Automatic Theorem Provers

Felix Wruck

June 27, 2017

1 Introduction

In this work an overview of the history and state of the art of automatic theorem proving is given. The early work of Hilbert, Herbrand and others is discussed, as well as the ground resolution, DPLL and Robins’s resolution method. Afterwards some of the most impacting results will be discussed in more detail. At the end ‘E’, a current, powerful automatic theorem proving program (ATP) will be discussed along with the concepts it uses. Also the foundations of those concepts will be presented.

2 Historical Overview

Today Automatic Theorem Provers are fairly powerful, e.g. being able automated proof of the Robins Conjecture. However until achieving this power, a long chain of scientific work had to be done. Furthermore there is still a lot of work to be done in the area of automated theorem proving. The current ATP are still far from being able to prove many complex theorems. This section tries to give a short overview over the scientific results in the area of automatic theorem proving that most influenced subsequent work. The concepts mentioned in this section will be explained in more detail their respective sections below.

The history of the idea of mechanizing mathematical reasoning and proof finding sartet with Gottfried Wilhelm Leibniz (1646 - 1716). He had the concept of ”machanizing reason” [Plaisted2015]. However, the theoretical and technical means necesarry to make this concept reality did not yet exist in his time. After a long spare period the next important milestones in the automatization of matematheical reasoning were achieved by Frege, Russell, Whitehead and Hilbert in the years around 1900. Frege developed a formal system, ”which, in effect, constituted the first predicate calculus”
This achievement is a milestone, because in order to be able to automate reasoning, one needs a logical framework, wherein the automatic reasoning can be done. In this context one has two main goals: the logical framework has to be sufficiently expressive in order to express all the mathematical reasoning necessary for proving theorems. Furthermore in order to do automatic theorem proving it is necessary to have an automatizable procedure for finding proofs. These two goals are in some way opposed, because in less stronger logics it is may easier to find an automatizable decision procedure, however this procedure is not of much use, if its expressibility is limited. On the other hand strong logics may are able to encode nearly every mathematical reasoning, however if they make it too complicated to find an automatizable proving procedure they are not suitable as a logical foundation of ATPs. In this context first-order logic provides a good tradeoff between expressability and the ability to automatize proof finding. Roughly at the same time as Frege, Russell and Whitehead tried to construct a universal formalization of mathematics, their 'Principia Matematica'. Hilbert sought to find the so called "Hilbert Program", a finite set of axioms along with a consistence proof in order to form the foundation of mathematics. But it was shown by Gödel, that this goal was impossible to achieve. However the result of Gödel did not lead to resignation in research efforts towards automatic theorem proving. The research in the area of automatic theorem proving in the 1920s and afterwards focused on theorems and reasoning expressed in first-order logic. This is because, as stated above, first-order logic provides a basis for automatization efforts. In the 1920s and 1930s Skolem and Herbrand worked on automatic proving of FO-formulas, despite "there is no general decision procedure for first-order logic"[Davis2001, p.7]. They provided basic theoretical result enabling the further development of automatized theorem proving: The Skolem Functions, Herbrand's Theorem, between others [Davis2001]. In the subsequent years, the first automatic theorem provers were built. Two of these first provers were the ones of Gilmore (1960) and Prawitz and Voghera (1960), published in 1960. These first provers, like many others until nowadays, were based on a search of the Herbrand universe, but did not yet use Skolem functions. [Davis2001]. These first ATP were not very powerful yet and just able to prove simple statements. Building up on these first provers, Davis and Putnam published one of the papers with most impact in the area of ATP until now. In 1958 they developed a solution method for logical formulas in conjunctive normal form, a method on top of which many ATP have been build. Many other authors have built on top of
their method called ”ground resolution”\(^1\). Among other points they noted that one problems of the first provers was the reliance on disjunctive normal form for solving logic formulas. Also they assumed, that the most central problem was the unavailability of effective solving procedures for huge logic formulas, which later turned out to be not the only problem to overcome. [Davis2001] The next very central proceeding was made by Logemann and Loveland four years later. They noted that the rule for ground resolution was too unefficient in terms of space complexity and they proposed to use their ”splitting rule” instead. This led to the satisfiability testing procedure called ”DPLL”, that is used until now. However with this approach there are still unnecessary terms of the Herbrandt universe being examined. This was noted by Pravitz in 1960. He proposed a algorithm ”that did not generate elements of the Herbrandt universe until needed” [4, p. 9]. This led to great performance improvements in automatic proving. However, the proposed algorithm led to huge expansions in the generated formulas, so it was not yet possible to prove more complicated theorems. In the following years further improvements were made by combining the aforementioned results. A further milestone was Robinson’s resolution procedure in 1965. Instead of having multiple rules of inference he found a single rule of inference to check the unsatisfiability of a logic formula. The rule he found was ”complete for first-order logic” [4, p. 11]. Since then many automatic theorems provers have been implemented and further research has been done, but it would exceed the intention of this work to mention them all. Despite the lot of work done since then, the results above form the basis of automated theorem proving until now.

