THE MARKGRAF KARL REFUTATION PROCEDURE
(FALL 1981)

K. Bl&sius, N. Eisinger, J. Siekmann, G. Smolka, A. Herold, C. Walther

Institut fur Informatik I
UNIVERSITAT KARLSRUHE
D-7500 Karlsruhe, Postfach 6380

Abstract: The current state of a Theorem Proving System (The Markgraf Karl Refutation Procedure) at the University of Karlsruhe is presented. The goal of this project can be summarized by the following three claims: it is possible to program a theorem prover (TP) and augment it by appropriate heuristics and domain-specific knowledge such that

(i) it will display an 'active' and directed behaviour in its striving for a proof, rather than the 'passive' combinatorial search through very large search spaces, which was the characteristic behaviour of the TPs of the past. Consequently
(ii) it will not generate a search space of many thousands of irrelevant clauses, but will find a proof with comparatively few redundant derivation steps.
(Hi) Such a TP will establish an unprecedented leap in performance over previous TPs expressed in terms of the difficulty of the theorems it can prove.

The results obtained thus far corroborate the first two claims.

O. INTRODUCTION

The working hypothesis of this TP project [DS77], [DS79], first formulated in an early proposal in 1975, reflects the then dominating themes of artificial intelligence research, namely that TPs have attained a certain level of performance, which will not be significantly improved by:

(i) developing more and more intricate, refinement strategies (like unit preference, linear resolution, TOSS, MTOSS,____), whose sole purpose is to reduce the search space, nor by
(ii) using different logics (like natural deduction logics, sequence logics, matrix reduction methods etc)

although this was the main focus of theorem proving research in the past.

The relative weakness of current TP-systems as compared to human performance is due to a large extent to their lack of the rich mathematical and extramatematical knowledge that human mathematicians have: in particular, knowledge about the subject and knowledge of hpw to find proofs in that subject.

Hence the object of this project is to make this knowledge explicit for the case of *automata theory,* to find appropriate representations for this knowledge and to find ways of using it. As a testease and for the final evaluation of the projects success or failure, the theorems of a standard mathematical textbook [DE71] shall be proved mechanically.

In the first section of this paper we give a general overview of the system as it is designed, albeit not completed. The second section concentrates on those parts of the system, whose implementation is finished and evaluated. In the third section experimental results are given and the final two sections present an evaluation based on the present findings.

1. OVERVIEW OF THE SYSTEM

A Bird's-eye View

Proving a theorem has two distinct aspects: the creative aspect of how to find a proof, usually regarded as a problem of *psychology,* and secondly the logical aspect as to what constitutes a proof and how to write it down on a sheet of paper, usually referred to as *proof theory.*

These two aspects are in practice not as totally separated as this statement suggest (see e.g. [SZ69]), however we found it sufficiently important to let it dominate the overall design of the system:

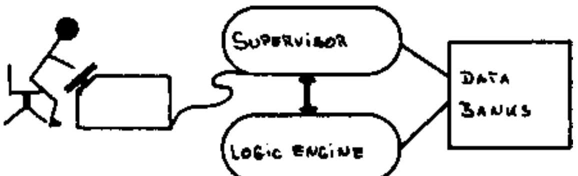


Figure 1

The *Supervisor* consists of several independent modules and has the complex task of generating an overall proposal (or several such) as to how the given theorem may best be proved, invoking the necessary knowledge that may be helpful in the course of the search for a proof and finally transforming both proposal and knowledge into technical information sufficient to guide the Logic Engine through the search space.

The *Logic Engine* is at heart a traditional theorem prover based on Kowalski's connection graph proof procedure [K075], augmented by several components that account for its strength.

The *Data Bank* consists of the factual knowledge of the particular mathematical field under investigation, i.e. the definitions, axioms, previously proved theorems and lemmata, augmented as far as possible by local knowledge about their potential use.

A View from a Leaser Altitude

The diagram of figure 2 sufficiently refines figure 1 to gain a feeling for the overall working of the system:

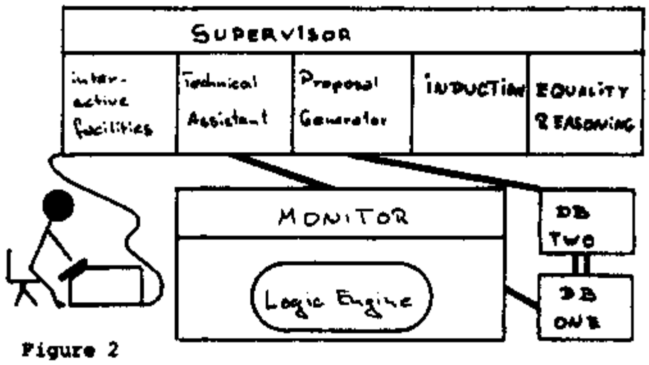


Figure 2

nical Assistant (TA), whose task is to transform this information, which up to this point is intelligible for a human user, into technical advice and code, which will then govern the top level behaviour of the Logic Engine and is for that reason passed on to the Monitor.

