Boolean satisfiability problem

In logic and computer science, the **Boolean satisfiability problem** (sometimes called **propositional satisfiability problem** and abbreviated **SATISFIABILITY**, **SAT** or **B-SAT**) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE. If this is the case, the formula is called *satisfiable*. On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the formula is *unsatisfiable*. For example, the formula "*a* AND NOT *b*" is satisfiable because one can find the values *a* = TRUE and *b* = FALSE, which make (*a* AND NOT *b*) = TRUE. In contrast, "*a* AND NOT *a*" is unsatisfiable.

SAT is the first problem that was proven to be NP-complete—this is the Cook–Levin theorem. This means that all problems in the complexity class NP, which includes a wide range of natural decision and optimization problems, are at most as difficult to solve as SAT. There is no known algorithm that efficiently solves each SAT problem, and it is generally believed that no such algorithm exists, but this belief has not been proven mathematically, and resolving the question of whether SAT has a polynomial-time algorithm is equivalent to the P versus NP problem, which is a famous open problem in the theory of computing.

Nevertheless, as of 2007, heuristic SAT-algorithms are able to solve problem instances involving tens of thousands of variables and formulas consisting of millions of symbols,[1] which is sufficient for many practical SAT problems from, e.g., artificial intelligence, circuit design,[2] and automatic theorem proving.

Definitions

A *propositional logic formula*, also called *Boolean expression*, is built from variables, operators AND (conjunction, also denoted by ∧), OR (disjunction, ∨), NOT (negation, ¬), and parentheses. A formula is said to be *satisfiable* if it can be made TRUE by assigning appropriate logical values (i.e. TRUE, FALSE) to its variables. The *Boolean satisfiability problem* (SAT) is, given a formula, to check whether it is satisfiable. This decision problem is of central importance in many areas of computer science, including theoretical computer science, complexity theory,[3][4] algorithmics, cryptography[5][6] and artificial intelligence.[7]

Conjunctive normal form

A *literal* is either a variable (in which case it is called a *positive literal*) or the negation of a variable (called a *negative literal*). A *clause* is a disjunction of literals (or a single literal). A clause is called a *Horn clause* if it contains at most one positive literal. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses (or a single clause).

For example, x_1 is a positive literal, $\neg x_2$ is a negative literal, and $x_1 \vee \neg x_2$ is a clause. The formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$ is in conjunctive normal form; its first and third clauses are Horn clauses, but its second clause is not. The formula is satisfiable, by choosing x_1 = FALSE, x_2 = FALSE, and x_3 arbitrarily, since (FALSE ∨ ¬FALSE) ∧ (¬FALSE ∨ FALSE ∨ x_3) ∧ ¬FALSE evaluates to (FALSE ∨ TRUE) ∧ (TRUE ∨ FALSE ∨ x_3) ∧ TRUE, and in turn to TRUE ∧ TRUE ∧ TRUE (i.e. to TRUE). In contrast, the CNF formula $a \wedge \neg a$, consisting of two clauses of one literal, is unsatisfiable, since for *a*=TRUE or *a*=FALSE it evaluates to TRUE ∧ ¬TRUE (i.e., FALSE) or FALSE ∧ ¬FALSE (i.e., again FALSE), respectively.

For some versions of the SAT problem, it is useful to define the notion of a *generalized conjunctive normal form* formula, viz. as a conjunction of arbitrarily many *generalized clauses*, the latter being of the form $R(l_1,...,l_n)$ for some Boolean function R and (ordinary) literals l_i. Different sets of allowed Boolean functions lead to different problem versions. As an example, $R(\neg x,a,b)$ is a generalized clause, and $R(\neg x,a,b) \wedge R(b,y,c) \wedge R(c,d,\neg z)$ is a generalized conjunctive normal form. This formula is used below, with R being the ternary operator that is TRUE just when exactly one of its arguments is.

Using the laws of Boolean algebra, every propositional logic formula can be transformed into an equivalent conjunctive normal form, which may, however, be exponentially longer. For example, transforming the formula $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee ... \vee (x_n \wedge y_n)$ into conjunctive normal form yields

$$
\begin{aligned}
&(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge \\
&(y_1 \vee x_2 \vee \ldots \vee x_n) \wedge \\
&(x_1 \vee y_2 \vee \ldots \vee x_n) \wedge \\
&(y_1 \vee y_2 \vee \ldots \vee x_n) \wedge ... \wedge \\
&(x_1 \vee x_2 \vee \ldots \vee y_n) \wedge \\
&(y_1 \vee x_2 \vee \ldots \vee y_n) \wedge \\
&(x_1 \vee y_2 \vee \ldots \vee y_n) \wedge \\
&(y_1 \vee y_2 \vee \ldots \vee y_n);
\end{aligned}
$$

while the former is a disjunction of n conjunctions of 2 variables, the latter consists of 2^n clauses of n variables.

However, with use of the Tseytin transformation, we may find an equisatisfiable conjunctive normal form formula with length linear in the size of the original propositional logic formula.

Complexity

SAT was the first problem known to be NP-complete, as proved by Stephen Cook at the University of Toronto in 1971[8] and independently by Leonid Levin at the Russian Academy of Sciences in 1973.[9] Until that time, the concept of an NP-complete problem did not even exist. The proof shows how every decision problem in the complexity class NP can be reduced to the SAT problem for CNF[note 1] formulas, sometimes called **CNFSAT**. A useful property of Cook's reduction is that it preserves the number of accepting answers. For example, deciding whether a given graph has a 3-coloring is another problem in NP; if a graph has 17 valid 3-colorings, then the SAT formula produced by the Cook–Levin reduction will have 17 satisfying assignments.

NP-completeness only refers to the run-time of the worst case instances. Many of the instances that occur in practical applications can be solved much more quickly. See §Algorithms for solving SAT below.

