, 1971

PSYCHOLOGY & EDUCATION SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

TECHNICAL REPORTS

PSYCHOLOGY SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

(Place of publication shown in parentheses; if published title is different from title of Technical Report, this is also shown in parentheses.)

(For reports no. 1 - 44, see Technical Report no. 125.)

50 R. C. Atkinson and R. C. Calfee. Mathematical learning theory. January 2, 1963. (In B. B. Wolman (Ed.), Scientific Psychology. New York: Basic Books, Inc., 1965. Pp. 254-275)

51 P. Suppes, E. Crothers, and R. Weir. Application of mathematical learning theory and linguistic analysis to vowel phoneme matching in Russian words. December 28, 1962.

52 R. C. Atkinson, R. Calfee, G. Sommer, W. Jeffrey and R. Shoemaker. A test of three models for stimulus compounding with children. January 29, 1963. (J. exp. Psychol., 1964, 67, 52-58)

53 E. Crothers. General Markov models for learning with inter-trial forgetting. April 8, 1963.

54 J. L. Myers and R. C. Atkinson. Choice behavior and reward structure. May 24, 1963. (Journal math. Psychol., 1964, 1, 170-203)

55 R. E. Robinson. A set-theoretical approach to empirical meaningfulness of measurement statements. June 10, 1963.

56 E. Crothers, R. Weir and P. Palmer. The role of transcription in the learning of the orthographic representations of Russian sounds. June 17,. 1963.

57 P. Suppes. Problems of optimization in learning a list of simple items. July 22, 1963. (In Maynard W. Shelly, II and Glenn L. Bryan (Eds.), Human Judgments and Optimality. New York: Wiley. 1964. Pp. 116-126)

58 R. C. Atkinson and E. J. Crothers. Theoretical note: all-or-none learning and intertrial forgetting. July 24, 1963.

59 R. C. Calfee. Long-term behavior of rats under probabilistic reinforcement schedules. October 1, 1963.

60 R. C. Atkinson and E. J. Crothers. Tests of acquisition and retention, axioms for paired-associate learning. October 25, 1963. (A comparison of paired-associate learning models having different acquisition and retention axioms, J. math. Psychol., 1964, 1, 285-315)

61 W. J. McGill and J. Gibbon. The general-gamma distribution and reaction times. November 20, 1963. (J. math. Psychol., 1965, 2, 1-18)

62 M. F. Norman. Incremental learning on random trials. December 9, 1963. (J. math. Psychol., 1964, 1, 336-351)

63 P. Suppes. The development of mathematical concepts in children. February 25, 1964. (On the behavioral foundations of mathematical concepts. Monographs of the Society for Research in Child Development, 1965, 30, 60-96)

64 P. Suppes. Mathematical concept formation in children. April 10, 1964. (Amer. Psychologist, 1966, 21, 139-150)

65 R. C. Calfee, R. C. Atkinson, and T. Shelton, Jr. Mathematical models for verbal learning. August 21, 1964. (In N. Wiener and J. P. Schoda (Eds.), Cybernetics of the Nervous System: Progress in Brain Research. Amsterdam, The Netherlands: Elsevier Publishing Co., 1965. Pp. 333-349)

66 L. Keller, M. Cole, C. J. Burke, and W. K. Estes. Paired associate learning with differential rewards. August 20, 1964. (Reward and information values of trial outcomes in paired associate learning. (Psychol. Monogr., 1965, 79, 1-21)

67 M. F. Norman. A probabilistic model for free-responding. December 14, 1964.

68 W. K. Estes and H. A. Taylor. Visual detection in relation to display size and redundancy of critical elements. January 25, 1965, Revised 7-1-65. (Perception and Psychophysics, 1966, 1, 9-16)

69 P. Suppes and J. Donio. Foundations of stimulus-sampling theory for continuous-time processes. February 9, 1965. (J. math. Psychol., 1967, 4, 202-225)

70 R. C. Atkinson and R. A. Kinchla. A learning model for forced-choice detection experiments. February 10, 1965. (Br. J. math stat. Psychol., 1965, 18, 184-206)

71 E. J. Crothers. Presentation orders for items from different categories. March 10, 1965.

72 P. Suppes, G. Groen, and M. Schlag-Rey. Some models for response latency in paired-associates learning. May 5, 1965. (J. math. Psychol., 1966, 3, 99-128).

73 M. V. Levine. The generalization function in the probability learning experiment. June 3, 1965.

74 D. Hansen and T. S. Rodgers. An exploration of psycholinguistic units in initial reading. July 6, 1965.

75 B. C. Arnold. A correlated urn-scheme for a continuum of responses. July 20, 1965.

76 C. Izawa and W. K. Estes. Reinforcement-test sequences in paired-associate learning. August 1, 1965. (Psychol. Reports, 1966, 18, 879-919)

77 S. L. Blehart. Pattern discrimination learning with Rhesus monkeys. September 1, 1965. (Psychol. Reports, 1966, 19, 311-324)

78 J. L. Phillips and R. C. Atkinson. The effects of display size on short-term memory. August 31, 1965.

79 R. C. Atkinson and R. M. Shiffrin. Mathematical models for memory and learning. September 20, 1965.

80 P. Suppes. The psychological foundations of mathematics. October 25, 1965. (Colloques Internationaux du Centre National de la Recherche Scientifique. Editions du Centre National de la Recherche Scientifique. Paris: 1967. Pp. 213-242)

81 P. Suppes. Computer-assisted instruction in the schools: potentialities, problems, prospects. October 29, 1965.

82 R. A. Kinchla, J. Townsend, J. Yellott, Jr., and R. C. Atkinson. Influence of correlated visual cues on auditory signal detection. November 2, 1965. (Perception and Psychophysics, 1966, 1, 67-73)

83 P. Suppes, M. Jerman, and G. Groen. Arithmetic drills and review on a computer-based teletype. November 5, 1965. (Arithmetic Teacher, April 1966, 303-309.

84 P. Suppes and L. Hyman. Concept learning with non-verbal geometrical stimuli. November 15, 1968.

85 P. Holland. A variation on the minimum chi-square test. (J. math. Psychol., 1967, 3, 377-413).

86 P. Suppes. Accelerated program in elementary-school mathematics -- the second year. November 22, 1965. (Psychology in the Schools, 1966, 3, 294-307)

87 P. Lorenzen and F. Binford. Logic as a dialogical game. November 29, 1965.

88 L. Keller, W. J. Thomson, J. R. Tweedy, and R. C. Atkinson. The effects of reinforcement interval on the acquisition of paired-associate responses. December 10, 1965. (J. exp. Psychol., 1967, 73, 268-277)

89 J. I. Yellott, Jr. Some effects on noncontingent success in human probability learning. December 15, 1965.

90 P. Suppes and G. Groen. Some counting models for first-grade performance data on simple addition facts. January 14, 1966. (In J. M. Scandura (Ed.), Research in Mathematics Education. Washington, D. C.: NCTM, 1967. Pp. 35-43.

91 P. Suppes. Information processing and choice behavior. January 31, 1966.

92 G. Groen and R. C. Atkinson. Models for optimizing the learning process. February 11, 1966. (Psychol. Bulletin, 1966, 66, 309-320)

93 R. C. Atkinson and D. Hansen. Computer-assisted instruction in initial reading: Stanford project. March 17, 1966. (Reading Research Quarterly, 1966, 2, 5-25)

94 P. Suppes. Probabilistic inference and the concept of total evidence. March 23, 1966. (In J. Hintikka and P. Suppes (Eds.), Aspects of Inductive Logic. Amsterdam: North-Holland Publishing Co., 1966. Pp. 49-65.

95 P. Suppes. The axiomatic method in high-school mathematics. April 12, 1966. (The Role of Axiomatics and Problem Solving in Mathematics. The Conference Board of the Mathematical Sciences, Washington, D. C. Ginn and Co., 1966. Pp. 69-76.

(Continued on inside back cover)

FIVE NOTES ON THE APPLICATION OF PROOF THEORY TO COMPUTER SCIENCE

by

Georg Kreisel

TECHNICAL REPORT NO. 182

December 10, 1971

PSYCHOLOGY AND EDUCATION SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

Table of Contents

Five Notes on the Application of Proof Theory to Computer Science[1]

Georg Kreisel

Institute for Mathematical Studies in the Social Sciences
Stanford University

Introduction.[2] Let me begin with a manifesto. As will be evident throughout these notes my background is in proof theory, not in computer science. However--and this fact might easily confuse the reader--my interests here are almost diametrically opposite to those of most logicians who publish on computer science. There will be little general mathematics; and what there is, will always be subordinate to the applications, as I understand them. I realize quite well that professional computer scientists (that is, people paid by computer science departments or affluent firms) are often as much attracted by beautiful generalities about computation as my fellow logicians. Perhaps they can afford it because, through experience, they have developed judgment on the relevance of these generalities. Evidently I cannot hope to have this kind of judgment. Be that as it may, let me set out here theoretical reasons for the philosophy followed in these notes.

We know of course that there is something common to proof theory and computer science: the formal rules studied in proof theory and the computation rules, both for numerical and non-numerical computation, studied in computer science are mechanical; indeed, the basic 'grand' idea behind formalization was the mechanization of mathematical reasoning, long before an even approximate physical realization of this idea by electronic machines could be proposed. Turing's analysis of the notion of mechanical rule in terms of an (idealized) machine led to recursion theory, the mathematical theory of mechanical rules or, at least, of functions definable by such rules.[3] Consequently, general results 'about' mechanical rules will be applicable both to formal rules (in proof theory) and computational rules. In particular, this applies to well-known recursive unsolvability results.

But no subject (except possibly some mathematics) lives on negative results. They indicate what not to do. This is the content of the slogans: Proof theory begins where recursion theory ends, and similarly for computer science. More soberly, by picking out *specific*, 'narrow' subclasses from among all mechanical rules we shall find those that are relevant to proof theory and those (others) relevant to computer science. For example, in proof theory the significant results are about *specific* formal systems. The brutal well-known ones, so-called Frege-Hilbert style axiomatic systems, are picked out by the mathematical content of the so-called non-logical symbols in the axioms (relation, function symbols); the sophisticated ones, discovered by Gentzen and Prawitz, are picked out by an operational meaning assigned to the logical symbols. In computer science there are additional conditions on *feasibility* (complexity of programs, length of computation). It stands to reason that for those narrower classes of rules more significant positive results are possible. To use slightly hysterical language: if there simply aren't general positive results (to be found for the class of all recursive functions), there just isn't a possibility of a 'deep' study of the general case.

The other extreme--or, better, error--that has to be avoided is sometimes expressed by saying that 'computation is an art'. What is meant, in terms of the preceding paragraph, is that there are no *rational or theoretical principles for choosing relevant classes of rules* (formal systems, respectively computation rules).[4] I believe that experience in proof theory may be of heuristic if indirect value here. The early hope in proof theory was that we could use a partial but very *simple* criterion for choosing formal systems, namely, their *completeness for the intended interpretation*. For example, when Artin and Schreier produced their axioms for real closed fields (which seemed adequate for current algebra) their successful choice appeared like magic. Tarski's discovery that these axioms produced precisely those (first order) statements which are true for the field of real numbers, gave a truly satisfactory *explanation* of the choice made by Artin and Schreier. (I do not know if Tarski emphasized this aspect.) It is an explanation because each theorem of current algebra was expected to be valid for the reals. Now Gödel's incompleteness theorem shows that *the simple criterion of completeness cannot be used for the choice of formal systems for arithmetic, analysis, set theory*, etc. This does not at all mean that there is *no* principle of choice; only it will have to be subtler than mere completeness. Besides, even when, as in the case of first order classical predicate calculus, we have formal systems complete w.r.t. validity, there are many different axiomatizations (as already mentioned, completeness makes only a partial, in fact, rather brutal choice). The further choices needed, and made by Gentzen or Prawitz, are just of a quite different order of sophistication. Also, the discovery of significant 'subsystems' of analysis in the literature is incomparably more sophisticated than, say, that of (complete) systems for dense orderings (I use the word 'subsystem' current in the literature; actually *every* formal system is a subsystem of the set of true statements if the formal language contains arithmetic).