The *Monitor* governs and controls the global behaviour of the Logic Engine: immediately after activation it checks for an easy proof using the terminator heuristic (see section 2) and only upon failure activates the full machinery of the Logic Engine. Typical *control* tasks are detecting constant replications of the same lemma, detecting a circular development in the search space and keeping track of the 'self resolving' clauses. A good example how the monitor *governs* the top level behaviour is in its prevention of the unsteady behaviour which the system showed during earlier experimentation: The selection heuristics constantly suggest 'interesting' steps to take and forced the system to vacillate between different parts of the search speace -very unlike the behaviour of people, who, if put into the same situation, would tackle an interesting path until they either succeed or become somehow convinced that it was a blind alley.

Up to this point the decisions and activities of the PG and the TA are to a large extent based on the semantics of the theory under investigation and knowledge about proofs in this theory and their top-goal may be formulated as: to be helpful *"in finding a proof"*. Once they have done so, that top-goal becomes *"to derive a contradiction (the empty clause)"* and although this goal is of course identical to the previous one, it implies that different kinds of information may be useful: the original information provided by the PG based on the *semantics* (which is by now coded into various parameters, priority values and activation modules) as well as information based on the *syntax* (of the connection graph or the potential resolvent).

It may be objected that this is the main goal of a traditional TP also. While this of course true, there is the important difference that a traditional (resolution based) TP is not dirextly guided towards this goal in a step by step fashion, as no refinement [L078] specifies which literal to resolve upon next. For example linear resolution reduces the search space as compared to binary resolution, but within the remaining space the search is as blind as ever.

The PG and the TA are currently under development and not implemented at the time of writing.

2. THE LOGIC ENGINE

The *Logic Engine* is based on Kowalski's connection graph proof procedure [K075], which has several advantages over previous resolution based proof procedures: there is no unseccessfull search for potentially unifiable literals, every resolution step is done once at most and the deletion of links and subsequently of clauses leads to a remarkable improvement in performance, which is heavily exploited in the Deletion-Module. Most crucial to our approach however is the observation that since every link represents a potential resolvent, the *selection* of a proper sequence of links leads to the alleged active, goaldirected behaviour of the sy-

2. l *Input and Output*

The *interactive facilities* are too numereous to account for here and instead a protocol of a typical session is presented at the conference. An interesting point to note is that the interaction at this level was only designed for the intermediate stages of development. It is now to an increasing degree taken over by the Supervisor as it develops, with the intention to move the interface with the user althogether to the outside and to make the Supervisor take most of the low level decisions. Two sets of instructions however are to stay, the *IN-Module* is used to set up (and to read) the Data Bank in a way easily intelligible for the user. It also performs a *syntactical* and *semantical* analysis of the Data Bank, which is of considerable practical importance in view of the fact that it eventually contains a whole standard mathematical textbook.

The PROTOCOL-Module provides several facilities for tracing the behaviour of the system at different degrees of abstraction in order to cope with its complexity.

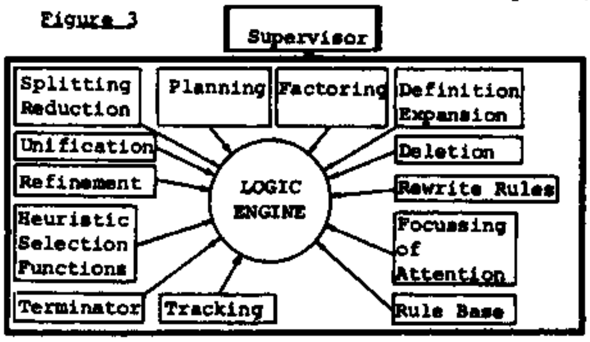
2.2 *The Monitor*

The deduction steps within the connection graph are governed and controlled by several modules, which are conceptually collected in the MONITOR.

We shall give a brief summary of the task of each of these modules (although each module represents a software development of about the size of a traditional theorem proving system). Only the heuristic selection functions are presented in more detail below.

Figure 3 gives an overview and refines figure 2:

Figure 3



The *Splitting and Reduction Module* is activated first and converts the initial formulas into cn-form subject to possible splitting and reduction operations [BL71]. The resulting set(s) of en-formulas are then transformed into the initial connection graph(s), which again are substantically reduced by subsumption and purity checks [EI81], [WA81].

The *Unification Module* contains special purpose unification algorithms for frequently occurring axioms like associativity, commutativity, idempotence and their combinations, and hence these axioms are removed from the data base prior to activation.