3-satisfiability

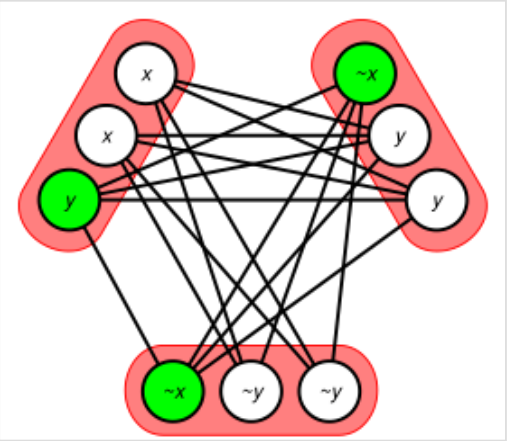
Like the satisfiability problem for arbitrary formulas, determining the satisfiability of a formula in conjunctive normal form where each clause is limited to at most three literals is NP-complete also; this problem is called **3-SAT**, **3CNFSAT**, or **3-satisfiability**. To reduce the unrestricted SAT problem to 3-SAT, transform each clause $l_1 \vee \cdots \vee l_n$ to a conjunction of $n - 2$ clauses

$$
\begin{aligned}
&(l_1 \vee l_2 \vee x_2) \wedge \\
&(\neg x_2 \vee l_3 \vee x_3) \wedge \\
&(\neg x_3 \vee l_4 \vee x_4) \wedge \cdots \wedge \\
&(\neg x_{n-3} \vee l_{n-2} \vee x_{n-2}) \wedge \\
&(\neg x_{n-2} \vee l_{n-1} \vee l_n)
\end{aligned}
$$

where $x_2, \cdots, x_{n-2}$ are fresh variables not occurring elsewhere. Although the two formulas are not logically equivalent, they are equisatisfiable. The formula resulting from transforming all clauses is at most 3 times as long as its original; that is, the length growth is polynomial.[10]

3-SAT is one of Karp's 21 NP-complete problems, and it is used as a starting point for proving that other problems are also NP-hard.[note 2] This is done by polynomial-time reduction from 3-SAT to the other problem. An example of a problem where this method has been used is the clique problem: given a CNF formula consisting of c clauses, the corresponding graph consists of a vertex for each literal, and an edge between each two non-contradicting[note 3] literals from different clauses; see the picture. The graph has a c-clique if and only if the formula is satisfiable.[11]

There is a simple randomized algorithm due to Schöning (1999) that runs in time $(4/3)^n$ where n is the number of variables in the 3-SAT proposition, and succeeds with high probability to correctly decide 3-SAT.[12]



The 3-SAT instance $(x \vee x \vee y) \wedge (\neg x \vee \neg y \vee \neg y) \wedge (\neg x \vee y \vee y)$ reduced to a clique problem. The green vertices form a 3-clique and correspond to the satisfying assignment x=FALSE, y=TRUE.

The exponential time hypothesis asserts that no algorithm can solve 3-SAT (or indeed k-SAT for any $k > 2$) in $\exp(o(n))$ time (that is, fundamentally faster than exponential in n).

Selman, Mitchell, and Levesque (1996) give empirical data on the difficulty of randomly generated 3-SAT formulas, depending on their size parameters. Difficulty is measured in number recursive calls made by a DPLL algorithm. They identified a phase transition region from almost-certainly-satisfiable to almost-certainly-unsatisfiable formulas at the clauses-to-variables ratio at about 4.26.[13]

3-satisfiability can be generalized to **k-satisfiability** (**k-SAT**, also **k-CNF-SAT**), when formulas in CNF are considered with each clause containing up to k literals. However, since for any $k \geq 3$, this problem can neither be easier than 3-SAT nor harder than SAT, and the latter two are NP-complete, so must be k-SAT.

Some authors restrict k-SAT to CNF formulas with **exactly k literals**. This does not lead to a different complexity class either, as each clause $l_1 \vee \cdots \vee l_j$ with $j < k$ literals can be padded with fixed dummy variables to $l_1 \vee \cdots \vee l_j \vee d_{j+1} \vee \cdots \vee d_k$. After padding all clauses, 2^k–1 extra clauses[note 4] must be appended to ensure that only $d_1 = \cdots = d_k = \text{FALSE}$ can lead to a satisfying assignment. Since k does not depend on the formula length, the extra clauses lead to a constant increase in length. For the same reason, it does not matter whether **duplicate literals** are allowed in clauses, as in $\neg x \vee \neg y \vee \neg y$.

Special cases of SAT

Conjunctive normal form

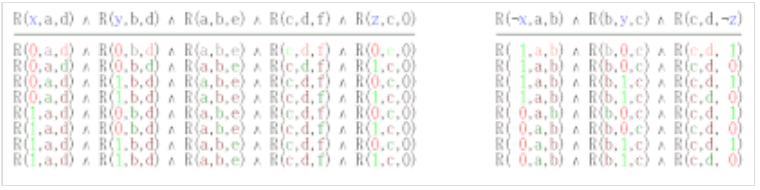
Conjunctive normal form (in particular with 3 literals per clause) is often considered the canonical representation for SAT formulas. As shown above, the general SAT problem reduces to 3-SAT, the problem of determining satisfiability for formulas in this form.

Disjunctive normal form

SAT is trivial if the formulas are restricted to those in **disjunctive normal form**, that is, they are a disjunction of conjunctions of literals. Such a formula is indeed satisfiable if and only if at least one of its conjunctions is satisfiable, and a conjunction is satisfiable if and only if it does not contain both x and NOT x for some variable x. This can be checked in linear time. Furthermore, if they are restricted to being in **full disjunctive normal form**, in which every variable appears exactly once in every conjunction, they can be checked in constant time (each conjunction represents one satisfying assignment). But it can take exponential time and space to convert a general SAT problem to disjunctive normal form; to obtain an example, exchange "∧" and "∨" in the above exponential blow-up example for conjunctive normal forms.

