A History of Satisfiability

Armin Biere, Hans van Maaren, Toby Walsh, editors [1]

August 7, 2007

[1] With sections contributed by John Franco, John Martin, Hantao Zhang, and others to be named.

Chapter 1

A History of Satisfiability

1.1 Preface: the concept of satisfiability

Logic is about *validity* and *consistency*. The two ideas are interdefinable if we make use of negation ($\neg$): the argument from $p_1, \ldots, p_n$ to q is valid if and only if the set $\{p_1, \ldots, p_n, \neg q\}$ is inconsistent. Thus, validity and consistency are really two ways of looking at the same thing and each may be described in terms of syntax or semantics.

The syntactic approach gives rise to *proof theory*. Syntax is restricted to definitions that refer to the syntactic form (that is, grammatical structure) of the sentences in question. In proof theory the term used for the syntactic version of validity is *derivability*. Proofs are derived with respect to an *axiom system* which is defined syntactically as consisting of a set of axioms with a specified grammatical form and a set of inference rules that sanction proof steps with specified grammatical forms. Given an axiom system, derivability is defined as follows: q is derivable from $p_1, \ldots, p_n$ (in symbols, $p_1, \ldots, p_n \vdash q$) if and only if there is a proof in the axiom system of q (derivable) from $p_1, \ldots, p_n$. Because the axioms and rules are defined syntactically, so is the notion of derivability. The syntactic version of consistency is simply called *consistency*, and is defined as follows: $\{p_1, \ldots, p_n\}$ is *consistent* if and only if it is not possible to derive a contradiction from $\{p_1, \ldots, p_n\}$. It follows that $\{p_1, \ldots, p_n\}$ is inconsistent if and only if, there is some contradiction $q \wedge \neg q$ such that $\{p_1, \ldots, p_n\} \vdash q \wedge \neg q$. Since derivability has a definition that only makes reference to the syntactic shapes, and since consistency is defined in terms of derivability, it follows that consistency too is a syntactic concept, and is defined ultimately in terms of grammatical form alone. The inference rules of axiom systems, moreover, are always chosen so that derivability and consistency are interdefinable: that is, $p_1, \ldots, p_n \vdash q$ if and only if $\{p_1, \ldots, p_n, \neg q\}$ is inconsistent.

The semantic approach gives rise to *model theory*. Semantics studies the way sentences relate to “the world.” Truth is the central concept in semantics because a sentence is said to be true if it “corresponds to the world.” The concept of algebraic structure is used to make precise the meaning of “corresponds to the world.”

An *algebraic structure*, or simply structure, consists of a non-empty set of objects existing in the world w, called the domain and denoted below by D, and a function, called an *interpretation* and denoted below by R, that assigns to each constant an entity in D, to each predicate a relation among entities in D, and to each functor a function among entities in D. A sentence p is said to be true in w if the entities chosen as the interpretations of the sentence's terms and functors *stand* to the relations chosen as the interpretation of the sentence's predicates. We denote a structure by $\langle D, R\rangle$. Below we sometimes use $\mathcal{A}$ to stand for $\langle D, R\rangle$ by writing $\mathcal{A} = \langle D, R\rangle$. A more traditional, algebraic notation for structure is stated in the glossary, Page 81, and used in Section 1.6. We will speak of *formulas* instead of sentences to allow for the possibility that a sentence contains free variables. The customary notation is to use $\mathcal{A} \models p$ to say *p is true in the structure $\mathcal{A}$*.

The semantic versions of validity and consistency are defined in terms of the concept of structure. In model theory validity is just called *validity.* Intuitively, an argument is valid if whenever the premises are true, so is the conclusion. More precisely, the argument from $p_1, \ldots, p_n$ to q is *valid* (in symbols, $p_1, \ldots, p_n \models q$) if and only if, for all structures $\mathcal{A}$, if $\mathcal{A} \models p_1, \ldots, \mathcal{A} \models p_n$, then $\mathcal{A} \models q$.

We are now ready to encounter, for the first time, *satisfiability*, the central concept of this book. Satisfiability is the semantic version of consistency. A set of formulas is said to be satisfiable if there is some structure in which all its component formulas are true: that is, $\{p_1, \ldots, p_n\}$ is *satisfiable* if and only if, for some $\mathcal{A}$, $\mathcal{A} \models p_1$ and $\ldots$ and $\mathcal{A} \models p_n$. It follows from the definitions that validity and satisfiability are mutually definable: $p_1, \ldots, p_n \models q$ if and only if $\{p_1, \ldots, p_n, \neg q\}$ is unsatisfiable.

Although the syntactic and semantic versions of validity and consistency - namely derivability and consistency, on the one hand, and validity and satisfiability, on the other - have different kinds of definitions, the concepts from the two branches of logic are systematically related. As will be seen later, for the languages studied in logic it is possible to devise axiom systems in such a way that the syntactic and semantic concepts match up so as to coincide exactly. Derivability coincides with validity (*i.e.* $p_1, \ldots, p_n \vdash q$ if and only if $p_1, \ldots, p_n \models q$), and consistency coincides with satisfiability (*i.e.* $\{p_1, \ldots, p_n\}$ is consistent if and only if $\{p_1, \ldots, p_n\}$ is satisfiable). Such an axiom system is said to be complete.

We have now located satisfiability, the subject of this book, in the broader geography made up of logic's basic ideas. Logic is about both validity and consistency, which are interdefinable in two different ways, one syntactic and one semantic. Among these another name for the semantic version of consistency is satisfiability. Moreover, when the language possesses a complete axiom system, as it normally does in logic, satisfiability also coincides exactly with syntactic consistency. Because of these correspondences, satisfiability may then be used to "characterize" validity (because $p_1, \ldots, p_n \models q$ if and only if $\{p_1, \ldots, p_n, \neg q\}$ is unsatisfiable) and derivability (because $p_1, \ldots, p_n \vdash q$ if and only if $\{p_1, \ldots, p_n, \neg q\}$ is unsatisfiable).

There is a further pair of basic logical ideas closely related to satisfiability: *necessity* and *possibility.* Traditionally, a sentence is said to be necessary (or necessarily true) if it is true in all possible worlds, and possible (or possibly true) if it is true in at least one possible world. If we understand a possible world to be a structure, possibility turns out to be just another name for satisfiability. A possible truth is just one that is satisfiable. In logic, the technical name for a necessary formula is logical truth: p is defined to be a *logical truth* (in symbols, $\models p$) if and only if, for all $\mathcal{A}$, $\mathcal{A} \models p$. (In sentential logic a logical truth is called a *tautology.*) Moreover, *necessary* and *possible* are predicates of the metalanguage (the language of logical theory) because they are used to describe sentences in the "object" language (the language that refers to entities in the world that is the object of investigation in logical theory).

There is one further twist. In the concept of consistency we have already the syntactic version of satisfiability. There is also a syntactic version of a logical truth, namely a theorem-in-an-axiom-system. We say p is a theorem of the system (in symbols $\models p$) if and only if p is derivable from the axioms alone. In a complete system, theorem-hood and logical truth coincide: $\vdash p$ if and only if $\models p$. Thus, in logical truth and theorem-hood we encounter yet another pair of syntactic and semantic concepts that, although they have quite different sorts of definitions, nevertheless coincide exactly. Moreover, a formula is necessary if it is not possibly not true. In other words, $\models p$ if and only if it is not the case that p is unsatisfiable. Therefore, satisfiability, theorem-hood, logical truths and necessities are mutually "characterizable."

This review shows how closely related satisfiability is to the central concepts of logic. Indeed, relative to a complete axiom system, satisfiability may be used to define, and may be defined by, the other basic concepts of the field - validity, derivability, consistency, necessity, possibility, logical truth, tautology, and theorem-hood.

However, although we have taken the trouble to clearly delineate the distinction between syntax and semantics in this section, it took over 2000 years before this was clearly enunciated by Tarski in the 1930s. Therefore, the formal notion of satisfiability was absent until then, even though it was informally understood since Aristotle.

The early history of satisfiability, which will be sketched in the next sections, is the story of the gradual enrichment of languages from very simple languages that talk about crude physical objects and their properties, to quite sophisticated languages that can describe the properties of complex structures and computer programs. For all of these languages, the core concepts of logic apply. They all have a syntax with constants that stand for entities and with predicates that stand for relations. They all have sentences that are true or false relative to possible worlds. They all have arguments that are valid or invalid. They all have logical truths that hold in every structure. They all have sets that are satisfiable and others that are unsatisfiable. For all of them, logicians have attempted to devise complete axiom systems to provide syntactic equivalents that capture, in the set of theorems, exactly the set of logical truths, that replicate in syntactic derivations exactly the valid arguments, and provide derivations of contradictions from every unsatisfiable set. We shall even find examples in these early systems of attempts to define decision procedures for logical concepts. As we shall see, in all these efforts the concept of satisfiability is central.

1.2 The ancients

It was in Athens that logic as a science was invented by Aristotle (384-322 B.C.). In a series of books called the Organon, he laid the foundation that was to guide the field for the next 2000 years. The logical system he invented, which is called the syllogistic or *categorical* logic, uses a simple syntax limited to subject-predicate sentences.

Aristotle and his followers viewed language as expressing ideas that signify entities and the properties they instantiate in the "world" outside the mind. They believed that concepts are combined to "form" subject-predicate propositions in the mind. A mental thought of a proposition was something like being conscious of two concepts at once, the subject S and the predicate P. Aristotle proposed four different ways to capture this notion of thought, with respect to a given "world" w, depending on whether we link the subject and predicate universally, particularly, positively, or negatively: that is, every S is P, no S is P, some S is P, and some S is not P. These *categorical propositions* were called, respectively, A (universal affirmative), E (universal negative), I (particular affirmative), and O (particular negative) propositions. Their truth-conditions are defined as follows:

A:	Every S is P is true in w	iff	Everything in w signified by S is something signified in w by S and P
E:	No S is P is true in w	iff	Some S is P is false in w

I:	Some S is P is true in w	iff	There is some T such that everything signified in w by S and P is something that is signified in w by P and T
O:	Some S is not P is true in w	iff	Every S is P is false in w

These propositions have the following counterparts in set theory:

$$\begin{array}{lcl}
S \subseteq P & \text{iff} & S = S \cap P \\
S \cap P = \emptyset & \text{iff} & \neg(S \cap P \neq \emptyset) \\
S \cap P \neq \emptyset & \text{iff} & \exists T : S \cap P = P \cap T \\
S \cap \bar{P} \neq \emptyset & \text{iff} & \neg(S = S \cap P)
\end{array}$$

The resulting logic is two-valued: every proposition is true or false. It is true that Aristotle doubted the universality of this bivalence. In a famous discussion of so-called future contingent sentences, such as "there will be a sea battle tomorrow," he pointed out that a sentence like this, which is in the future tense, is not now determined to be either true or false. In modern terms such a sentence "lacks a truth-value" or receives a third truth-value. In classical logic, however, the law of excluded middle (commonly known as *tertium non datur*), that is, p or not p is always true, was always assumed.

Unlike modern logicians who accept the empty set, classical logicians assumed, as a condition for truth, that a proposition's concepts must signify at least one existing thing. Thus, the definitions above work only if T is a non-empty set. It follows that A and E propositions cannot both be true (they are called contraries), and that I and O propositions cannot both be false. The definitions are formulated in terms of identity because doing so allowed logicians to think of mental proposition formulation as a process of one-to-one concept comparison, a task that conciousness seemed perfectly capable of doing.

In this theory of mental language we have encountered the first theory of satisfiability. A proposition is satisfiable (or possible, as traditional logicians would say) if there is some world in which it is true. A consistent proposition is one that is satisfiable. Some propositions were recognized as necessary, or always true, for example: every S is S.