Needless to say it is not claimed that the same considerations are relevant to the choices of both formal systems and computation rules. Quite simply: at least in the most articulate parts of proof theory the choices are made on *epistemological* grounds, that is, according to the kind of understanding needed to recognize the validity of *principles* (axioms or rules); the complexity of iterating these principles is rarely a principal object of research. In computer science the complexity is paramount. So one of the main problems will be to *find* the parts of proof theory that are useful to computer science. The particular examples mentioned in the last paragraph are illustrations only; not only of difficulties (of finding formal rules and of judging the choice) but also of solutions to these difficulties, at least in *some* cases.[5]

It should also be remembered that, quite apart from the philosophical aims mentioned, proof theory often looked for systems fairly close to our ordinary reasoning. This suggests a further reason why quite different principles of choice are needed, simply on the assumption that human reasoning (or, in crazy language, 'information processing by the brain') has a different structure from mechanical computers: for example, very mechanical instructions confuse the human computer. The practical effects of this simple observation are far reaching: The mere fact that *we* find it easy to solve some class of problems in, say, propositional calculus is no reason at all for supposing that they are easy for a computer; or, more formally, that there *exists* a short program. So it is not even easy to judge whether we are stupid when we are not able to mechanize reasoning which is easy for us. In fact, Section 4 contains proposals for *studying* such questions. The principle adopted is the simplest in the world: *The single most surprising innovation of logical foundations is the reduction of our mathematical vocabulary*: *Does this destroy the practical possibility of reasoning*?

Finally, a word on terminology. I shall not use the jargon of computer science (and shall try to avoid most proof theoretic jargon, except in the title of Section 3 which uses both). First of all, I don't know it well enough--Fremdwörter sind Glücksache. Secondly, perhaps in order to compensate for the 'inhumanly' systematic character of computer languages, computer jargon is full of quite different words for distinct, but closely related ideas, like compiler and translator. I should prefer a civilized language with a word for the common idea (translator) and adjectives to qualify it.[6] Thirdly, some terms suggest, it seems to me, a sensational but ill considered aim, for example, 'theorem proving'. (For those who know the catechism, an outward and visible sign of an inward and invisible disgrace.) I shall use 'non-numerical computing' or the name of the *particular* area where I believe such computing to be appropriate, for example, in the currently active subject of mechanical checking of programs (Section 2). Evidently there is nothing sensational in the idea of a 'mere' checking of a *computer* program, much less so than in checking mathematical proofs which are prima facie (and probably intrinsically) non-mechanical. But, by being less sensational the idea is perhaps also more feasible.

I. *Basic Notions and Distinctions*

NB. According to temperament the reader can either glance through this note before reading the others or simply go back to it when one of the basic notions or distinctions is actually needed.

I shall consider here principally (computation and proof theory in connection with) *arithmetic problems*, and discuss afterwards modifications needed in other contexts. Since we have 2000 years of experience in numerical computation it should be easy to judge whether our general considerations are truly applicable (and not only aesthetically or theoretically pleasing).

1. *General orientation*. To describe the relation between the aims of computer science and proof theory briefly, one can use the slogans:

Theory (or principles) of formalist mathematics

versus

Formalist theory (or reduction) of mathematics.

Computer science has as its objects (formal) calculations and tries to develop a theory of these objects. The intended *meaning* of, say, an equation $a = b$ *is* that the formal objects a and b have been or can be reduced, by given formal rules, to the same term (their formal value). In contrast, proof theory, as originally understood, tries to establish that, for a *different* (mathematical) meaning of the formula $a = b$,

$a = b$ is true for that meaning if (and, possibly, only if)
a and b have the same value.

Example. Suppose we are given the formal rules for *addition* in arithmetic, with the symbol 0 and the successor symbol s: $a + 0 \to a$, $a + sb \to s(a + b)$. Then $0 + s0 = (s0) + 0$ means[7] that each side can be reduced to the same term, namely $s0$ (by using the second and first rule resp.).

The expressions $a + b$ and $b + a$ with variables a and b can evidently not be reduced to the same term by means of the rules above. Suppose, however, we *interpret*[7]

(*) $$a + b = b + a$$

to mean: for each pair of *numerical* terms (0,s0,ss0,...) n and m, n + m and m + n have the same formal value for the rules of addition above. Then, for example by the well-known Presburger-algorithm, there *are* formal rules generating all equations which are true *about* computations (in the sense above); but it is *not* enough to use the axioms: $a + 0 = a$, $a + sb = s(a + b)$ 'corresponding' to the rules of addition.

In the modern proof theoretic literature (for example, under the slogan *operative Logik* or operational logic) there is a great deal of material which is *not* directly relevant to the original aims of proof theory, but rather provides a *theory of formal rules*; it extends the interpretation (*) of equations to logically complicated formulae. The theory does not deal with special properties of *computation rules* actually used in, say, arithmetic, but with general *Post-rules*. (Evidently, the rules for numerical addition are set out as Post rules.)

This theory is coherent and has great aesthetic appeal. I have no idea of its usefulness to computer science, simply because it is not clear *whether computationally interesting properties of* (computation) *rules are expressed by the logically complicated formulae used* (in contrast to commutativity in the example above, a property of addition which we use all the time, e.g., when checking the addition of columns in different order).

The theory is often presented, particularly by Curry, in connection with a formalist reduction of mathematics. As a result it is usually (i) not taken seriously by those who find such a reduction implausible (though the theory is not at all needed for such a reduction) and (ii) not examined for its interest *qua* theory of formal rules (though, if it has such an interest, this would be independent of formalist doctrine; after all, *some* mathematics we do, unquestionably has formalist character, for example, computing). Some defects of the notions used in that theory will be apparent from the next paragraph.

2. *Computations*. Anyone who has read this far, has no doubt a pretty good idea of what a *computation rule* or *computation diagram* is (for terms in some language L, the diagram takes the form $t_i \to t_i'$, $1 \le i \le N$, t_i containing 'meta' variables). There are a relatively small number

of useful *notions about* and, consequently, *distinctions between* various kinds of computation diagrams which are scattered in the literature and perhaps not even explicit.

(a) *Numerical and non-numerical computation* (of numerical valued terms) or: the *choice* of computation diagram. In computations we normally use well defined functions and rules which give a *numerical value*, that is, a value 0, s0, ss0, ..., or, more usually, a value in *decimal* or *binary* notation. In short there *is* a *standard* or *normal* form. This fact is used a good deal in the theoretical literature.

It should be noted that the use of normal forms is, in practice if not in theory, *in conflict with the practical requirement* that answers to (numerical) problems must be *comprehensible*. Thus while

$$10^{1000}$$

or even: the largest prime $< 10^{1000}$
are comprehensible answers, neither of the standard representations above would do.

More formally, practical computation diagrams 'for' the *exponential function* or for the so-called *bounded least number operator* (used in the two examples above) must not permit further reduction; in other words, though normal forms exist they must not be used. But it seems difficult at this stage to be much more specific about an efficient choice of normal forms; cf. also III §2c and III §3a. The reader should note that the *choice of language*, stressed throughout these notes in connection with proofs ('automatic theorem proving'), is also significant for computation.

Technical remark. If one wants to, one can easily state some precise 'general' theorems here. Given any of the usual *codes* for partial recursive functions, we consider non-numerical terms, built up from these codes, for computing *numbers* (for 'functions' with zero arguments!); for example, by use of Kleene's bracket notation. In this way *every recursive function* f *is represented by a rudimentary function from numerals* n (not to numerals but) *to codes of the value of* f(n). However, there is no evidence that this particular *general* scheme of using non-numerical terms is also efficient.[8]

(b) Consistency of computation rules w.r.t. (a congruence relation between) normal forms. For any term t, if t can be reduced to the normal (irreducible) terms t_1 and t_2 then t_1 and t_2 must be congruent.

Examples. One familiar class of examples comes from computation rules which are (i) valid for, say, an arithmetic interpretation, that is, there are functions which satisfy the given rules in the sense of footnote 7 and (ii) non-congruent normal terms denote distinct objects in the interpretation. Outside arithmetic, a well-known example is provided by combinatory logic. (This may be relevant to computer science because Curry's explicit intention was to analyze the basic steps in human mathematical reasoning in terms of combinators.. Even if this intention is absurd, the possibility remains that this analysis applies to formal operations which Curry assumed to constitute mathematical reasoning.[9])

Warning. In the light of (a), consistency (in the present and current sense!) is not a necessary condition for a computationally adequate diagram.

(c) Computability and strong computability (w.r.t. the notion of normal, that is, irreducible form determined by the computation diagram itself). Some, resp. every sequence of applications of the computation rules to any term t stops after a finite number of steps; that is, it reaches a term to which no further reduction can be applied.

Evidently, rules can be perfectly valid in the sense of b(i) without possessing even the (weak) computability property; for example, in case of addition, we may have added 'by mistake': $a \to a + 0$.

(d) Hilbert-Post completeness (of the congruence relation between normal forms). If, for two non-congruent normal terms t_1 and t_2, the rules $t_1 \to t_2$ and $t_2 \to t_1$ are added to the computation diagram then all terms become interreducible.

This is a specialization of the traditional notion of Hilbert-Post completeness for arbitrary formal systems: if a non-derivable formula is added to the formal rules then all formulae become derivable. (In the case of computation rules the only relevant formulae are equations.)

Discussion. The idea behind the Hilbert-Post notion is this: to establish *maximality* of the rules considered with a *minimum of assumptions about the intended meaning* (or, as formalists like to say, 'uses') *of the formal rules*. As is well known, most familiar H-P complete systems satisfy--like classical propositional calculus--the abstractly stronger conditions: For each formula F there is a substitution instance F_1 for which either F_1 or $\neg F_1$ is derivable. Also, quite often, we simply know enough about the intended meaning, e.g., of classical predicate (or, for that matter, propositional!) calculus to know that we have enough formal rules even if they are (or were) not H-P complete. Similarly, in view of (a), we shall not, in general, expect computation diagrams to be H-P complete *if we have chosen them after careful analysis of practical requirements*.

3. (Formal) *deductions*. Once again (as in §2), the reader may be assumed to know what a formal system or, equivalently, a system of deduction rules is.[10] As in §2, we may distinguish between the problems connected with (analyzing) *given* formal systems and with *choosing* formal systems; cf. §2a concerning the choice of computation rules and even of the concept of normal form. So much is clear.

Now there is a further element of choice, namely, the choice in the order of applying (computation and deduction) *rules*. Certainly not all computation rules need be deterministic; if we want to compute $0 \cdot t$, for a complex term t, we may either first work out the numerical value of t and apply the recursion rules for multiplication or we may apply the rule $0 \cdot a \to 0$ (if the latter is included; the recursion rules will always be included). Again, if we have a rule for computing a function of two arguments there is a choice in the order in which the arguments are treated. But it may be expected that *within such a well-organized framework as a system of computation rules, this freedom of choice causes little havoc*. In contrast, as we know from experience in mathematics, being given formal deduction rules is of little help. (In the analyses below of the 'help' involved the lettering of subsections does *not* correspond to §2.)

(a) *Looking at deductions and searching for deductions* (in a given formal system). Without wishing to be 'mathematical' in an affected sort of way, I'll begin by setting out the distinction in the form of two kinds of computation diagrams.

Evidently, since *operations on deductions* are involved, the objects or 'terms' of the computation diagram will be (whole) deductions, not, say, derived formulae.

Case 1 (the *familiar* set-up). This is relevant to a *search* for deductions. We start with: $d \rightarrow d'$ if (the deduction figure) d' is obtained from d by adding a formula which is an *immediate* consequence of formulae in d; 'immediate' by application of the given rules.

There is no end to this procedure, and an *independent* criterion has to be added to create a semblance of a computation diagram with its irreducible or normal forms. What is indeed implicit in our ordinary way of thinking is that we wish to decide some formula A: Writing $\underset{A}{\rightarrow}$, we add the *restriction* $d \underset{A}{\rightarrow} d'$ *only if* $d \rightarrow d'$ and d contains neither the formula A nor the formula $\neg A$ (because in either case the search is over).

