an input tape, and an unbounded amount of storage in the form of a pushdown stack. The input tape has k read-only heads communicating with the finite control. The pushdown stack has a single read-write head communicating with the finite control. The stack head may read anywhere in the stack. However, symbols may be added or deleted only at the top of the stack. A number of variations on this basic model have been studied. The principal variations considered in this paper are non-erasing stack automata, checking stack automata, auxiliary stack automata, and auxiliary pda's. In non-erasing stack automata, the device is constrained never to delete a symbol from the stack. In a checking stack automaton the device must do all its writing before it does any reading in the stack. Auxiliary stack automata are provided with a bounded amount of general purpose storage, in addition to their stack. Auxiliary pda's are similar to auxiliary stack automata. In all cases, both deterministic and non-deterministic models are considered.

This paper is a complete and detailed summary of the equivalences among the above devices and of the equivalences between these devices and both tape-bounded Turing machines and time-bounded Turing machines. Many of the results were already known for the case where the automaton has only a single input head. However, the proofs do involve techniques other than straightforward generalizations of the methods used for the single-head case. The following two equivalences are representative of the kinds of results presented. In both cases n refers to the length of the input.

A language is accepted by a deterministic (or non-deterministic) k-head stack automaton if and only if it is accepted by some deterministic ($n^k \log n$)-tape-bounded auxiliary pda (or deterministic n^{2k}-tape-bounded auxiliary pda respectively).

A language is accepted by a deterministic (or non-deterministic) non-erasing stack automaton if and only if it is accepted by an ($n^k \log n$)-tape-bounded deterministic Turing machine (or n^{2k}-tape-bounded non-deterministic Turing machine respectively). WALTER J. SAVITCH

J. A. ROBINSON. *Automatic deduction with hyper-resolution.* ***International journal of computer mathematics,*** vol. 1 no. 3 (1965), pp. 227–234.

With the intention of providing the basis for more efficient computer programs to find proofs for theorems from various fields of mathematics, the author in the paper under discussion introduces a refinement of resolution (see XXXI 515), *hyper-resolution.* The clause C is a P_1-*resolvent* of the clauses A and B if C is a resolvent of A and B and either A or B contains no negative literals. A P_1-*deduction* is one in which all of the clauses which are both in the deduction and obtained by resolution are P_1-resolvents of earlier clauses therein. The clause C is a *hyper-resolvent* of the clauses $A_1, A_2, \cdots, A_k$ and B if: (1) $A_1, \cdots, A_k$ and C contain no negative literals; and (2) there exists a sequence $B_1, B_2, \cdots, B_k$ with B_1 a P_1-resolvent of A_1 and B, B_2 a P_1-resolvent of A_2 and $B_1, \ldots, B^k = C$ a P_1-resolvent of A_k and B_{k-1}.

The key theorems of the paper are Theorems 2 and 3. Theorem 2 states that a finite set S of clauses is unsatisfiable if and only if there exists a P_1-deduction of the empty clause from S. (A deduction of the empty clause is equivalent to the mathematician's proof by contradiction.) Theorem 3 states that a finite set S of clauses is unsatisfiable if and only if there exists a deduction by hyper-resolution of the empty clause from S, i.e., a deduction in which all resolvents therein are hyper-resolvents.

The author suggests that hyper-resolution can be accomplished by generating a family of sets of P_1-resolvents and then discarding from each set all clauses containing at least one negative literal. Such a procedure would result in a number of computations being repeated many times. The author concludes that, since the use of hyper-resolution would result in a sharp reduction in the number of retained clauses during proof search, a program based on hyper-resolution would be much more efficient than those already in existence. Efficiency in automated theorem proving, however, can be measured in, among other ways, the computer memory required or the time required to obtain a proof. It seems clear that, for many theorems, the memory required to complete the proof search would be sharply reduced if resolution were replaced by hyper-resolution. However, for two reasons the corresponding reduction in the time required might not occur. First, there is the potentially serious problem of computational redundancy referred to earlier. Secondly, for a number of theorems, a hyper-resolution-based theorem-proving program will be forced much deeper into the space of inferences before a proof is found.