Satisfiability can be used to show that propositions $p_1, \ldots, p_n$ do not logically imply q: one only needs to show that there is some assignment of concepts to the terms so that all the propositions in $\{p_1, \ldots, p_n, \neg q\}$ come out true. For example, consider the statement:

$$(\text{some } M \text{ is } A \wedge \text{some } C \text{ is } A) \rightarrow \text{every } M \text{ is } C.$$

Aristotle would show this statement is false by replacing the letters with familiar terms to obtain the requisite truth values: some man is an animal and some cow is an animal are both true, but every man is a cow is false. In modern terms, we say the set

$$\{ \text{ some } M \text{ is } A, \text{ some } C \text{ is } A, \neg(\text{ every } M \text{ is } C \text{ }) \text{ } \}$$

is satisfiable.

The means to deduce (that is, provide a valid argument) was built upon syllogisms, using what is essentially a complete axiom system for any conditional $(p_1, \ldots, p_n) \rightarrow q$ in which $p_1, \ldots, p_n$, and q are categorical propositions [197]. A syllogism is defined as a conditional $(p \wedge q) \rightarrow r$ in which p,q, and r are A, E, I, or O propositions. To show that propositions are valid, that is $(p_1 \wedge \ldots \wedge p_n) \rightarrow q$, Aristotle would create syllogisms $(p_1 \wedge p_2) \rightarrow r_1, (r_1 \wedge p_3) \rightarrow r_2, \ldots, (r_{n-2} \wedge p_n) \rightarrow q$, then repeatedly reduce valid syllogisms to one of

A1: (every X is Y $\wedge$ every Y is Z) $\rightarrow$ every X is Z
A2: (every X is Y $\wedge$ no Y is Z) $\rightarrow$ no X is Z

The reduction, when viewed in reverse, is an axiom system where A1 and A2 are axiom schemata, from which are deduced the valid syllogisms, and from the valid syllogisms are deduced all valid conditionals. The system used four inference rules:

R1: From $(p \wedge q) \rightarrow r$ infer $(\neg r \wedge q) \rightarrow \neg p$
R2: From $(p \wedge q) \rightarrow r$ infer $(q \wedge p) \rightarrow r$
R3: From no X is Y infer no Y is X
R4: From $(p \wedge q) \rightarrow$ no X is Y infer $(p \wedge q) \rightarrow$ some X is not Y

For example, to prove

$$(\text{ every } P \text{ is } M \wedge \text{ no } S \text{ is } M) \rightarrow \text{ some } S \text{ is not } P$$

deduce

1. (every P is M $\wedge$ no M is S) $\rightarrow$ no P is S Axiom A2
2. (every P is M $\wedge$ no S is M) $\rightarrow$ no S is P Rule R3
3. (every P is M $\wedge$ no S is M) $\rightarrow$ some S is not P Rule R4

The logic of the Stoics (c. 300s-200s BC) developed into a sophisticated sentential logic, using operators $\rightarrow$, $\wedge$, $\neg$, and $\vee$, where a proposition is the *meaning* of a sentence that expresses it and the truth of a proposition may change over time. They combined this with the standard definition of validity to discover a series of propositional inferences that have remained part of logical lore ever since:

$p, p \rightarrow q \models q$	(modus ponens)
$\neg q, p \rightarrow q \models \neg p$	(modus tollens)
$\neg q, p \vee q \models p$	(disjunctive syllogism)
$p \rightarrow q, q \rightarrow r \models p \rightarrow r$	(hypothetical syllogism)

1.3 The medieval period

The development of concepts open to decision by an effective process (such as a *mechanical process*) was actually an important goal of early modern logic, although it was not formulated in those terms. A goal of symbolic logic is to make epistemically transparent judgments that a formula is a theorem of an axiom system or is deducible within an axiom system. An effective process ensures this transparency because it is is possible to know with relative certainty that each stage in the process is properly carried out.

Logicians of the medieval period knew all of the logic of Aristotle and the Stoics, and much more. The syntax of the languages they used was rich, incorporating combined categorical propositions (with and without modal and epistemic operators), other quantifiers, restrictive and non restrictive relative clauses, and the propositional connectives into complex sentences. Moreover, although they did not have set theory, they described interpretions of predicates using set-like notions such as “group” or “collection.”

The work of Ramon Lull (1232-1315) was influential beyond the medieval period. He devised the first system of logic based on diagrams expressing truth and rotating wheels to achieve some form of deduction. It had similarities to important later work, for example Venn circles, and greatly influenced Leibniz in his quest for a system of deduction that would be universal.

1.4 The renaissance

In the 17^{th} century Descartes and Leibniz began to understand the power of applying algebra to scientific reasoning. To this end, Leibniz devised a language that could be used to talk about either ideas or the world. He thought, like we do, that sets stand to one another in the subset relation $\subseteq$, and that a new set can be formed by intersection $\cap$. He also thought that concepts can be combined by definitions: for example the concepts *animal* and *rational* can be combined to form the concept *rational+animal*, which is the definition of the concept *man*, and the concept animal would then be a "part" of the concept *man*. The operator $\preceq$, called concept inclusion, was introduced to express this notion: thus, animal $\preceq$ man.

Leibniz worked out dual Boolean interpretations of syllogistic propositions joined with the propositional connectives. The first (*intensional*) interpretation assigns terms to "concepts" within a structure of concepts ordered by $\preceq$ and organized by operations that we would call *meet* and *join*. The dual (*extensional*) interpretation is over a Boolean algebra of sets.

The logical operations of multiplication, addition, negation, identity, class inclusion, and the null class were known at this time, well before Boole, but Leibniz published nothing on his ideas related to formal logic. In addition to $\preceq$ his logic is rooted in the operators of identity ($=$), and a conjunction-like operator ($\oplus$) called *real addition*, which obeys the following:

$$\begin{array}{ll} t \oplus t = t & \text{(idempotency)} \\ t \oplus t' = t' \oplus t & \text{(commutativity)} \\ t \oplus (t' \oplus t'') = (t \oplus t') \oplus t'' & \text{(associativity)} \end{array}$$

where t, t', and t'' are terms representing substance or ideas. The following, Leibniz's equivalence, shows the tie between set inclusion and real addition that is the basis of his logics.

$$t \preceq t' \text{ if and only if } t \oplus t' = t'$$

Although Leibniz's system is simplistic and ultimately implausible as an account of science, he came very close to defining with modern rigor complete axiom systems for well-defined formal languages that also possessed decision procedures for identifying the sentences satisfied in every interpretation. It would be 250 years before modern logic accomplished the same for its more complex languages. His vision is relevant to modern times in other ways too. As examples, he invented binary arithmetic, and his *calculus ratiocinator* is regarded by some as a formal inference engine, its use not unlike that of a computer programming language, and by others as referring to a "calculating machine" that was a forerunner to the modern digital computer. In fact, Leibniz constructed a machine, called a Stepped Reckoner, for mathematical calculations. Yet, at this point in history, the notion of satisfiability still had not been enunciated.

1.5 The first logic machine

According to Gardner [106] the first *logic machine*, that is the first machine able to solve problems in formal logic (in this case syllogisms), was invented by Charles Stanhope, 3rd Earl Stanhope (1753-1816). It employed methods similar to Venn circles (Section 1.6) and therefore can in some sense be regarded as

Boolean. However, it was also able to go beyond traditional syllogisms to solve numerical syllogisms such as the following: 8 of 10 pigeons are in holes, and 4 of the 10 pigeons are white (conclude at least 2 holes have white pigeons). See [106] for details.

1.6 Boolean algebra

George Boole (1815-1864) advanced the state of logic considerably with the introduction of the algebraic structure $\langle B, \vee, \wedge, \neg, 0, 1\rangle$ that bears his name[1]: a structure $\langle B, \vee, \wedge, \neg, 0, 1\rangle$ is a Boolean algebra if and only if $\vee$ and $\wedge$ are binary operations and $\neg$ is a unary operation on B under which B is closed, $1, 0 \in B$, and

$$\begin{array}{ll}
x \wedge y = y \wedge x; & x \vee \neg x = 1; \\
x \vee y = y \vee x; & 1 \wedge x = x; \\
x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z); & 0 \vee x = x; \\
x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z); & x \wedge \neg x = 0;
\end{array}$$

Boole's main innovation was to develop an algebra-like notation for the elementary properties of sets. His own objective, at which he succeeded, was to provide a more general theory of the logic of terms which has the traditional syllogistic logic as a special case. He was one of the first to employ a symbolic language. His notation consisted of term variables s,t,u,v,w,x,y,z etc., which he interpreted as standing for sets, the standard "Boolean" operators on sets (complementation indicated by $-$, union indicated by $+$, intersection indicated by $\cdot$), constants for the universal and empty set (1 and 0), and the identity sign ($=$) used to form equations. He formulated many of the standard "laws" of Boolean algebra, including association, commutation, and distributions, and noted many of the properties of complementation and the universal and empty sets. He symbolized Aristotle's four categorical propositions, relative to subject y and predicate x, by giving names of convenience V ($V \neq 0$) to sets that intersect appropriately with y and x forming subsets:

	Boole	**Sets**
Every x is y:	$x = x \cdot y$	$x \subseteq y$
No x is y:	$0 = x \cdot y$	$x \subseteq \bar{y}$
Some x is y:	$V = V \cdot x \cdot y$	$x \cap y \neq \emptyset$
Some x is not y:	$V = V \cdot x \cdot (1 - y)$	$x \cap \bar{y} \neq \emptyset$

Boole was not interested in axiom systems, but in the theory of inference. In his system it could be shown that the argument from $p_1, \ldots, p_n$ to q is valid

[1] The structures we call 'Boolean algebras' were not defined by Boole but by Jevons (see below) who advocated the use of the inclusive or.

(*i.e.* $p_1, \ldots, p_n \models q$) by deriving the equation q from equations $p_1, \ldots, p_n$ by applying rules of inference, which were essentially cases of algebraic substitution.

However, Boole's algebras needed a boost to advance their acceptance. Gardner [106] cites John Venn (1834-1923) and William Stanley Jevons (1835-1882) as two significant contributors to this task. Jevons regarded Boole's work as the greatest advance since Aristotle but saw some flaws that he believed kept it from significantly influencing logicians of the day, particularly that it was too mathematical. To fix this he introduced, in the 1860's, the "method of indirect inference" which is an application of *reductio ad absurdum* to Boolean logic. For example, to handle 'All x is y' and 'No y is z' Jevons would write all possible "class explanations" as triples

$$xyz,\ xy\neg z,\ x\neg yz,\ x\neg y\neg z,\ \neg xyz,\ \neg xy\neg z,\ \neg x\neg yz,\ \neg x\neg y\neg z,$$

where the first represents objects in classes x, y, and z, the second represents objects in classes x and y but not in z, and so on[2]. Then some of these triples would be eliminated by the premises which imply they are empty. Thus, $x\neg yz$ and $x\neg y\neg z$ are eliminated by 'All x is y' and xyz and $\neg xyz$ are eliminated by 'No y is z'. Since no remaining triples contain both x and z, one can conclude 'No x is z' and 'No z is x'.

Although Jevons' system is powerful it has problems with some statements: for example, 'Some x is y' cannot eliminate any of the triples xyz and $xy\neg z$ since at least one but maybe both represent valid explanations. Another problem is the exponential growth of terms, although this did not seem to be a problem at the time[3]. Although the reader can see x, y and z taking truth values 0 or 1, Jevons did not appreciate the fact that truth-value logic would eventually replace class logic and his attention was given only to the latter. He did build a logic machine (called a "logic piano" due to its use of keys) that can solve five term problems [150] (1870). The reader is refered to [106] for details.

What really brought clarity to Boolean logic, though, was the contribution of Venn [272] (1880) of which the reader is almost certainly acquainted since it touches several fields beyond logic. We quote the elegantly stated passage of [106] explaining the importance of this work:

[2] Jevons did not use ¬ but lower and upper case letters to distinguish inclusion and exclusion.

[3] Nevertheless, Jevons came up with some labor saving devices such as inked rubber stamps to avoid having to list all possibilities at the outset of a problem.

> It was of course the development of an adequate symbolic notation that reduced the syllogism to triviality and rendered obsolete all the quasi-syllogisms that had been so painfully and exhaustively analyzed by the 19th century logicians. At the same time many a controversy that once seemed so important no longer seemed so. ... Perhaps one reason why these old issues faded so quickly was that, shortly after Boole laid the foundations for an algebraic notation, John Venn came forth with an ingenious improvement to Euler's circles[4]. The result was a diagrammatic method so perfectly isomorphic with the Boolean class algebra, and picturing the structure of class logic with such visual clarity, that even a nonmathematically minded philosopher could "see" what the new logic was all about.