The *Refinement Module* simulates standard derivation strategies and most of the time only a small fraction of all links in the graph is declared *active*. For example if only the links emanating from a clause in the initial set of support are declared active and subsequently only the links of each resolvent in turn are declared active and the previous one *passive* (by appropriate •on-off-switches'), the resulting derivation is *linear.* The Refinement-Module allows to chose among some of the standard strategies and setting refinements CL078]. This has turned out to be advantageous, since it substantially reduces the number of active links and hence the expense for the computation of the heuristic selection functions and the most successful runs were obtained with this 'mixed approach'. The decision, which refinement to choose, is taken by the Monitor based on some genereal heuristics.

The *Planning Module* is currently under development and contains additional, heuristically motivated, splitting and simplification routines for the connection graph as well as planning capabilities for proof plans.

The *Tracking Module* controls the logic engine and detects circular developments in the search space, constant reapplication of the same lemma etc.

The *Focussing of Attention Module* prevents the unsteady behaviour mentioned in section one and ensures that theorems and lemmas selected by the supervisor are used in preference to other clauses. Both activities are achieved by special weights attached to the links in question.

The *Definition Expansion Module* exists in a rudimentary form only: as more complicated examples are tried, more refined heuristics are necessary to control the expansion process, i.e. the replacement of the defined literal by its defining clauses.

The *Rule Base* contains lemmas in the form of a production system and each time the *if-part* is satisfied the *then-part* is added to the current connection graph.

The *Rewrite Rule Module* contains simplification and reduction rules in the form of rewrites, which are applied to the terms of the initial graph, as well as to every generated resolvent.

The *Deletion-Module* heavily exploits the crucial property of this proof procedure that distinguishes it from other proof procedures based on graphs [SH76], [SI76]: the fact that the deletion of links and clauses can lead to a snowball effect of further deletions. Because of this effect it is worth every effort to find and compute as many criteria for potential deletions as possible. At present a clause is marked for deletion if:

(i) It is *pure* [WA81]
(ii) it is a *tautology* [NASI]
(iii) it is *subsumed* by some other clause [EI81]

and this information has *absolute priority* over the *relative priorities* to be discussed below.

The *Factoring Module* controls the factoring of clauses, which is based on special types of links in the connection graph.

In contrast to the global search strategies and global heuristics, the heuristic selection functions of the *Heuristic-Module* are based on *local eyntaotical information* about the graph or the resolvent (pararoodulant) respectively UseiL These heuristics wars obtained with two types of experiments: In the first experinent a human testperson is asked to prove a given set of formulas by resolution *without any* information on the intended meaning of the predicate or function symbols, Then the same set is proved by the system and in case its performance is inferior, the analysis of the deviation (and introspection of the testperson on why a particular step was taken) can give valueable hints for further heuristics.

In the second type of experiment the system is set to prove a theorem. In the case of success, an analysis of the protocol, where in the listing of all steps the steps necessary for the proof are marked, provides a remarkably good source of ideas for improvement, particularly if the reason why a certain step was chosen, is also printed in the protocol listing. During the last two years several hundreds of such listings were analyzed.

Initially we experimented with about 20 different *heuristic feature**, where each feature attaches a certain value to every link k in G.. 6. is the present graph, G. is the resulting graph after resolution upon linx K and Res is the resolvent resulting from this step:

(i) Sum of literals in G_{i+1}
(ii) Sum of olauses in G_{i+1}
(iii) Sum of links in G_{i+1}
(iv) Average length of olauses in G_{i+1}
(v) Average sum of links on literals in G_{i+1}
(iv) Sum (resp. average sum) of oonstant symbols in G_{i+1}
(vii) Number of distinot predioate symbols in G_{i+1}
(viii) Number of distinot variables in G_{i+1}
(ix) Sum of literals of Res
(x) Sum of links of Res
(xi) Sum of oonstant symbols in Res
(xii) Sum of distinot variables in Res
(xiii) Number of distinot predioate symbols in Res
(xiv) Term oomplexity of Res
(xv) Minimum of links on literals in Res
(xvi) Complexity of the most general unifier σ_k attaohed to link k
(xvii) Age of Res
(xviii) Degree of isolation of Res
(xix) Degree of isolation of the parents of Res.

The problem is that although each heuristic feature has a certain worth, the cost of its computation can by far outweigh its potential contribution. Also it may not be independent of the other heuristic features; for example features (xi) and (xii) both measure the "degree of groundness of Res", but in a different way. Similarly the values for Res and for G_{i+1} are not independent for certain features (e.g. xiii and vli). Also there are the two problems of finding an appropriate *mtrio* for each feature and to decide upon their *relative worth* in case of *conflict* with other features.

