

Logical reasoning and programming

SAT solving—resolution and DPLL

Karel Chvalovský

CIIRC CTU

CNFs

A *literal* is a propositional variable p (positive l.) or a negation of propositional variable $\neg p$ (negative l.). In this context we write \bar{p} instead of $\neg p$.

A *clause* is any disjunction of finitely many literals. An important special case is the empty clause, we write \square .

A formula φ is in *conjunctive normal form* (CNF) if φ is a conjunction of clauses. Analogously *disjunctive normal form* (DNF) is defined as a disjunction of conjunctions of literals.

Remark

The clause

$$\bar{p}_1 \vee \bar{p}_2 \vee \cdots \vee \bar{p}_m \vee q_1 \vee q_2 \vee \cdots \vee q_n$$

is equivalent (for example) to

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_m) \rightarrow (q_1 \vee q_2 \vee \cdots \vee q_n).$$

SAT problem

Given a formula φ in CNF decide whether $\varphi \in \text{SAT}$.

Why is satisfiability important? Among other things it is possible to express other notions through it.

For any formula φ we have

$$\models \varphi \quad \text{iff} \quad \neg\varphi \text{ is a contradiction} \quad \text{iff} \quad \neg\varphi \notin \text{SAT}.$$

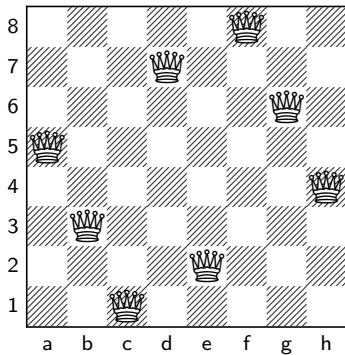
Moreover, for any formula φ and a finite set of formulae Γ we have

$$\Gamma \models \varphi \quad \text{iff} \quad \bigwedge \Gamma \wedge \neg\varphi \text{ is a contradiction} \quad \text{iff} \quad \bigwedge \Gamma \wedge \neg\varphi \notin \text{SAT}.$$

Example

$$p, p \rightarrow q, q \rightarrow r \models r \quad \text{iff} \quad p \wedge (p \rightarrow q) \wedge (q \rightarrow r) \wedge (\neg r) \notin \text{SAT}.$$

Example: n -queens problem

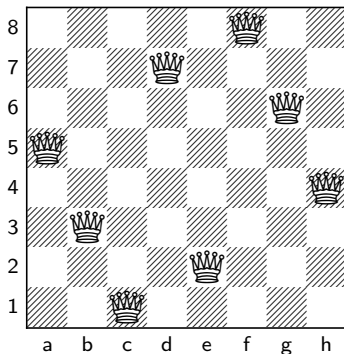


Example: n -queens problem

8	q_{a8}	q_{b8}	q_{c8}	q_{d8}	q_{e8}	q_{f8}	q_{g8}	q_{h8}
7	q_{a7}	q_{b7}	q_{c7}	q_{d7}	q_{e7}	q_{f7}	q_{g7}	q_{h7}
6	q_{a6}	q_{b6}	q_{c6}	q_{d6}	q_{e6}	q_{f6}	q_{g6}	q_{h6}
5	q_{a5}	q_{b5}	q_{c5}	q_{d5}	q_{e5}	q_{f5}	q_{g5}	q_{h5}
4	q_{a4}	q_{b4}	q_{c4}	q_{d4}	q_{e4}	q_{f4}	q_{g4}	q_{h4}
3	q_{a3}	q_{b3}	q_{c3}	q_{d3}	q_{e3}	q_{f3}	q_{g3}	q_{h3}
2	q_{a2}	q_{b2}	q_{c2}	q_{d2}	q_{e2}	q_{f2}	q_{g2}	q_{h2}
1	q_{a1}	q_{b1}	q_{c1}	q_{d1}	q_{e1}	q_{f1}	q_{g1}	q_{h1}
	a	b	c	d	e	f	g	h

- ▶ We represent the fact that a queen occupies a square by propositional variables $q_{a1}, q_{b1}, \dots, q_{h8}$.
 - ▶ q_{b1} means $b1$ contains a queen,
 - ▶ $\overline{q_{b1}}$ means $b1$ is empty.

Example: n -queens problem



- ▶ There should be clauses expressing
 - ▶ n queens on the board (or at least one per row or column), **and**
 - ▶ no two queens on the same row, **and**
 - ▶ no two queens on the same column, **and**
 - ▶ no two queens on the same diagonal.