Exactly-1 3-satisfiability

A variant of the 3-satisfiability problem is the **one-in-three 3-SAT** (also known variously as **1-in-3-SAT** and **exactly-1 3-SAT**). Given a conjunctive normal form with three literals per clause, the problem is to determine whether there exists a truth assignment to the variables so that each clause has *exactly* one TRUE literal (and thus exactly two FALSE literals). In contrast, ordinary 3-SAT requires that every clause has *at least* one TRUE literal. Formally, a one-in-three 3-SAT problem is given as a generalized conjunctive normal form with all generalized clauses using a ternary operator R that is TRUE just if exactly one of its arguments is. When all literals of a one-in-three 3-SAT formula are positive, the satisfiability problem is called **one-in-three positive 3-SAT**.



Left: Schaefer's reduction of a 3-SAT clause $x \vee y \vee z$. The result of R is TRUE (1) if exactly one of its arguments is TRUE, and FALSE (0) otherwise. All 8 combinations of values for x,y,z are examined, one per line. The fresh variables $a,...,f$ can be chosen to satisfy all clauses (exactly one green argument for each R) in all lines except the first, where $x \vee y \vee z$ is FALSE. **Right:** A simpler reduction with the same properties.

One-in-three 3-SAT, together with its positive case, is listed as NP-complete problem "LO4" in the standard reference *Computers and Intractability: A Guide to the Theory of NP-Completeness* by Michael R. Garey and David S. Johnson. One-in-three 3-SAT was proved to be NP-complete by Thomas Jerome Schaefer as a special case of Schaefer's dichotomy theorem, which asserts that any problem generalizing Boolean satisfiability in a certain way is either in the class P or is NP-complete.[14]

Schaefer gives a construction allowing an easy polynomial-time reduction from 3-SAT to one-in-three 3-SAT. Let "(x or y or z)" be a clause in a 3CNF formula. Add six fresh Boolean variables a, b, c, d, e, and f, to be used to simulate this clause and no other. Then the formula $R(x,a,d) \wedge R(y,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(z,c,\text{FALSE})$ is satisfiable by some setting of the fresh variables if and only if at least one of x, y, or z is TRUE, see picture (left). Thus any 3-SAT instance with m clauses and n variables may be converted into an equisatisfiable one-in-three 3-SAT instance with $5m$ clauses and $n + 6m$ variables.[15] Another reduction involves only four fresh variables and three clauses: $R(\neg x,a,b) \wedge R(b,y,c) \wedge \mathrm{R}(c,d,\neg z)$, see picture (right).

Not-all-equal 3-satisfiability

Another variant is the **not-all-equal 3-satisfiability** problem (also called **NAE3SAT**). Given a conjunctive normal form with three literals per clause, the problem is to determine if an assignment to the variables exists such that in no clause all three literals have the same truth value. This problem is NP-complete, too, even if no negation symbols are admitted, by Schaefer's dichotomy theorem.[14]

Linear SAT

A 3-SAT formula is *Linear SAT* (*LSAT*) if each clause (viewed as a set of literals) intersects at most one other clause, and, moreover, if two clauses intersect, then they have exactly one literal in common. An LSAT formula can be depicted as a set of disjoint semi-closed intervals on a line. Deciding whether an LSAT formula is satisfiable is NP-

complete.[16]

2-satisfiability

SAT is easier if the number of literals in a clause is limited to at most 2, in which case the problem is called **2-SAT**. This problem can be solved in polynomial time, and in fact is complete for the complexity class NL. If additionally all OR operations in literals are changed to XOR operations, then the result is called **exclusive-or 2-satisfiability**, which is a problem complete for the complexity class SL = L.

Horn-satisfiability

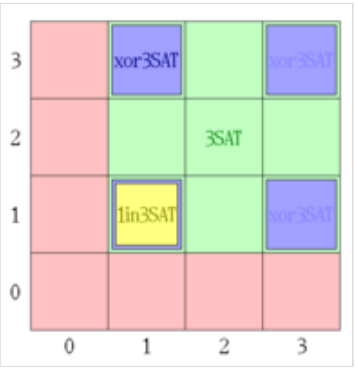
The problem of deciding the satisfiability of a given conjunction of Horn clauses is called **Horn-satisfiability**, or **HORN-SAT**. It can be solved in polynomial time by a single step of the unit propagation algorithm, which produces the single minimal model of the set of Horn clauses (w.r.t. the set of literals assigned to TRUE). Horn-satisfiability is P-complete. It can be seen as P's version of the Boolean satisfiability problem. Also, deciding the truth of quantified Horn formulas can be done in polynomial time.[17]

Horn clauses are of interest because they are able to express implication of one variable from a set of other variables. Indeed, one such clause $\neg x_1 \vee \ldots \vee \neg x_n \vee y$ can be rewritten as $x_1 \wedge \ldots \wedge x_n \rightarrow y$; that is, if $x_1,\ldots,x_n$ are all TRUE, then y must be TRUE as well.

A generalization of the class of Horn formulas is that of renameable-Horn formulae, which is the set of formulas that can be placed in Horn form by replacing some variables with their respective negation. For example, $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$ is not a Horn formula, but can be renamed to the Horn formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg y_3) \wedge \neg x_1$ by introducing y_3 as negation of x_3. In contrast, no renaming of $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$ leads to a Horn formula. Checking the existence of such a replacement can be done in linear time; therefore, the satisfiability of such formulae is in P as it can be solved by first performing this replacement and then checking the satisfiability of the resulting Horn formula.

