Topics in Automated Theorem Proving

Course (236714, 2013/14)

Johann A. Makowsky*

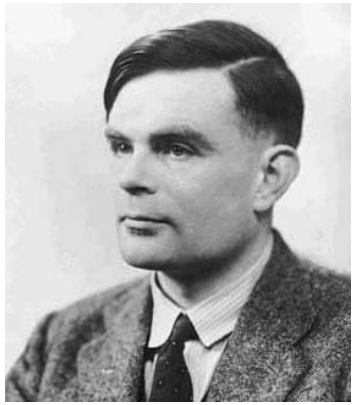
* Faculty of Computer Science,
Technion - Israel Institute of Technology,
Haifa, Israel
janos@cs.technion.ac.il

Course homepage
http://www.cs.technion.ac.il/~janos/COURSES/THPR

The consequence relation of First Order Logic is semi-computable (1929), but not computable (1931, 1936)

Kurt Gödel, 1906-1978

Alonzo Chruch, 1903-1995



Alain Turing, 1912-1954

-

The basis of Automated Theorem Proving (ATP)

Thoralf Skolem, 1887-1963

Jacques Herbrand, 1908-1931

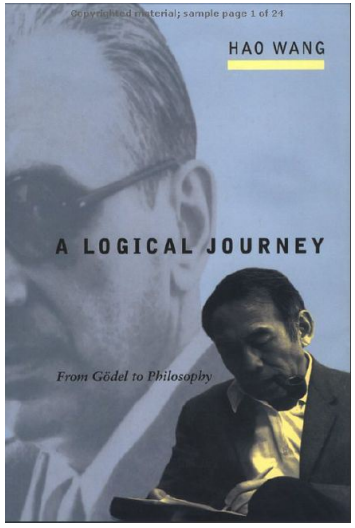
- Skolem, Thoralf (1920), Logisch-kombinatorische Untersuchungen ber die Erfllbarkeit oder Beweisbarkeit mathematischer Stze nebst einem Theoreme ber dichte Mengen, Videnskapsselskapet Skrifter, I. Matematisk-naturvidenskabelig Klasse 6: 136

- J. Herbrand: Recherches sur la theorie de la demonstration.
  Travaux de la Societe des Sciences et des Lettres de Varsovie, Class III, Sciences Mathematiques et Physiques, 33, 1930.

Skolem's and Herbrand's Theorem

- Skolem 1920: Every countable set of sentences which has a model has a countable or finite model. In fact, if the set is in Skolem Normal Form, it has a term model.
- Herbrand 1930: Used Skolem's theorem to reduce First Order Logic to propositional Logic.
- Herbrand's Theorem was syntactical and can be viewed as a completeness theorem for cut elimination.
- One can give a purely model theoretic proof of this.

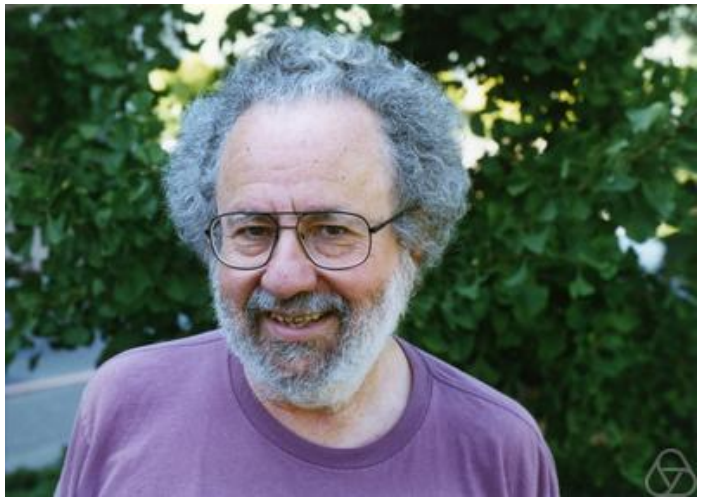
Early Automated Theorem Proving



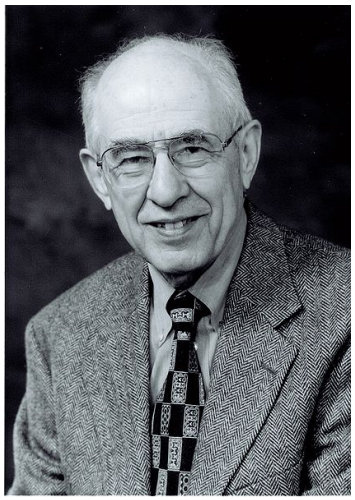
Wang, Hao (January 1960).
Toward Mechanical Mathematics.
IBM Journal of Research and Development,
4:1 (January 1960), 2-22

Hao Wang, 1921-1995

Resolution and the Davis-Putnam Procedure



Martin Davis, * 1928



Hilary Putnam, * 1926

- Davis, Martin; Putnam, Hillary (1960).
  A Computing Procedure for Quantification Theory.
  Journal of the ACM 7 (3): 201-215

- Davis, Martin; Logemann, George, and Loveland, Donald (1962).
  A Machine Program for Theorem Proving
  Communications of the ACM 5 (7)

Resolution, the propositional case, I

In this lecture we introduce a syntactic method of checking whether a set of formulas Σ is satisfiable called resolution.