Note that the questions of §2a in connection with a *choice* of irreducible forms have a parallel here. Specifically, d need not contain the formula A itself but say a formula A' which is propositionally equivalent to A. This induces a *congruence relation* between 'normal' forms, that is, derivations containing A' or $(\neg A)'$. The concepts of §2(b)-(d), with 'computability' replaced by 'formal decidability (of A)', are self-explanatory. What makes formal rules of 'little help' is that *for the usual formal rules* (with modus ponens and without 'strategies', that is, additional rules of choice) *we do not have strong computability*. This is a good negative criterion; but of course strong computability is not enough practically if the execution of the computation is not practical.

Case 2 (the *sophisticated* set-up). This is relevant to the *analysis* of (given) deductions. According to previous training the reader may think here of *cut-elimination* or *elimination of redundancies* where, in aptly-named natural deductions, an introduction (of a logical operation) is followed immediately by its elimination. In both cases we shall speak of a *normalization step*. The computation diagram runs as follows:

$d \to d'$ if d' is obtained from d by applying a single reduction step.

Now, in contrast to *Case* 1, d can be irreducible, namely, if it is in normal form. Note here, for the normalization steps mentioned, d and d' automatically have the same end formula. The concepts of §2(b)-(d) distinguish between different *normalization procedures*. For example, it is open whether, in familiar systems, cut-elimination rules are *strongly* computable, even when the allegedly 'equivalent' natural deduction rules are.[11] Naturally the terminology, e.g., of 'consistency' in §2b, is meaningful only if the choice of the normalization procedures themselves is intended to respect some preassigned *meaning* of the normal deductions. The terminology is justified if non-congruent deductions are intended to express different proofs (in the ordinary sense of 'difference of proof' as used, e.g., in arguments about plagiarism) and normalization steps are supposed to preserve identity of proofs.

(b) *Conflicting requirements*. Quite naively, it would be surprising that the same criterion, for example, the selection of *normal* derivation, should be useful for both the analysis of given deductions and for the discovery of new deductions of some given formula. Perhaps one should not ignore the ridiculous-*sounding* advice of mathematicians (who are, after all, well known to be terribly inarticulate outside their own domain) such as: *one must not be too logical in research*. As in footnote 4 of the introduction, precise and merely logically satisfactory formulations may be much worse than no formulation at all because their defect is likely to be not obvious irrelevance but something subtler.

Example. In the discussion of *Case* 1 in (a) above, failure of strong computability of the usual formal rules was mentioned; they are wildly indeterminate. On the other hand, normal derivation rules or the rules introduced by Herbrand are essentially deterministic: a formula determines very closely from which premise it must be deduced if it has a normal derivation at all. But as already mentioned, this is no reason at all for supposing that normal derivation rules (which were introduced for the analysis of given proofs) should also provide *efficient*[12] deterministic search procedures.

The practical conclusion which I have drawn from this observation and which I apply in Notes 2 and 3 is to look for areas where the analysis of given deductions is relevant.

(c) Digression on search procedures (to be taken up in Note 4). The immediate purpose of this digression is to serve as a kind of mental hygiene. There is a natural, quite traditional aim, to find a theory of mathematical reasoning. Through formalization it is possible to restrict this (evidently hopelessly) general aim to, say, reasoning within some limited context such as predicate logic. In addition we have some isolated quite striking examples. For example, the logical theorems at the beginning of *Principia* fall into simple decidable classes for which, as observed by Wang, there is a quite effective mechanization. True, these theorems do not at all have the look of typical mathematical theorems. But as long as no alternatives are set out, there is a lurking doubt: Shouldn't something similar be possible for much wider and more interesting areas? It may well be objected that, even *if*, objectively, such a possibility exists, this is no good reason why any particular person should pursue it; he may not have the necessary talent or he may have more talent in another area. But it seems to me reasonable to present some alternatives which could remove the doubts without the appeal to self-analysis made above.

We are able to prove or refute quite a large number of formulae F in predicate logic with great ease. Often we are not aware of having followed a method.

The lurking doubt is:

There ought to be a mechanical method, applicable to a large class C of formulae of predicate logic, which will also decide the formulae in F in a reasonable time (where the greater speed of electronic computers is to make up for the subroutines supposedly stored by the human computer). Usually one makes the additional assumption that the class C is characterized by such syntactic conditions as prefix complexity or the number of arguments of the relation symbols used.

Proposal. Granted that, in simple cases, we are not aware of following a method, let's look at the *nature* of the methods which are used when we *do*!

The very first method that strikes one when inspecting the facts is the extension of the language. (Indeed, no inspection is needed, because the principal step taken in the formalization of ordinary intelligible reasoning was a reduction of the language!) Here we have two quite different applications, which are both logically insignificant, but in different senses, as made precise below.

Definitional extensions. We consider consequences from axioms $\textcircled{A}$, say for real closed fields. We extend the language by a binary relation symbol O (to stand for 'order') and derive a number of implications of the form $A_O \to B_O$ where A_O are axioms for an ordering. We make the definitional extension

$$\forall x \forall y[O(x,y) \leftrightarrow \exists z(y = x + z^2)]$$

and derive from $\textcircled{A}$ the formula A_O. Each B_O is then, demonstrably equivalent (on the basis of $\textcircled{A}$) to a formula in the language of fields.

These extensions are logically insignificant in the sense that (not only in the present case, but under very general conditions) the introduction of the symbols can be eliminated. But: at what cost? for example, in length of derivation? The example is by no means farfetched: it is simply the formal counterpart to saying that our proofs are guided by the meaning of the relation $\exists z(y = x + z^2)$. Amusingly, electronic computers furnish an excellent means for answering this question of cost! Cf. Note 4.

Introducing abstract concepts (in the sense of higher type or set theory). We can use the language of the previous example, and consider ordered fields, i.e., the axioms A_O and the axioms for fields (among $\textcircled{A}$). To derive B_O we may proceed as follows. We use the fact that every ordered (ground) field has a real closure. This is a set theoretic statement. We can now apply set theoretic constructions to this real closure and then deduce results about the ground field. As a quite trivial illustration consider a simple formula of predicate logic. Writing $<$ for O and p for the general polynomial of degree n.

$$A_< \to \forall x_1 \ldots \forall x_{n+1}([\bigwedge_{1 \leq i \leq n} (x_i < x_{i+1} \wedge px_i \cdot px_{i+1} < 0)] \to 0 < px_n \cdot px_{n+1}) .$$

A set theoretic proof proceeds by embedding the ground field in its real closure where $px_i \cdot px_{i+1} < 0 \rightarrow \exists z(x_i < z < x_{i+1} \wedge pz = 0)$ and using the fact that a polynomial of degree n has at most $\underline{n}$ zeros.

In contrast to the first example, the present use of set theoretic concepts is logically insignificant (in the sense of being eliminable) for the quite special reason that first order predicate calculus is complete. Of course the theorem above has also a first order proof, in fact a quite simple one. We need only use Lagrange's formula

$$p(x_{n+1}) = \sum_{1 \leq i \leq n} p(x_i) \prod_{j \neq i} \left(\frac{x_{n+1} - x_j}{x_i - x_j} \right) .$$

and observe that

$$\prod_{j \neq k} \left(\frac{x_{n+1} - x_j}{x_k - x_j} \right)$$

has different signs when $k = i$ and when $k = i + 1$. But it is a <u>subject for research to see whether in general the use of set theoretic methods is decisive for feasibility</u>, that is, for effective execution in time and space by an electronic computer or for intelligibility by a human one. (I have in mind mathematically or logically intelligible theorems, not the kind of *ad hoc* formulae usually constructed in the literature on 'speed-up'.)

Preview. As the reader will see, only Notes 2 and 3 contain genuine applications of proof theory. Both analyze *given* deductions, illustrating the two important directions in proof theory, the methods of normalization and interpretation resp. In Note 4 we experiment on search procedures, using facts from proof theory merely as a *guide* for the design of experiments.

II. Checking of Computer Programs: An Example of Non-numerical Computation

We shall consider principally, but not exclusively, programs for solving some arithmetic problem with a parameter, n say. In terms of I §2(c) and I §3(a), the formal counterpart to the informal distinction between checking and proving (the validity of) a program is this:

Checking a program is a decidable procedure, by use of strongly computable rules; a check provides indeed a proof, by very elementary rules, of

(*) the function defined by the program solves the problem for all n.

In contrast, for the usual formal rules of inference, it is not (recursively) decidable whether or not a universal statement of the form (*) is derivable. This corresponds to the meaning of 'checking' as a procedure which does not require ingenuity (in contrast to 'cross checking' which does, but which need not constitute a complete check).

It will be best to begin with a closer look at the familiar ideas of 'arithmetic problem' and 'solution', including the choice of language used to present the solution; cf. I § 2(a). The reader should not forget for one moment that, in the context of checking, the program and our knowledge of its properties are the principal objects of study; we are not merely concerned with the sequence of states of the machine which the program, objectively, determines; and *a fortiori*, not merely with the graph of the function which the machine puts out; cf. footnote 3 of the Introduction.

1. Numerical analysis and programmation. The general form of an arithmetic problem (which we consider here) is

$$\exists m A(n,m) \text{ with parameter } n ,$$

where the relation A may be defined by use of quite abstract concepts; for example, it may say that m is (a code for) an approximation, to accuracy $1/n$, to the solution, at the origin say, of some differential equation in Hilbert space.

Numerical analysis, by use of proofs not merely by mechanical checking, provides more or less explicit solutions in terms of a function f (of n)

determined in familiar terms, for example, 2^n or $[\log n]$ (the integral part of log n). One sometimes says that numerical analysis provides an *algorithm*; but, strictly speaking, the algorithm is only tacitly understood because *in numerical estimates not the procedure of calculation but only the values of the function are relevant*. In fact, numerical analysis provides in general not an algorithm but rather a *simple functional equation* for f which *suggests* an algorithm, e.g., in the case of 2^n

$$\forall p[f(0) = 1 \wedge f(p + 1) = 2f(p)] .$$

Numerical analysis has done its job when it finds such a functional property or equation $\Phi(f)$ and a proof of

$$\Phi(f) \rightarrow A(n, fn) .$$

(In §3c below we give some formal conditions ensuring that Φ *does* determine an algorithm.)

Programmation consists in making explicit, in mechanical terms, the algorithm 'suggested' by the property Φ. Roughly speaking, *checking* a proposed program π_f consists in establishing by the elementary methods implicit in the idea of checking that π_f satisfies Φ. Our problem here consists in making explicit what these elementary methods are. This involves of course some general conjecture on the type of properties Φ which occur in practice (but not on the type of relation A!).

The following example does not merely illustrate the difference between numerical analysis and programmation, but gives a good idea of the *qualitative* difference between a recursive and a feasible solution, discussed at length at the end of Note 1.

Instead of the single variable n, we consider the quadruples n_1, n_2, n_3 and n' and the relation

$$(m = 0 \;\&\; n_1^{2+n'} + n_2^{2+n'} = n_3^{2+n'}) \vee (m = 1 \;\&\; n_1^{2+n'} + n_2^{2+n'} \neq n_3^{2+n'}) .$$

Without numerical analysis, we have the obvious program of 'simply evaluating'

$$n_1^{2+n'} + n_2^{2+n'}, \text{ and } n_3^{2+n'}$$

for given positive integers n_1, n_2, n_3 and n' and 'comparing' them. The program must translate these instructions to the machine. *With* numerical analysis, that is a (partial or complete) proof of Fermat's conjecture, we should use a totally different program (for the same function of n_1, n_2, n_3, n'!), namely

$$n_1, n_2, n_3, n' \to 1 .$$

Amusingly, on the basis of current knowledge,[13] as far as *existing* computing machinery is concerned, *only the second program is feasible* (for *numerical* computing): for those numbers for which Fermat's conjecture is open at least one of the numbers $n_1^{2+n'}$, $n_2^{2+n'}$ and $n_3^{2+n'}$ is so large that it could not be stored in any machine's memory at all! A much more sophisticated example will be given in IV §2b.

2. *Formalization of the concept of 'checking': definitional equality* (brutal approach). To start with, I want to convey an *idea* for such a formalization which a programmer, relying on his judgment and experience, may occasionally decide to use. It is evident that any application will depend on the *programming language*, that is, what precisely the symbol π stands for in §1, and the possibly *non-numerical* terms (cf. I §2a) chosen to present the (numerical valued) solution in the particular application. I should try to refer the formalization to theories of programming languages or of (non-numerical) notation systems for natural numbers if I were convinced that any known theory is even remotely correct. As it is, the chance of introducing a *systematic* error by relying on current theories seems more damaging than the *vagueness* which results from giving only an illustration (as I do below). In the next section I shall try to discuss limitations.