The information contained in the heuristic features is entered In different ways: certain facts (e.g. decreasing sise of the graph) have *absolute priority* and override all other information (see also the merge feature of TT in [DA78]). Most of the information of the other features is expressed as a real number in [0,1], where we experimented with several (linear, nonlinear) metrics. This information is then entered in a weighted polynomial and the resulting real number (the priority value) expresses the *relative worth* of the particular link and is attached to each link. In case of no overriding information the Control-Module selects the link with the highest priority value. Still other information is entered using the "window-technique": among the links whose priority value is within a certain interval (the 'window'), the Control-Module choses the one which minimizes some other feature.

The system has been designed such that heuristic features can easily be added and deleted, however after two years of experimentation the system has stabilized with the following solution (stabilized in the sense that neither the addition of heuristics from the above list nor the use of different metrics will significantly change the performance of the system on an appropriately large set of tests):

1. Complexity of the Graph
1.1 FCLSUM = (Σ of clauses of G_{i+1}) - (Σ of clauses of G_i)
1.2 FLINKSUM = (Σ of links of G_{i+1}) - (Σ of links of G_i)
1.3 FCANCEL = #{P|P is predicate symbol occuring in G_{i+1}}
1.4 FTERMINATE: (see below)

2. Complexity of the Resolvent
2.1 FAGE = Age of Res
2.2 FLITSUM = Sum of literals in Res
2.3 FTERM = Term complexity of Res
2.4 FRESISO = Degree of isolation of Res

3. Complexity of the Parents of Res
3.1 FPARISO = Degree of isolation of the parents

These features influence the actual derivation in the following way: all steps that lead to a reduction in the size of the graph have *absolute priority* and are immediately executed. That is, every link which leads to a graph with fewer clauses or fewer links or both is put into a special class, which is executed before any further evaluation takes place. The decision whether or not a link leads to a reduction is based on information from the Deletion-Module and is optionally taken for *every* link or for the *active* links only. Note that the reduction in the size of the graph may lead to further deletions, hence a potential snowball effect of deletions is carried out *immediately,* which accounts for the first main source of the strength of the system.

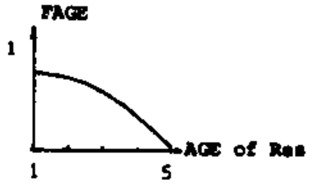
All other features have a *relative priority* and are classified as situation dependent and situation independent respectively, since the cost of their initial computation and later updating differs fundamentally [SS80].

A successful usage of the relative priorities depends on an appropriate *metric* for each feature, which expresses its estimated worth. A discussion of why the following metrics where chosen is outside of the scope of this paper and may be found in [SS80]; but the important point to notice is, that each metric displays a particular *characterietio*, which expresses the heuristic worth relative to its argument.

$\overline{\underline{\text{FAGE}}} := 1 - \left(\frac{\text{Age}}{D_{\text{max}}}\right)^{3/2}$

Characteristic:

(for $D_{\text{max}} = 5$)



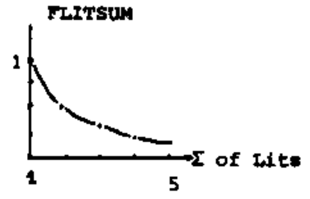
Age: max{Age Parent1, Age Parent2}+1

D_{max}: user defined maximally admitted depth of derivation

This feature is mainly used to avoid "infinite holes" in the search space.

$\overline{\underline{\text{FLITSUM}}} := \frac{1}{\Sigma \text{ of Lit of Res}}$

Characteristic:



This feature adds a strong "unit-preference-behaviour" to the system.

$$\overline{\underline{\text{FTERM}}} := \begin{cases} 1 & \text{if no nested or no terms in Res} \\ \frac{1}{2}\left(1 - \frac{\frac{1}{n}\sum_{1}^{n}\left(s_i\right)^{3/2}}{\left(s_{\text{max}}\right)^{3/2}}\right) & \text{otherwise} \end{cases}$$

where: s_i: maximal nesting of i^{th} term in Res
s_{max}: user defined maximally admitted depth of nesting
n: number of terms in Res

$\overline{\underline{\text{FCANCEL}}} := \text{LIT}(G_i) - \text{LIT}(G_{i+1})$

where $\text{LIT}(G_i) = \#\{P \mid P \text{ occurs as predicate in } G_i\}$

Note that the value is either 0 or 1 and this information is useful for a simulation of Colmerauer's cancellation strategy [KO75].