Basic constraints

We want to express that

$$p_1 + p_2 + \cdots + p_n \bowtie 1,$$

where $\bowtie \in \{\leq, \geq, =\}$.

$= 1$ is expressed as both ≤ 1 and ≥ 1 ,

≥ 1 is

(ATLEASTONE)

$$p_1 \vee \cdots \vee p_n,$$

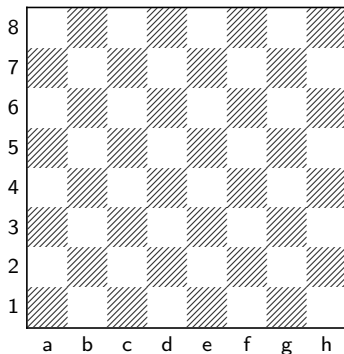
≤ 1 is (pairwise encoding)

(ATMOSTONE)

$$\bigwedge_{1 \leq i < j \leq n} \overline{p_i} \vee \overline{p_j},$$

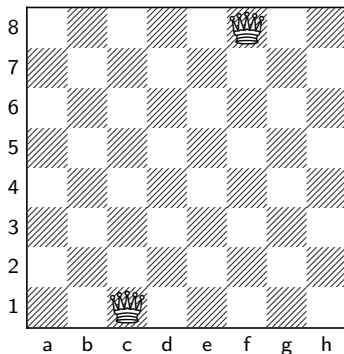
- ▶ we want $p_i \rightarrow \overline{p_j}$, for $i \neq j$,
- ▶ it means $\mathcal{O}(n^2)$ clauses.

Example: n -queens problem



- ▶ Is it satisfiable that
 - ▶ at least one queen per row (or column), **and**
 - ▶ at most one queen per row, **and**
 - ▶ at most one queen per column, **and**
 - ▶ at most one queen per diagonal?

Example: n -queens problem



- ▶ Is it satisfiable that
 - ▶ at least one queen per row (or column), and
 - ▶ at most one queen per row, and
 - ▶ at most one queen per column, and
 - ▶ at most one queen per diagonal, and
 - ▶ q_{c1} and q_{f8} ?

SAT solving applications

SAT solving is one of success stories in computer science. We are able to solve industrial problems containing millions of variables.

It is used, e.g., in

- ▶ formal verification — chip makers check correctness of their designs,
- ▶ security,
- ▶ bioinformatics — mutations in DNA,
- ▶ train safety,
- ▶ planning and scheduling,
- ▶ automated theorem proving.

How to solve SAT?

It is not easy; $\varphi \in \text{SAT}$ is an NP-complete problem (and hence $\varphi \in \text{TAUT}$ is a co-NP-complete problem).

We can use truth tables, but that is in many cases too complicated; we test all possible valuations and for example

$$p \wedge \bar{p} \wedge (q_1 \vee \cdots \vee q_n)$$

is clearly unsatisfiable regardless of values of q_1, \dots, q_n .

Another way, we can think about transformations of formulae that preserve satisfiability; a trivial example is to handle clauses as sets of literals and formulae in CNF as sets of clauses.

CNF as a set of sets

We know that conjunctions and disjunctions are associative, commutative, and idempotent. Therefore a clause can be seen as a set of literals and a formula in CNF as a set of clauses.

Hence from now on we freely use

$$\varphi = \{\{\bar{p}, q\}, \{\bar{q}, r\}, \{\bar{r}, s\}, \{\bar{s}, t\}\}$$

instead of

$$(\bar{p} \vee q) \wedge (\bar{q} \vee r) \wedge (\bar{r} \vee s) \wedge (\bar{s} \vee t).$$

Note that φ is, for example, also a representation of

$$(t \vee \bar{s} \vee t) \wedge (\bar{q} \vee r) \wedge (r \vee \bar{q}) \wedge (\bar{r} \vee s) \wedge (\bar{p} \vee q).$$

Satisfying a clause

For the clause

$$\overline{p_1} \vee \overline{p_2} \vee \cdots \vee \overline{p_m} \vee q_1 \vee q_2 \vee \cdots \vee q_n$$

there are 2^{m+n} possible different truth assignments to literals in it.

How many of them make the clause false?