Proposal. Assume that the critical properties Φ of §1 supplied by numerical analysis are conjunctions of *equations* $t_i = t_i'$ $(1 \leq i \leq N)$ between terms built up from *f*, the numerical *variable* p, and signs for specific programs (constants, successor operation, addition, multiplication, etc.). Assume further that, for any program π_f and any term $t[\pi]$ built up from π_f, p and the signs for specific programs (as above), a program is assigned to $t[\pi]$. Then we define:

π_f is *checked* for Φ if for each i, $1 \leq i \leq N$, the *same* programs are assigned to $t_i[\pi]$ and $t'_i[\pi]$.

The question whether or not the same program is assigned to two terms is supposed to be *decidable*, and this constitutes the elementary character of 'checking'.

Remark. It is familiar from recursion theory that for quite *elementary formal systems*, specifically any consistent r.e. extension of formal primitive arithmetic PRA, it is *not* decidable whether an equation between two terms containing a variable (p) is formally *provable* since the sets

$$\{t_1, t_2 : \vdash_{\text{PRA}} t_1 = t_2\} \quad\text{and}\quad \{t_1, t_2 : \exists n \vdash_{\text{PRA}} t_1[p/n] \neq t_2[p/n]\}$$

are recursively inseparable. (Here n ranges over numerical terms and $t[p/n]$ is the result of substituting n for the variable p in *t*.)

Illustrations. Probably the reader understands the proposal pretty well, that is, the ideas of: program assigned to a term, and identity criteria for programs or, equivalently, definitional equality between terms. For deepening his understanding he should look at the relevant literature cited in footnote 9 of Note 1. Finally, since in terms of I §3a we are looking here at genuine, terminating programs only, perhaps an even *better* illustration of (or approximation to) proposed applications would be got from a language in which *every* term denotes a terminating program, as is the case with *typed* combinators; for arithmetic problems one adds (typed) *recursion operators*.[14] Whatever their mathematical appeal (or use in other contexts) the usual combinators do not seem to present any advantages here; specifically the definability of an (even) type free recursion operator does not seem worth the price of introducing expressions for which one does not know whether or not the corresponding program terminates. And--perhaps this is a principal point--it cannot be assumed that the BASIC IDEA OF THE WHOLE PROPOSAL (which comes from the *familiar* experience of knowing whether or not two *familiar* definitions denote the same program) EXTENDS UNAMBIGUOUSLY TO THE UNFAMILIAR (full) LANGUAGE of combinators. So without any prejudice against future theoretical studies on this point, I suggest that the reader concentrate on *strongly computable computation diagrams* in the sense of I §2(c).

Example. We suppose *descriptions* of programs to be given by terms built up from typed combinators (of type: $0 \to 0$); cf. footnote 9 of Note 1. Two such terms denote the *same* program if and only if *their* (unique) *normal forms are congruent*, that is, the normal forms are canonical or *standard* descriptions of programs. Since each term has a normal form, the required decision can be carried out by working out the normal forms.

Corollary (to footnote 12 of Note 1). For practically efficient solutions it will be necessary to find (sub) classes of terms for which an *equivalent* but computationally simpler criterion of definitional equality can be given. Naturally, as in footnote 14 above, the equivalence *proof* need not at all be elementary.

3. *Discussion of the formalization* (proposed in the last section). (a) At first sight the sense of *checking* seems to be unduly narrow. In fact, a program π_f is, of course, a perfectly good solution even if π_f does not check Φ, but simply satisfies Φ. The justification for the narrow definition, though simple, is often overlooked.

> If, as a matter of empirical fact, programmers tend to write programs which *do* check the conditions discovered by numerical analysis, it makes good practical sense to exploit this fact.

In this sense the proposal involves an (empirical) hypothesis about programmers which has to be verified.

(b) Note, however, that the notion of definitional equality(on which the formalization is based) is rather wide *unless we are planning to use the program for a large number of values of* p. It ignores the steps needed to go from the *description* $t[\pi]$ to its normal form, that is, to the program described, or, equivalently, to its *canonical description*. It is of course conceivable that an even sharper notion of *checking* is valid which is based on a narrower notion of definitional equality;

'valid' in the practical sense above of being respected by actual programmers. The discovery of such a notion would probably be at least as useful as the kind of mathematical discovery considered in the Corollary at the end of §2.

(c) Concerning the general hypothesis on the form of functional properties Φ supplied by numerical analysis (according to §1), one can distinguish three kinds.

(i) The equations, read from left to right, translate into strongly computable computation rules; this is the case in the example given: $f(0) \to 1$, $f(p+1) \to 2f(p)$ (when supplemented by rules for doubling: $2.0 \to 0$, $2(p+1) \to (((2p)+1)+1)$).

(ii) Φ, that is, $\forall p\Phi_0(p,f)$ provides a finite[15] definition of f, that is; $\forall m \exists n \exists q([(\forall p \leq q)\Phi_0(p,f)] \to fm = n)$.
In this case f is recursive but the equations in Φ_0 do not necessarily provide a computation diagram.

(iii) Conceivably numerical analysis provides a Φ that defines f uniquely but not computationally, for example,

$$\forall p[f(p) = 2f(p+1)] \ .$$

Only $f = \dot{0}$, i.e., $\forall p[f(p) = 0]$ can satisfy this condition since if $f(p) = k$, $f(p+k) = k \cdot 2^{-k}$ which is not integral. But, presumably, the numerical analyst will himself provide this little argument and thereby replace the condition by

$$\forall p[f(p) = 0) \ .$$

(d) Concerning the existence of genuine uses of the present proposal there can, of course, be no doubt inasmuch as I was able to appeal to such familiar material as exponentiation or doubling (only the range of application is in doubt). For the same reason the weakest point in the illustration is the use of mathematically 'interesting' and therefore unfamiliar material, such as the typed combinators with recursion. No doubt more experienced people would be better able to judge at the present stage.

III. *Consistency Proofs and Programs for Translators*

The kind of 'translator' meant here is supposed to start from a (formal) language relatively close to actual reasoning in arithmetic practice and to go over to computer programs. As a matter of empirical fact, actual reasoning uses non-constructive principles; as is well known the relation between non-constructive proofs and 'corresponding' programs is problematic. Roughly speaking, consistency proofs and their developments solve these problems; in fact, this kind of use of consistency proofs is often considered to constitute their *mathematical significance*.[16]

Since the mathematics involved has been developed actively over half a century, it will be possible to be quite brief by referring to the literature. The main point of interest is to indicate in what respects the existence of *fast* computers offers new (practical) uses for some of the existing consistency proofs, or makes developments desirable which would have no value to the human computer.

1. *Generalities*. (a) To avoid confusion when reading the literature the point made already in the introduction must not be forgotten: the *official* purpose of consistency proofs (formulated by Hilbert for philosophical reasons) is not only irrelevant to computer science but contrary to computer interests; cf. also footnote 14 of Note 2. Hilbert's aim was to show, by *elementary* metamathematical methods, that if a universal statement $\ulcorner\forall x A\urcorner$ (with primitive recursive A) is non-constructively provable then (*) *each* numerical instance can be checked by computation. But for the computer it is quite sufficient to know the *fact* (*); he gains nothing from knowing an elementary proof. The fact alone permits him to add an *additional computation rule*, namely, the subroutine: whenever $A[x/n]$, for numerical n, is supposed to be worked out (by computation) the additional rule says: write 'true'. From the illustration in II §2 in connection with Fermat's conjecture, it is clear that a computer scientist (that is, numerical analyst) who knows only elementary constructive methods of proof is liable to be at a disadvantage here: *he* could not justify the use of this subroutine which can greatly shorten the work; more pedantically, which shortens the work whenever computations of $A[x/n]$ are

needed for many $\underline{n}$ and we have a non-constructive, but no elementary proof of $\forall xA$.. The 'constructivist' could, at best, make the empirical prediction that the subroutine will 'work'.[17]

(b) In point of fact, most of the metamathematical studies which are described as 'consistency proofs' establish more. For *arithmetic problems* in the sense of II §1, but with A restricted to decidable relations they provide provide an *explicit scheme* for defining number theoretic functions such that

> If p is a derivation (in the non-constructive system) of $\exists mA(n,m)$, they provide π_p defined by means of the scheme such that $\forall nA(n,\pi_p n)$.

Of course, once again the mere *validity* of the rules considered (when applied to arithmetic) is sufficient to ensure the *existence of a program* π_A, quite independently of any hypothetical proof p of $\exists mA(n,m)$, namely,

> Compute $A(n,0)$, $A(n,1)$, ... until you find an m_n such that $A(n,m_n)$ holds.

If $\exists mA(n,m)$ is (seen to be) *true* (by non-constructive means) the program *terminates* (or: is seen, by the same means, to do so).

In view of the irrelevance to computer science of the metamathematical methods used, *only a detailed study can decide whether the program supplied by* π_p *is more efficient than* π_A. (It is tempting to say here that one surely knows 'more' if one knows a proof p than if one merely knows the truth of $\exists mA(n,m)$; sure, but is this additional knowledge *computationally relevant*?[18])

Remark. In terms of II §1, the justification of the program π_p requires numerical analysis. Suppose Φ_p is the functional property used for the defining equations of π_p; then we have to have shown:

$$\Phi_p(f) \to A(n,fn) .$$

In contrast, the program π_A requires no numerical analysis.

(c) It is well known that for logically more complicated relations $A(n,m)$, of the form

$$\forall qB(n,m,q)$$

(B primitive recursive), $\forall n \exists m A$ may be non-constructively provable, but there is simply no recursive function f at all which satisfies: $\forall n A(n, fn)$. In this case, 'consistency' proofs for the non-constructive principles considered, may be used to provide constructive interpretations, discussed at length in the papers cited in footnote 16 above.

In general the value of these interpretations is purely logical. Specifically, and this is in sharp contrast to (b) above, for logically complicated A, the non-constructive meaning of $\forall n \exists m A$ may be (mathematically) interesting--bien entendu: to the non-constructive mathematician who understands this meaning!--but the constructive interpretation (of $\forall n \exists m A$) is of no interest--to anybody. So the principal problem here is to look for those cases which are interesting.

Exception (from the Digression on pp 135-136 of the paper cited in footnote 2 of the Introduction). Little has to be added to Herbrand's constructive interpretation of classical predicate calculus to convert Roth's non-constructive proof[19] of the well-known Thue-Siegel-Roth theorem into the refinement by Davenport-Roth.[19] This refinement was of interest inasmuch as its authors decided to devote a separate paper to it.

2. Programming translators: the present evidence. (a) There is no shadow of doubt of the usefulness of (knowing) consistency proofs. People working in number theory, quite consciously and explicitly, wanted to know bounds for existential theorems proved by non-constructive means and, as a matter of historical fact, got lost when trying to obtain these bounds without theoretical knowledge of consistency proofs: the(ir) light of nature was not enough. Without having to apply the theoretical knowledge in detail, just having the general principles in mind, made it easy, in accordance with footnote 8 of Note 1, to obtain the desired bounds. But this does not yet constitute a reason for actually programming a translator; quite trivially, inasmuch as general knowledge was sufficient (for the bounds above) there is no need to do more about it unless one has new, ambitious tasks in mind!

(b) It seems plausible that, as far as applications to computer science are concerned, the general ideas of consistency proofs but not necessarily the specific schemata of functions (leading to the programs π_p in §1b) are useful. Quite generally, the currently open problems really concern the details of the specific schemata, since the *theoretical possibility* of constructivizing the bulk of mathematical practice has been established. Neglect of this fact has led people to rehash, at length, dead issues.[20]

(c) The *general* conflict between logical and computational purposes (which we are talking about) can also be put as follows. Consistency proofs qua logical contribution tell us the *nature* of the problems involved in constructivization, in other words, what is, generally, *possible*. It is not reasonable to suppose that the same piece of work will *often* be useful for *actual* understanding of particular problems.