$\overline{\underline{\text{FLINKSUM}}}$: has either *absolute priority* if it decreases or else is entered into the selection process of the CD-Module with the window-technique

$\overline{\underline{\text{FRESISO}}}$: has either *absolute priority* if Res is pure, else

$$:= \begin{cases} 0 & \text{each literal of Res has at least 3 links} \\ 1/3 & \text{there is a literal in Res with 2 links} \\ 1 & \text{there is a literal in Res with 1 link} \end{cases}$$

$\overline{\underline{\text{FPARISO}}}$: similar to FRESISO

Note that FPARISO and FRESISO are useful, since they provide the Control-Module with the information that after one (or less useful, after two) further steps another deletion process starts, which potentially leads again to a snowball effect of deletions.

The main constraint for these heuristics is that after each resolution step the values of the heuristic features have to be updated and it is essential that the cost of this updating is much less than the cost of performing every possible resolution. For that reason not every value of the arguments for the metrics is computed exactly but estimated by some heuristic estimation function.

The *Terminator Module* is used in two ways: First it acts like a simple (and relatively fast) theorem prover and is activated on the initial connection graph. If it fails, the full machinery of the logic engine is loaded.

Secondly it is used to overcome the problem that the above heuristics have the very limited horizon of one step ahead, since the computation of a further n-level look ahead for n>2, is so prohibitively expensive that it outweighs the advantage. For that reason we implemented in addition a different n-level look ahead technique, which checks at tolerable cost if there is a proof within a predefined complexity bound. This terminator heuristic FTERMINATE is the no-loop-requirement of [SI76, p. 832] and is akin to the n-level-look-ahead heuristic proposed by [K075, p. 593] and is the second main source for the success of the current system. The essential idea is an elaboration of the following observation:

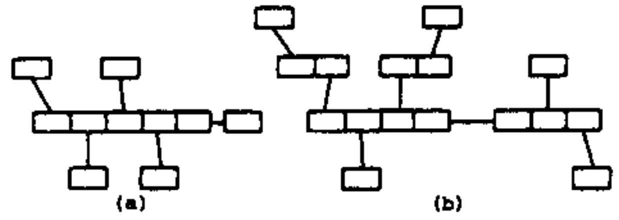


Figure 4

Each box in figure 4 represents a literal, a string of boxes is a clause and complementary boxes (literals) are connected by a link. If all unifiers attached to the links in figure 4a are compatible it represents a one-level-terminator, since it immediately allows for the derivation of the empty clause. Similarly figure 4b represents a two-level-terminator if the unifiers are compatible (see [SS80] for a detailed presentation). At regular intervals (e.g. after n derivation steps) the Terminator Module searches for such configurations.

2.3 Searching for a Proof

Once the Logic Engine is set into "prove-mode", the *CSS-Module* converts the *activated* part of the Data Bank into Skolemized clausal normal form and performs various splitting and truth functional simplification tasks. The resulting set of clauses is passed on to the *CG-Module,* which constructs the connection graph and if possible performs an evaluation of terms, reductions or algebraic simplifications of terms with the aid of the above modules. After these activities the *initial connection graph* is set up and now the search for a proof within this graph commences.

This search is locally controlled by the *Control-Module,* which decides which link to resolve upon next, based on nformation provided by the Heuristic-Module, the Refinement-Module, the Deletion-Module etc. The Control-Module turns the initial *repreeen-*

tation of clauses (the connection graph) into a *proof procedure* (baaed on connection graphs) as it defines a particular selection function, which maps graphs to links. This mapping is complex and based on information provided by several modules, but for clarity it is entirely contained in the Control-Nodule.

It should not be necessary to say that the complex interplay of the various modules, which 'suggest' which step to take next, prevents of course the overall deduction from being 'standard* and the respective completeness results do not necessarily hold. However, by an appropriate setting of the weight for FACE, it is impossible for a link to be ignored indefinitely for selection and for that reason we believe the system is complete.

The potential explosion of links is the bottleneck of the connection graph: the following 'challenge' proposed by P. Andrews, Carnegie Mellon at the 1979 Deduction Workshop, provides a point of demonstration: $(\exists x\ Qx \equiv \forall y\ Qy) \equiv (\exists x\ \forall y\ Qx \equiv Qy)$.

The initial graph of this formula consists of almost *10 000 links* and several hundred new links are added to the grpah for *each* resolution step. If all these links were declared 'active', the computation of the selection functions would become intolerably expensive.

In our system the example is split, reduced and subsequently deleted to a graph never exceeding *50 links* and easily proved within a few steps. Even for more 'natural' examples, the number of deletion steps is about one third and sometimes over one half of the total number of steps.

Scorn Technical Data about the Project

Name of Project:	Incorporation of Mathematical Knowledge into an ATP-System, investigated for the Case of Automata Theory.
Funding Agency:	German Research Organisation (DFG) Bonn, De 238/1, De 238/2.
Tim Period:	1976 to 1982 (six years)
Machine:	SIEMENS 7.760
Minimally Required Storage Space:	6.000 K (virtual memory)
Languages:	SIEMENS-INTERLISP CEP75]
Present size of system:	* 500 K of source code
Effort:	* 10 manyears for its implementation

With more than 500 K of actual code at present and approximatly 1.000 K under design for the next two years, the system is the largest software development undertaken in the history of automated theorem proving and it may be indicative for the changing pattern of research in this field.