As an example, draw a circle each for x, y, and z and overlap them so all possible intersections are visible. A point inside circle x corresponds to x and a point outside circle x corresponds to $\neg x$ and so on. Thus a point inside circles x and y but outside circle z corresponds to Jevons' $xy\neg z$ triple. Reasoning about syllogisms follows Jevons as above except that the case 'Some x is y' is handled by placing a mark on the z circle in the region representing xy which means it is not known whether the region xyz or $xy\neg z$ contains the x that is y. Then, if the next premise is, say, 'All y is z', the $xy\neg z$ region is eliminated so the mark moves to the xyz region it borders allowing the conclusion 'Some z is x'. Since the connection between 0-1 logic and Venn circles is obvious, syllogisms can be seen as just a special case of 0-1 logic.

The results of Boole, Jevons, and Venn rang down the curtain on Aristotelian syllogisms, ending a reign of over 2000 years. In the remainder of this chapter syllogisms will appear only once, and that is to show cases where they are unable to represent common inferences.

1.7 Frege, logicism, and quantification logic

In the nineteenth century mathematicians like Cauchy and Weierstrass put analysis on a clear mathematical footing by precisely defining its central terms. George Cantor (1845-1918) extended their work by formulating a more global theory of sets that allowed for the precise specification of such important details as when sets exist, when they are identical, and what their cardinality is. Cantor's set theory was still largely intuitive and imprecise, though mathematicians like Dedekind and Peano had axiomatized parts of number theory. Motivated as much by philosophy as logic, the mathematician Gottlob Frege (1848-1925) conceived a project called logicism to deduce from the laws of logic alone "all of mathematics," by which he meant set theory, number theory, and analysis [184]. Logicism as an attempt to deduce arithmetic from the axioms of logic was ultimately a failure. As a research paradigm, however, it made the great contribution of making clear the boundaries of "formal reasoning,"

[3] The difference between Euler circles and Venn circles is that Venn circles show all possible overlaps of classes while there is no such requirement for Euler circles. Therefore, Euler circles cannot readily be used as a means to visualize reasoning in Boolean logic. Leibniz and Ramon Lull also used Euler-like circles [27].

and has allowed deep questions to be posed, but which are still unanswered, about the nature of mathematical truth.

Frege employed a formal language of his own invention, called “concept notation” (Begriffsschrift). In it he invented a syntax which is essential to the formal language we know today for sentential and quantificational logic, and for functions in mathematics. In his *Grundgesetze der Arithmetik* (1890) he used a limited number of “logical axioms,” which consisted of five basic laws of sentential logic, several laws of quantifiers, and several axioms for functions. When he published this work, he believed that he had succeeded in defining within his syntax the basic concepts of number theory and in deducing their fundamental laws from his axioms of logic alone.

1.8 Russell and Whitehead

Bertrand Russell (1882-1970), however, discovered that he could prove in Frege’s system his notorious paradox: if the set of all sets that are not members of themselves is a member of itself, then, by definition, it must not be a member of itself; and if it is not a member of itself then, by definition, it must be a member of itself.

In *Principia Mathematica* (1910-13), perhaps the most important work of early modern logic, Russell and Whitehead simplified Frege’s logical axioms of functions by substituting for a more abstract set describing sets and relations. One of the great weaknesses of traditional logic had been its inability to represent inferences that depend on relations. For example, in the proof of Proposition 1 of his *Elements* Euclid argues, regarding lines, that if AC=AB and BC=AB, it follows by commutativity that CA=AB, and hence that AC=BC (by either the subsitutivity or transitivity or identity). But, the requirement of Aristotelian logic that all propositions take subject-predicate form ‘S is P’ makes it impossible to represent important grammatical facts: for example, that in a proposition $AC = AB$, the subject AC and direct object AB are phrases made up of component terms A, B, and C, and the fact that the verb = is transitive and takes the direct object AB. The typical move was to read “=AC” as a unified predicate and to recast equations in subject-predicate form:

$AC = AB$: the individual AC has the property being-identical-to-AB
$BC = AB$: the individual BC has the property being-identical-to-AB
$AC = BC$: the individual AC has the property being-identical-to-BC

But the third line does not follow by syllogistic logic from the first two.

One of the strange outcomes of early logic is that syllogistic logic was so unsuccessful at representing mathematical reasoning that in the 2000 years in which it reigned there was only one attempt to reproduce geometrical proofs using syllogisms; it did not occur until the 16th century, and it was unsuccessful [134]. Boole's logic shared with Aristotle's an inability to represent relations. Russell, however, was well aware both of the utility of relations in mathematics and the need to reform syntax to allow for their expression [239].

Accordingly, the basic notation of *Principia* allows for variables, one-place predicates standing for sets, and n-place predicates standing for relations. Formulas were composed using the sentential connectives and quantifiers. With just this notation it was possible to define all the constants and functions necessary for arithmetic, and to prove with simplified axioms a substantial part of number theory. It appeared at first that *Principia* had vindicated logicism: arithmetic seemed to follow from logic.

1.9 Gödel's incompleteness theorem

In 1931, Kurt Gödel astounded the mathematical world by proving that the axiom system of *Principia*, and indeed any axiom system capable of formulating the laws of arithmetic, is and must be incomplete in the sense that there will always be some truth of arithmetic that is not a theorem of the system. Gödel proved, in short, that logicism is false - that mathematics in its entirety cannot be deduced from logic. Gödel's result is sweeping in its generality. It remains true, however, that limited parts of mathematics are axiomatizable, for example first-order logic. It is also true that, although incomplete, mathematicians as a rule still stick to axiomatic formulations as their fundamental methodology even for subjects that they know must be incomplete. Axiomatic set theory, for example, contains arithmetic as a part and is therefore provably incomplete. But axiomatics is still the way set theory is studied even within its provable limitations.

Although logicism proved to be false, the project placed logic on its modern foundations. *Principia* standardized the modern notation for sentential and quantificational logic. In the 1930s Hilbert and Bernays christened "first-order" logic as that part of the syntax in which quantifiers bind only variables over individuals, and "higher-order" logic as the syntax in which variables bind predicates, and predicates of predicates, etc.

1.10 Effective process and recursive functions

As logic developed in the 20th century, the importance of effective decidablilty (see Section 1.3) increased and defining effective decidablility itself became a research goal. However, since defining any concept that incorporates as a defining term "epistemic transparency" would require a background theory that explained "knowledge," any direct definition of effective process remained elusive. This difficulty motivated the inductive definitions of recursive function theory, Turing machines, lambda calculus, and more, which, by Church's thesis, succeeds in providing an indirect definition of effective process.

One outcome of logicism was the clarification of some of the basic concepts of computer science: for example, the notion of recursive functions which are supposed to be what a mathematician would intuitively recognize as a "calculation" on the natural numbers. Gödel, in his incompleteness paper of 1931, gave a precise formal definition of the class of primitive recursive functions, which he called "recursive functions." Gödel first singled out three types of functions that all mathematicians agree are calculations and called these recursive functions. He then distinguished three methods of defining new functions from old. These methods moreover were such that everyone agreed they produced a calculation as an output if given calculations as inputs. He then defined the set of recursive functions as the closure of the three basic function types under the three construction methods.

Given the definition of recursive function, Gödel continued his incompleteness proof by showing that for each calculable function, there is a predicate in the language of *Principia* that has that function as its extension. Using this fact, he then showed, in particular, that *Principia* had a predicate T that had as its extension the theorems of *Principia*. In an additional step he showed that *Principia* also possesses a constant c that stands for the so-called liar sentence $\neg Tc$, which says in effect "This sentence is not a theorem." Finally, he demonstrated both that $\neg Tc$ is a truth of arithmetic and that it is not a theorem of *Principia*. He proved, therefore, that the axiom system failed to capture one of the truths of arithmetic and was therefore incomplete.

Crucial to the proof was its initial definition of recursive function. Indeed, by successfully analyzing "effective process," Gödel made a major contribution to theoretical computer science: the computable function. However, in his Princeton lectures of 1934, Gödel, attributing the idea of general recursive functions to a suggestion of Herbrand, did not commit himself to whether all effective functions are characterized by his definition. In 1936, Church [57] and Turing [268] independently proposed a definition of the effectively computable functions. It's equivalence with Gödel's definition was proved in 1943 by Kleene [166]. Emil Post (1897-1954), Andrei Markov (1903-1979), and others confirmed Gödel's work by providing alternative analyses of computable function that are also provably coextensive with his.

1.11 Herbrand's Theorem

Herbrand's theorem relates issues of validity and logical truth into ones of satisfiability, and issues of satisfiability into ones concerning the definition of computable functions on syntactic domains. The proof employs techniques important to computer science. A *Herbrand model* for a formula of first-order logic has as its domain literally those terms generated from terms that occur in the formula p. Moreover, the predicates of the model are true of a term if and only if the formula asserting that the predicate holds of the term occurs in p. The first part of Herbrand's theorem says that p is satisfiable if and only if it is satisfied in its Herbrand model.

Using techniques devised by Skolem, Herbrand showed that the quantified formula p is satisfiable if and only if a specific set of its truth-functional instantiations, each essentially a formula in sentential logic, is satisfiable. Thus, satisfiability of p reduces to an issue of testing by truth-tables the satisfiability of a potentially infinite set S of sentential formulas. Herbrand showed that, for any first-order formula p, there is a decision function f such that $f(p) = 1$ if p is unsatisfiable because, eventually, one of the truth-functions in S will come out false in a truth-table test, but $f(p)$ may be undefined when p is satisfiable because the truth-table testing of the infinite set S may never terminate.

1.12 Model theory and Satisfiability

Although ideas in semantics were central to logic in this period, the primary framework in which logic was studied was the axiom system. But the seemingly obvious need to define the grammar rules for a formal language was skipped over until Gödel gave a completely formal definition of the formulas of his version of simple type theory in his 1931 paper [111].

In the late nineteenth and early twentieth centuries Charles Sanders Peirce, see [221], and Ludwig Wittgenstein [276] had employed the two-valued truth-tables for sentential logic. In 1936 Marshall Stone proved that the more general class of Boolean algebras is of fundamental importance for the interpretation of classical logic. His "representation theorem" showed that any interpretation of sentential logic that assigns to the connectives the corresponding Boolean operators and defines validity as preserving a "designated value" defined as a maximal upwardly closed subset of elements of the algebra (called a "filter") has as its logical truths and valid arguments exactly those of classical logic [258]. The early decades of the twentieth century also saw the development of many-valued logic, in an early form by Peirce [224] and then in more developed versions by Polish logicians lead by Jan Łukasiewicz (1878-1956). Thus, in sentential logic the idea of satisfiability was well understood as "truth relative to an assignment of truth-values" in a so-called "logical matrix" of truth-functions.

A precise notion of satisfiability for first-order logic, however, was not developed until the 1930s in the work of Alfred Tarski (1902-1983) [261, 262, 263]. Tarski's task was to define a set of necessary and sufficient conditions for "p

is true," for any formula p of first-order syntax. His solution was not to define the idea in a single phrase applicable to all formulas, but, like Gödel, to give an inductive definition, first defining *truth* for basic formulas and then extending the definition to more complex formulas. The problem was made complex, however, by the fact that, unlike sentential logic in which the truth-value of the parts immediately determine that of the whole (by reference to truth-tables), when the whole expression is universally quantified, it is unclear how its truth is determined by the interpretation of its part. How does the "truth" of the open formula Fx determine that of $\forall x : Fx$?

Tarski solved the problem in two stages. In the first stage he assigned fixed interpretations to the variables. Having done so, it is possible to say when $\forall x : Fx$ is true if we know whether Fx is true under its various interpretations. If Fx is true under all interpretation of x, then $\forall x : Fx$ is also true under each of these interpretations. If Fx is false under even one interpretation of x, however, $\forall x : Fx$ is false under any interpretation of the variables. Tarski coined the technical term *satisfaction* to refer to truth relative to an interpretation of variables.