XOR-satisfiability

Another special case is the class of problems where each clause contains XOR (i.e. exclusive or) rather than (plain) OR operators.[note 5] This is in P, since an XOR-SAT formula can also be viewed as a system of linear equations mod 2, and can be solved in cubic time by Gaussian elimination;[18] see the box for an example. This recast is based on the kinship between Boolean algebras and Boolean rings, and the fact that arithmetic modulo two forms a finite field. Since a XOR b XOR c evaluates to TRUE if and only if exactly 1 or 3 members of $\{a,b,c\}$ are TRUE, each solution of the 1-in-3-SAT problem for a given CNF formula is also a solution of the XOR-3-SAT problem, and in turn each solution of XOR-3-SAT is a solution of 3-SAT; see the picture. As a consequence, for each CNF formula, it is possible to solve the XOR-3-SAT problem defined by the formula, and based on the result infer either that the 3-SAT problem is solvable or that the 1-in-3-SAT problem is unsolvable.



A formula with 2 clauses may be unsatisfied (red), 3-satisfied (green), xor-3-satisfied (blue), or/and 1-in-3-satisfied (yellow), depending on the TRUE-literal count in the 1st (hor) and 2nd (vert) clause.

Provided that the complexity classes P and NP are not equal, neither 2-, nor Horn-, nor XOR-satisfiability is NP-complete, unlike SAT.

Schaefer's dichotomy theorem

The restrictions above (CNF, 2CNF, 3CNF, Horn, XOR-SAT) bound the considered formulae to be conjunctions of subformulas; each restriction states a specific form for all subformulas: for example, only binary clauses can be subformulas in 2CNF.

Schaefer's dichotomy theorem states that, for any restriction to Boolean functions that can be used to form these subformulas, the corresponding satisfiability problem is in P or NP-complete. The membership in P of the satisfiability of 2CNF, Horn, and XOR-SAT formulae are special cases of this theorem.[14]

The following table summarizes some common variants of SAT.

Solving an XOR-SAT example by Gaussian elimination

Given formula

("⊕" means XOR, the red clause is optional)

$(a \oplus c \oplus d) \wedge (b \oplus \neg c \oplus d) \wedge (a \oplus b \oplus \neg d) \wedge (a \oplus \neg b \oplus \neg c) \wedge (\neg a \oplus b \oplus c)$

Equation system

("1" means TRUE, "0" means FALSE)

Each clause leads to one equation.

$$\begin{aligned}
a \oplus c \oplus d &= 1 \\
b \oplus \neg c \oplus d &= 1 \\
a \oplus b \oplus \neg d &= 1 \\
a \oplus \neg b \oplus \neg c &= 1 \\
\neg a \oplus b \oplus c &\simeq 1
\end{aligned}$$

Normalized equation system

using properties of Boolean rings ($\neg x = 1 \oplus x$, $x \oplus x = 0$)

$$\begin{aligned}
a \oplus c \oplus d &= \mathbf{1} \\
b \oplus c \oplus d &= \mathbf{0} \\
a \oplus b \oplus d &= \mathbf{0} \\
a \oplus b \oplus c &= \mathbf{1} \\
a \oplus b \oplus c &\simeq \mathbf{0}
\end{aligned}$$

(If the red equation is present, it contradicts the last black one, so the system is unsolvable. Therefore, Gauss' algorithm is used only for the black equations.)

Associated coefficient matrix

a	***b***	***c***	***d***		**line**
1	0	1	1	**1**	A
0	1	1	1	**0**	B
1	1	0	1	**0**	C
1	1	1	0	**1**	D

Transforming to echelon form

a	***b***	***c***	***d***		**operation**
1	0	1	1	**1**	A
1	1	0	1	**0**	C
1	1	1	0	**1**	D
0	1	1	1	**0**	B (swapped)
1	0	1	1	**1**	A
0	1	1	0	**1**	E = C⊕A
0	1	0	1	**0**	F = D⊕A
0	1	1	1	**0**	B
1	0	1	1	**1**	A
0	1	1	0	**1**	E
0	0	1	1	**1**	G = F⊕E
0	0	0	1	**1**	H = B⊕E

Transforming to diagonal form

a	***b***	***c***	***d***	**operation**

1	0	1	0	**0**	I = A⊕H
0	1	1	0	**1**	E
0	0	1	0	**0**	J = G⊕H
0	0	0	1	**1**	H
1	0	0	0	**0**	K = I⊕J
0	1	0	0	**1**	L = E⊕J
0	0	1	0	**0**	J
0	0	0	1	**1**	H

Solution:

If the red clause is present: Unsolvable

Else:
a = 0 = FALSE
b = 1 = TRUE
c = 0 = FALSE
d = 1 = TRUE

As a consequence:

$R(a,c,d) \wedge R(b,\neg c,d) \wedge R(a,b,\neg d) \wedge R(a,\neg b,\neg c) \wedge R(\neg a,b,c)$ is not 1-in-3-satisfiable,

while $(a \vee c \vee d) \wedge (b \vee \neg c \vee d) \wedge (a \vee b \vee \neg d) \wedge (a \vee \neg b \vee \neg c)$ is 3-satisfiable with a=c=FALSE and b=d=TRUE.