3 Basic terms

3.1 Propositional logic normal forms

A formula is in conjunctive normal form if it has the form \((x_1 \lor \ldots \lor x_n) \land \ldots \land (y_1 \lor \ldots \lor y_n)\). It is said to be in disjunctive normal form if it has the form \((x_1 \land \ldots \land x_n) \lor \ldots \lor (y_1 \land \ldots \land y_n)\).

\(^1\)Their research was supported by the United States NSA and the original report is still unpublished.[Davis2001]. Their results were first published in [6] two years after their unpublished report.
3.2 Interpretations and Instances

An instance $I$ of a formula $F$ is a formula obtained by replacing some or all variables in the formula by concrete interpretations. For example take the inference rule:

All $X$ are $Y$

$a$ is $X$

Therefore $a$ is $Y$

Now we take the interpretations:

$X$: Athenians

$Y$: mortal

$a$: Socrates

and substitute the variables $X$, $Y$ and $a$ by the interpretations mentioned above. By doing so we obtain the instance:

All Athenians are mortal

Socrates is Athenian

Therefore Socrates is mortal.

When a formula contains no variables it is "ground". Accordingly a instance not containing any variables is called ground instance. Loosely speaking an instance is obtained by replacing the variables by other terms. An interpretation assigns meaning to the formula. An interpretation satisfies a formula if the application of the interpretation the formula results in a valid formula for all ground instances of the original formula. A formula is valid if all its interpretations satisfy the formula.

3.3 First-order logic normal forms

A FO-formula is in prenex normal form if it consists in a prefix containing only quantified variables and a second part free of quantifiers. Every first-order formula has an equivalent formula in prenex normal form and it exists an algorithm for transforming any FO-formula into prenex normal form [10].

E.g. the following formula is in prenex normal form:

$$\exists w \forall x \forall y \exists z P(x) \land (Q(w) \lor Q(z))$$
But the following formula is not:

$$\exists w \forall x \forall y P(x) \land (Q(w) \lor Q(z) \rightarrow \exists z R(z))$$

### 3.4 Skolemization

A Formula F is in skolemized form if it is in prenex normal form and contains only universal quantifiers. The elimination of existential quantifiers in a formula is called skolemization. A skolemized formula is not necessarily equivalent to the original formula, but Skolemization preserves satisfiability. Skolemization is very important in automatic theorem proving. It reduces the complexity of the proof finding procedure significantly, because "if existential quantifiers are eliminated in favor of Skolem functions at the outset, instead of systems of equations, one has the simple problem of unifying pairs of terms" [12]. Because of this, nearly every current ATP uses skolemization as the first step in the procedure of automated proof finding.[10] Intuitively, the advantage using skolemization is, that it helps to translate FO-formulas into predicate logical formulas, while preserving satisfiability. The predicate logic like representation can then be used to actually search a proof for the given theorem.