Specifically, to take one of the non-constructive proofs considered in the papers cited in footnote 16, knowledge of consistency proofs allowed one to see (the theoretically interesting fact) that *without* new ideas one obtained a bound

$$(\exists n < 2^{2^{2^{8}}})[\pi(n) > \mathrm{li}(n)]$$

from Littlewood's non-constructive proof of

$$\exists n[\pi(n) > \mathrm{li}(n)]$$

where $\pi(n)$ denotes the number of primes $\leq n$ and li the logarithmic integral. But we now know (the practically more interesting fact) that by *judicious* use of *new* ideas *and* a fast computer one has

$$(\exists n < (1.65)\cdot 10^{1165})[\pi(n) - \mathrm{li}(n)] .$$[21]

In the last analysis the reader will simply have to decide for himself, if or to what extent the theoretical knowledge helps him in his practice.

3. *Speculation on systematic uses of fast computers.* As already stressed at the beginning of §2, there is no doubt about the possibility of intelligent *ad hoc* use of computers in extracting bounds from non-constructive proofs; for example, in the paper cited in footnote 21. But it is more difficult to formulate good *general principles* for such uses; or, put differently, it is difficult to *apply* to the present topic of translations (from non-constructive proofs to programs) the well-known general criteria on fruitful uses of computers.

(a) *Principle of the middle-distance.* I suppose the most familiar criterion is this (which, as pointed out at the end of Note 1, is not satisfied by so-called automatic theorem proving). We have available 3 conditions: (i) a *formal analysis* of a problem, in our case of extracting bounds from formal derivations in non-constructive systems; (ii) the analysis is *not so simple* that we can apply it 'efficiently' ourselves; (iii) the analysis is *not so complicated* that it cannot be used by (existing) computers either.

To make progress we must look at the *details* of consistency proofs and the preparation needed in their application. The first point that helps is that we do *not* need a 'fully formalized' argument. One soon learns to separate irrelevant inferences and concentrates on *critical* ones.[22] The bounds obtained by use of a consistency proof depend (only) on the *syntactic properties* of the formulae appearing in these critical inferences. What can the computer do that we can't or won't do?

Speaking from my (limited) personal experience a human computer soon gets bored trying to determine the syntactic complexity of the relevant formulae, and then makes quite rough estimates. The loss of *computational efficiency is hidden because we use non-numerical terms in presenting the solution* (cf. I §1a). Thus in §2c we have

$$2^{2^{2^{8}}} \quad \text{and} \quad (1.65)\, 10^{1165} \;;$$

size, not (human) intelligibility, makes a marked distinction between these two terms.

Suggestion. Perhaps there are occasions where a computer can produce a *significantly* lower bound without any new ideas, but as follows. We take the *explicit definitions* in an informal proof, we take the *critical inferences* (expressed of course by use of those explicit definitions) and simply let the computer work out the syntactic complexity of the formulae obtained after eliminating the explicit definitions.[23]

(b) Developing the suggestion above, one could look for a *very special kind of 'automatic' theorem proving*, namely *reductions* of the syntactic complexity of the critical formulae (of course, reductions by means of non-critical inferences).

This particular proposal is related to the general aim of using computers to *fill in 'gaps' in elementary informal proofs*. But even if, perhaps for the reasons given at the beginning of I §3c, this *general* aim has no feasible mechanical solution, the narrower problem of reducing syntactic complexity may do: it certainly has a more mechanical *look*. Also this problem does not involve any (artificial) reduction of the language used (which is the 'logical' device that, in these notes, is held responsible for inefficiency).

Warning. All I can hope to have done here is to have made a case for studying *problems which*, for the widely accepted doctrinaire views described in footnote 18, *are underestimated*. There can, I think, be no true progress until we have found a specific problem, where *we really want to know the bounds* and therefore can judge whether some given solution is satisfactory. Afterwards we can build on the solution.

IV. Experiments with Computers on the Complexity of Non-numerical Computations

In Note I §3c, in connection with search procedures for proofs, that is, 'automatic theorem proving', the role of *extending the language of predicate calculus* (in human proofs of logical[24] theorems) was discussed. The formulation of this role in precise mathematical terms is quite obvious. Moreover, given a particular set of deduction rules the problem of determining bounds for the efficiency of search procedures (average length of searches) is of course mathematically well defined. But all this is irrelevant, on the practical philosophy guiding these notes, since the general problem is too complicated combinatorially and therefore not likely to be (practically) rewarding.

Rather, we should like to make use of the experience we all have of mathematical proofs to pick out *particular cases which have a chance of being typical*. This is all the more important in the present context since there is a genuine possibility of using computers to help study such a particular case, but none of using them *directly* for the general mathematical problem above. The reader should note that this proposal, of using computers on such typical particular cases, is itself a good example of the *ad hoc* uses of computers described at the beginning of III §3 (so-called 'man-machine interaction'): *searching through search procedures* would be hopeless even for computers; however, for some particular examples and search procedures, analyzing the length of the search seems hopeless for a human computer (even if he is allowed to use all his mathematical ingenuity[25]); but by the very nature of the problem, a computer *can*, always, be used to decide whether the particular procedure is practical for *this* computer to solve a particular problem: just set the computer going and let it run for the time you have available!

The basic question is of course: *how good is our judgment of which cases are typical*? Since a fair amount of computer time (where time *is* money, and not merely *ennui*) is needed for the kind of experiments I have in mind, let me begin by expanding the argument in I §3c for giving up the 'logical' point of view. But the discussion is not needed for §§2-3.

1. *Some consequences of the chase after logical completeness.*[26] The kind of completeness meant concerns not only completeness (of proof procedures) w.r.t. validity, but also so-called functional completeness (of the language used).

(a) In connection with predicate calculus, *if* logical selection criteria are to be used at all, the classification according to *prefix complexity* is no doubt as natural as any. For *this* classification essentially only (negative) undecidability results are available.

More importantly, for the most outstanding decision methods so far discovered (for various classes of *fields*), the subclass of formulae considered, namely,

$$(*) \qquad A \rightarrow B$$

for fixed A and arbitrary B, is quite unnatural from a purely *syntactic* or logical view: the choice of the particular A is not explained in syntactic terms.

Conversely, at least at present, if we take such a known decision *method* and ask for a more general class $\textcircled{A}$ of formulae (with $A \in \textcircled{A}$)

to which the method applies, no syntactically natural definition of Ⓐ suggests itself. In the words of Wang's paper, cited in footnote 26, p. 102: we have here a powerful (decision) method applied to the restricted 'range' of formulae (*). But we don't really know what to do with the method *within* the general framework of logical complexity (how to 'explore' it, loc. cit., p. 102, l. 11). Incidentally, the same situation arises with more advanced mathematical results when they are formulated in some *reductive vocabulary* like that of axiomatic set theory.

We have here a typical example of the frustration produced when one wants to apply the much vaunted dialectical method: it's all very well to say we should *build on previous knowledge* (= synthesizing thesis and antithesis); but which *general* framework (for a synthesis) is valid?

(b) In connection with (partial) recursive functions we have a similar situation to the *reduction classes* in (a). As shown by various 'axiomatic' recursion theories[27] or combinators: provided we have a few 'simple' basic functions and some kind of composition, *all* partial recursive functions can be defined and hence, once again, we have a host of (recursive) undecidability results.

If we stay closer to computations, not the partial recursive functions, but *rudimentary* functions used to code the sequences of states of a Turing machine (or, equivalently, used to enumerate r.e. sets) are fundamental; the partial recursive functions are defined from them as described. (This example was already used in the *Technical remark* at the end of I §2a). While of course equations between *closed* rudimentary terms are, those between open ones are not, in general, decidable; cf. the *Remark* in II §2.

(c) Granted (a) and (b), a study of logically 'complete' languages (for the whole of predicate logic, resp. for all partial recursive functions) would still make practical sense if there were at least a *fair number* of significant cases where results about the whole, undecidable, class *specialize to practical computation procedures in some practical subclass*; in other words, if the 'logically complete' theory were a *good* practical theory.

I know from experience in proof theory that, at least there, one has to make a fresh start; as was mentioned earlier, proof theory *begins* where recursion theory ends; cf. specific examples in §2b below. As to computation, for example, Rabin's useful decision method for trees[28] does not seem to be a special case of a general result (perhaps on definability--of course not on decidability) for, say, the whole second order predicate calculus.

Discussion. By footnote 25, the general issue is far too interesting simply to adopt some point of view (for example, of the present notes) as a working hypothesis. One wants to test it. The difficulty in testing is of course this: whatever view one adopts one is going to get some 'results'; and if the view is coherent one will get mathematically satisfactory results, even if the view itself is mistaken.

I suspect that the present issue is closely connected with the so-called 'prejudice' of mathematicians against logic. Since most of those mathematicians do not know technical logic their opinion is not based on the actual technical level or mathematical quality of the methods used in logic. What they do know is the ordinary meaning of 'logic' as the science of reasoning or of the principles of reasoning; in other words, that the subject of logic (applied to mathematics) is concerned with possibilities of proof. And, in their inarticulate way, mathematicians want to say that the kind of general knowledge of such possibilities provided by logic, does not help them with actual understanding of a given proof; cf. also III §2c.

The only part of the mechanistic view mentioned in footnote 26 that I am using in these notes, is this:

> Human intelligibility is a qualitative indication of mechanical feasibility.

As a kind of corollary:

> A proof which surprises us will be difficult to find by a general mechanical method.

We shall consider two measures of difficulty in §§2,3.

2. First experiment: The intention is to find a problem, which is formulated in first order language and is of undisputed mathematical interest (and hence of feasible length!), but has no proof of feasible length in the language of predicate calculus.

In other words, though some proof in predicate calculus exists, such a proof is *not only difficult to find*, *but difficult to write down* (even if found). Consider the proposition

> There are no skew fields or division algebras over the reals of dimension n except for $n = 1, 2, 4, 8$ (real and complex numbers, quaternions and Cayley numbers resp.).[29]

For any fixed n, there is of course a formula in the first order language of real closed fields which expresses the existence of such an algebra of dimension n by means of an $\exists\forall$ formula, say $\exists\vec{\alpha}\forall\vec{x}\forall\vec{y}D_n$, namely, the existence of elements (real numbers) α^k_{ij} $(1 \le i,j,k \le n)$ such that the *composition law* or 'multiplication table'

$$(x_1,\ldots,x_n)\cdot(y_1,\ldots,y_n) = (\Sigma\Sigma\alpha^1_{ij}x_iy_j,\ldots,\Sigma\Sigma\alpha^n_{ij}x_iy_j)$$

satisfies the axioms for a division algebra (of course for all real numbers x and y).

A statement of this form is *decided* by the (first order) axioms for real closed fields. Therefore for each $n \neq 1, 2, 4, 8$ there exists *some* derivation of $\neg\exists\vec{\alpha}\,\forall\vec{x}\,\forall\vec{y}\,D_n$ from those axioms.

CONJECTURE. There is an n, say $n = 64$ or $n = 256$, for which the formula $\neg\exists\vec{\alpha}\,\forall\vec{x}\,\forall\vec{y}\,D_n$ can be stored in the memory of a moderately priced computer, but the *shortest* first order deduction cannot be carried out in a week on the fastest available computer.

The assumption about *feasibility* here is that at most one week's computer time on the fastest computer is available to the experimenter.

Evidently, in principle, the problem is purely mathematical once the intended deduction rules for predicate calculus (or, more precisely, not for logical validity but for *consequence from axioms for real closed fields*) are given. I assume that this mathematical problem is too difficult. The experimenter should reflect, perhaps guided by the topological proofs cited in footnote 29, on the most promising strategies and then let a computer work them out. Here are some comments for orientation.

(a) The general reason for choosing this problem is stated quite convincingly by Wang (cf. footnote 26, p. 107, second paragraph). The topological proofs were found to be surprising. Moreover, the theorem in question had been conjectured a long time ago; so it was the proof, not the result which was surprising. Hence it seems not unreasonable to suppose that some intrinsic properties of possible proofs--and not only of the search for proofs--constituted an obstacle to its discovery; cf. also the Remark at the end of this Note. (The particular conjecture is problematic, just because the general reason is quite commonplace.)

The present issue should be distinguished from a superficially similar problem which is definitely interesting for computer science but of a different character, namely, the computational analysis of the various decision methods for the theory of real closed fields. To discuss this matter we need a distinction.