Code	Name	Restrictions	Requirements	Class
3SAT	3-satisfiability	Each clause contains 3 literals.	At least one literal must be true.	NP-c
2SAT	2-satisfiability	Each clause contains 2 literals.	At least one literal must be true.	NL-c
1-in-3-SAT	Exactly-1 3-SAT	Each clause contains 3 literals.	Exactly one literal must be true.	NP-c
1-in-3-SAT+	Exactly-1 Positive 3-SAT	Each clause contains 3 positive literals.	Exactly one literal must be true.	NP-c
NAE3SAT	Not-all-equal 3-satisfiability	Each clause contains 3 literals.	Either one or two literals must be true.	NP-c
NAE3SAT+	Not-all-equal positive 3-SAT	Each clause contains 3 positive literals.	Either one or two literals must be true.	NP-c
PL-SAT	Planar SAT	The incidence graph (clause-variable graph) is planar.	At least one literal must be true.	NP-c
LSAT	Linear SAT	Each clause contains 3 literals, intersects at most one other clause, and the intersection is exactly one literal.	At least one literal must be true.	NP-c
HORN-SAT	Horn satisfiability	Horn clauses (at most one positive literal).	At least one literal must be true.	P-c
XOR-SAT	Xor satisfiability	Each clause contains XOR operations rather than OR.	The XOR of all literals must be true.	P

Extensions of SAT

An extension that has gained significant popularity since 2003 is **satisfiability modulo theories** (**SMT**) that can enrich CNF formulas with linear constraints, arrays, all-different constraints, uninterpreted functions,[19] etc. Such extensions typically remain NP-complete, but very efficient solvers are now available that can handle many such kinds of constraints.

The satisfiability problem becomes more difficult if both "for all" (∀) and "there exists" (∃) quantifiers are allowed to bind the Boolean variables. An example of such an expression would be $\forall x\, \forall y\, \exists z\ (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$; it is valid, since for all values of *x* and *y*, an appropriate value of *z* can be found, viz. *z*=TRUE if both *x* and *y* are FALSE, and *z*=FALSE else. SAT itself (tacitly) uses only ∃ quantifiers. If only ∀ quantifiers are allowed instead, the so-called **tautology problem** is obtained, which is co-NP-complete. If both quantifiers are allowed, the problem is called the **quantified Boolean formula problem** (**QBF**), which can be shown to be PSPACE-complete. It is widely believed that PSPACE-complete problems are strictly harder than any problem in NP, although this has not yet been proved. Using highly parallel *P systems*, QBF-SAT problems can be solved in linear time.[20]

Ordinary SAT asks if there is at least one variable assignment that makes the formula true. A variety of variants deal with the number of such assignments:

- **MAJ-SAT** asks if the majority of all assignments make the formula TRUE. It is known to be complete for PP, a probabilistic class.
- **#SAT**, the problem of counting how many variable assignments satisfy a formula, is a counting problem, not a decision problem, and is #P-complete.
- **UNIQUE SAT**[21] is the problem of determining whether a formula has exactly one assignment. It is complete for US,[22] the complexity class describing problems solvable by a non-deterministic polynomial time Turing machine that accepts when there is exactly one nondeterministic accepting path and rejects otherwise.
- **UNAMBIGUOUS-SAT** is the name given to the satisfiability problem when the input is restricted to formulas having at most one satisfying assignment. The problem is also called **USAT**.[23] A solving algorithm for UNAMBIGUOUS-SAT is allowed to exhibit any behavior, including endless looping, on a formula having several satisfying assignments. Although this problem seems easier, Valiant and Vazirani have shown[24] that if there is a practical (i.e. randomized polynomial-time) algorithm to solve it, then all problems in NP can be solved just as easily.
- **MAX-SAT**, the maximum satisfiability problem, is an FNP generalization of SAT. It asks for the maximum number of clauses which can be satisfied by any assignment. It has efficient approximation algorithms, but is NP-hard to solve exactly. Worse still, it is APX-complete, meaning there is no polynomial-time approximation scheme (PTAS) for this problem unless P=NP.
- **WMSAT** is the problem of finding an assignment of minimum weight that satisfy a monotone Boolean formula (i.e. a formula without any negation). Weights of propositional variables are given in the input of the problem. The weight of an assignment is the sum of weights of true variables. That problem is NP-complete (see Th. 1 of [25]).

Other generalizations include satisfiability for first- and second-order logic, constraint satisfaction problems, 0-1 integer programming.

Finding a satisfying assignment

While SAT is a decision problem, the search problem of finding a satisfying assignment reduces to SAT. That is, each algorithm which correctly answers whether an instance of SAT is solvable can be used to find a satisfying assignment. First, the question is asked on the given formula Φ. If the answer is "no", the formula is unsatisfiable. Otherwise, the question is asked on the partly instantiated formula $\Phi\{x_1{=}\text{TRUE}\}$, that is, Φ with the first variable x_1 replaced by TRUE, and simplified accordingly. If the answer is "yes", then x_1=TRUE, otherwise x_1=FALSE. Values of other variables can be found subsequently in the same way. In total, *n*+1 runs of the algorithm are required, where *n* is the number of distinct variables in Φ.

This property is used in several theorems in complexity theory:

$\text{NP} \subseteq \text{P/poly} \Rightarrow \text{PH} = \Sigma_2$ (Karp–Lipton theorem)
$\text{NP} \subseteq \text{BPP} \Rightarrow \text{NP} = \text{RP}$

P = NP ⇒ FP = FNP

Algorithms for solving SAT

Since the SAT problem is NP-complete, only algorithms with exponential worst-case complexity are known for it. In spite of this, efficient and scalable algorithms for SAT were developed during the 2000s and have contributed to dramatic advances in the ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints (i.e. clauses).[1] Examples of such problems in electronic design automation (EDA) include formal equivalence checking, model checking, formal verification of pipelined microprocessors,[19] automatic test pattern generation, routing of FPGAs,[26] planning, and scheduling problems, and so on. A SAT-solving engine is also considered to be an essential component in the electronic design automation toolbox.

Major techniques used by modern SAT solvers include the Davis–Putnam–Logemann–Loveland algorithm (or DPLL), conflict-driven clause learning (CDCL), and stochastic local search algorithms such as WalkSAT. Almost all SAT solvers include time-outs, so they will terminate in reasonable time even if they cannot find a solution. Different SAT solvers will find different instances easy or hard, and some excel at proving unsatisfiability, and others at finding solutions. Recent attempts have been made to learn an instance's satisfiability using deep learning techniques.[27]

SAT solvers are developed and compared in SAT-solving contests.[28] Modern SAT solvers are also having significant impact on the fields of software verification, constraint solving in artificial intelligence, and operations research, among others.