### 4 Herbrand’s theorem and the Herbrand Universe

The Herbrands Theorem exists in various versions [2]. The version discussed here allows the reduction of the unsatisfiability problem for FO-formulas to the unsatisfiability problem for formulas in propositional logic. Thus it allows to prove (by negation) the validity of a theorem expressed in first-order logic by checking the unsatisfiability of propositional logic formulas:

A quantifier-free formula is first-order satisfiable iff all finite sets of ground instances are (propositionally) satisfiable

Herbrand Universes are defined over first-order languages $L_{\sigma}$. $\sigma$ is a vocabulary containing e.g. constants, function symbols, etc. The first-order language $L_{\sigma}$ contains all sentences (first-order logic formulas) that can be built by using the vocabulary $\sigma$. The Herbrand Universe H over $L_{\sigma}$ is the set of all ground terms that can be built over $L_{\sigma}$. If $n$ consants are contained in $L_{\sigma}$ a constant $c$ is added. If for example $\sigma = \{\text{d, f(x)}\}$ then the Herbrand Universe $H(L_{\sigma})$ is $\{\text{d, f(x)}$,
Intuitively, the Herbrand Universe over a set of FO-Terms is the set of all FO-Formulas not containing any variables that can be built using the given terms.

5 1960: Resolution (Davis and Putnam)

The resolution method was published by Davis and Putnam in 1960 [6]. It is an algorithm designed to decide if a given first-order-logic formula in conjunctive normal form is valid. A FO-formula is valid, if it is true for all its interpretations. Equivalently, it is valid iff its negation is unsatisfiable, which is what the Davis-Putnam algorithm proves. In order to proof the unsatisfiability of a FO-formula the algorithm uses Herbrand’s theorem. As said above, the theorem states that that for any unsatisfiable formula it exists an unsatisfiable ground instance. Exploiting this fact, the procedure searches for an unsatisfiable ground instance of a formula and if it finds one, by Herbrand’s theorem, it has found a proof for the unsatisfiability of the whole formula. In order to find such an unsatisfiable ground instance, the algorithm first converts the FO-formula in prenex normal form, generates all ground instances and checks for each instance if it is satisfiable. It only terminates on valid formulas, as the set of valid formulas is recursively countable. In the case of a theorem expressed as a FO-formula showing that the formula is valid means to prove the theorem. The satisfiability-check works by successively replacing a formula $F$ by a formula $F'$, satisfying the condition that $F$ is valid iff $F'$ is valid. This is done through the following rules:

1. Rule for the Elimination of One-Literal Clauses / Unit Propagation
2. Affirmative-Negative Rule
3. Rule for Eliminating Atomic Formulas

By just applying the third rule, one is able to successively reduces the amount of literals and thus to deduce the empty clause. The rule 3 is the central rule and it is the only one that is strictly necessary in order to achieve completeness [10]. However the other two rules have a positive impact on performance. In the following $F$ is a FO-Formula, $F'$ a formula derived from $F$ by applying the beforementioned rules, $c_0, ..., c_n$ are clauses of $F'$ and $p$ is an atomic formula. Rule 1 has the following cases:

(a) If $F$ contains $p$ and $\neg p$ as one-literal clauses, then $F$ is unsatisfiable, thus the unnegated formula is valid.
(b) If 1 is not the case, and if there is a one-literal-clause $p$, delete all clauses containing $p$ and delete $\neg p$ from all remaining clauses, obtaining $F'$.

(c) This case is the negation of 2

In cases (b) and (c) $F'$ is unsatisfiable iff $F$ is unsatisfiable. If $F'$ contains the empty clause it is unsatisfiable and thus the unnegated formula is valid. If $F'$ itself is empty, then it is valid and thus also satisfiable.

Rule 2 states, that if $F$ contains $p$ either only in positive form or only in negative form, we can delete all clauses from $F$ containing $p$ or $\neg p$ respectively, obtaining $F'$. As above, $F'$ is unsatisfiable iff $F$ is unsatisfiable. If $F'$ is empty then the unnegated formula is valid.

Rule 3 is special in the way, that it is the only of the three rules that can increase the size of the formula. Therefore it is only applied after exhaustively applying rule 1 and 2. The rule 3 replaces a formula $F$ of the form $(A \lor p) \land (B \lor \neg p)R$ by $F' = (A \lor B) \land R$. (A, B and R are free of p). Satisfiability of $F'$ is as above.