Resolution rule — example

Assume we want to satisfy two clauses that contain contradicting literals simultaneously

$$\frac{q \vee p \quad \bar{p} \vee r}{q \vee r}$$

If $v \models q \vee p$ and $v \models \bar{p} \vee r$, then clearly $v \models q \vee r$.

Resolution rule

Let $l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}$ be literals and p be a propositional variable.

$$\frac{\{l_1, \dots, l_m, p\} \quad \{\bar{p}, l_{m+1}, \dots, l_{m+n}\}}{\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}}$$

The clause $\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}$ produced by the resolution rule is called the *resolvent* of the two *input* clauses. We call p and \bar{p} a *complementary pair*. We also say that it is a *p-resolvent* to emphasize the complementary pair.

Theorem (correctness)

For any valuation v , if $v \models \{l_1, \dots, l_m, p\}$ and $v \models \{\bar{p}, l_{m+1}, \dots, l_{m+n}\}$, then $v \models \{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}$.

Hence the resolution rule preserves satisfiability.

Resolution calculus

The resolution calculus has no axioms and the only deduction rule is the resolution rule.

Resolution proof

A (resolution) proof of clause c from clauses c_1, \dots, c_n is a finite sequence of clauses d_1, \dots, d_m such that

- ▶ every d_i is among c_1, \dots, c_n or is derived by the resolution rule from input clauses d_j and d_k , for $1 \leq j < k < i \leq m$,
- ▶ $c = d_m$.

We say that a clause c is *provable* (derivable) from a set of clauses $\{c_1, \dots, c_n\}$, we write $\{c_1, \dots, c_n\} \vdash c$, if there is a proof of c from c_1, \dots, c_n .

Resolution proof

Example

$$\frac{\frac{\{p\} \quad \{\bar{p}, q\}}{\{q\}} \quad \frac{\{\bar{q}, r\} \quad \{\bar{r}\}}{\{\bar{q}\}}}{\square}$$

is a proof of $\{\{p\}, \{\bar{p}, q\}, \{\bar{q}, r\}, \{\bar{r}\}\} \vdash \square$.

Strictly speaking the presented derivation is not a sequence, but it is easy to produce a sequence from it.

Completeness of resolution calculus

It is not true that we can derive every valid formula in the resolution calculus, e.g., from the empty set we derive nothing. However, it is so called *refutationally complete*.

Theorem (completeness)

Let φ be a set of clauses. If $\varphi \notin \text{SAT}$, then $\varphi \vdash \square$.

Note that from the correctness theorem we already know the converse implication.

Theorem

Let φ be a set of clauses. If $\varphi \vdash \square$, then $\varphi \notin \text{SAT}$.

Deciding SAT using resolution

If we have a formula φ in CNF, a finite set of clauses, then we can clearly derive only finitely many clauses from it, say $\psi = \{c: \varphi \vdash c\}$. Note that if $\psi \vdash c$, then $c \in \psi$. We call such a set of clauses *saturated*—it is closed under the resolution rule.

This gives us a decision procedure for SAT. Either we produce

- ▶ a (saturated) set of clauses containing the empty clause and hence $\varphi \notin \text{SAT}$, or
- ▶ a saturated set of clauses not containing the empty clause and hence $\varphi \in \text{SAT}$.

Example

Let $\varphi = \{\{\bar{p}, \bar{q}\}, \{p, r\}, \{q, s\}\}$. From φ we obtain the saturated set of clauses $\{\{\bar{p}, \bar{q}\}, \{p, r\}, \{q, s\}, \{\bar{q}, r\}, \{\bar{p}, s\}, \{r, s\}\}$. Hence $\varphi \in \text{SAT}$.

Ordered resolution

Assume the set of clauses $\{\{\bar{p}, \bar{q}\}, \{p, r\}, \{q, s\}\}$ and two possible derivations that differ only in the order of performed steps

$$\frac{\frac{\frac{\{\bar{p}, \bar{q}\}}{\{\bar{q}, r\}} \quad \{p, r\}}{\{q, s\}}}{\{r, s\}} \qquad \frac{\frac{\frac{\{\bar{p}, \bar{q}\}}{\{\bar{p}, s\}} \quad \{q, s\}}{\{r, s\}} \quad \{p, r\}}{\{r, s\}}$$

Is it necessary to try all such possible orderings?

No, we can use an ordered resolution. We can always impose an ordering on variables and resolve using this order. Say $p < q$, meaning all p -resolvents must precede all q -resolvents.