(b) In its ordinary sense a decision method is given by a function δ from (closed) formulae A (of the theory) to $\{T, \bot\}$, where

$$\delta(A) = T \text{ if } A \text{ holds,} \qquad \delta(A) = \bot \text{ if } A \text{ does not hold;}$$

or, equivalently, in the case of a complete theory, where

$$\delta(A) = T \text{ if } A \text{ is derivable,} \qquad \delta(A) = \bot \text{ if } \neg A \text{ is derivable.}$$

Actually, constructive proofs of decision methods, for example, by quantifier elimination, yield more, namely, a function δ^F from formulae to formal deductions in a system F such that (for example, in the case of complete theories)

$$\delta^F(A) \text{ is a formal derivation in } F \text{ of } A \text{ if } A \text{ is derivable,}$$

$$\delta^F(A) \text{ is a formal derivation in } F \text{ of } \neg A \text{ if } \neg A \text{ is derivable.}$$

Clearly the CONJECTURE would be refuted if we had a practical δ^F for the theory of real closed fields (where F is the formal first order theory). Equally clearly, the topological proof provides a splendid δ for the class of formulae $\exists\vec{\alpha}\,\forall\vec{x}\,\forall\vec{y}\,D_n$ for $n = 1, 2, \ldots$! (This is the 'sophisticated example' promised at the end of II §1.)

From non-constructive proofs establishing the decidability by means of a function δ we can of course also derive an explicit $\delta^{\mathfrak{F}}$ where, however, $\mathfrak{F}$ contains also the principles used in the metamathematical proof of decidability.

$\delta^{\mathfrak{F}}(A)$ consists in evaluating $\delta(A)$ and then attaching the metamathematical proof of the properties of δ.

The reader will recognize the parallel between the relation of δ^{F} to $\delta^{\mathfrak{F}}$ on the one hand and the relation of numerical to non-numerical but numerical-valued terms (in I §2a or III §3a) on the other. The main difference is that the *values* of $\delta^{F}(A)$, $\delta^{\mathfrak{F}}(A)$ are proofs and not numbers.

An even more brutal use of non-constructive proofs of decidability (which need not even provide an explicit δ) is this: one defines the *recursive* function δ^{D} ('D' for mere decidability)

> $\delta^{D}(A)$ is the first derivation $\underline{d}$ (in some ω-order of the formal derivations of the given system) which ends up with *either* A *or* with $\neg A$.

δ^{D} is recursive by (the mere fact of) decidability and recursive enumerability of the theorems of any formal system. There is no apparent way of refining the brutal definition δ^{D} to get a practical δ, let alone δ^{F}. The reader will have recognized here a particular instance of the limitations of recursion theory discussed in §1c above. (He would be amazed to know how often the simple facts above are ignored in the literature, with hilarious results.)

(c) It goes without saying that (as is familiar from all sciences which involve both theory and experimentation) it cannot be expected that all details of the experiment proposed are perfect! A skillful[30] computer scientist may think of a more promising example to work on than the theorem about division algebras. But it is possible to analyze in advance at least in a general way *what we can expect to learn from this kind of experiment*.

If the conjecture is refuted (for some n which 'barely' misses being a solution), one would simply have an *unexpected success of automatic theorem proving* in its *current* sense. Also, I should expect from this a contribution to algebra: a mathematician reflecting on this hypothetical piece of first order algebra may discover in it some general features of mathematical interest.

If the conjecture is established, we have only evidence against the (whole) current trend in non-numerical computation, not against the possibility of using computers.

> We have to look for relatively few additional notions about the real numbers (enrich the field structure a 'little') in order to shorten relatively many proofs considerably. And we expect to find a not negligible area of problems which are of interest, and have computer-solutions by use of these additional notions (problems which do not have computer-solutions without them nor humanly performable solutions with them).

To remove a quite unjustified air of unreality from this task, it should be observed that it is analogous to the task which the 'abstract' mathematician sets himself (except of course that the reference to 'computer-solutions' is to be dropped); 'humanly performable' means: comprehensible. The mathematician's task is not finished even when there is a practical decision method for all formulae of a language! He still wants to find infinite or even large finite classes of formulae for which he can tell us the result of applying the method, in short, he really wants to solve problems.

Inasmuch as the organization of a computer is different from our way of thinking, the computer scientist cannot expect to use exactly the same solutions as the mathematician. But one would expect the mathematician's 'unsystematic' work to provide a better general guide than the systematic parts of logic. On this score proof theory seems to stand a better chance of being useful than say model theory or recursion theory just because of--what to many logicians is--its principal 'defect': it studies specific systems, specific processes derived from an examination of our intended meanings, and has to search for principles of choice (which even after they are found may be less convincing than the choices themselves).

3. Second experiment: Here the intention is to find a theorem (bien entendu, again of undisputed interest) which we know to have relatively short proofs in a given system; but no reasonable universal search procedure could be expected to find such proofs in a feasible number of steps.

Consider the arithmetic propositions:

(i) $$n = 0 \vee \neg\, n^2 = 2m^2$$

i.e., the irrationality of the square root of 2, and

(ii) $$n \cdot m \cdot p = 0 \vee \neg\, n^q + m^q = p^q \quad \text{for } q = 3, 4, \ldots,$$

i.e., instances of Fermat's conjecture.

Both (i) and (ii) are written in the language of the *ring* of integers (that is, + and × only). Certainly (i) has a relatively short proof in the first order theory of the ring of integers, including induction. And (ii) has such a proof, for instance, for $q = 3$ or $q = 4$.

The known proofs of (i) and of (ii) with $q = 3$, were certainly felt to be *surprising*. One striking feature in both cases is that, as ordinarily formulated, the proofs use *concepts which do not occur in the statements of the theorems*. More formally, they use concepts which cannot be defined in the language of rings without the use of *logical* symbols, for example, in (i) one uses *divisibility* or *prime factorization*, in (ii) for $q = 3$ one uses *Legendre's symbol*. These proofs can be translated into the first order language of rings by using, throughout, explicit definitions of the new concepts (and relatively simple proofs of their 'defining equations' when the new symbols are replaced by the definitions).

It is a mathematical fact[31] that the theorems mentioned cannot be proved in the theory of rings with induction, but without the use of logically complicated formulae. Put differently, the *selection* of those auxiliary concepts and of their definitions in the first order language of rings is the *central* problem.

My impression (or, as people nowadays like to say, 'intuition') is that the selection depends, practically speaking, on understanding the intended arithmetic meaning of (i) and (ii). A possible computational effect could be that *universal* procedures (for predicate logic or perhaps even ring theory) would not find proofs *quickly*.

Evidently the first step would be to take an *existing* procedure; or rather choose among existing universal procedures the one that strikes the computer scientist as best suited to this problem. Perhaps someday one could even solve the mathematical problem for all universal procedures (satisfying some uniformity conditions which exclude building the known proofs of (i) or (ii) into the first few steps of the procedure); for some q in (ii) the search is inordinately long.

The general points made in §2 apply also here mutatis mutandis.

Remark. In line with footncte 25 (and Wang's paper cited in footnote 26) it would seem to me quite interesting to analyze striking specific facts in the history of mathematics in terms of complexity of non-numerical computation. For example--as a foil to the conjecture in §2--it was proved a long time ago that the only associative division algebras over the reals have dimensions 1, 2, or 4:

QUESTION: Is there a relatively short first order proof of the fact that there is no such algebra of dimension 64 or 256?

V. Remarks on Relevant Proof-theoretical Literature

Granted the basic phi.osophy going through these notes, namely, the stress on the difference between logical and computational significance, the following corollary is plausible.

The logical highlights of proof theory, which are of course the best-known parts of proof theory, are not likely to be of direct interest to the computer scientist. And, in particular, negative results are something in the background, telling him what not to do or to expect.

For the same reason the computer scientist, especially if he is interested in non-numerical computing, may find useful information in the neglected proof theoretical literature!--'neglected' because, from a logical point of view, it was marginal. The kind of thing I have in mind is some new particularly elegant formalization or a particularly direct treatment of some problems which, as far as logical interest was concerned, had really been solved completely. There is of course no guarantee that elegance or directness (for the human reader) will always have computational value but the possibility is worth keeping in mind.[32] NB: The neglect of this kind of work I have in mind is not accidental: without the additional computational analysis, the work *is* useless.

1. *Propositional operators*: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. Logically, most of these are *unemployed*, at least for the usual truth functional interpretation since all of them can be defined by superposition from, say $(\neg, \wedge)$. Similarly, the propositional *constants* T and $\perp$ can be defined since, for variable p, $p \wedge \neg p = \perp$. For logical metamathematical arguments it is convenient to use only $(\neg, \wedge)$ in order not to have too many cases to consider. But, for example, as long as the operators occurring in a formula determine the rules to be applied in some non-numerical computation it seems plausible that a *compromise* is most efficient, where one uses not only $\neg$ and $\wedge$, nor, of course, a separate symbol for every truth function![33] The following remarks may provide some perspective.

(a) Logically sophisticated readers seeing novel, more elegant formalizations tried to use them to get new logically significant results, for example, by applying the new work to intuitionistic systems which (are known to have logical interest but) do *not* allow explicit definition of the other propositional operations from $(\neg, \wedge)$. This applies also to §2 below.

(b) The step (a) simplified the analysis by the-mathematically-very popular device of using a purely *qualitative* (indefinability) criterion so that there is no need for closer *quantitative* analysis. In contrast, the proposed computational use of novel formalizations needs at least some *assumptions* about a correct measure of computational complexity. To judge *significantly* between two given formalizations it will not be necessary to find a particular measure, but only a *few* evident properties of a correct measure.

(c) Logicians will of course try to exploit the last observation in (b) above; see also the remark at the end of this note. But it is certainly possible that good *knowledge* of computer 'hardware' may help one find an explicit measure of complexity, and thus replace the *judgment* needed to see what is significant or evident about such measures.

(d) Practically speaking, the reduction of our problem to the kind of judgment just mentioned constitutes progress: probably more people (experienced in computer science) have such judgment than the *taste* needed to recognize elegance in the proof theoretic literature. (It is quite unnecessary to claim that matters of taste are not objective; it is enough if people lack taste; just as it is of little help to blind people that differences in color are objective: what they need is an *extensionally* equivalent criterion which *they* can use, such as those famous pointer-coincidences. Only the person who establishes the equivalence must be able to see, and the same applies mutatis mutandis to our problems about *significance* of computational measures.)

2. *Prenex normal forms*. Here again there is no doubt that for (classical) logic, for example, in completeness proofs or in the statement of such interpretations as provided by Herbrand's theorem (cf. the end of III §1) it is rewarding to exploit the existence of prenex normal forms. But we have already seen the computational absurdity of the classification by prefix complexity in IV §1a.

More importantly, if one is interested in the formal deductions themselves (not only easily comprehensible *interpretations* of the kind needed for Note III), it is morally certain that it is better to use formulations in systems of natural deduction[34] and normalization procedures; these are not significantly simpler for prenex formulae. In contrast to Herbrand's treatment they also analyze quantifier free but propositionally complex formulae. The moral certainty will become mathematical if a correct measure of complexity of deductions and bounds for the complexity (of the particular deductions involved) are available.

3. *Equality*. It is one of the more memorable facts of elementary logic that the theory of predicate logic when $=$ has the intended meaning is reducible to ordinary predicate logic (in contrast, e.g., to the case where we have a binary relation symbol $<$ with intended meaning: $a < b$ if and only if a and b are natural numbers and a is less than b). One simply takes all relation symbols R_j and function symbols f_j occurring in the formula F to be treated and considers the implication

$$(*)\qquad \forall\vec{x}\,\forall\vec{x}\,'\{\bigwedge_i x_i = x_i') \rightarrow \bigwedge_j[(R_j \leftrightarrow R_j') \wedge (f_j = f_j')]\} \rightarrow F$$

where $\vec{x}$ is a sequence of variables containing the arguments of all the R_j and f_j and R_j', F_j' are obtained by substituting x_i' for x_i in R_j, resp. f_j. (No two symbols, R or f, have any argument in common.)

But as any writer on proof procedures knows,[35] derivations of the *implication* (*) have a quite different structure in *detail* from, say, natural deductions of F using relevant *equality rules*. (I cannot recall *exact* references, but I seem to remember material in Japanese journals and in a letter from K. Schütte.)