For valid formulas the resolution algorithm terminates for the negated counterpart.

5.1 1965: First order resolution (Robinson)

In 1965 Robinson published a new form of resolution, based on Davis and Putnam’s approach and taking into account results provided by Pravitz. His resolution procedure needed one single rule of inference in order to achieve completeness for proving unsatisfiability of first-order logic formulas. In his original publication Robinson proposed to construct "a sequence of finite subsets $P_0, P_1, \ldots$ of the Herbrand Universe $H$ of $S$, such that $P_j \subseteq P_{j+1}$ for each $j \geq 0$, and such that $\bigcup_{j=0}^{\infty} P_j = H$" [13] Then successively each of these sets is checked for satisfiability. If any of the subsets is not satisfiable, then by Herbrand's theorem the whole formula is not satisfiable. The procedure is based on the version of Herbrand's theorem cited above. It uses unification in order to directly resolve the most general ground instances of the formula directly [10]. The Unification technique was also developed by Robinson and published together with his resolution procedure. Unification is used to find "appropriate" [10] instantiations of the formula that is analyzed. It is defined as follows [10]:

Given a set of pairs of terms

$$S = (s_1, t_1), \ldots, (s_n, t_n),$$
a unifier of the Set $S$ is an instantiation $\sigma$ such that

$$t_{\text{subst}}\sigma s_i = t_{\text{subst}}\sigma t_i$$

for each $i = 1, \ldots, n$. In the special case of a single pair of terms, we often talk about a 'unifier of $s$ and $t$', meaning a unifier of $(s, t)$.

Considering for example the following formula in CNF containing two clauses [10]:

$$P(x, f(y)) \lor Q(x, y), \neg P(g(u), v)$$

In the Davis-Putnam method one would now successively create more and more instances and enumerate them. Opposed to this approach, using unification the following instantiations are chosen s.t. "$P(x, f(y))$ and $\neg P(g(u), v)$ become complementary" [10]:

$$P(g(u), f(y)) \lor Q(g(u), y), \neg P(g(u), f(y))$$

In this situation one can then by the resolution rule derive the clause

$$Q(g(u), y)$$

cf. [10].

The first-order resolution rule replaces the clauses:

$$p_1 \lor \ldots \lor p_j \lor \ldots \lor p_m, q_1 \lor \ldots \lor \neg q_k \lor \ldots \lor q_n$$

by

$$\text{Subst}(\theta, p_1 \lor \ldots \lor p_{j-1} \lor p_{j+1} \lor \ldots \lor p_m \lor q_1 \lor \ldots \lor q_{k-1} \lor \ldots \lor q_n)$$

resolving $p_j$ ans $q_k$ where $F_1$ and $F_2$ don’t have variables in common. $\theta$ denotes the most general unifier of the two clauses, cf. [3].

An algorithm can be constructed using unification and the resolution rule for first-order logic as follows [3]:

At the beginning the initial state is defined by taking together the negated, original theorem and the axioms all in expressed as first-order logic formulas in CNF. Then the algorithm repeatedly picks two clauses from the current search space, constructs a unifier and adds the derived clause to the search
space. After each application of the resolution rule for first-order logic the algorithm checks if the empty clause has been derived and so the formula has been shown to be unsatisfiable. This resolution procedure turned out to be very efficient and forms the basis of many current automatic theorem provers.

6 Example: E

E is an automatic theorem prover written by Stephan Schulz at TUM. It is among the best ATPs available today and performed well in various competitions. E can operate in fully automatic mode or accept input parameters used to control the proof finding process, which in automatic mode are chosen automatically based on heuristics. In this section E will be presented along with the concepts it uses. E accepts a set of axioms and a theorem to be proved assuming the axioms, both given as a formula in first-order logic. The proving process has four phases. The first phase is to parse the given theorem and axioms and translate it into a set of clauses and formulas. E also negates the given theorem. The negation allows it to check the theorem for unsatisfiability instead of validity, because for a formula \( \phi \) is valid if and only if \( \phi \) is unsatisfiable. E checks the resulting set of clauses for unsatisfiability and if it indeed is unsatisfiable, a proof by contradiction has been found.