See also

- Unsatisfiable core
- Satisfiability modulo theories
- Counting SAT
- Planar SAT
- Karloff–Zwick algorithm
- Circuit satisfiability

Notes

1. The SAT problem for *arbitrary* formulas is NP-complete, too, since it is easily shown to be in NP, and it cannot be easier than SAT for CNF formulas.
2. i.e. at least as hard as every other problem in NP. A decision problem is NP-complete if and only if it is in NP and is NP-hard.
3. i.e. such that one literal is not the negation of the other
4. viz. all maxterms that can be built with $d_1, \cdots, d_k$, except $d_1 \vee \cdots \vee d_k$
5. Formally, generalized conjunctive normal forms with a ternary Boolean function *R* are employed, which is TRUE just if 1 or 3 of its arguments is. An input clause with more than 3 literals can be transformed into an equisatisfiable conjunction of clauses á 3 literals similar to above; i.e. XOR-SAT can be reduced to XOR-3-SAT.

External links

- SAT Game (http://www.cril.univ-artois.fr/~roussel/satgame/satgame.php?lang=eng): try solving a Boolean satisfiability problem yourself
- The international SAT competition website (http://www.satcompetition.org/)

- International Conference on Theory and Applications of Satisfiability Testing (http://www.satisfiability.org/)
- Journal on Satisfiability, Boolean Modeling and Computation (https://web.archive.org/web/20060219180520/http://jsat.ewi.tudelft.nl/)
- SAT Live, an aggregate website for research on the satisfiability problem (http://www.satlive.org)
- Yearly evaluation of MaxSAT solvers (http://www.maxsat.udl.cat/)

References

1. Ohrimenko, Olga; Stuckey, Peter J.; Codish, Michael (2007), "Propagation = Lazy Clause Generation", *Principles and Practice of Constraint Programming – CP 2007*, Lecture Notes in Computer Science, vol. 4741, pp. 544–558, CiteSeerX 10.1.1.70.5471 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.5471), doi:10.1007/978-3-540-74970-7_39 (https://doi.org/10.1007%2F978-3-540-74970-7_39), ISBN 978-3-540-74969-1, "modern SAT solvers can often handle problems with millions of constraints and hundreds of thousands of variables".
2. Hong, Ted; Li, Yanjing; Park, Sung-Boem; Mui, Diana; Lin, David; Kaleq, Ziyad Abdel; Hakim, Nagib; Naeimi, Helia; Gardner, Donald S.; Mitra, Subhasish (November 2010). "QED: Quick Error Detection tests for effective post-silicon validation". *2010 IEEE International Test Conference*. pp. 1–10. doi:10.1109/TEST.2010.5699215 (https://doi.org/10.1109%2FTEST.2010.5699215). ISBN 978-1-4244-7206-2. S2CID 7909084 (https://api.semanticscholar.org/CorpusID:7909084).
3. Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems" (https://web.archive.org/web/20110629023717/http://www.cs.berkeley.edu/~luca/cs172/karp.pdf) (PDF). In Raymond E. Miller; James W. Thatcher (eds.). *Complexity of Computer Computations*. New York: Plenum. pp. 85–103. ISBN 0-306-30707-3. Archived from the original (http://www.cs.berkeley.edu/~luca/cs172/karp.pdf) (PDF) on 2011-06-29. Retrieved 2020-05-07. Here: p.86
4. Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley. p. 403. ISBN 0-201-00029-6.
5. Massacci, Fabio; Marraro, Laura (2000-02-01). "Logical Cryptanalysis as a SAT Problem". *Journal of Automated Reasoning*. **24** (1): 165–203. doi:10.1023/A:1006326723002 (https://doi.org/10.1023%2FA%3A1006326723002). S2CID 3114247 (https://api.semanticscholar.org/CorpusID:3114247).
6. Mironov, Ilya; Zhang, Lintao (2006). "Applications of SAT Solvers to Cryptanalysis of Hash Functions" (https://link.springer.com/chapter/10.1007%2F11814948_13). In Biere, Armin; Gomes, Carla P. (eds.). *Theory and Applications of Satisfiability Testing - SAT 2006*. Lecture Notes in Computer Science. Vol. 4121. Springer. pp. 102–115. doi:10.1007/11814948_13 (https://doi.org/10.1007%2F11814948_13). ISBN 978-3-540-37207-3.
7. Vizel, Y.; Weissenbacher, G.; Malik, S. (2015). "Boolean Satisfiability Solvers and Their Applications in Model Checking". *Proceedings of the IEEE*. **103** (11): 2021–2035. doi:10.1109/JPROC.2015.2455034 (https://doi.org/10.1109%2FJPROC.2015.2455034). S2CID 10190144 (https://api.semanticscholar.org/CorpusID:10190144).
8. Cook, Stephen A. (1971). "The complexity of theorem-proving procedures" (http://www.cs.toronto.edu/~sacook/homepage/1971.pdf) (PDF). *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. pp. 151–158. CiteSeerX 10.1.1.406.395 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.406.395). doi:10.1145/800157.805047 (https://doi.org/10.1145%2F800157.805047). S2CID 7573663 (https://api.semanticscholar.org/CorpusID:7573663). Archived (https://ghostarchive.org/archive/20221009/http://www.cs.toronto.edu/~sacook/homepage/1971.pdf) (PDF) from the original on 2022-10-09.
9. Levin, Leonid (1973). "Universal search problems (Russian: Универсальные задачи перебора, Universal'nye perebornye zadachi)". *Problems of Information Transmission (Russian: Проблемы передачи информа́ции, Problemy Peredachi Informatsii)*. **9** (3): 115–116. (pdf) (https://www.mathnet.ru/php/getFT.phtml?jrnid=ppi&paperid=914&volume=9&year=1973&issue=3&fpage=115&what=fullt&option_lang=eng) (in Russian), translated into English by Trakhtenbrot, B. A. (1984). "A survey of Russian approaches to *perebor* (brute-force searches) algorithms". *Annals of the History of Computing*. **6** (4): 384–400. doi:10.1109/MAHC.1984.10036 (https://doi.org/10.1109%2FMAHC.1984.10036). S2CID 950581 (https://api.semanticscholar.org/CorpusID:950581).
10. Aho, Hopcroft & Ullman (1974), Theorem 10.4.
11. Aho, Hopcroft & Ullman (1974), Theorem 10.5.