4. *Axioms and rules*. The particular case of equality in §3 is distinguished by the fact that, for a given F, a single (equality) hypothesis is added, which, moreover, is determined by F itself. In the case of the induction schema,

$$(**)\qquad \forall x[\{A(0,x) \wedge \forall n[A(n,x) \rightarrow A(n+1,x)]\} \rightarrow \forall nA(n,x)]$$

we have also a reduction to predicate logic; it needs a *single* premise if we use the schema only for formulae A of bounded complexity; instead of the equality axiom in (*) of §3, we use (**) as an hypothesis on F.

But, as any up-to-date account of the matter shows, we cannot *deduce* significant results for arithmetic from such results as Herbrand's theorem; at best we can hope to generalize the *ideas* involved in the proof of the theorem, and apply them to the rule:

Derive $\forall nA(n)$ from $A(0)$ and $\forall n[A(n) \rightarrow A(n+1)]$.[36]

5. *Computation diagrams for various classes of functions*. In view of the connections between computations and deductions discussed at length in Notes 1 and 2, it is clear that the proof theoretic literature must also contain several computation diagrams or schemata which define the same class of functions but have quite different computational properties.[37] However, I am not competent to give a detailed bibliography.

CONJECTURE. Probably a good deal of information which is useful to the computer scientist is contained in the *details* of work by people who are best known for the logically important results they have obtained while the elegance of their *mathematical* ideas may or may not be recognized; as examples of the former we have the family J. and R. M. Robinson or Schütte; as examples of the latter, Kleene and some Japanese authors.

It must be remembered that mathematical elegance (which is relevant to computational applications) need not be apparent from the look of the printed page, let alone from the grammar of the introduction or the general literary form--and therefore it is not easy for us to judge at a glance. This is only one further addition to the many examples in these notes of the differences in detail between human and mechanical requirements.

Remark. As far as I can see, a rather simpleminded idea of choosing between formalizations, which I have pursued for some years, has not yet been *refuted*:

> To consider infinitary languages, where of course differences are very much magnified (the ratio of $\omega^\omega : \omega$ is 'bigger' than the ratio $2^2 : 2!$), and to *hope* that obvious ordinal measures in the infinitary case will be a reliable guide to the subtle measures needed in the ordinary finite case.

Cf. the paper cited in footnote 36, top of p. 330 (and, for yet another direction, App. VIII on pp. 383-384).

Footnotes

[1] This research was supported by National Science Foundation Grant NSFGJ-443X2.

[2] Some of the material in these notes appeared in: Hilbert's programme and the search for automatic proof procedures, Springer Lecture Notes 125 (1970) [pp. 128-146, reviewed Zbl. 206 (1971) 277-278], which are the proceedings of a conference held in Versailles in December 1968 (on automatic demonstration). The present version is not only better organized and more articulate but, especially in Note 2, makes essential use of recent material published in: Second Scand. Logic Symp., ed. J. E. Fenstad, Amsterdam, 1971.

[3] Mathematicians have studied only relatively recently so-called non-extensional properties (of rules) which distinguish between two rules even if both assign the same value to each particular argument. For an example from arithmetic, consider the following two rules which determine quite different computation procedures. Letting $\mapsto$ mean: is replaced by, the first rule says: $n \mapsto 0$. The second rule says: $n \mapsto n \mathbin{\dot-} n$ with two 'subroutines' determined by the familiar rules for the predecessor (pred) and subtraction ($\dot-$, as above), that is, $\text{pred}\ 0 \mapsto 0$ and $\text{pred}\ (m + 1) \mapsto m$; $m \mathbin{\dot-} 0 \mapsto m$, and $m \mathbin{\dot-} (p + 1) \mapsto \text{pred}\ (m \mathbin{\dot-} p)$.

[4] The business about computation being an art supposes of course that there are no aesthetic laws. Actually, I believe, quite often we have to do with an illiterate expression of a sensible objection, namely, when quite unimaginative classes of rules are preferred for no other reason than that they are precisely defined (say, primitive recursive computation rules). A vague idea which is fairly close to good sense is often much more useful, though--by the very meaning of the terms--*we should not be able to give* (explicit) *reasons* for this fact.

[5] At the risk of sounding arrogant I should like to add that, just because of this difficulty, not only particularly gifted, but also quite weak minds are attracted to these problems, the latter because it is more difficult to establish the *in*adequacy of their proposed solutions. Whatever my motives in saying this, I believe the remark is *practically important*

for computer science. Otherwise one is tempted to assume that, for example, the *whole* subject of automatic theorem proving is hopeless just because the quite massive existing literature contains so little significant work.

[6] Old-fashioned experimental genetics and pseudo creative recursion theory have a similar jargon but like to use long words.

[7] This metamathematical meaning is not what we are taught in university mathematics when the integers and (cardinal) addition are defined set theoretically. There we prove, using our knowledge of set theoretic notions, that the assertion $\forall x \forall y(x + 0 = x \; \& \; x + sy = s(x + y))$ is true for the set theoretic meaning. In a careful course it is then explained why, for numerical terms n and m, $n = m$ is true for the set theoretic meaning if and only if n and m have the same formal value (completeness of the rules of addition; see para. 2 below).

[8] It is certainly true that in some highly developed subjects (electric networks) theory actually predicts experimental results; in such cases a developed formal theory is always useful. But theoretical or 'idealized' notions can also be useful in more subtle areas by drawing attention to general features or, as one says, by providing a 'point of view'. This can be achieved either by means of a mathematical formulation or by an instructive example.

[9] For further support, see J. T. Kearns, JSL 34 (1969) 561-575 and (the supplementary) Part II of the dissertation by H. P. Barendregt, Utrecht, 1971. There it is shown how Turing machine programs π (and other computation diagrams) map into combinatory terms t in the sense that the *sequence* of states determined by π corresponds to the *sequence* of terms appearing in the reduction of the term t_π.

[10] In the present context it would be pointless to 'identify' a formal system with its set of theorems, that is, to forget everything about it except the set of theorems: one wouldn't be making any deductions at all! One may still 'identify', say, the rules for forming wff with the set of wff as long as, quite realistically, it is easy to decide whether or not an expression is a wff.

[11] In §1a of: A survey of proof theory II, Second Scand. Logic Symposium, ed. J. E. Fenstad, Amsterdam (1971) 109-170, I discuss some Kleinarbeit which is required here. In the same volume there is also an excellent account which 'accentuates the positive', by Prawitz, particularly pp. 282-283 on a connection between derivations and terms.

[12] As always, maximizing an efficiency ratio does not consist in taking some given class of cases (say all formulae of predicate logic or all diophantine equations) and looking for a most effective procedure of handling them all: even if such a procedure exists it may be hopelessly ineffective. The problem is to discover a subclass which is (i) still useful, but which (ii) admits a much more effective procedure. For this reason 'logically' complete languages are rarely useful when efficiency is the principal issue.

[13] See H. S. Vandiver, Fermat's last theorem, Amer. Math. Monthly 60 (1953) 164-167 and the literature referred to there.

[14] The reader should, of course, make use of the knowledge he happens to have (for an illustration, it doesn't make sense to learn a new exposition even if it is much more elegant, provided a familiar one conveys the same idea). A fairly adequate introduction is on pp. 225-227 of Shoenfield's standard text Mathematical Logic, Addison-Wesley, 1967. For more detail see: L. E. Sanchis, Notre Dame J. Formal Logic 8 (1967) 161-174, another is W. A. Howard, pp. 443-458 in: Intuitionism and proof theory, Amsterdam, 1970. A word of warning: the fact that the metamathematical methods used by Sanchis in the analysis of computability are less elementary than Howard's is totally irrelevant for computational applications, just as--at the end of §1 above--any proof of Fermat's conjecture, however non-constructive, makes the difference between the first, useless, (primitive) recursive programmation and the second which appeals to (hypothetical) numerical analysis. To return to the two papers above: it is more significant computationally that Sanchis establishes strong computability for a greater variety of reduction rules than are considered by Howard. (Computation is closer to genuine mathematics than to foundations.) See also the papers cited in footnote 11 of Note 1.

[15] This is studied in Kreisel-Tait, Z. math. Logik u. Grundlagen 7 (1961) 28-38.

[16] For example, in my paper: JSL 23 (1958) 155-182 and its predecessors in JSL 16 (1951) 241-267 and 17 (1952) 43-58.

[17] At the present time actual mathematical (in contrast to metamathematical) number theory does not know examples of such A; more precisely there are no known examples of such $\forall xA$ which strike the experienced number theorists as mathematically interesting. By II §3b, it may well be of great practical use to know the *status* of this fact: in a practical theory we want to take into account the statistical distribution of the material to which the theory will be applied; cf. Section 2 below.

[18] This quite essential problem is usually overlooked in the *doctrinaire* constructivist literature for the following obvious reason (of which the doctrinaires are certainly not always conscious). Since the very idea of arithmetic *truth* or non-constructive *proof* of $\exists mA(n,m)$ is rejected, the transformation of p into the program π_p provides, generally, the *first constructive* proof of $\exists mA(n,m)$ and therefore--according to the doctrine!--the first *valid* proof. From the doctrinaire point of view this step is 'fundamental' and the 'detailed study' required above is, at best, regarded as a refinement.

[19] K. F. Roth, Mathematika 2 (1955) 1-20; and H. Davenport and K. F. Roth, Mathematika 2 (1955) 160-167.

[20] Cf., e.g., the rephrasing by E. Bishop, Mathematics as a numerical language, pp. 53-71 in: Intuitionism and proof theory, Amsterdam 1970. In particular it ignores the fact how much mathematical practice can be developed in *subsystems* of familiar systems (of analysis or set theory).

[21] R. S. Lehman, Acta arithmetica 11 (1966) 397-410.

[22] Indeed, in one of the well-known methods in the literature, Hilbert's ϵ-substitution method, one speaks of 'critical formulae'; see Hilbert-Bernays, vol. 2 (1970), p. 21.

[23] There is also an interesting and (therefore?) neglected theoretical problem of seeing whether the *order of magnitude* of the existing bounds, as a function of the syntactic parameters used, is right. For interesting exceptions, see B. Dreben, P. Andrews, S. Aandera, Bull. A. M. S. 69 (1963) 699-706 and current work by the Leningrad school of proof theorists.

[24] In the many cases where the formalization of ordinary reasoning involves a *reduction* to the language of predicate logic; as stressed in III §3b, for instance, in the case of quantifier manipulation to minimize logical complexity, no such reduction is involved. N.B.: To avoid (a most unlikely) misunderstanding. It is not the purpose of this note (nor of I §3c) to argue against the possibility of using properly computers for 'theorem-proving' or non-numerical computation; I am skeptical of (the current fashion of) using programs in the language of predicate logic; cf. also §2c below.

[25] It is sometimes forgotten that, for a wide class of problems such as multiplication of large numbers, computers are superior to a mathematician even if the latter *is* allowed to use all his ingenuity. The (practical) philosophy in these notes can now be restated as follows: find *new* areas where *this* kind of superiority is relevant--rather than finding complicated mechanical procedures to replace simple uses of ingenuity. The theoretical objections to the 'opposite' philosophy were discussed at the beginning of I §3c. Incidentally, in the literal sense of the words a 'theory of knowledge' might be expected to give us an idea *which* (areas of) problems are more easily understood by use of available mechanical devices and which by available ingenuity. Traditional epistemology simply did not possess these alternatives and therefore *had* to confine itself to questions of *validity* (of knowledge), though some philosophers--from Kant to Wittgenstein--tried to widen the subject (before it was ripe, I think).

[26] A general discussion of additional (negative) consequences is given, e.g., on p. 105, in H. Wang's paper: On the long range prospects of automatic theorem proving, Springer Lecture Notes 125 (1970) 101-111. The conclusions drawn overlap (of course) occasionally with those below, but the general view is not only different from, but in contradiction to, the

philosophy of these notes; cf. the crucial passage, p. 106, l. 3 to l. 9. Specifically if (a major part of) mathematical reasoning were indeed mechanical in nature, also the *selection* of (the major part of the) useful extensions of the language discussed in I §3c and of restrictions to subclasses would have to be mechanized.