Resolution is a machine friendly method which involves some preprocessing transforming the set $\Sigma \subseteq$ WFF into a set $clause(\Sigma)$ of clauses.

Clauses, I

- A literal is either a propositional variable, p_i,
  or the negation of a proopositional variable, $\neg p_i$.
  The constant F is also a literal. We denote literals by l_j.

- A clause is a finite set of literals $\{l_1, l_2, ..., l_k\}$.
  We denote clauses by C_j.
  We denote the empty clause (the empty set of literals) by $\Box$.

- Let z be a propositional assignment.
  The meaning function M_{clause} for clauses is defined inductively as follows:

  Basis: $M_{clause}(\{l_j\}, z) = M_{PL}(l_j, z)$. $M_{clause}(\Box, z) = 0$.

  Closure: $M_{clause}(\{l_1, l_2, ..., l_k\}, z) = max\{M_{clause}(\{l_j\}, z) : j \leq k\}$.
  If S is a set of clauses $M_{clause}(S, z) = min\{M_{clause}(C, z) : C \in S\}$.

Clauses, II

- Let $\phi \in$ **CNF** be formula in conjunctive normal form.

  We define a set $clause(\phi)$ of clauses inductively as follows:

  Basis:
  $clause(p_i) = \{p_i\}$.
  $clause(\neg p_i) = \{\neg p_i\}$.
  $clause(F) = \{\Box\}$.

  Closure:
  (1) If $clause(\phi_1) = \{C_1\}$ and $clause(\phi_2) = \{C_2\}$ then
  $clause(\phi_1 \vee \phi_2) = \{C_1 \cup C_2\}$.
  (2) $clause(\phi_1 \wedge \phi_2) = clause(\phi_1) \cup clause(\phi_2)$.
  (3) If $\Sigma \subseteq$ **CNF** is a set of formulas in conjunctive normal form then
  $clause(\Sigma) = \bigcup\{clause(\phi) : \phi \in \Sigma\}$.

- If $\Sigma \subseteq$ WFF is a set of well formed formulas (not necessarily in conjunctive normal form) then
  $clause(\Sigma) = \bigcup\{clause(cnf(\phi)) : \phi \in \Sigma\}$.

  Here $cnf(\phi)$ denotes a formula in **CNF** logically equivalent to ϕ.

Some easy facts

(i) Let $\Sigma \in$ WFF be a set of well formed formulas and z be a propositional assignment. Then

$$M_{PL}(\Sigma, z) = M_{clause}(clause(\Sigma), z).$$

(ii) In particular Σ is satisfiable if and only if
there is a propositional assignment z such that $M_{clause}(clause(\Sigma), z) = 1$.

Resolution trees

(i) Let $C_1 \cup \{p_j\}, C_2 \cup \{\neg p_j\}$ be two clauses. We say that the clause $C_1 \cup C_2$ is obtained from $C_1 \cup \{p_j\}, C_2 \cup \{\neg p_j\}$ by one resolution step, and we write

$$\frac{C_1 \cup \{p_j\}, C_2 \cup \{\neg p_j\}}{C_1 \cup C_2}$$

(ii) A resolution tree T is a binary (directed) labeled tree $T = \langle V, E \rangle$ such that the labels of V are clauses.
Furthermore, if C_1, C_2 are labels of the two sons of a vertex labeld with C then $C_1 = D_1 \cup \{p_j\}$, $C_2 = D_2 \cup \{\neg p_j\}$ and $C = D_1 \cup D_2$.
In other words, the label of the father is the result of performing a resolution step on the labels of its two sons. Note that several vertices may carry the same label.

(iii) Let S be a set of clauses and C be a clause.
We say that S proves C by resolution, if there is a resolution tree with the root labeled C and all the leaves labeled with clauses from S. We write $S \vdash_{res} C$ for S proves C by resolution.

Examples

(i) Draw a resolution tree for $S \vdash_{res} \Box$ for

$$S = \{\{p_1\}, \{p_2\}, \{\neg p_1, \neg p_2\}\}.$$

(ii) Draw a resolution tree for $S \vdash_{res} \Box$ for

$$S = \{\{p_1, p_2, p_3\}, \{\neg p_1, p_2\}, \{\neg p_3, p_2\}, \{\neg p_2\}\}.$$

(iii) Invent your own example and draw the tree!