With the introduction of computers with large memories, time rather than space is the more significant measure of efficiency in automated theorem proving. However, since a sharp reduction in clause retention has an eventual indirect effect on running time, hyper-resolution should be implemented and studied. Since computations by hand with sets consisting of but a few clauses can be quite misleading, the value of a strategy or inference rule can be judged only by experiments with a computer program. L. Wos

J. A. Robinson. *A review of automatic theorem-proving.* ***Mathematical aspects of computer science,*** Proceedings of symposia in applied mathematics, vol. 19, American Mathematical Society, Providence 1967, pp. 1–18.

Robinson begins with a brief review of the language, theorems, and structures underlying the field of automated theorem proving. About Prawitz he says that Prawitz was the first to observe that, given a finite set C of formulas, the question whether or not there exists a substitution θ such that $C\theta$ is truth-functionally unsatisfiable can be settled by an algorithm which, appropriately, computes θ or indicates no such substitution exists; the computational problem of Prawitz-like methods is that the number of unifiable partitions rapidly increases with an increase in the number of atoms present.

Robinson then turns to resolution-type procedures and introduces the concept of a *clash.* A clash is a finite set of ground clauses $A_1, A_2, \cdots, A_n, B$ such that: B contains at least $n > 0$ literals, $L_1, L_2, \cdots, L_n$; and for $1 \leqq i \leqq n$, A_i contains the complement $\bar{L}_i$ of L_i but does not contain the complement of any other L_j nor the complement of any other literal of B or of A_j, $1 \leqq j \leqq n$. The unique resolvent of the clash $[A_1, \cdots, A_n, B]$ is the clause $(A_1 - \{\bar{L}_1\}) \cup \cdots \cup (A_n - \{\bar{L}_n\}) \cup (B - \{L_1, \cdots, L_n\})$. He then proves the quite pleasing lemma: If the finite set K of ground clauses is truth-functionally unsatisfiable, then either K contains the empty clause or there exists a clash N contained in K whose resolvent is not in K. Two of the author's remarks in connection with this lemma are in error. First, if the model in the proof of the lemma is not chosen correctly, even when A is the set of positive clauses, the author is incorrect in stating that the resolvents will be hyper-resolvents. The counterexample is the set $K = \{\{P\}, \{Q\}, \{-P\}, \{-Q, R\}\}$ with the model of true assigned to P, Q, and R, for the clause $-R$ will be a resolvent. Secondly, it is incorrect to imply that, in the proof of the lemma, the maximally clash-free A can always be chosen to be the set of positive clauses. The counterexample is $K = \{\{P, Q, -S\}, \{P, -Q, -S\}, \{-P, Q, -S\}, \{-P, -Q\}, \{S\}\}$.

If a set N of clauses is such that there exists a substitution θ with $N\theta$ a clash, then N is a *latent clash.* The main result of the paper is the latent clash theorem which states that the set C is latently truth-functionally unsatisfiable if and only if there exists a deduction by means of latent clashes of the empty clause from C, where C is a finite set of clauses having no variables in common. The sufficiency, as stated, of the theorem is false and it should instead read: If there exists a deduction by means of latent clashes of the empty clause, then some finite set, having no variables in common, of copies of C is latently truth-functionally unsatisfiable. Although the use of latent clashes is of questionable computational value, especially if the author's suggestion of computing by means of binary resolution is followed, he makes the excellent suggestion that advantage be taken of the generally correct statement that the latent clash can be computed in $M!$ ways. In fact, for the so-called unit section of theorem-proving programs, it is virtually mandatory to implement the correspondent of Robinson's suggestion and thereby avoid the generation of all but one of the $M!$ in general identical clauses inferable from a non-unit clause when resolved upon with n units.

The most significant contribution of the paper is the author's astute suggestion that computational advantage be taken of the relations of inclusion in set theory and, more importantly, of equality. L. Wos

V. F. Turchin and V. I. Serdobol'skiĭ. *The REFAL language and its use in transforming algebraic expressions.* English translation of XXXV 349. ***Cybernetics*** (New York), vol. 5 (for 1969, pub. 1972), pp. 307–312.