After reading the input the second step E optionally performs is relevancy pruning, which can discard "clauses and formulas deemed unlikely to contribute to a proof" [14, p. 1]. In the third phase the theorem is converted into conjunctive normal form using the algorithm by Nonnengart and Weidenbach. The fourth phase does preprocessing of the formula. Preprocessing includes elimination of clauses that are tautologies, that is containing \( p \lor \neg p \) for some literal \( p \), and pure literals. Also preprocessing can "expand equational definitions" [14, p. 1] The fifth phase is the real proof-finding phase, checking the set of clauses representing the theorem for unsatisfiability. E’s unsatisfiability check is based on the superposition calculus which among others is based on the resolution method by Robinson discussed earlier. During the search for a proof E can store information about the steps performed, which enables E in the sixth and last phase to construct a tree of steps performed to find the proof which can be used to verify the proof.
6.1 How E proves theorems: The superposition calculus

Kalkül für gleichheit, paramodulation beschreiben-harrison buch E finds proofs using the superposition calculus. This calculus is a combination of the resolution method of Robinson and the Knuth-Bendix completion and is refutation complete. Refutation completeness is the property, that if a set of clauses is not valid, a contradiction can be derived from the set of clauses using the rules of inference that are part of the calculus. This means, that if a theorem is provable by contradiction the proof can be found using the superposition calculus. E is also able to detect when all non-redundant applications of rules of inference to the set of clauses have been made and terminates in such an event. When all non-redundant applications of rules of inference have been made, the resulting saturated set describes a model (satisfying interpretation) and so the negated formula can’t be unsatisfiable. During the fifth execution phase of E, the proof finding phase, E applies the mentioned rules of inference seeking to derive a contradiction starting with the set of clauses. The current working set of clauses is called the ‘search state’. During the application if the rules of inference it is critical
to maintain the current set of clauses as compact as possible at all times. In order to achieve this, E applies various simplification rules, e.g. the subsumption check. This means that when E infers a clause it checks, if it subsumes any clauses already in the current search state. If there are such clauses that can be deleted, because they are subsumed in the current clause. The amount of simplification rules applied by E (more than 10) gives an idea their importance for the performance of the theorem prover. These rules in order to be useful preserve the unsatisfiability of a set of clauses. Besides the theoretical model a actual algorithm is required to apply the rules correctly and efficiently. In order to achieve completeness it is necessary to consider all possible combinations of clauses to infer new clauses using the inference rules. For the superposition rule of inference all combinations of two clauses have to be considered. For the other rule of inference ”typically a single clause” [14]. An algorithm satisfying this property is the given-clause-algorithm. This algorithm keeps track of the current state $S = P \cap U$ of the proof. $P$ is the set of clauses between whom all possible inferences have been generated, $U$ is the current set of clauses not in $P$ The basic algorithm is the following:

```plaintext
while $U \neq \{\}$
    $g = \text{delete best}(U)$
    if $g = \square$
        SUCCESS, Proof found //the empty clause, a contradiction, has been generated
    $P = P \cup \{g\}$
    $U = U \cup \text{generate}(g, P)$
SUCCESS, original $U$ is satisfiable //all possible rules have been applied without generating a contradiction
```

However the plain algorithm does not yet apply the simplification rules. Because of this it is necessary to extend the simple form of the algorithm in order to take these rules into account. This extension is called ‘DISCOUNT Loop’ [15].
while $U \neq \{\}
    g = \text{delete best}(U)
    g = \text{simplify}(g, P)
    \text{if } g == \square
        \text{SUCCESS, Proof found}

\text{if } g \text{ is not subsumed by any clause in } P
    // (or otherwise redundant w.r.t. } P
    T = \{c \in P | c \text{ redundant or simplifiable w.r.t. } g\}
    P = (P \setminus T) \cup \{g\}
    T = T \cup \text{generate}(g, P)
    \text{foreach } c \in T
        c = \text{cheap simplify}(c, P)
        \text{if } c \text{ is not trivial}
            U = U \cup \{c\}
    \text{SUCCESS, original } U \text{ is satisfiable}