Let $\mathcal{A} = \langle D, R\rangle$ be a structure and define a variable assignment as any function s that assigns to each variable an entity in D. Given R and s, all the basic expressions of the syntax have a referent relative to D. We can now inductively define "p is satisfied relative to $\mathcal{A}$ and s." The atomic formula $Ft_1, \ldots, t_n$ is satisfied relative to $\mathcal{A}$ and s if and only if the interpretations of $t_1, \ldots, t_n$ in R and s stand in the relation assigned by R to F. The "satisfaction" of molecular formulas made up by the sentential connectives are determined by the two-valued truth-tables depending on whether or not their parts are satisfied. Finally, $\forall x : Fx$ is satisfied relative to $\mathcal{A}$ and s if and only if Fx is satisfied relative to R and every variable assignment s. The notation for "p is satisfied relative to $\mathcal{A}$ and s" is $\mathcal{A} \models_s p$.

The second stage of Tarski's definition is to abstract away from a variable assignment and define the simpler notion "p is true relative to $\mathcal{A}$." His idea at this stage is to interpret an open formula Fx as true in this general sense if it is "always true" in the sense of being satisfied under every interpretation of its variables. That is, he adopts the simple formula: p is true relative to $\mathcal{A}$ if and only if, for all variable assignments s, p is satisfied relative to $\mathcal{A}$ and s. In formal notation, $\mathcal{A} \models p$ if and only if, for all s, $\mathcal{A} \models_s p$.

Logicians have adopted the common practice of using the term "satisfied in a structure" to mean what Tarski called "true in a structure." Thus, it is common to say that p is satisfiable if there is some structure $\mathcal{A}$ such that p is true in $\mathcal{A}$, and that a set of formulas X is satisfiable if and only if there is some structure $\mathcal{A}$ such that for all formulas p in X, p is true (satisfied) in $\mathcal{A}$.

1.13 Completeness of first-order logic

First-order logic has sufficient expressive power for the formalization of virtually all of mathematics. To use it requires a sufficiently powerful axiom system such as the Zermelo-Fraenkel set theory with the axiom of choice (ZFC). It is generally accepted that all of classical mathematics can be formalized in ZFC.

Proofs of completeness of first-order logic under suitable axiom systems date back at least to Gödel in 1929. In this context, completeness means that all logically valid formulas of first-order logic can be derived from axioms and rules of the underlying axiom system. This is not to be confused with Gödel's incompleteness theorem which states that there is no consistent axiom system for the larger set of truths of number theory (which includes the valid formulas of first-order logic as a proper subset) because it will fail to include at least one truth of arithmetic.

Tarski's notion of truth in a structure introduced greater precision. It was then possible to give an elegant proof that first-order logic is complete under its usual axiom systems and sets of inference rules. Of particular interest is the proof due to Leon Henkin (1921-2006) that makes use of two ideas relevant to this book: satisfiability and a structure composed of syntactic elements [132] (1949). Due to the relationship of validity to satisfiability, Henkin reformulated the difficult part of the theorem as: if a set of formulas is consistent, then it is satisfiable. He proved this by first extending a consistent set to what he calls a maximally consistent saturated set, and then showing that this set is satisfiable in a structure made up from the syntactic elements of the set. Although differing in detail, the construction of the structure is similar to that of Herbrand.

Herbrand models and Henkin's maximally consistent saturated sets are relevant prototypes of the technique of constructing syntactic proxies for conventional models. In complexity theory, the truth of predicates is typically determined relative to a structure with a domain of entities that are *programs* or *languages* which themselves have semantic content that allow one to determine a corresponding, more conventional, model theoretic structure. In that sense, the program or language entities can be said to be a proxy for the conventional model and the predicates are second-order, standing for a property of sets.

1.14 Application of logic to circuits

Claude Shannon provided one of the bridges connecting the path of logic over the centuries to its practical applications in the information and digital age. Another bridge is considered in the next section. Whereas the study of logic for thousands of years was motivated by a desire to explain how humans use information and knowledge to deduce facts, particularly to assist in decision making, Shannon, as a student at MIT, saw propositional logic as an opportunity to make rigorous the design and simplification of switching circuits. Boolean algebra, appreciated by a relatively small group of people for decades, was ready to be applied to the fledgling field of digital computers.

In his master's thesis [256] (1940), said by Howard Gardner of Harvard University to be "possibly the most important, and also the most famous, master's thesis of the century," Shannon applied Boolean logic to the design of minimal circuits involving relays. Relays are simple switches that are either "closed," in which case current is flowing through the switch or "open," in which case current is stopped. Shannon represented the state of a relay with a variable taking value 1 if open and the value 0 if closed. His algebra used the operator '+' for "or" (addition - to express the state of relays in series), '$\cdot$' for "and" (multiplication - to express the state of relays in parallel), and his notation for the negation of variable x was x'.

The rigorous synthesis of relay circuits entails expressing and simplifying complex Boolean functions of many variables. To support both goals Shannon developed two series expansions for a function, analogous, in his words, to Taylor's expansion on differentiable functions. He started with

$$f(x_1, x_2, ..., x_n) = x_1 \cdot f(1, x_2, ..., x_n) + x_1' \cdot f(0, x_2, ..., x_n)$$

which we recognize as the basic splitting operation of DPLL algorithms and the *Shannon expansion* which is the foundation for Binary Decision Diagrams (Section 1.19), and its dual

$$f(x_1, x_2, ..., x_n) = (f(0, x_2, ..., x_n) + x_1) \cdot (f(1, x_2, ..., x_n) + x_1').$$

Using the above repeatedly he arrived at the familiar DNF canonical form

$$\begin{aligned} f(x_1, x_2, ..., x_n) \quad = \quad & f(0, 0, ..., 0) \cdot x_1' \cdot x_2' \cdot ... \cdot x_n' \; + \\ & f(1, 0, ..., 0) \cdot x_1 \cdot x_2' \cdot ... \cdot x_n' \; + \\ & f(0, 1, ..., 0) \cdot x_1' \cdot x_2 \cdot ... \cdot x_n' \; + \\ & \quad ... \\ & f(1, 1, ..., 1) \cdot x_1 \cdot x_2 \cdot ... \cdot x_n \end{aligned}$$

and its familiar CNF dual. These support the expression of any relay circuit and, more generally, any combinational circuit. To simplify, he introduced the following operations:

$$\begin{aligned} & x = x + x = x + x + x = ... \\ & x + x \cdot y = x \\ & x \cdot f(x) = x \cdot f(1) \\ & x' \cdot f(x) = x' \cdot f(0) \\ & x \cdot y + x' \cdot z = x \cdot y + x' \cdot z + y \cdot z \end{aligned}$$

and their duals. In the last operation the term $y \cdot z$ is the *consenus* of terms $x \cdot y$ and $x' \cdot z$. The dual of the last operation amounts to adding a propositional *resolvent* to a CNF clause set. The first two operations are *subsumption* rules.

In his master's thesis Shannon stated that one can use the above rules to achieve minimal circuit representations but did not offer a systematic way to do so. Independently, according to Brown [44], Archie Blake, an all but forgotten yet influential figure in Boolean reasoning, developed the notion of consensus in his Ph.D. thesis [34] of 1937. Blake used consensus to express Boolean functions in a minimal DNF form with respect to their *prime implicants* (product g is an implicant of function h if $g \cdot h' = 0$ and is prime if it is minimal in literals) and subsumption. An important contribution of Blake was to show that DNFs are not minimized unless consensus is not possible. The notion of concensus was rediscovered by Samson and Mills [240] (1954) and Quine [233] (1955). Later, Quine [234] (1959) and McCluskey [199] (1959) provided a systematic method for the minimization of DNF expressions through the notion of *essential prime implicants* (necessary prime implicants) by turning the 2-level minimization problem into a covering problem. This was perhaps the first confrontation with complexity issues: although they did not know it at the time, the problem they were trying to solve is $\mathcal{NP}$-complete and the complexity of their algorithm was $O(3^n/\sqrt{n})$, where n is the number of variables.

1.15 Resolution

Meanwhile, the advent of computers stimulated the emergence of the field of automated deduction. Martin Davis has written a history of this period in [81] on which we base this section. Early attempts at automated deduction were aimed at proving theorems from first-order logic because, as stated in Section 1.13, it is accepted that given appropriate axioms as premises, all reasoning of classical mathematics can be expressed in first-order logic. Propositional satisfiability testing was used to support that effort.

However, since the complexity of the problem and the space requirements of a solver were not appreciated at the time, there were several notable failures. At least some of these determined satisfiability either by simple truth table calculations or expansion into DNF and none could prove anything but the simplest theorems. But, each contributed something to the overall effort. According to Davis [81], Gilmore's [109] (1960) system served as a stimulus for others and Prawitz [229] (1960) adopted a modified form of the method of semantic tableaux. Also notable were the "Logic Theory Machine" of [215] (1957), which used the idea of a search heuristic, and the Geometry machine of [107] (1959), which exploited symmetry to reduce proof size.

Things improved when Davis and Putnam proposed using CNF for satisfiability testing as early as 1958 in an unpublished manuscript for the NSA [82].

According to Davis [81], that manuscript cited all the essentials of modern DPLL variants. These include, in the words of Davis:

1. The one literal rule also known as the unit clause rule.
2. The affirmative-negative rule also known as the pure literal rule.
3. The rule for eliminating atomic formulas: that is, replace
$$(v \vee l_{1,1} \vee \ldots \vee l_{1,k_1}) \wedge (\neg v \vee l_{2,1} \vee \ldots \vee l_{2,k_2}) \wedge C$$
with
$$(l_{1,1} \vee \ldots \vee l_{1,k_1} \vee l_{2,1} \vee \ldots \vee l_{2,k_2}) \wedge C$$
if literals $l_{1,i}$ and $l_{2,j}$ are not complementary for any i, j.
4. The splitting rule, called in the manuscript 'the rule of case analysis.'

Observe that rule 3. is ground resolution: the CNF expression it is applied to having come from a prenex form with its clauses grounded. The published version of this manuscript is the often cited [83] (1960). The Davis-Putnam procedure, or DPP, reduced the size of the search space considerably by eliminating variables from a given expression. This was done by repeatedly choosing a target variable v still appearing in the expression, applying all possible ground resolutions on v, then eliminating all remaining clauses still containing v or $\neg v$.

Loveland and Logemann attempted to implement DPP but they found that ground resolution used too much RAM, which was quite limited in those days. So they changed the way variables are eliminated by employing the splitting rule: recursively assigning values 0 and 1 to a variable and solving both resulting subproblems [84] (1962). Their algorithm is, of course, the familiar DPLL.

Robinson also experimented with DPP and, taking ideas from both DPP and Prawitz, he generalized ground resolution so that instead of clauses having to be grounded to use resolution, resolution was lifted directly to the Skolem form [235, 236] (1963,1965). This is, of course, a landmark result in mechanically proving first-order logic sentences.

Resolution was extended by Tseitin in [267] (1968) who showed that, for any pair of variables a, b in a given CNF expression ϕ, the following expression may be appended to ϕ:

$$(z \vee a) \wedge (z \vee b) \wedge (\neg z \vee \neg a \vee \neg b)$$

where z is a variable not in ϕ. The meaning of this expression is: either a and b both have value 1 or at least one of a or b has value 0. Judicious use of such extensions can result in polynomial size refutations for problems that have no polynomial size refutations without extension, a notable example being the pigeon hole formulas. By adding variables not in ϕ one obtains, in linear time, a satisfiability-preserving translation from any propositional expression to CNF with at most a constant factor blowup in expression size.

After this point the term satisfiability was used primarily to describe the problem of finding a model for a Boolean expression. The complexity of Satisfiability became the major issue due to potential practical applications for Satisfiability. Consequently, work branched in many directions. The following sections describe most of the important branches.