12. Schöning, Uwe (Oct 1999). "A probabilistic algorithm for k-SAT and constraint satisfaction problems" (http://homepages.cwi.nl/~rdewolf/schoning99.pdf) (PDF). *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. pp. 410–414. doi:10.1109/SFFCS.1999.814612 (https://doi.org/10.1109%2FSFFCS.1999.814612). ISBN 0-7695-0409-4. S2CID 123177576 (https://api.semanticscholar.org/CorpusID:123177576). Archived (https://ghostarchive.org/archive/20221009/http://homepages.cwi.nl/~rdewolf/schoning99.pdf) (PDF) from the original on 2022-10-09.
13. Selman, Bart; Mitchell, David; Levesque, Hector (1996). "Generating Hard Satisfiability Problems". *Artificial Intelligence*. **81** (1–2): 17–29. CiteSeerX 10.1.1.37.7362 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.7362). doi:10.1016/0004-3702(95)00045-3 (https://doi.org/10.1016%2F0004-3702%2895%2900045-3).
14. Schaefer, Thomas J. (1978). "The complexity of satisfiability problems" (http://www.ccs.neu.edu/home/lieber/courses/csg260/f06/materials/papers/max-sat/p216-schaefer.pdf) (PDF). *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. San Diego, California. pp. 216–226. CiteSeerX 10.1.1.393.8951 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.393.8951). doi:10.1145/800133.804350 (https://doi.org/10.1145%2F800133.804350).
15. Schaefer (1978), p. 222, Lemma 3.5.
16. Arkin, Esther M.; Banik, Aritra; Carmi, Paz; Citovsky, Gui; Katz, Matthew J.; Mitchell, Joseph S. B.; Simakov, Marina (2018-12-11). "Selecting and covering colored points" (https://doi.org/10.1016%2Fj.dam.2018.05.011). *Discrete Applied Mathematics*. **250**: 75–86. doi:10.1016/j.dam.2018.05.011 (https://doi.org/10.1016%2Fj.dam.2018.05.011). ISSN 0166-218X (https://search.worldcat.org/issn/0166-218X).
17. Buning, H.K.; Karpinski, Marek; Flogel, A. (1995). "Resolution for Quantified Boolean Formulas" (https://doi.org/10.1006%2Finco.1995.1025). *Information and Computation*. **117** (1). Elsevier: 12–18. doi:10.1006/inco.1995.1025 (https://doi.org/10.1006%2Finco.1995.1025).
18. Moore, Cristopher; Mertens, Stephan (2011), *The Nature of Computation* (https://books.google.com/books?id=z4zMiZyAE1kC&pg=PA366), Oxford University Press, p. 366, ISBN 9780199233212.
19. R. E. Bryant, S. M. German, and M. N. Velev, Microprocessor Verification Using Efficient Decision Procedures for a Logic of Equality with Uninterpreted Functions (http://portal.acm.org/citation.cfm?id=709275), in Analytic Tableaux and Related Methods, pp. 1–13, 1999.
20. Alhazov, Artiom; Martín-Vide, Carlos; Pan, Linqiang (2003). "Solving a PSPACE-Complete Problem by Recognizing P Systems with Restricted Active Membranes" (https://www.researchgate.net/publication/220444503). *Fundamenta Informaticae*. **58**: 67–77. Here: Sect.3, Thm.3.1
21. Blass, Andreas; Gurevich, Yuri (1982-10-01). "On the unique satisfiability problem" (https://doi.org/10.1016%2FS0019-9958%2882%2990439-9). *Information and Control*. **55** (1): 80–88. doi:10.1016/S0019-9958(82)90439-9 (https://doi.org/10.1016%2FS0019-9958%2882%2990439-9). hdl:2027.42/23842 (https://hdl.handle.net/2027.42%2F23842). ISSN 0019-9958 (https://search.worldcat.org/issn/0019-9958).
22. "Complexity Zoo:U - Complexity Zoo" (https://web.archive.org/web/20190709142353/https://complexityzoo.uwaterloo.ca/Complexity_Zoo:U#US). *complexityzoo.uwaterloo.ca*. Archived from the original (https://complexityzoo.uwaterloo.ca/Complexity_Zoo:U#US) on 2019-07-09. Retrieved 2019-12-05.
23. Kozen, Dexter C. (2006). "Supplementary Lecture F: Unique Satisfiability" (https://www.springer.com/gp/book/9781846282973). *Theory of Computation*. Texts in Computer Science. Springer. p. 180. ISBN 9781846282973.
24. Valiant, L.; Vazirani, V. (1986). "NP is as easy as detecting unique solutions" (http://www.cs.princeton.edu/courses/archive/fall05/cos528/handouts/NP_is_as.pdf) (PDF). *Theoretical Computer Science*. **47**: 85–93. doi:10.1016/0304-3975(86)90135-0 (https://doi.org/10.1016%2F0304-3975%2886%2990135-0).
25. Buldas, Ahto; Lenin, Aleksandr; Willemson, Jan; Charnamord, Anton (2017). "Simple Infeasibility Certificates for Attack Trees". In Obana, Satoshi; Chida, Koji (eds.). *Advances in Information and Computer Security*. Lecture Notes in Computer Science. Vol. 10418. Springer International Publishing. pp. 39–55. doi:10.1007/978-3-319-64200-0_3 (https://doi.org/10.1007%2F978-3-319-64200-0_3). ISBN 9783319642000.
26. Gi-Joon Nam; Sakallah, K. A.; Rutenbar, R. A. (2002). "A new FPGA detailed routing approach via search-based Boolean satisfiability" (https://web.archive.org/web/20160315003856/http://cs-rutenbar.web.engr.illinois.edu/wp-content/uploads/2012/10/rutenbar-sattranscad02.pdf) (PDF). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. **21** (6): 674. doi:10.1109/TCAD.2002.1004311 (https://doi.org/10.1109%2FTCAD.2002.1004311). Archived from the original (http://cs-rutenbar.web.engr.illinois.edu/wp-content/uploads/2012/10/rutenbar-sattranscad02.pdf) (PDF) on 2016-03-15. Retrieved 2015-09-04.