The main loop proceeds as follows: First delete_best(U) deletes the clause ”with the best heuristic evaluation” [14] from the set of unprocessed clauses. This clause will be processed in this run of the while loop. Next the set of clauses is simplified using the beforementioned simplification rules. A check follows, wether the empty clause has been derived or not. Then E checks, if any yet processed clause is subsuming the current clause. In this case the run on this clause would be redundant and E continues directly with the next clause, without performing any further operation on this clause. If the current clause is not redundant, E deletes all clauses in the set P of processed clauses, that are subsumed by the current clause, because they are redundant. Afterwards a temporary set T is created, that contains all clauses $c \in P$ that ”can be simplified using g” [14]. This set is deleted from P and g is added to P. Then generate() applies the rules of the superposition calculus in order to derive new clauses from g and P. The generated clauses are added to tT. As the last step each clause in c is simplified using the already processed clauses and if the result of the simplification is non-trivial the resulting clause c is added to the set of unprocessed clauses U again. In this extended version with the DISCOUNT loop the algorithm takes into account both, rules of inference and simplification rules. This is a sketch of the algorithm that is used by E to automatically generate proofs.
6.2 Complexity of proof generation

After discussing the theory behind E and the algorithm that E uses for proof generation, this section discusses the complexity of automatic theorem proving. This last section aims to give an intuitive idea of why it is so hard to automatically generate proofs for theorems. Statistical data over E’s performance can e.g. be found in [14]. The following example by Schulz [15, p. 20] makes it visible that automatic theorem proving is a hard problem. The data is an example of one current state during one run of E on a concrete theorem:

- Initial clauses: 160
- Processed clauses: 16,322
- Generated clauses: 204,436
- Paramodulations: 204,395
- Current number of processed clauses: 1,885
- Current number of unprocessed clauses: 94,442
- Number of terms: 5,628,929

Figure 2: cf. [15, p. 20]

The initial theorem consists of 160 clauses. During the proof procedure a total of more than 204,000 clauses have been generated up to the point in time the data was captured. More than 5 millions of terms have been generated so far. Nearly 95,000 clauses are still to be processed currently. This shows that for proofs of hard theorems millions of clauses and terms have to be generated and processed. For the processing tens of thousands of applications of rules have to be done and rules like the subsumption rule have to consider tens of thousands of candidates [14, p. 20] Schulz therefore points out that need for good heuristics for the selection of clauses and on the website of the E theorem prover Schulz states that one of his research interests is the work on these heuristics. IN resume it can be said that a big challenge for E and other automatic theorem provers is the huge amount of clauses and terms generated and processed during the attempt to find a proof for a particular theorem.
7 Successes of Automatic Theorem Proving

The most famous success of automated theorem provers was the fully automatic proof of the Robbins Conjecture. This conjecture was postulated in 1933 and was an open problem until 1996, when William McCune found a proof for it using the ATP EQP. The prover needed 172 hours to find the proof and the final proof is 6182 lines long [9]. Besides this outstanding result there are many other theorems that have been automatically proved, refer to the TPTP library for more examples.

8 Conclusion

In this work an overview of the history of automatic theorem proving has been given. The overview started with the first ideas to mechanize mathematical reasoning by Leibniz. The work on foundations by Hilbert, Herbrandt and other has been presented. The as further central work the very impactful publications by Davis, Putnam, Logemann, Loveland and Robinson have been mentioned and have been discussed in more detail and lastly the state-of-the-art theorem prover E has been presented. It became clear, that the current power of automatic theorem provers was made possible by a large chain of scientific work building up one onto the other. In state of the art provers like E the legacy of early work in this area is still visible. But it also has to be kept in mind that despite successes like the automatic proof of the Robinson’s Conjecture there is still a lot of research to be done in the area of automatic theorem proving in order to make ATP’s more powerful.
References