1.16 The complexity of resolution

Perhaps the theoretically deepest branch is the study of the complexity of resolution. As seen in previous sections, the question of decidability dominated research in logic until it became important to implement proof systems on a computer. Then it became clear that decidable problems could still be effectively unsolvable due to space and time limitations of available machinery. Thus, many researchers turned their attention to the complexity issues associated with implementing various logic systems. The most important of these, the most relevant to the readers of this chapter, and the subject of this section is the study of resolution refutations of contradictory sets of CNF clauses (in this section clause will mean CNF clause).

Rephrasing the "rule for eliminating atomic formulas" from the previous section: if $A \vee l$ and $B \vee \neg l$ are clauses, then the clause $A \vee B$ may be inferred by the resolution rule, *resolving on* the literal l. A *resolution refutation* of a set of clauses Σ is a proof of the empty clause from Σ by repeated applications of the resolution rule.

Refutations can be represented as trees or as sequences of clauses; the worst case complexity differs considerably depending on the representation. We shall distinguish between the two by describing the first system as "tree resolution," the second simply as "resolution."

Lower bounds on the size of resolution refutations provide lower bounds on the running time of algorithms for the Satisfiability problem. For example, consider the familiar DPLL algorithm that is the basis of many of the most successful algorithms for Satisfiability. If a program based on the splitting rule terminates with the output "The set of clauses Σ is unsatisfiable," then a trace of the program's execution can be given in the form of a binary tree, where each of the nodes in the tree is labeled with an assignment to the variables in Σ. The root of the tree is labeled with the empty assignment, and if a node other than a leaf is labeled with an assignment ϕ, then its children are labeled with the assignments $\phi[v := 0]$ and $\phi[v := 1]$ that extend ϕ to a new variable v; the assignments labeling the leaves all falsify a clause in Σ. Let us call such a structure a "semantic tree," an idea introduced by Robinson [237] and Kowalski and Hayes [170].

A semantic tree for a set of clauses Σ can be converted into a tree resolution refutation of Σ by labeling the leaves with clauses falsified by the assignment at the leaves, and then performing resolution steps corresponding to the splitting moves (some pruning may be necessary in the case that a literal is missing from one of the premisses). It follows that a lower bound on the size of tree resolution refutations for a set of clauses provides a lower bound on the time required for a DPLL-style algorithm to certify unsatisfiability. This lower bound applies no matter what strategies are employed for the order of variable elimination.

The first results on the complexity of resolution were proved by Grigori Tseitin in 1968 [267]. In a remarkable pioneering paper, Tseitin showed that for all $n > 0$, there are contradictory sets of clauses Σ_n , containing $O(n^2)$ clauses with at most four literals in each clauses, so that the smallest tree resolution refutation of Σ_n has $2^{\Omega(n)}$ leaves. Tseitin's examples are based on graphs. If we assign the values 0 and 1 to the edges of a finite graph G, we can define a vertex v in the graph to be *odd* if there are an odd number of vertices attached to v with the value 1. Then Tseitin's clauses $\Sigma(G)$ can be interpreted as asserting that there is a way of assigning values to the edges so that there are an odd number of odd vertices. The set of clauses Σ_n mentioned above is $\Sigma(G_n)$, where G_n is the $n \times n$ square grid.

Tseitin also proved some lower bounds for resolution but only under the restriction that the refutation is *regular*. A resolution proof contains an *irregularity* if there is a sequence of clauses $C_1, \ldots, C_k$ in it, so that C_{i+1} is the conclusion of a resolution inference of which C_i is one of the premisses, and there is a variable v so that C_1 and C_k both contain v, but v does not occur in some intermediate clause C_j, $1 < j < k$. In other words, an irregularity occurs if a variable is removed by resolution, but is later introduced again in a clause depending on the conclusion of the earlier step. A proof is *regular* if it contains no irregularity. Tseitin showed that the lower bound for Σ_n also applies to regular resolution. In addition, he showed that there is a sequence of clauses Π_n so that there is a superpolynomial speedup of regular resolution over tree resolution (that is to say, the size of the smallest tree resolution refutation of Π_n is not bounded by any fixed power of the size of the smallest regular refutation of Π_n).

Tseitin's lower bounds for the graph-based formulas were improved by Zvi Galil [105], who proved a truly exponential lower bound for regular resolution refutations of sets of clauses based on expander graphs E_n of bounded degree. The set of clauses $\Sigma(E_n)$ has size $O(n)$, but the smallest regular resolution refutation of $\Sigma(E_n)$ contains $2^{\Omega(n)}$ clauses.

The most important breakthrough in the complexity of resolution was made by Armin Haken [124], who proved exponential lower bounds for the pigeonhole clauses PHC_n. These clauses assert that there is an injective mapping from the set $\{1, \ldots, n+1\}$ into the set $\{1, \ldots, n\}$. They contain $n+1$ clauses containing n literals asserting that every element in the first set is mapped to some element of the second, and $O(n^3)$ two-literal clauses asserting that no two elements are mapped to the same element of $\{1, \ldots, n\}$. Haken showed that any resolution refutation of PHC_n contains $2^{\Omega(n)}$ clauses.

Subsequently, Urquhart [271] adapted Haken's argument to prove a truly exponential lower bound for clauses based on expander graphs very similar to those used earlier by Galil. The technique used in Urquhart's lower bounds were employed by Chvátal and Szemerédi [59] to prove an exponential lower bound on random sets of clauses. The model of random clause sets is that of the constant width distribution discussed below in Section 1.18. Their main result is as follows: if c, k are positive integers with $k \geq 3$ and $c2^{-k} \geq 0.7$, then there is an $\epsilon > 0$, so that with probability tending to one as n tends to infinity, the random family of cn clauses of size k over n variables is unsatisfiable and its resolution complexity is at least $(1+\epsilon)^n$.

The lower bound arguments used by Tseitin, Galil, Haken and Urquhart have a notable common feature. They all prove lower bounds on size by proving lower bounds on width – the *width* of a clause is the number of literals it contains, while the *width* of a set of clauses is the width of the widest clause in it. If Σ is a contradictory set of clauses, let us write $w(\Sigma)$ for the width of Σ, and $w(\Sigma \vdash 0)$ for the minimum width of a refutation of Σ.

The lower bound techniques used in earlier work on the complexity of resolution were generalized and unified in a striking result due to Ben-Sasson and Wigderson [31]. If Σ is a contradictory set of clauses, containing the variables V, let us write $S(\Sigma)$ for the minimum size of a resolution refutation of Σ. Then the main result of [31] is the following lower bound:

$$S(\Sigma) = \exp\left(\Omega\left(\frac{(w(\Sigma \vdash 0) - w(\Sigma))^2}{|V|}\right)\right).$$

This lower bound easily yields the lower bounds of Urquhart [271], as well as that of Chvátal and Szemerédi [59] via a width lower bound on resolution refutations.

Tseitin was the first to show a separation between tree resolution and general resolution, as mentioned above. The separation he proved is fairly weak, though superpolynomial. Ben-Sasson, Impagliazzo and Wigderson [32] improved this to a truly exponential separation between the two proof systems, using contradictory formulas based on pebbling problems in directed acyclic graphs.

These results emphasize the inefficiency of tree resolution, as opposed to general resolution. A tree resolution may contain a lot of redundancy, in the sense that the same clause may have to be proved multiple times. The same kind of inefficiency is also reflected in SAT solvers based on the DPLL framework, since information accumulated along certain branches is immediately discarded. This observation has led some researchers to propose improved versions of the DPLL algorithm, in which such information is stored in the form of clauses. These algorithms, which go under the name "clause learning," lead to dramatic speedups in some cases – the reader is referred to the paper of Beame, Kautz and Sabharwal [30] for the basic references and some theoretical results on the method.

1.17 Upper bounds

Deciding satisfiability of a CNF formula ϕ with n Boolean variables can be performed in time $O(2^n|\phi|)$ by enumerating all assignments of the variables. The number m of clauses as well as the number l of literals of ϕ are further parameters for bounding the runtime of decision algorithms for SAT. Note that l is equal to length(ϕ) and this is the usual parameter for the analysis of algorithms. Most effort in designing and analyzing algorithms for SAT solving, however, are based on the number n of variables of ϕ.

A first non-trivial upper bound of $O(\alpha_k^n \cdot |\phi|)$, where α_k^n is bounding the Fibonacci-like recursion

$$\begin{aligned}
&T(1) = T(2) = \ldots = T(k-1) = 1 \quad \text{and} \\
&T(n) = T(n-1) + T(n-2) + \ldots + T(n-k+1), \quad \text{for } n \geq k,
\end{aligned}$$

for solving k-SAT, $k \geq 3$, was shown in [212, 213]. For example, $\alpha_3 \geq 1.681$, $\alpha_4 \geq 1.8393$, and $\alpha_5 \geq 1.9276$.

The algorithm supplying the bound looks for a shortest clause c from the current formula. If c has $k-1$ or less literals, e.g. $c = (x_1 \vee \ldots \vee x_{k-1})$ then ϕ is split into $k-1$ subformulas according to the $k-1$ subassignments

$$\begin{array}{rl}
1) & x_1 = 1; \\
2) & x_1 = 0,\, x_2 = 1; \\
 & \ldots \\
k\text{-}1) & x_1 = x_2 = \ldots = x_{k-2} = 0,\, x_{k-1} = 1.
\end{array}$$

If all clauses c of ϕ have length k, then ϕ is split into k subformulas as described, and each subformula either contains a clause of length at most $k-1$ or one of the resulting subformulas only contains clauses of length k. In the former case, the above mentioned bound holds. In the latter case, the corresponding subassignment is *autark*, that is, all clauses containing these variables are satisfied by this subassignment, so ϕ can be evaluated according to this autark subassignment thereby yielding the indicated upper bound.

For the case $k = 3$, the bound was later improved to $\alpha_3 = 1.497$ by a sophisticated case analysis (see [246]). Currently, from [78], the best deterministic algorithms for solving k-SAT have a run time of

$$O\left((2 - \frac{2}{k+1})^n\right).$$

For example, the bounds are $O(1.5^n)$, $O(1.6^n)$, $O(1.666...^n)$ for 3,4,and 5 literal clause formulas. These bounds were obtained by the derandomization of a multistart-random-walk algorithm based on covering codes. In the case of $k = 3$ the bound has been further improved to $O(1.473^n)$ in [43]. This is currently the best bound for deterministic 3-SAT solvers.

Two different probabilistic approaches to solving k-SAT formulas have resulted in better bounds and paved the way for improved deterministic bounds. The first one is the algorithm of Paturi-Pudlak-Zane [227] which is based on the following procedure:

> Determine a truth assignment of the variables of the input formula ϕ by iterating over all variables v of ϕ in a randomly chosen order: If ϕ contains a unit clause $c = (x)$, where $x = v$ or $\bar{v}$, set $t(v)$ such that $t(x) = 1$. If neither v nor $\bar{v}$ occur in a unit clause of ϕ, then randomly choose $t(v)$ from $\{0, 1\}$. Evaluate $\phi := t(\phi)$. Iterate this assignment procedure at most r times or until a satisfying truth assignment t of ϕ is found, starting each time with a randomly chosen order of variables for the assignment procedure.

After $r = 2^{n(1-\frac{1}{k})}$ rounds a solution for a satisfiable formula is found with high probability [227]. By adding to ϕ clauses originating from resolving input clauses up to a certain length this bound can be improved. In case of $k = 3, 4$, and 5, the basis of the exponential growth function is 1.36406, $1, 49579$, and $1, 56943$ [228]. For $k \geq 4$ this is currently the best probabilistic algorithm for solving k-SAT.

The second approach is due to Schöning [244, 245] and extends an idea of Papadimitriou [226]. That procedure is outlined as follows:

> Repeat the following for r rounds: randomly choose an initial truth assignment t of ϕ; if t does not satisfy ϕ, then repeat the following for three times the number of variables of ϕ or until a satisfying assignment t is encountered: select a falsified clause c from ϕ and randomly choose and flip a literal x of c.

If the algorithm continues, round after round, without finding a satisfying assignment nothing definite can be said about the satisfiability of ϕ. However, in this case the higher the number of rounds r, the lower the probability that ϕ is satisfiable. To guarantee an error probability of $e^{-\lambda}$, the number of rounds should be at least $O(\lambda(2\frac{k-1}{k})^n)$. For $k = 3, 4$, and 5 the basis of the exponential growth is 1.3334, 1.5, and 1.6.