[27] E.g., H. R. Strong, IBM Research and Development 12 (1968) 465-475 and E. G. Wagner, Trans. A. M. S. 144 (1969) 1-42. The composition $a \circ b$ means in Kleene's notation $\{a\}(b)$ or $\{\{a\}(b)\}$, depending on the context.

[28] Which led him to *discover* certain decidable second order theories which have some aesthetically very satisfactory completeness properties, cf. Trans. A. M. S. 141 (1969) 1-35.

[29] J. Milnor, Ann. of Math. 68 (1958) 444-449 and J. F. Adams, Ann. of Math. 75 (1962) 603-632 for the non-associative case (permitting $n = 8$).

[30] It also goes without saying that, in many cases, the experimenter's problem, that is, the step from the (theoretically perfectly sound) theoretical solution, to a workable experiment may simply be much more difficult than the theory.

[31] J. C. Shepherdson, Bull. Acad. Pol. Sc. 12 (1964) 79-86 and pp. 342-358 in: The theory of models, ed. Addison, Amsterdam, 1965.

[32] As stressed throughout these notes, *general* aesthetic criteria are certainly not reliable since, as elaborated in the introduction, for example, logical completeness has very great aesthetic appeal; indeed abstract questions which we make up ourselves are often more manageable than the more significant problems and have attraction just because we can do something about them.

[33] Significant *quantitative* work in the propositional case is contained in a paper by L. Hodes and E. P. Specker, pp. 175-188 in: Contributions to mathematical logic, ed. H. A. Schmidt and K. Schütte, Amsterdam, 1968. It does not deal with complexity of propositional *deductions* but of (propositional) *definitions* of truth functions. See also footnote 23 of Note 3.

[34] D. Prawitz, Natural deduction, A proof theoretical study, Stockholm 1965.

[35] Cf., e.g., the paper cited in IV, footnote 26, where also the, incidentally quite unrelated, problem of including = in decidability procedures is mentioned.

[36] Cf., e.g., §6 of my paper JSL 33 (1968) 321-388, with information on axioms and rules on pp. 366-368.

[37] Cf., for example, the elegant schemata by means of iteration for the class of primitive recursive functions by R. M. Robinson, Bull. A. M. S. 53 (1947) 925-942, where recursion is replaced by iteration.

(Continued from inside front cover)

96 R. C. Atkinson, J. W. Breisford, and R. M. Shiffrin. Multi-process models for memory with applications to a continuous presentation task. April 13, 1966. (J. math. Psychol., 1967, 4, 277-300).

97 P. Suppes and E. Crothers. Some remarks on stimulus-response theories of language learning. June 12, 1966.

98 R. Bjork. All-or-none subprocesses in the learning of complex sequences. (J. math. Psychol., 1968, 1, 182-195).

99 E. Gammon. The statistical determination of linguistic units. July 1, 1966.

100 P. Suppes, L. Hyman, and M. Jerman. Linear structural models for response and latency performance in arithmetic. (In J. P. Hill (ed.), Minnesota Symposia on Child Psychology. Minneapolis, Minn.: 1967. Pp. 160-200).

101 J. L. Young. Effects of intervals between reinforcements and test trials in paired-associate learning. August 1, 1966.

102 H. A. Wilson. An investigation of linguistic unit size in memory processes. August 3, 1966.

103 J. T. Townsend. Choice behavior in a cued-recognition task. August 8, 1966.

104 W. H. Batchelder. A mathematical analysis of multi-level verbal learning. August 9, 1966.

105 H. A. Taylor. The observing response in a cued psychophysical task. August 10, 1966.

106 R. A. Bjork. Learning and short-term retention of paired associates in relation to specific sequences of interpresentation intervals. August 11, 1966.

107 R. C. Atkinson and R. M. Shiffrin. Some Two-process models for memory. September 30, 1966.

108 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the third year. January 30, 1967.

109 P. Suppes and I. Rosenthal-Hill. Concept formation by kindergarten children in a card-sorting task. February 27, 1967.

110 R. C. Atkinson and R. M. Shiffrin. Human memory: a proposed system and its control processes. March 21, 1967.

111 Theodore S. Rodgers. Linguistic considerations in the design of the Stanford computer-based curriculum in initial reading. June 1, 1967.

112 Jack M. Knutson. Spelling drills using a computer-assisted instructional system. June 30, 1967.

113 R. C. Atkinson. Instruction in initial reading under computer control: the Stanford Project. July 14, 1967.

114 J. W. Breisford, Jr. and R. C. Atkinson. Recall of paired-associates as a function of overt and covert rehearsal procedures. July 21, 1967.

115 J. H. Stelzer. Some results concerning subjective probability structures with semiorders. August 1, 1967

116 D. E. Rumelhart. The effects of interpresentation intervals on performance in a continuous paired-associate task. August 11, 1967.

117 E. J. Fishman, L. Keller, and R. E. Atkinson. Massed vs. distributed practice in computerized spelling drills. August 18, 1967.

118 G. J. Groen. An investigation of some counting algorithms for simple addition problems. August 21, 1967.

119 H. A. Wilson and R. C. Atkinson. Computer-based instruction in initial reading: a progress report on the Stanford Project. August 25, 1967.

120 F. S. Roberts and P. Suppes. Some problems in the geometry of visual perception. August 31, 1967. (Synthese, 1967, 17, 173-201)

121 D. Jamison. Bayesian decisions under total and partial ignorance. D. Jamison and J. Kozielecki. Subjective probabilities under total uncertainty. September 4, 1967.

122 R. C. Atkinson. Computerized instruction and the learning process. September 15, 1967.

123 W. K. Estes. Outline of a theory of punishment. October 1, 1967.

124 T. S. Rodgers. Measuring vocabulary difficulty: An analysis of item variables in learning Russian-English and Japanese-English vocabulary parts. December 18, 1967.

125 W. K. Estes. Reinforcement in human learning. December 20, 1967.

126 G. L. Wolford, D. L. Wessel, W. K. Estes. Further evidence concerning scanning and sampling assumptions of visual detection models. January 31, 1968.

127 R. C. Atkinson and R. M. Shiffrin. Some speculations on storage and retrieval processes in long-term memory. February 2, 1968.

128 John Holmgren. Visual detection with imperfect recognition. March 29, 1968.

129 Lucille B. Mlodnosky. The Frostig and the Bender Gestalt as predictors of reading achievement. April 12, 1968.

130 P. Suppes. Some theoretical models for mathematics learning. April 15, 1968. (Journal of Research and Development in Education, 1967, 1, 5-22)

131 G. M. Olson. Learning and retention in a continuous recognition task. May 15, 1968.

132 Ruth Norene Hartley. An investigation of list types and cues to facilitate initial reading vocabulary acquisition. May 29, 1968.

133 P. Suppes. Stimulus-response theory of finite automata. June 19, 1968.

134 N. Moler and P. Suppes. Quantifier-free axioms for constructive plane geometry. June 20, 1968. (In J. C. H. Gerretsen and F. Oort (Eds.), Compositio Mathematica. Vol. 20. Groningen, The Netherlands: Wolters-Noordhoff, 1968. Pp. 143-152.)

135 W. K. Estes and D. P. Horst. Latency as a function of number of response alternatives in paired-associate learning. July 1, 1968.

136 M. Schlag-Rey and P. Suppes. High-order dimensions in concept identification. July 2, 1968. (Psychom. Sci., 1968, 11, 141-142)

137 R. M. Shiffrin. Search and retrieval processes in long-term memory. August 15, 1968.

138 R. D. Freund, G. R. Loftus, and R.C. Atkinson. Applications of multiprocess models for memory to continuous recognition tasks. December 18, 1968.

139 R. C. Atkinson. Information delay in human learning. December 18, 1968.

140 R. C. Atkinson, J. E. Holmgren, and J. F. Juola. Processing time as influenced by the number of elements in the visual display. March 14, 1969.

141 P. Suppes, E. F. Loftus, and M. Jerman. Problem-solving on a computer-based teletype. March 25, 1969.

142 P. Suppes and Mona Morningstar. Evaluation of three computer-assisted instruction programs. May 2, 1969.

143 P. Suppes. On the problems of using mathematics in the development of the social sciences. May 12, 1969.

144 Z. Domotor. Probabilistic relational structures and their applications. May 14, 1969.

145 R. C. Atkinson and T. D. Wickens. Human memory and the concept of reinforcement. May 20, 1969.

146 R. J. Titiev. Some model-theoretic results in measurement theory. May 22, 1969.

147 P. Suppes. Measurement: Problems of theory and application. June 12, 1969.

148 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the fourth year. August 7, 1969.

149 D. Rundus and R.C. Atkinson. Rehearsal in free recall: A procedure for direct observation. August 12, 1969.

150 P. Suppes and S. Feldman. Young children's comprehension of logical connectives. October 15, 1969.

(Continued on back cover.)

(Continued from inside back cover)

151 Joaquim H. Laubsch. An adaptive teaching system for optimal item allocation. November 14, 1969.

152 Roberta L. Klatzky and Richard C. Atkinson. Memory scans based on alternative test stimulus representations. November 25, 1969.

153 John E. Holmgren. Response latency as an indicant of information processing in visual search tasks. March 16, 1970.

154 Patrick Suppes. Probabilistic grammars for natural languages. May 15, 1970.

155 E. Gammon. A syntactical analysis of some first-grade readers. June 22, 1970.

156 Kenneth N. Wexler. An automaton analysis of the learning of a nimiature system of Japanese. July 24, 1970.

157 R. C. Atkinson and J.A. Paulson. An approach to the psychology of instruction. August 14, 1970.

158 R.C. Atkinson, J.D. Fletcher, H.C. Chetin, and C.M. Stauffer. Instruction in initial reading under computer control: the Stanford project. August 13, 1970.

159 Dewey J. Rundus. An analysis of rehearsal processes in free recall. August 21, 1970.

160 R.L. Klatzky, J.F. Juola, and R.C. Atkinson. Test stimulus representation and experimental context effects in memory scanning.

161 William A. Rottmayer. A formal theory of perception. November 13, 1970.

162 Elizabeth Jane Fishman Loftus. An analysis of the structural variables that determine problem-solving difficulty on a computer-based teletype. December 18, 1970.

163 Joseph A. Van Campen. Towards the automatic generation of programmed foreign-language instructional materials. January 11, 1971.

164 Jamesine Friend and R.C. Atkinson. Computer-assisted instruction in programming: AID. January 25, 1971.

165 Lawrence James Hubert. A formal model for the perceptual processing of geometric configurations. February 19, 1971.

166 J. F. Juola, I.S. Fischler, C.T. Wood, and R.C. Atkinson. Recognition time for information stored in long-term memory.

167 R.L. Klatzky and R.C. Atkinson. Specialization of the cerebral hemispheres in scanning for information in short-term memory.

168 J.D. Fletcher and R.C. Atkinson. An evaluation of the Stanford CAI program in initial reading (grades K through 3). March 12, 1971.

169 James F. Juola and R.C. Atkinson. Memory scanning for words versus categories.

170 Ira S. Fischler and James F. Juola. Effects of repeated tests on recognition time for information in long-term memory.

171 Patrick Suppes. Semantics of context-free fragments of natural languages. March 30, 1971.

172 Jamesine Friend. Instruct coders' manual. May 1, 1971.

173 R.C. Atkinson and R.M. Shiffrin. The control processes of short-term memory. April 19, 1971.

174 Patrick Suppes. Computer-assisted instruction at Stanford. May 19, 1971.

175 D. Jamison, J.D. Fletcher, P. Suppes and R.C.Atkinson. Cost and performance of computer-assisted instruction for compensatory education.

176 Joseph Offir. Some mathematical models of individual differences in learning and performance. June 28, 1971.

177 Richard C. Atkinson and James F. Juola. Factors influencing speed and accuracy of word recognition. August 12, 1971.

178 P. Suppes, A. Goldberg, G. Kanz, B. Searle and C. Stauffer. Teacher's handbook for CAI courses. September 1, 1971.

179 Adele Goldberg. A generalized instructional system for elementary mathematical logic. October 11, 1971.

180 Max Jerman. Instruction in problem solving and an analysis of structural variables that contribute to problem-solving difficulty. November 12, 1971.

181 Patrick Suppes. On the grammar and model-theoretic semantics of children's noun phrases. November 29, 1971.

182 Georg Kreisel. Five notes on the application of proof theory to computer science. December 10, 1971.