3. PERFORMANCE STATISTICS

To gain a feeling for the improvement achieved by the system, figure 5 gives a sample of some test runs. In order to avoid one of the pitfalls of statistical data, which is to show the improvement achieved on certain examples and not showing the deterioration on others, the system is to be tested on all of the main examples quoted in the ATP literature: [WM76], [M0W76], [KRYKU72J. Of all exanples tested thus far, the examples of figure 5 are representative (and wore tease, i.e. all other examples were even more favourable for our system). The examples of figure 5 are taken from the extensive, comparative study undertaken at University of Maryland [WM76], where eight different proof procedures were tested and statistically evaluated on a total of 152 examples.

The table is to be understood as follows: the first column gives the name of the set of axioms in [WM76], e.g. LS-35 in line 9. *The* next three columns quote the findings of [WM76], where the figure in brackets gives the value for the worst proof procedure among the eight tested procedures and the other figure gives the value for the proof procedure that performed best. The final three columns give the corresponding values for the Markgraf Karl Procedure. For example, in order to prove the axiom set LS-35 (line 9) the best proof procedure of [WM76] had to generate 335 clauses in order to find the proof, which consisted of 14 clauses, and the worst proof procedure had to generate 1.521 clauses in order to find that proof. In contrast our system *generated only 9 clauses* and as these figures are typical and hold uniformly for *all* cases, they are the statistical expression and justification for the first two claims put forward in the abstract.

4. KINSHIP TO OTHER DEDUCTION SYSTEMS

The advent of PLANNER [HE72] marked an important point in the history of automatic theorem proving research [AH72], and although none of the techniques proposed there are actually present in our system it is none the less the product of the shift of the research paradigm, of which PLANNER was an early hallmark.

The work most influential, which more permeates our system than is possible to credit in detail, is that of W. Bledsoe, University of Texas [BT75], [BB75], [BB72], [BL71], [BL77], In contrast to [BT75], we tried to separate as much as possible the logic within which the proofs are carried out from the heuristics which are helpful in finding the proof.

The strongest resolution based system at present is [M0W76], and we have tested their examples in our system. Comparison with their reported results, shows that if our system finds a proof it is superior to the same degree as reported in figure 5.

However, there are still several more difficult examples reported in [MOW76] which we can not prove at present. The strength of the system [M0W76] derives mainly from a successful technique to handle equality axioms and almost all the examples quoted in [M0W76] rely on this technique. For that reason, as long as our paramodulation module is not fully equipped with proper heuristics there is no fair comparison (the test cases were obtained with the full set of equality axioms and no special treatment for the equality predicate).

Finally among the very large systems which presently dominate theorem proving research is the system developed by R. Boyer and J.S. Moore at SRI [BM78]. Their system relies on powerful induction techniques and although some of the easier examples quoted in [BM78] could be proved by our system at present, a justifiable comparison is only possible once our induction modules are completed.

Example	Maryland Refutation Procedure			Markgraf Karl Refutation Procedure		
	NOC-P	NOC-G	G-P	NOC-P	NOC-G	G-P
Ances	19 (18)	62 (943)	0,306 (0,019)	7	7	1
Ew 33	19 (21)	63 (2585)	0,302 (0,008)	8	16	0,5
Prim	21 (20)	89 (221)	0,236 (0,09)	12	27	0,444
Wos 3	7 (7)	17 (154)	0,412 (0,045)	3	3	1
Wos 7	13 (12)	241 (244)	0,054 (0,049)	9	10	0,9
Wos 8	12 (12)	210 (360)	0,057 (0,033)	6	9	0,667
LS-17	20 (14)	98 (1273)	0,204 (0,011)	4	7	0,571
LS-21	12 (12)	252 (684)	0,048 (0,018)	7	12	0,583
LS-35	14 (14)	335 (1521)	0,042 (0,009)	8	9	0,889
LS-65	17 (17)	48 (880)	0,354 (0,019)	11	58	0,189
LS-115	13 (13)	20 (227)	0,65 (0,057)	7	11	0,636
LS-121	31	536	0,058	12	30	0,4

NOC-P = Number of Clauses in the Proof
NOC-G = Number of Clauses generated
G-P = G-Penetrance

$$G\text{-}P = \frac{NOC\text{-}P}{NOC\text{-}G}$$

Figure 5

A theorem prover based on heuristic evaluation was also reported by Slagle and Farrell [SF71] however it appears that such heuristics are not too successful for an ordinary resolution based prover.