The case of $k = 3$ has since been improved to $O(1.324^n)$ by Iwama and Tamaki [148]. For CNF formulas of unrestricted clause length the best time bound for a deterministic solution algorithm is $O(2^{n(1-\frac{1}{log(2m)})})$, where n is the number of variables and m the number of clauses. This result, by Dantsin and Wolpert [79], is obtained from a derandomization of a randomized algorithm by Schuler [252], of the same time complexity. Dantsin and Wolpert recently improved this bound for randomized algorithms to $O\left(2^{n(1-\frac{1}{ln(\frac{m}{n})+O\ ln(ln(m))})}\right)$ [80].

1.18 Classes of easy expressions

An entire book could be written about the multitude of classes of the Satisfiability problem that can be solved in polynomial time so only some of the classical results will be mentioned here. It is often not enough that a class of problems is solved in polynomial time - an instance may have to be recognized as a member of that class before applying an efficient algorithm. Perhaps surprisingly, for some classes the recognition step is unnecessary and for some it is necessary but not known to be tractable.

All clauses of a 2-SAT expression contain at most two literals. A two literal clause describes two inferences. For example, inferences for the clause $(x \vee y)$ are: 1) if x is 0 then y is 1; and 2) if y is 0 then x is 1. For a given 2-SAT expression an implication graph may be constructed where directed edges between pairs of literals represent all the inferences of the expression. A cycle in the inference graph that includes a pair of complementary literals is proof that no model exists. Otherwise a model may be determined easily with a depth-first search of the graph, which amounts to applying unit propagation. A full algorithm is given in [90] (1976). A linear time algorithm is given in [22] (1979).

All clauses of an expression said to be *Horn* have at most one *positive literal*. This class is important because of its close association with Logic Programming: for example, the clause $(\neg v_1 \vee \neg v_2 \vee v)$ expresses the rule $v_1 \wedge v_2 \rightarrow v$ or $v_1 \rightarrow v_2 \rightarrow v$. However, the notion of causality is generally lost when translating from rules to Horn expressions. An extremely important property of Horn expressions is that every satisfiable one has a unique minimum model with respect to 1 (the unique minimum model is the intersection of all models of the expression). Finding a model is a matter of applying unit propagation on positive literals until all positive literals are eliminated, then assigning all remaining literals the value 0 (if satisfiable, a unique minimum model is the result). It took a few iterations in the literature to get this universally understood and the following are the important citations relating to this:[146]

(1982), [85] (1984), [254] (1990).

A given expression may be *renameable Horn*, meaning a change in the polarity of some variables results in an equivalent Horn expression. Renameable Horn expressions were shown to be recognized and solved in linear time by [186] (1978) and [23] (1980).

A number of polynomially solvable relaxations of Linear Programming problems were shown to be equivalent to classes of Satisfiability; this work, quite naturally, originated from the Operations Research community. Representing CNF expressions as $(0, \pm 1)$ matrices where columns are indexed on variables and rows indexed on clauses, Satisfiability may be cast as an Integer Programming problem. If the matrix has a particular structure the Integer Program can be relaxed to a Linear Program, solved, and the non-integer values of the solution rounded to 0 or 1. Notable classes based on particular matrix structures are the extended Horn expressions and what we call the CC-balanced expressions.

The class of *extended Horn* expressions was introduced by Chandru and Hooker [52] (1991). Their algorithm is due to a theorem of Chandrasekaran [51] (1984). Essentially, a model for a satisfiable expression may be found by applying unit propagation, setting values of unassigned variables to 1/2 when no unit clauses remain, and rounding the result by a matrix multiplication. This algorithm cannot, however, be reliably applied unless it is known that a given expression is extended Horn and, unfortunately, the problem of recognizing an expression as extended Horn is not known to be solved in polynomial time.

The class of *CC-balanced* expressions has been studied by several researchers (see [61] (1994) for a detailed account of balanced matrices and a description of CC-balanced formulas). The motivation for this class is the question, for Satisfiability, when do Linear Programming relaxations have integer solutions? The satisfiability of a CNF expression can be determined in linear time if it is known to be CC-balanced and recognizing that a formula is CC-balanced takes linear time.

Horn, Renameable Horn, Extended Horn, CC-balanced expressions, and other classes including that of [259] (1991) turn out to be subsumed by a larger, efficiently solved class called *SLUR* for Single Lookahead Unit Resolution [251] (1995). The SLUR class is peculiar in that it is defined based on an algorithm rather than on properties of expressions. The SLUR algorithm recursively selects variables sequentially and arbitrarily, and considers a one-level lookahead, under unit propagation in both directions, choosing only one, if possible. If a model is found, the algorithm is successful, otherwise it "gives up." An expression is SLUR if, for all possible sequences of variable selections, the SLUR algorithm does not give up. Observe that due to the definition of this class, the question of class recognition is avoided. In fact, SLUR provides a way to avoid preprocessing or recognition testing for several polynomial time solvable classes of SAT when using a reasonable variant of the DPLL algorithm.

The worst case time complexity of the SLUR algorithm would appear to be quadratic. However, a simple modification brings the complexity down to linear: run both calls of unit propagation simultaneously, alternating execution of their repeat blocks. When one terminates without an empty clause in its output formula, abandon the other call.

The *q-Horn* class also originated in the Operations Research community [37, 38] (1990) and for several years was thought to be what was described as the largest, succinctly expressed class of polynomial time solvable expressions. This claim was due to a measure on an underlying $(0, \pm 1)$ matrix representation of clauses called the *satisfiability index* [39] (1994). The q-Horn class was also studied as a special case of the maximum monotone decomposition of matrices [265] (1994). We find it easier to describe the efficient solution of q-Horn expressions by following [266] (1998). Using the $(0, \pm 1)$ matrix representation, an expression is q-Horn if columns can be multiplied by -1, and permuted, and rows can be permuted with the result that the matrix has four quadrants as follows: northeast - all 0s; northwest - a Horn expression; southeast - a 2-SAT expression; southwest - no +1s. A model may be found in linear time, if one exists, by finding a model for the northwest quadrant Horn expression, cancelling rows in the southern quadrants whose clauses are satisfied by that model, and finding a model for the southeast quadrant 2-SAT expression. It was shown in [39] (1994) that a CNF expression is q-Horn if and only if its satisfiability index is no greater than 1 and the class of all expressions with a satisfiability index greater than $1 + 1/n^{\epsilon}$, for any fixed $\epsilon < 1$, is $\mathcal{NP}$-complete. The SLUR and q-Horn classes are incomparable [100] (2003).

The class of *linear autarkies* was developed by van Maaren [193] (2000) and shown to include the class of q-Horn formulas. It was also shown to be incomparable with the SLUR class. An autarky for a CNF formula ϕ is a partial assignment that satisfies all those clauses of ϕ affected by it: for example, a pure literal is an autarky. Therefore, a subformula obtained by applying an autarky to ϕ is satisfiable if and only if ϕ is. A formula with $(0, \pm 1)$ matrix representation A has a linear autarky $x \in Q^n$, $x \neq 0$, if $Ax \geq 0$. In [178] it was shown that a linear autarky can be found in polynomial time. There exists a simple, efficient decomposition that results in a partial, autark assignment. Applying this decomposition repeated results in a unique, linear-autarky-free formula. If the decomposition is repeatedly applied to a renameable Horn formula without unit clauses what is left is empty and if it is applied repeatedly to a 2-SAT formula, the formula is unsatisfiable if what is left is not empty and satisfiable otherwise.

The class of *matched expressions* was analyzed to provide a benchmark for testing the claim made for the q-Horn class. This is a class first described in [264] (1984) but not extensively studied, probably because it seems to be a rather useless and small class of formulas. Establish a bipartite graph $G_\phi = (V_1, V_2, E)$ for an expression ϕ where V_1 vertices represent clauses, V_2 vertices represent variables, and edge $\langle c, v\rangle \in E$ if and only if clause c contains v or its complement. If there is a total matching in G_ϕ, then ϕ is said to be matched. Clearly, matched expression are always satisfiable and are trivially solved. The matched class is incomparable with the q-Horn and SLUR classes. However, as is shown in Section 1.21, with respect to frequency of occurrence on random expressions, matched expressions are far more common than both those classes together [100] (2003).

The worst case complexity of *nested satisfiability*, a class inspired by Lichtenstein's theorem of planar satisfiability [188] (1982), has been studied in [176] (1990). Index all variables in an expression consecutively from 1 to n and let positive and negative literals take the index of their respective variables. A clause c_i is said to *straddle* another clause c_j if the index of a literal of c_j is strictly between two indices of literals of c_i. Two clauses are said to *overlap* if they straddle each other. A formula is said to be *nested* if no two clauses overlap. For example, the following formula is nested

$$(v_6 \vee \bar{v}_7 \vee v_8) \wedge (v_2 \vee v_4) \wedge (\bar{v}_6 \vee \bar{v}_9) \wedge (v_1 \vee \bar{v}_5 \vee v_{10}).$$

The class of Nested formulas is quite limited in size for at least the reason that a nested expression can contain no more than $2m + n$ literals. Thus, no expression consisting of k-literal clauses is a nested formula unless $m/n < 1/(k-2)$. The class of nested expressions is incomparable with both the SLUR and q-Horn classes. However, by the measure of Section 1.21 (also in [100] (2003)) a random expression is far more likely to be matched, q-Horn, or even SLUR than nested. The algorithm for nested expressions is notable for being quite different than those mentioned above: instead of relying on unit propagation, it uses dynamic programming to find a model in linear time. The question of whether the variable indices of a given formula can, in linear time, be permuted to make the formula nested appears to be open. An extension to nested satisfiability, also solvable in linear time, has been proposed in [128] (1993).

None of the classes above covers a significant proportion of unsatisfiable expressions. Nevertheless, several classes of unsatisfiable expressions have been identified. It is interesting that most known polynomial time solvable classes with clauses containing three or more literals either are strongly biased toward satisfiable expressions or strongly biased toward unsatisfiable expressions.

An expression is said to be *minimally unsatisfiable* if it is unsatisfiable and removing any clause results in a satisfiable expression. A minimally unsatisfiable expression with n variables must have at least $n + 1$ clauses [11] (1986). Every variable of a minimally unsatisfiable expression occurs positively *and* negatively in the expression. The class of minimally unsatisfiable formulas is solved in $n^{O(k)}$ if the number of clauses exceeds the number of variables by a fixed positive constant k [171] (1999) and [172] (2000). Szeider improved this to $O(2^k)n^4$ [260] (2003). Kullmann has generalized this class in [179] (2003) and continues to find larger versions. Some SAT solvers look for minimally unsatisfiable sets of clauses to reduce search space size.

A CNF expression is said to be k-BRLR if all resolvents derived from it have a number of literals bounded by k or if the null clause is deriveable from resolvents having at most k literals. Obviously, this class is solved in time bounded by $2^k\binom{n}{k}$.

1.19 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) were considered for a while to be the best way to handle some problems that are rooted in real applications, particularly related to circuit design, testing, and verification. They are still quite useful in various roles [77, 101, 144, 151, 214, 219, 60] and in some ways are complementary to search [269, 270].

A BDD may be regarded as a DAG representation of the truth table of a Boolean function. In a BDD non-leaf vertices are labeled as variables, there are two out-directed edges for each non-leaf vertex, each labeled with value 0 or 1, and two leaves, labeled 1 and 0. There is a single root and any path from root to a "1" ("0") leaf indicates an assignment of values to the variables which causes the represented function to have value 1 (0). A BDD may also be viewed as representing a search space in which paths to the 1 leaf represent models.