Why is this enough? We try to produce the empty clause, it does not matter in which order we eliminate literals to achieve that goal.

Davis–Putnam algorithm

[Davis and Putnam 1960]

It was originally developed for first-order logic.

We have a set of clauses φ . We choose a variable p such that both p and \bar{p} occur in φ and eliminate it—we produce all possible p -resolvents and add them to φ and then we remove all clauses in φ that contain p or \bar{p} . This operation preserves satisfiability and is also complete even with an imposed ordering.

Example

From $\{\{\bar{p}, \bar{q}\}, \{p, r\}, \{q, s\}\}$ we obtain $\{\{\bar{q}, r\}, \{q, s\}\}$ by eliminating p and then we obtain $\{r, s\}$ by eliminating q . We cannot proceed and hence the original formula is satisfiable.

(Note that r and s are pure literals.)

We can use many tricks, many proposed already by Davis and Putnam (unit propagation, pure literal elimination), to simplify searching, but in general the size of space needed to store clauses can grow exponentially.

Proof of completeness I.

Let φ be a set of clauses and we define

$$\varphi_p^+ = \{c \in \varphi : p \in c\},$$

$$\varphi_p^- = \{c \in \varphi : \bar{p} \in c\},$$

$$\varphi_p^0 = \varphi \setminus (\varphi_p^+ \cup \varphi_p^-),$$

$$\varphi_p = \varphi_p^0 \cup \{c : c \text{ is a } p\text{-resolvent of clauses from } \varphi_p^+ \text{ and } \varphi_p^-\}.$$

Lemma

If $\varphi \notin \text{SAT}$, then $\varphi_p \notin \text{SAT}$.

Proof.

By contraposition. If $\varphi_p \in \text{SAT}$, then there exists v s.t. $v \models \varphi_p$. Let v_p and $v_{\bar{p}}$ only differ from v by setting $v_p(p) = 1$ and $v_{\bar{p}}(p) = 0$, respectively. Clearly, $v_p \models \varphi_p$ and $v_{\bar{p}} \models \varphi_p$. If both $v_p \not\models \varphi$ and $v_{\bar{p}} \not\models \varphi$, then there are $c_p \in \varphi_p^+$ and $c_{\bar{p}} \in \varphi_p^-$ s.t. $v_p \not\models c_{\bar{p}}$ and $v_{\bar{p}} \not\models c_p$. Hence the resolvent of c_p and $c_{\bar{p}}$ is not satisfied by v , a contradiction. Hence $\varphi \in \text{SAT}$. □

Completeness theorem

It is easy to see that the previous lemma gives us the completeness theorems for all resolution calculi we have defined so far. We prove it for the Davis–Putnam algorithm with ordered variables. Because if this calculus derives \square from φ , then a less restrictive calculus derives \square as well.

Theorem (completeness)

Let φ be a set of clauses. If $\varphi \notin \text{SAT}$, then $\varphi \vdash \square$.

Proof.

Let $\varphi \notin \text{SAT}$ and p_1, \dots, p_n be a sequence of all variables occurring in φ . We repeatedly apply the previous lemma (and remove all tautological clauses). We obtain a set of clauses that contains no variables from φ and is unsatisfiable. Hence it is equal to $\{\square\}$. \square

Note that we assume that φ is a finite set of clauses. However, the compactness theorem for propositional logic gives us that an infinite unsatisfiable set of clauses contains a finite unsatisfiable subset.

Some properties of resolution

Subsumption

A clause c_1 is said to (syntactically) *subsume* a clause c_2 if $c_1 \subseteq c_2$.

If $c_1, c_2 \in \varphi$ and $c_1 \subseteq c_2$, then $\varphi \equiv \varphi \setminus c_2$, because $c_1 \models c_2$. Hence $\varphi \in \text{SAT}$ iff $\varphi \setminus c_2 \in \text{SAT}$. Moreover, this can shorten a proof.

Example

From $\{\{p\}, \{p, r\}, \{\bar{p}, q\}, \{\bar{r}, q\}\}$ we obtain $\{\{p\}, \{\bar{p}, q\}, \{\bar{r}, q\}\}$ that is equivalent (and hence equisatisfiable).

Multiple resolvents

If it is possible to obtain more different resolvents from two clauses, then all these resolvents are tautologies, hence always satisfiable, and hence we can ignore them.