5. CONCLUSION

At present the system performs substantially better than most other automatic theorem proving systems, however on certain classes of examples (induction, equality) the comparison is unfavourable for our system (section 4). But there is little doubt that these shortcomings reflect the present state of development; once the other modules (T-unification, paramodulated connection graphs, a far more refined monitoring, induction, improved heuristics etc) are operational, traditional theorem provers will no longer be competitive.

This statement is less comforting than it appears: the comparison is based on measures of the search space and it *totally neglecte* the (enormous) resources needed in order to achieve the behaviour described. Within this frame of reference it would be easy to design the "perfect" proof procedure: the supervisor and the look-ahead heuristics would find the proof and then guide the system without any unnecessary steps through the search space.

Doubtlessy, the TP systems of the future will have to be evaluated in totally different terms, which take into account the *total* (time and space) resources needed in order to find the proof of a given theorem.

In summary, although there are good fundamental arguments supporting the hypothesis that the future of TP research is with the finely knowledge engineered systems as proposed here, there is at present no evidence that a traditional TP with its capacity to quickly generate many ten thousands of clauses is not just as capable. The situation is reminiscent of todays chess playing programs, where the programs based on intellectually more interesting principles are outperformed by the brute force systems relying on advances in hardware technology.

REFERENCES

[AH72] B. Anderson, P. Hayes: An Arraignment of Theorem Proving or the Logicians Folly, Coop. Logic Memo 54, Univ. of Edinburgh

[BB75] N. Ballayntyne, W. Bledsoe: Autom. Proofs and Theorems in Analysis using nonstandard Techniques, ATP-23, 1975, Univ. of Texas

[BBH72] W. Bledsoe, B. Boyer, Hemmeman: Computer Proofs of Limit Theorems, J. Art. Intellig. vol 3, 1972

[BL71] W. Bledsoet Splitting and Reduction Heuristics in ATP, J. Art. Int., vol 2, 1971

BL77 W. Bledsoe: A maximal method for set variables in ATP, ATP-33, Univ. of Texas, 1977

[BN78] R.S. Boyer, J.S. Moore: A Computational Logic, SRI Intern., Comp. Sci. Lab., 1978

BT75 W. Bledsoe, M. Tyson: The UT Interactive Prover, Univ. of Texas, ATP-7, 1975

[DA78] J.L. Darlington: Connection Graphs and Fact Nets, Intern. Rep., GMD Bonn, I.S.T., 1978

[DA78] J.L. Darlington: Connected Fact Nets and Automatic Programming, GMD Bonn, I.S.T.,1978

CDA79] J.L. Darlington: A Net Based Theorem Proving Procedure for Program Verification and Synthesis, 4th Workshop on Art. Int., Bad Honnef, 1979

[DE71] P. Deussen: Halbgruppen und Automaten, Springer 1971

[DS77] P. Deussen, J. Siekmann: Neuantrag sum Forschungsvorhaben 'Untersuchung zur Einbesiehung mathematischen WIssens beim Automatischen Bewelsen am Beispiel der Automatentheorie*, DFG-Forschungsprojekt, As. De 238/1 1977

[DS79] P. Deussen, J. Siekmann: Zusatsantrag sum Forschungsvorhaben *Untersuchung sur Einbeziehung mathematischen WIssens beim Autom. Bewelsen am Beispiel der Automatentheorie*, DFG-Projekt De 238/1, 1979

[EI81] N. Eisinger: Subsumption and Connection Graphs, Universitlt Karlsruhe, 1981

[EP75] B. Epp: INTERLISP Programmierhandbuch, Institut far Deutsche Sprache, Mannheim

[HE72] C. Hewitt: Description and Theoretical Analysis Of PLANNER, AI-TR-258, MIT

[HU77] G. Huett Confluent Reductions: Abstract Properties and Applications to Tern Rewriting Systems, 18th IEEE symp. on Pound, of Coop. Scie., 1977

[K0753 R. Kowalski: A Proof Procedure based on Connection Graphs, JACM, vol 22, no 4, 1975

[IA77] D.S. Lankford, A.N. Ballantyne: Complete Sets of Pemutative Reductions, ATP-35, ATP-37, ATP-39, Univ. of Texas at Austin, 1977

[L078] D. Loveland: Automated Theorem Proving, North Holland, 1978

[N0N76] J. NcCharen, R. Overbeck, L. Nos: Problems and Experiments for and with Automated Theorem Proving Programs, IEEE Transae, on Computers, vol-C-25, no 6, 1976

[RRYKV72] Reboh, Raphael, Yates, Kling, Velarde: Study of Automatic Theorem Proving Programs, 1972, Stanford Research Institute, SRI-TN-75 Stanford