The attraction to BDDs is due in part to the fact that no subproblem is represented more than once in a collection of BDDs - this is in contrast to the tree-like search spaces of DPLL implementations of the 1980s. In addition, efficient implementations exist for BDD operations such as existential quantification, "or," "and," and others. The down side is: 1) each path from the root of a BDD must obey the same variable ordering, so BDDs are not necessarily a minimal representation for a function; and 2) repeatedly conjoining pairs of BDDs may create outragiously large intermediate BDDs even though the final BDD is small. By the late 1990s, DPLL advances such as conflict resolution (clausal learning), backjumping, restarts, and more gave rise to DAG search spaces with dynamic variable ordering. The result was improved performance for search over BDDs in some cases.

BDDs were introduced in [185] (1959) as a data structure based on the Shannon expansion (see Section 1.14). They were publicized in [40] (1976) and [12] (1978). But BDDs became most useful with the introduction of reduced order BDDs by Bryant [45, 46] (1986,1992) and their implementation [41] (1990) which supports subproblem sharing over collections of BDDs and the efficient manipulation of those BDDs.

A number of operations on BDDs have been proposed over the years to assist in the efficiency of combining BDDs. One of the most important and basic operations is `existential quantification` which arises directly from Shannon's expansion of Section 1.14: namely, replace f with $f|_{v=0} \vee f|_{v=1}$. This operation can be used to eliminate a variable at the expense of first conjoining all BDDs containing it. It is not clear when existential quantification was first used but it was probably known very early. An important problem in managing BDD size is how to simplify a function f (implemented as a BDD), given a constaint c (implemented as a BDD): that is, replace f with a function f' that is equal to f on the domain defined by c. The operation of `restrict` [66] (1990) does this by pruning paths from the BDD of f that are 'false' in the BDD of c. A more complex operation with restrict-like properties but admiting removal of a BDD from a collection of BDDs at the expense of increasing the size of some of the remaining BDDs is the `constrain` or `generalized cofactor` operation [67] (1990). Constraining f to c results in a function h with the property that $h(x) = f(\mu(x))$ where $\mu(x)$ is the closest point to x in c where distance between binary vectors $x, y \in \{0,1\}^n$ is measured by $d(x,y) = \sum_{1 \le i \le n} 2^{n-i} \cdot ((x_i + y_i) \bmod 2)$ under a BDD variable ordering which matches the index order. Additional minimization operations, based on `restrict`, that do not increase BDD size are proposed in [138] (1997).

BDDs were designed primarily for the efficient representation of switching circuits and their manipulation, but a number of variations on BDDs have appeared over the years to target related classes of problems. BDDs of various kinds have been used successfully to represent relations and formulas to support symbolic model checking [49, 201] (1992), although more recently, it has been found that SAT solvers (see Section 1.28) for Bounded Model Checking [33] (1999) can sometimes achieve even better results. The ZDD, for Zero-suppressed BDD, introduced in 1993 [205], differs from the BDD in that a vertex is removed if its 1 edge points to the 0 leaf. This helps improve efficiency when handling sparse sets and representing covers. Thus, the ZDD has been used successfully on problems in logic synthesis such as representing an irredundant DNF of an incompletely specified Boolean function [69] (1993), finding all essential prime implicants in DNF minimization [68] (1994), factorization [206] (1996), and decomposition [149] (2001). The BMD, for Binary Moment Diagram, is a generalization of the BDD for handling numbers and numeric computations, particularly multiplication [47] (1995). The ADD, for Algebraic Decision Diagram (also known as Multi-Terminal Decision Diagram), allows more than two leaves (possibly real-valued) and is useful for mapping Boolean functions to sets [24] (1993) as might be needed to model the delay of MOS circuits, for example [200] (2001). The XDD has been used to efficiently represent equations involving the xor operation. The essential idea is that vertex x with 1 edge to vertex a and 0 edge to vertex b represents the equation $(x \wedge a) \oplus b = 1$. Many other varieties of BDD have been proposed but there is not enough space to mention them all.

1.20 Probabilistic analysis: SAT algorithms

Probabilistic and average-case analysis of algorithms can give useful insight into the question of what SAT algorithms might be effective and why. Under certain circumstances, one or more structural properties shared by each of a collection of formulas may be exploited to solve such formulas efficiently; or structural properties might force a class of algorithms to require superpolynomial time. Such properties may be identified and then, using probabilistic analysis, one may argue that these properties are so common that the performance of an algorithm or class of algorithms can be predicted for most of a family of formulas. The first probabilistic results for SAT had this aim.

Probabilistic results of this sort depend on an underlying input distribution. Although many have been proposed, well-developed histories exist for just two, plus a few variants, which are defined over CNF inputs. All the results of this section are based on these two. In what follows, the *width* of a clause is the number of literals it contains. The distributions we discuss here are:

1. *Variable width distribution*: Given integers n, m, and a function $p : N^+ \times N^+ \rightarrow [0,1]$. Let $V = \{v_1, ..., v_n\}$ be a set of n Boolean variables. Construct a random clause (disjunction) c by independently adding literals to c as follows: for each variable v_i, $1 \leq i \leq n$, add literal v_i with probability $p(m,n)$, add literal $\neg v_i$ with probability $p(m,n)$ (therefore add neither v_i nor $\neg v_i$ with probability $1 - 2p(m,n)$). A random input consists of m independently constructed random clauses and is referred to below as a random (n,m,p)-CNF expression. In some cases $p(m,n)$ is independent of m and n and then we use p to represent $p(m,n)$.

2. *Constant width distribution*: Given integers n, m, and k. Let $V = \{v_1, ..., v_n\}$ be a set of n Boolean variables. Construct a random clause (disjunction) c by choosing, uniformly and without replacement, k variables from V and then complementing each, independently, with probability 1/2. A random input consists of m independently constructed random clauses and is referred to below as a random (n,m,k)-CNF expression. Such an input is widely known as a uniform random k-SAT instance or simply random k-SAT instance.

Variable width distribution

Goldberg was among the first to apply probability to the analysis of SAT algorithms. He investigated the frequency with which simple backtracking returns a model quickly on random CNF formulas by providing an average-case analysis of a variant of DPLL which does not handle pure literals or unit clauses [115] (1979). The result received a lot of attention when it was first presented and even 10 years afterward some people still believed it was the definitive probabilistic result on satisfiability algorithms.

Goldberg showed that, for random (n, m, p)-CNF expressions, the DPLL variant has average complexity bounded from above by $O(m^{-1/\log(p)}n)$ for any fixed $0 < p < 1$. This includes the "unbiased" sample space when $p = 1/3$ and all expressions are equally likely. Later work [116] (1982) showed the same average-case complexity even if pure literals are handled as well. Very many problems confronting the scientific and engineering communities are unsatisfiable, but Goldberg made no mention of the frequency of occurrence of unsatisfiable random (n, m, p)-CNF expressions.

However, Franco and Paull [99] (1983) pointed out that large sets of random (n, m, p)-CNF expressions, for fixed $0 < p < 1/2$, are dominated by trivial *satisfiable* expressions: that is, any random assignment of values to the variables of such a random expression is a model for that expression with high probability. This result is refined somewhat in [95] (1986) where it is shown that a random assignment is a model for a random (n, m, p)-CNF expression with high probability if $p > \ln(m)/n$ and a random expression is unsatisfiable with high probability if $p < \ln(m)/2n$. In the latter case, a "proof" of unsatisfiability is trivially found with high probability because a random (n, m, k)-CNF expression for this range of p usually contains at least one empty clause, which can easily be located, and implies unsatisfiability. The case $p = c\ln(m)/n$, $1/2 \leq c \leq 1$ was considered in [98] (1988) where it was shown that a random (n, m, p)-CNF expression is satisfiable with high probability if $\lim_{n,m\to\infty} m^{1-c}/n^{1-\epsilon} < \infty$, for any $0 < \epsilon < 1$.

Although these results might be regarded as early threshold results (see Section 1.21) the main impact was to demonstrate that probabilistic analysis can be highly misleading and requires, among other things, some analysis of input distribution to ensure that a significant percentage of non-trivial inputs are generated. They show that random (n, m, p)-CNF expressions, satisfiable or unsatisfiable, are usually trivially solved because either they contain empty clauses (we get the same result even if empty or unit clauses are disallowed) or they can be satisfied by a random assignment. In other words, only a small region of the parameter space is capable of supporting significantly many non-trivial expressions: namely, when the average clause width is $c\ln(m)/n$, $1/2 \leq c \leq 1$. These results demonstrate shortcomings in choosing random (n, m, p)-CNF expressions for analysis and, because of this, such generators are no longer considered interesting by many.

Nevertheless, some interesting insights were developed by further analysis and we mention the most significant ones here. The results are shown graphically in Figure 1.1 which partitions the entire parameter space of the variable width distribution according to polynomial-average-time solvability. The vertical axis $(p \cdot n)$ measures average clause width and the horizontal axis (m/n) measures density. Each result is presented as a line through the chart with a perpendicular arrow. Each line is a boundary for the algorithm labeling the line and the arrow indicates that the algorithm has polynomial average time performance in that region of the parameter space that is on the arrow side of the line (constant and even log factors are ignored for simplicity).

Goldberg's result is shown as the diagonal line in the upper right corner of the figure and is labeled **Goldberg**: it is not even showable as a region of the parameter space, so there is no arrow there. Iwama analyzed an algorithm which counts models using inclusion-exclusion [147] (1989) and has polytime-average-time complexity in the region above and to the right of the line labeled **Counting**. A random expression generated at a point in that region satisifes conditions, with high probability, under which the number of terms in the inclusion-exclusion expansion is polynomial in m and n. However, every clause of the same random expression is a tautology (contains a literal and its complement) with high probability. Therefore, this seems to be another example of a misleading result and, judging from the relation between the **Counting** and **Goldberg** regions in the figure, lessens the significance of Goldberg's result even more.

There have been a number of schemes proposed for limiting resolution steps to obtain polynomial complexity. A simple example is to perform resolution only if the pivot variable appears once as a positive literal and once as a negative literal - then, if an empty clause does not exist when no more resolutions are possible, do a full search for a solution. The conditions under which this algorithm runs in polynomial average time under random variable width expressions are too complex to be given here but they are represented by the regions in Figure 1.1 below the lines labeled **Limited Resolution** [96] (1991).

If pure literal reductions are added to the algorithm analyzed by Goldberg then polynomial average time is achieved in the **Pure Literals** region, considerably improving the result of Goldberg [232] (1985). But a better result, shown as the region bounded from above by the lines labeled **Unit Clauses**, is obtained by performing unit propagation [97] (1993) and is consistent with the empirically and theoretically demonstrated importance of unit propagation in DPLL algorithms. Results for Search Rearrangement Backtracking [231] (1983) are disappointing (shown bounded by the two lines labeled **Backtracking**) but a slightly different variant in which only variables from positive clauses (a *positive clause* contains only positive literals) are chosen for elimination (if there is no positive clause, satisfiability is determined by assigning all unassigned variables the value 0), has spectacular performance as is shown by the line labeled **Probe Order Backtracking** [230] (1997). The combination of probe order backtracking and unit clauses yield polynomial average time for random (n, m, p)-CNF expressions at every point of the parameter space except for some points along the thin horizontal region $p \cdot n = c \cdot \log(n)$.