Example

$$\frac{\{p, \bar{q}\} \quad \{\bar{p}, q\}}{\{q, \bar{q}\}} \qquad \frac{\{p, \bar{q}\} \quad \{\bar{p}, q\}}{\{p, \bar{p}\}}$$

Conditioning — simplifications

To avoid space problems of the Davis–Putnam algorithm we use a different approach. We try to produce a satisfying valuation by assigning values to variables and we backtrack if necessary.

We select a literal l and replace it by true (\top). Hence \bar{l} is replaced by false (\perp). This can lead to many simplifications of our set of clauses.

Require: A set of clauses φ , a literal l

function SIMPLIFY(φ, l)

$\varphi' \leftarrow \varphi$

for $c \in \varphi'$ **do**

if $l \in c$ **then** remove c from φ' \triangleright satisfied clause

else if $\bar{l} \in c$ **then** remove \bar{l} from c \triangleright unsatisfied literal

return φ'

Chronological backtracking algorithm

Using the previous simplification function, we can chronologically try to create a satisfying valuation.

Require: A set of clauses φ

function ISSAT(φ)

if $\varphi = \emptyset$ **then return** true ▷ no clause

else if $\square \in \varphi$ **then return** false ▷ empty clause

else

$l \leftarrow$ select a literal occurring in φ

if ISSAT(SIMPLIFY(φ, l)) **then return** true

else if ISSAT(SIMPLIFY(φ, \bar{l})) **then return** true

else return false

DPLL algorithm

[Davis, Logemann, and Loveland 1962]

The name stands for Davis, (Putnam), Logemann, and Loveland.

We improve our backtracking algorithm by the following two ideas:

Unit propagation

If a clause contains only a single literal l , then it is forced that l has to be true.

Example

For $\{\{p\}, \{\bar{p}, q\}, \{\bar{q}, r\}, \{\bar{r}\}\}$ we obtain unsatisfiability immediately after unit propagations and simplifications.

Unit propagation is a very powerful technique especially with clause learning.

Pure literal elimination

A literal l is *pure*, if \bar{l} does not occur in the formula. Hence we can satisfy all clauses containing l by assigning true to l .

Pure literal elimination is often ignored for efficiency reasons.

DPLL algorithm

Require: A set of clauses φ

function DPLL(φ)

while φ contains a unit clause $\{l\}$ **do** ▷ unit propagation
 delete clauses containing l from φ ▷ unit subsumption
 delete \bar{l} from all clauses in φ ▷ unit resolution

if $\square \in \varphi$ **then return** false ▷ empty clause

while φ contains a pure literal l **do**
 delete clauses containing l from φ

if $\varphi = \emptyset$ **then return** true ▷ no clause

else
 $l \leftarrow$ select a literal occurring in φ ▷ a choice of literal
 if DPLL($\varphi \cup \{\{l\}\}$) **then return** true
 else if DPLL($\varphi \cup \{\{\bar{l}\}\}$) **then return** true
 else return false

DPLL — data structures

In real implementations we use trail — we keep whole set and construct a partial assignment during a computation. An efficient implementation of unit propagations is crucial.

Watched literals

Instead of checking whole clauses all the time we select two distinct literals, called *watched literals*, in each clause. We also remember in which clauses a literal is selected. If we assign a value to a literal l , then we check only clauses where l is a watched literal. In these clauses we try to select another literal as a watched literal. If that is no longer possible, then we have a unit clause.

It has nice properties during backtracking, because there is no need to update current watched literals.

For details see, e.g., Knuth 2015; Biere, M. J. H. Heule, et al. 2009; Biere, M. Heule, et al. 2021.

How to select a SAT solver?

Try different solvers (based on CDCL), they use the same input format and hence it is easy to experiment. However, the good encoding of your problem is usually at least as important as a good solver.

MiniSat is free, fast, and very popular implementation in C. It won all three industrial categories in the SAT Competition 2005. A new version is called MiniSat 2. However, it is not the state of the art. A good choice if you want to use a SAT solver in your software.

For playing in Python you can use pycosat, a package that provides bindings to PicoSAT on the C level. A rapidly developing toolkit is PySAT.