[SF71] J.R. Slagle, CD. Farrell: Experiments in Automatic Learning for a Multipurpose Heuristic Program, CACM vol 14, 1971

[SI81] J. Siekmann: Mathematical Reasoning as done by Man and Machine, Report (forthcoming), Universitat Karlsruhe, 1981

[SH76] R.P. Shostak: Refutation Graphs, J. of Art. Intelligence, vol 7, no 1, 1976

[SI76] S. Sickel: Interconnectivity Graphs, IEEE Trans, on Computers, vol C-25, no 6, 1976

[SB79] J. Siekmann, K. Bl&sius: Forschungsvorhaben zur Behandlung des Gleichheitspradikates beim Automatischen Beweisen, Universitat Karlsruhe, 1979

[SS813 J. Siekmann, G. Smolka: Selection Heuristics, Deletion Strategies and N-Level-Terminator Situations in Connection Graphs, Universit&t Karlsruhe, SEKI-Report, 1980

[SW79] J. Siekmann, G. Nrightson: Paramodulated Connection Graphs, Acta Informatica, 13, 1981

[SZ69] P. Szabo: The collected papers of G. Gentsen, North Holland, 1969

[WA79] C. Neither: Forschungsvorhaben zur Automatisierung von Induktionsbeweisen, Universityt Karlsruhe, 1979

[WM76] G. Wilson, J. Minker: Resolution, Refinements and Search Strategies; A Comparative Study, IEEE Transac. on Comp., vol C-25, no 8, 1976

[MA81] C. Neither: Elimination of Redundant Links in Extended Connection Graphs, Springer Fachberichte, vol 42, 1981

REFERENCES

References for the paper by J. Siekmann, P. Szabo: "Universal Unification and Regular Equational ACFM Theories'*, this volume.

[FA79] M. Fay: First Order Unification in an Equational Theory, Proc. 4th Workshop on Autom. Deduction, Texas, 1979

[G066] W.E. Gould: A Matching Procedure for unorder Logic, (thesis), Air Force Cambridge Research Labs, 1966

[H080] G. Huet, D.C. Oppen: Equations and Rewrite Rules, in: "Formal Languages: Perspectives and Open Problems", Ed. R. Book, Acad. Press, 1980

[HT80] G. Huet: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, JACM, vol 27, no 4, 1980

[HU8o] J.M. Hullot: Canonical Forms and Unification, Proc. of 5th Workshop on Autom. Deduction, Springer Lecture Notes, 1980

[HU80a] J.M. Hullot: A Catalogue of Canonical Term Rewriting Systems, Research Rep. CSL-113, SRI-International, 1980

[KB70] D. Knuth, P. Bendix: Simple Word Problems in Universal Algebras, in: Comp. Problems in Abstract Algebra. J. Leech (ed), Pergamon Press 1970

[K075] R. Kowalski: A Proof Procedure based on Connection Graphs, JACM, vol 22, no 4, 1975

[LA79] D.S. Lankford: A Unification Algorithm for Abelian Group Theory, Rep. MTP-1, Louisiana Techn. Univ., 1979

[LB79] D. Lankford, M. Ballantyne: The Refutation Completeness of Blocked Permutative Narrowing and Resolution, 4th Workshop on Autom. Deduction, Texas, 1979

[PS81] G. Peterson, M. Stickel: Complete Sets of Reductions for Equational Theories with Complete Unification Algorithms, JACM, vol 28, no 2, 1981

[PL72] G. Plotkin: Building in Equational Theories, Machine Intelligence, vol 7, 1972

[RS78] P. Raulefs, J. Siekmann: Unification of Idempotent Functions, Univ. Karlsruhe, 1978

[RSS79] P. Raulefs, J. Siekmann, P. Szabo, E. Unvericht: A short Survey on the State of the Art in Matching and Unification Problems, SIGSAM Bulletin, 13, 1979

[SL74] J. Slagle: ATP for Theories with Simplifiers, Coromutativity and Associativity, JACM 21, 1974

[SS81a] P. Szabo, J. Siekmann: A Minimal Unification Algorithm for Idempotent Functions, Univ. Karlsruhe (forthcoming 1981)

[SS8lb] P. Szab6, J. Siekmann: Universal Unification, Univ. Karlsruhe (forthcoming 1981)

[SS81c] J. Siekmann, P. Szabo: Universal Unification and Regular ACFM Theories, Univ. Karlsruhe, Research Report, 1981

[VA75] J. van vaalen: An Extension of Unification to Substitutions with an Application to Automatic Theorem Proving, IJCAI-4, Proc. of 1975

[WF73] L. Nos, G. Robinson: Maximal Models and Refutation Completeness: Semidecision Procedures in Autom. Theorem Proving,in: Word problems (W.W.Boone, F.B.Cannonito, R.C. Lyndon, eds), North Holland, 1973