27. Selsam, Daniel; Lamm, Matthew; Bünz, Benedikt; Liang, Percy; de Moura, Leonardo; Dill, David L. (11 March 2019). "Learning a SAT Solver from Single-Bit Supervision". arXiv:1802.03685 (https://arxiv.org/abs/1802.03685) [cs.AI (https://arxiv.org/archive/cs.AI)].
28. "The international SAT Competitions web page" (http://www.satcompetition.org/). Retrieved 2007-11-15.

Sources

- This article includes material from https://web.archive.org/web/20070708233347/http://www.sigda.org/newsletter/2006/eNews_061201.html by Prof. Karem A. Sakallah.

Further reading

(by date of publication)

- Garey, Michael R.; Johnson, David S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. pp. A9.1: LO1–LO7, pp. 259–260. ISBN 0-7167-1045-5.
- Marques-Silva, J.; Glass, T. (1999). "Combinational equivalence checking using satisfiability and recursive learning". *Design, Automation and Test in Europe Conference and Exhibition, 1999. Proceedings (Cat. No. PR00078)* (http://eprints.soton.ac.uk/265003/1/jpms-date99a.pdf) (PDF). p. 145. doi:10.1109/DATE.1999.761110 (https://doi.org/10.1109%2FDATE.1999.761110). ISBN 0-7695-0078-1. Archived (https://ghostarchive.org/archive/20221009/http://eprints.soton.ac.uk/265003/1/jpms-date99a.pdf) (PDF) from the original on 2022-10-09.
- Clarke, E.; Biere, A.; Raimi, R.; Zhu, Y. (2001). "Bounded Model Checking Using Satisfiability Solving". *Formal Methods in System Design*. **19**: 7–34. doi:10.1023/A:1011276507260 (https://doi.org/10.1023%2FA%3A1011276507260). S2CID 2484208 (https://api.semanticscholar.org/CorpusID:2484208).
- Giunchiglia, E.; Tacchella, A. (2004). Giunchiglia, Enrico; Tacchella, Armando (eds.). *Theory and Applications of Satisfiability Testing*. Lecture Notes in Computer Science. Vol. 2919. doi:10.1007/b95238 (https://doi.org/10.1007%2Fb95238). ISBN 978-3-540-20851-8. S2CID 31129008 (https://api.semanticscholar.org/CorpusID:31129008).
- Babic, D.; Bingham, J.; Hu, A. J. (2006). "B-Cubing: New Possibilities for Efficient SAT-Solving" (http://www.domagoj-babic.com/uploads/Pubs/TCOM06/tcom06.pdf) (PDF). *IEEE Transactions on Computers*. **55** (11): 1315. doi:10.1109/TC.2006.175 (https://doi.org/10.1109%2FTC.2006.175). S2CID 14819050 (https://api.semanticscholar.org/CorpusID:14819050).
- Rodriguez, C.; Villagra, M.; Baran, B. (2007). "Asynchronous team algorithms for Boolean Satisfiability" (https://www.cc.pol.una.py/lcca/publicaciones/optimizacion/2007/Asynchronous%20Team%20Algorithms%20for%20Boolean%20Satisfiability.pdf) (PDF). *2007 2nd Bio-Inspired Models of Network, Information and Computing Systems*. pp. 66–69. doi:10.1109/BIMNICS.2007.4610083 (https://doi.org/10.1109%2FBIMNICS.2007.4610083). S2CID 15185219 (https://api.semanticscholar.org/CorpusID:15185219).
- Gomes, Carla P.; Kautz, Henry; Sabharwal, Ashish; Selman, Bart (2008). "Satisfiability Solvers". In Harmelen, Frank Van; Lifschitz, Vladimir; Porter, Bruce (eds.). *Handbook of knowledge representation*. Foundations of Artificial Intelligence. Vol. 3. Elsevier. pp. 89–134. doi:10.1016/S1574-6526(07)03002-7 (https://doi.org/10.1016%2FS1574-6526%2807%2903002-7). ISBN 978-0-444-52211-5.
- Vizel, Y.; Weissenbacher, G.; Malik, S. (2015). "Boolean Satisfiability Solvers and Their Applications in Model Checking". *Proceedings of the IEEE*. **103** (11): 2021–2035. doi:10.1109/JPROC.2015.2455034 (https://doi.org/10.1109%2FJPROC.2015.2455034). S2CID 10190144 (https://api.semanticscholar.org/CorpusID:10190144).
- Knuth, Donald E. (2022). "Chapter 7.2.2.2: Satifiability". *The Art of Computer Programming*. Vol. 4B: Combinatorial Algorithms, Part 2. Addison-Wesley Professional. pp. 185–369. ISBN 978-0-201-03806-4.