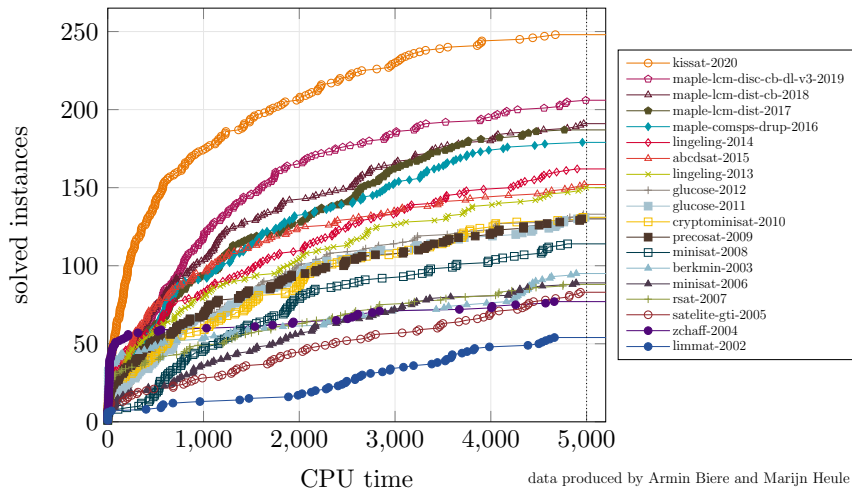
Check results of SAT Competition 2023 and from previous years for the state of the art.

Some modern solvers

There are many modern solvers available, e.g.,

- ▶ solvers developed by Armin Biere, for example,
 - ▶ PicoSAT,
 - ▶ Lingeling,
 - ▶ CaDiCaL,
 - ▶ Kissat,
 - ▶ SATCH—a simple and clean code base for explaining and experimenting with SAT solvers,
- ▶ Glucose
 - ▶ based on MiniSat,
- ▶ MapleSAT
 - ▶ based on Glucose,
- ▶ CryptoMiniSat,
- ▶ Sat4j—a Java library; good for Windows,
- ▶ UBCSAT—a Stochastic Local Search SAT Solver framework.

SAT Competition Winners on the SC2020 Benchmark Suite



DIMACS format

The standard format for SAT solvers.

Variables are enumerated $1, 2, \dots$. A variable x_i is represented by i and $\overline{x_i}$ by $-i$. A clause is a list of non-zero integers separated by spaces, tabs, or newlines. The end of a clause is represented by zero. The order of literals and clauses is irrelevant.

Input

```
c start with comments
```

```
c
```

```
p cnf 5 3 #variables #clauses
```

```
1 -5 4 0
```

```
-1 5 3 4 0
```

```
-3 -4 0
```

```
encodes
```

$$(x_1 \vee \overline{x_5} \vee x_4) \wedge (\overline{x_1} \vee x_5 \vee x_3 \vee x_4) \wedge (\overline{x_3} \vee \overline{x_4}).$$

DIMACS output format

There are three possible outcomes





- ▶ s SATISFIABLE
 - ▶ a satisfying assignment is returned: $v \ 1 \ -2 \ -3 \ 4 \ 0$
- ▶ s UNSATISFIABLE
 - ▶ a possible certificate in an external file
- ▶ s UNKNOWN

Certifying unsatisfiability

It is easy to convince someone that a formula is satisfiable by showing an assignment. To certificate that it is unsatisfiable is not so easy. It can be exponentially long and usually such a certificate is provided in a form of resolution proof.

A standard format currently used is called DRAT (Delete Resolution Asymmetric Tautologies).

Bibliography I

-  Biere, Armin, Marijn Heule, et al., eds. (2021). *Handbook of Satisfiability*. 2nd. Vol. 336. Frontiers in Artificial Intelligence and Applications. Washington: IOS Press. ISBN: 978-1-64368-161-0.
-  Biere, Armin, Marijn J. H. Heule, et al., eds. (Feb. 2009). *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, p. 980. ISBN: 978-1-58603-929-5.
-  Davis, Martin, George Logemann, and Donald Loveland (July 1962). “A Machine Program for Theorem-Proving”. In: *Communications of the ACM* 5.7, pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557.
-  Davis, Martin and Hilary Putnam (July 1960). “A Computing Procedure for Quantification Theory”. In: *Journal of the ACM* 7.3, pp. 201–215. ISSN: 0004-5411. DOI: 10.1145/321033.321034.

Bibliography II



Knuth, Donald E. (2015). *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. 1st. Addison-Wesley Professional. ISBN: 978-0-13-439760-3.