Soundness of resolution

The next proposition establishes that resolution steps preserve the meaning of the clauses on which they are based. We call this the soundness of resolution steps. More precisely:

(i) Let S be a set of clauses and $C_1 \cup \{p_j\}, C_2 \cup \{\neg p_j\} \in S$.
Let z be a propositional assignment such that $M_{clause}(S, z) = 1$.
Then $M_{clause}(S \cup \{C_1 \cup C_2\}, z) = 1$.

(ii) Let T be a resolution tree and S_0 be the set of clauses which are the labels of its leaves and S the set of all its labels.
Let z be a propositional assignment such that $M_{clause}(S_0, z) = 1$.
Then $M_{clause}(S, z) = 1$.

(iii) If S is a set of clauses such that $S \vdash_{res} \square$ then S is not satisfiable.

Proof of soundness, I

We prove only (i).

To prove (ii), we can proceed by induction on the depth of the tree applying (i) as the induction step.
(iii) is a direct consequence of (i).

- So let $C_1 \cup \{p_j\}, C_2 \cup \{\neg p_j\} \in S$ be two clauses and z be a propositional assignment such that

$$M_{clause}(S \cup \{C_1 \cup \{p_j\}, C_2 \cup \{\neg p_j\}\}, z) = 1.$$

- We have to show that

$$M_{clause}(S \cup \{C_1 \cup C_2\}, z) = 1.$$

- It suffices to prove the case where $z(p_j) = 1$, as the case $z(p_j) = 0$ is similar.

Proof of soundness, II

- $z(p_j) = 1$ implies that $M_{clause}(C_2, z) = 1$ and therefore $M_{clause}(C_1 \cup C_2, z) = 1$.
- As $M_{clause}(S, z) = 1$ we also have $M_{clause}(S \cup \{C_1 \cup C_2\}, z) = 1$.
- this completes the proof of (i).

Q.E.D.

Completeness of resolution, I

We would like to state a completeness theorem for resolution.

The obvious formulation would be:

(*) Let S be a set of clauses , and C be a clause.
Then $S \vdash_{res} C$ iff C is a logical consequence of S.

The following example shows that this is not true:

Counterexample: for (*)

Let S be $\{\{p_0\}\}$ and C be $\{p_0, p_1\}$.

Clearly C is a logical consequence of S.

However, there is no resolution step applicable given S only.

Completeness of resolution, II

The best we can hope for is the following:

Theorem: Completeness of Resolution for Satisfiability

(**) Let S be a set of clauses.

S is not satisfiable iff $S \vdash_{res} \square$.

This will follow from the Compactness Theorem

and the completeness of the Davis–Putnam Procedure.

The Davis–Putnam Procedure

Let S be a finite set of clauses.
Without loss of generality let $p_0, \ldots, p_n$ be all the variables occuring in S.

We define inductively sets of clauses

$$S_j,\ S_j(pos),\ S_j(neg),\ S_j(-)$$

for $j = 0, \ldots, n$ as follows:

- $S_0 = S$,
- $S_j(pos) = \{C \in S_j : p_j \in C\}$,
- $S_j(neg) = \{C \in S_j : \neg p_j \in C\}$,
- $S_j(-) = \{C \in S_j : p_j \notin C \ and \ \neg p_j \notin C\}$,
- $S_{j+1} = S_j(-) \cup \{C \cup D : C \cup \{p_j\} \in S_j(pos), D \cup \{\neg p_j\} \in S_j(neg)\}$.

Preserving satisfiability

For every $j = 0, \ldots, n$ S_j is satisfiable iff S_{j+1} is satisfiable.

Proof:

- If S_j is satisfiable then S_{j+1} is satisfiable, by the soundness of resolution.
- So let us assume that S_{j+1} is satisfiable by a truth assignment z.
- Let z_1 be the truth assignment obtained from z by putting $z_1(p_j) = 1 - z(p_j)$.
- If neither z nor z_1 satisfy S_j then there are clauses C, D with

$$C \cup D \in \{C \cup D : C \cup \{p_j\} \in S_j(pos)$$
$$D \cup \{\neg p_j\} \in S_j(neg)\}$$

  and $M_{clause}(C \cup D, z) = 0$.
- Which is a contradiction.

Termination

S_{n+1} is either empty or contains only the empty clause. Furthermore, S_{n+1} is empty iff S is satisfiable.

Proof: Exercise

Completeness of the Davis–Putnam Procedure

A finite set of clauses S is satisfiable iff the Davis–Putnam Procedure returns the empty set (not the empty clause).

Proof: Use the lemmas.

Complexity of Resolution

- The Davis–Putnam Procedure seems rather crude and may need an exponential number of resolution steps. Its performance is also sensitive to the numbering of the variables.

- It was shown in a sequence of papers (Tseitin, Galil, Haken, Urquhart, Szemeredi) that there are many sets of n clauses S which are unsatisfiable and which need an exponential number of resolution steps to discover the unsatisfiability.

- In the case of average complexity the situation is more complex as it depends on the input distribution for clauses. There are quite natural distributions for which resolution is polynomial on the average, and others, equally natural, for which resolution is exponential on the average.