By 1997, what Goldberg had set out to do in 1979, namely show DPLL has polynomial average time complexity for variable width distributions, had been accomplished. The research in this area had some notable successes such as the analysis of probe order backtracking which demonstrated that a slight

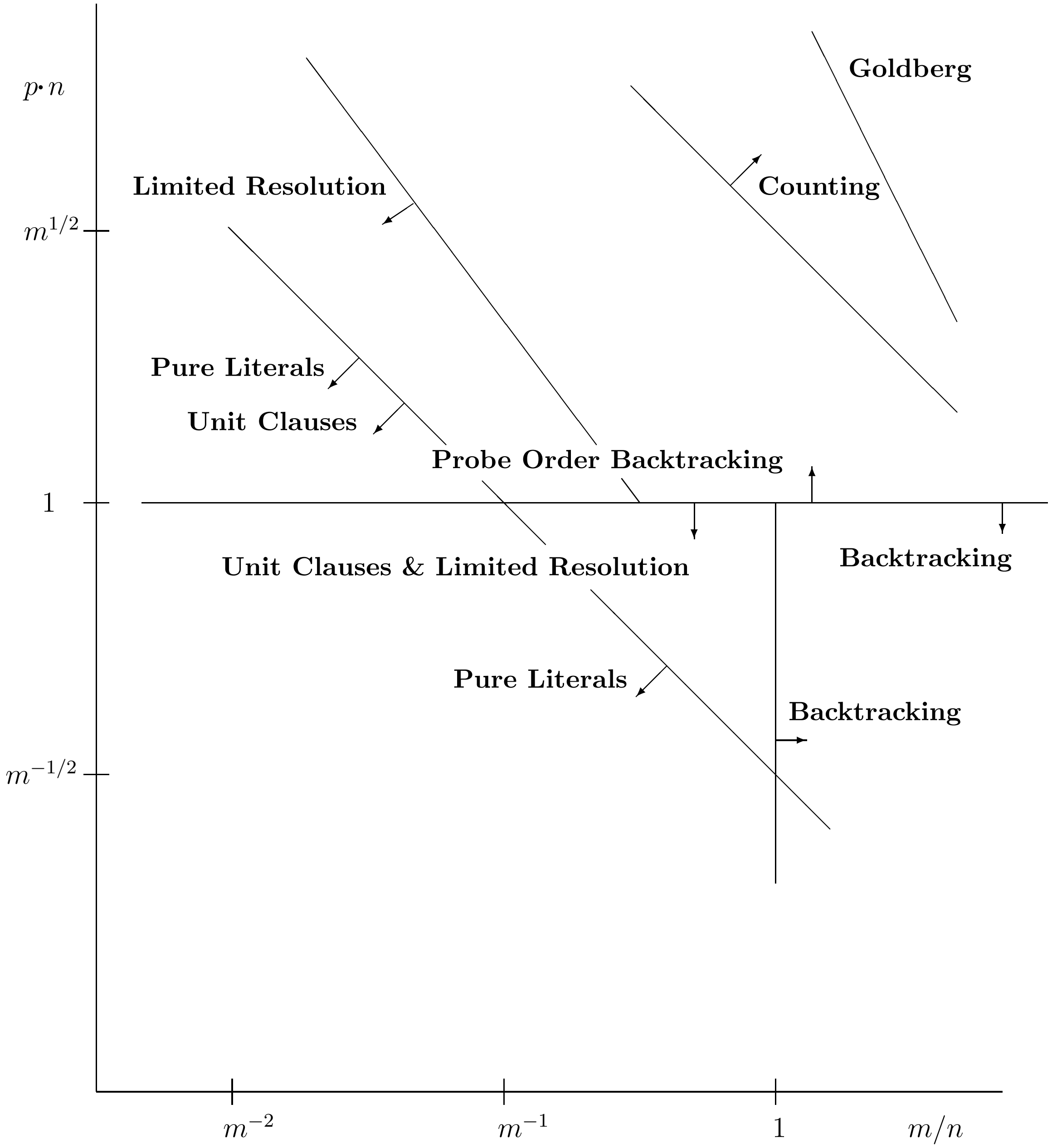

Figure 1.1: The parameter space of the variable width distribution partitioned by polynomial-average-time solvability. Pick a point in the parameter space. Locate the lines with names of algorithms on the side of the line facing the chosen point. Random formulas generated with parameters set at that point are solved in polynomial average time by the named algorithms.

algorithmic change can have a major effect on perfomance. But, the fact that nearly every point in the parameter space is covered by an algorithm that has polynomial average time complexity is rather disturbing, though, since there are many practical applications where SAT is considered extremely difficult. Largely for this reason attention shifted to the constant width distribution and, by the middle of the 1990s, research on the variable width distribution all but disappeared.

Constant width distribution

Franco and Paull (1983) in [99] (see [225], 1987, for corrections) set out to test Goldberg's result on other distributions. They formulated the constant width distribution and showed that for all $k \geq 3$ and every fixed $m/n > 0$, with probability $1 - o(1)$, the algorithm analyzed by Goldberg takes an *exponential* number of steps to report a result: that is, either to report all ("cylinders" of) models, or that no model exists. They also showed that a random (n, m, k)-CNF expression is unsatisfiable with probability tending to 1 when $m/n > -\ln(2)/\ln(1 - 2^{-k})$. For $k = 3$ this is $m/n > 5.19$.

Perhaps the first positive result using the constant width distribution was presented in [53] (1986) where a non-backtracking, incremental assignment algorithm employing unit propagation, called UC, was shown to find a model with bounded probability when $m/n < O(1) \cdot 2^k/k$. This work introduced an analysis tool called "clause-flows" which was refined by Achlioptas [1] (2001) using a theorem of Wormald [278]. Analysis of clause flows via differential equations was to be the basic analytic tool for most of the following results in this area. In [54] (1990) it was shown that a generalization of unit propagation, called GUC, in which a variable from a "smallest" clause is chosen for assignment finds a model with probability $1 - o(1)$ when $m/n < O(1) \cdot 2^k/k$, $k \geq 4$. This was improved by Chvátal and Reed [58] (1992) who showed that GUC, with assignment rules relaxed if no unit or two literal clauses are present (called SC for shortest clause), finds a model with probability $1 - o(1)$ when $m/n < O(1) \cdot 2^k/k$ for $k \geq 3$.

Observe that the results of the previous two paragraphs show a gap between the range $m/n > \ln(2)/\ln(1 - 2^{-k}) \approx 2^k \ln(2)$, where random expressions are unsatisfiable with high probability, and $m/n < O(1) \cdot 2^k/k$, where analyzed non-backtracking incremental assignment algorithms find models for random expressions with high probability. According to results of Friedgut (1999 - see Section 1.21) there is a sharp satisfiability threshold r_k for every $k \geq 2$ (it is still not known whether the threshold depends on n) and the results above show $2^k/k < r_k < 2^k \ln(2)$. The question of the location of r_k was pondered for quite some time. Most people intuitively believed that r_k was located near $2^k \ln(2)$ so the question became known as the '*Why* 2 *k*?' problem (Pittel, 1999). This was verified by Achlioptas, Moore, and Perez in [4, 5] (2002,2004) who showed that $r_k = 2^k \ln(2) - O(k)$ by applying the second moment method to a symmetric variant of satisfiability, known as not-all-equal k-SAT (NAE-k-SAT). In NAE-k-SAT the question is whether, for a given CNF expression, there is a satisfying assignment whose complement is also a satisfying assignment. Obviously, the class of k-CNF expressions satisfiable in NAE-k-SAT is a subset of the class satisfiable in the traditional sense. Hence, for the purposes of finding the threshold, NAE-k-SAT may analyzed instead of k-SAT. The symmetric nature of NAE-k-SAT results in a low variance in the number of satisfying assignments and this makes the second moment method work.

Most interest, though, has centered on the case $k = 3$. It is believed, from experiment, that $r_3 \approx 4.25$. In [53] (1986) it is shown that UC finds a model with bounded probability when $m/n < 8/3 = 2.66..$ and, when combined with a "majority" rule (that is, choose a variable with the maximum difference between the number of its positive and negative literals if unit propagation cannot be applied), this improves to $m/n < 2.9$. Frieze and Suen [103] (1996) considered SC and GUC and showed that for $m/n < 3.003..$, both heuristics succeed with positive probability. Moreover, they proved that a modified version of GUC, called GUCB, which uses a limited form of backtracking, succeeds almost surely for the same range of m/n. Later, Achlioptas [2] (2000) analyzed SC, modified slightly to choose two variables at a time from a clause with two unfalsified literals, for an improvement to $m/n \leq 3.145$ with probability $1 - o(1)$.

Nearly all the results above apply to *myopic* algorithms. A non-backtracking, variable assignment algorithm is called *myopic* [3] if, under the spectral coalescence of states, the distribution of expressions corresponding to a particular coalesced state can be expressed by its spectral components alone: that is, by the *number* of clauses of width i, for all $1 \leq i \leq k$, and the *number* of assigned variables. Being myopic considerably assists an analysis by preserving statistical independence from one algorithmic step to the next. Unfortunately, in [3] (2000) it is shown that no myopic algorithm can have good probabilistic performance when $m/n > 3.26$. In the same paper it is shown that at least one myopic algorithm almost surely finds a model if $m/n < 3.26$.

However, Kaporis et al. [155] (2002) found a workaround to this "barrier" and analyzed a simple non-myopic greedy algorithm for Satisfiability. They control the statistical dependence problem by considering a different generator of expressions such that probabilistic results also hold for random $(n, m, 3)$-CNF expressions. Their result is that a greedy algorithm for satisfiability almost always finds a model when $m/n < 3.42$. Similar algorithms have been shown to find a model, almost always, when $m/n < 3.52$ [123, 156] (2003). With this analysis machinery in place, better results are expected.

Although most of the results of the previous paragraph were motivated by threshold research (see Section 1.21), their analyses can influence algorithm development. It is hoped future probabilistic results will reveal new generally useful search heuristics, or at least explain the mechanics of existing search heuristics.

Another analysis should be noted for the following non-myopic incremental assignment algorithm: repeatedly assign values to pure literals to eliminate the clauses containing them. Broder, Frieze and Upfal [42] (1993) proved that this algorithm almost always finds a model when $m/n \leq 1.63$ and $k = 3$. They used Martingales to offset distribution dependency problems as pure literals are uncovered. Mitzenmacher [209] (1997) showed that this bound is tight.

This section concludes with a history of results on algorithms which are intended to return certificates of unsatisfiability. Most of these results, all pessimistic, are obtained for resolution. The best result, though, is obtained for an algorithm which finds lower bounds on the number of variable assignments that must be positive and the number that must be negative (via translation to two hitting set problems) and returns "unsatisfiable" if the sum is greater than the number of variables. This result is notable also because it makes use of a highly underused analysis tool in this area: eigenvalues. Recall that nearly all random constant width expressions are unsatisfiable if $m/n > 2^k \ln(2)$.

Chvátal and Szemerédi obtained the earliest result on constant width distributions, inspired by the work reported in Section 1.16, is that, with high probability, every resolution proof for a random unsatisfiable expression is exponentially long if $\lim_{n,m\to\infty} m/n = f(k) > 2^k \ln(2)$ [59] (1988). The analysis shows the root cause to be a property known as sparseness; which roughly indicates the number of times pairs of clauses have a common literal or complementary pair of literals. When an expression is sparse any "moderately large" subset of its clauses must contain a "large" number of variables that occur exactly once in that subset. Sparsity forces the proof to have a large resolvent. But, almost all "short" resolution refutations contain no long clauses after eliminating all clauses satisfied by a particular small random partial assignment ρ. Moreover, resolution refutations for almost all unsatisfiable random (n, m, k)-CNF expressions with clauses satisfied by ρ removed are sparse and, therefore, must have at least one high width resolvent. Consequently, almost all unsatisfiable random expressions have long resolution refutations. Beame, Karp, Pitassi, Saks, Ben-Sasson, and Widgerson, among others, contributed ideas leading up to a concise understanding of this phenomenon in [28] (1996), [29] (1998), [31] (2001).

Despite considerable tweaking, the best we can say right now is that, with probability tending to 0, the width of a shortest resolution proof is bounded by a polynomial in n when $m/n > 2^k \ln(2)$ and $\lim_{m,n\to\infty} m/n^{(k+2)/(4-\epsilon)} < 1$, where ϵ is some small constant; and with probability tending to 1, the width is polynomially bounded when $\lim_{m,n\to\infty} m/n > (n/\log(n))^{k-2}$.

The failure of resolution has led to the development of a new technique by Goerdt for certifying unsatisfiability, mentioned above, that uses eigenvalues. In [114] this spectral technique is used to obtain bounds sufficient to show that certifying unsatisfiability in polynomial time can be accomplished with high probability when $\lim_{m,n\to\infty} m/n > n^{k/2-1+o(1)}$ which is considerably better than resolution.

1.21 Probabilistic analysis: thresholds

Results of the 1980s (see Section 1.20) showed that, for random width k expressions density, that is the ratio m/n, is correlated with certain interesting expression properties. For example, it was shown that if $m/n > 2^k \ln(2)$ then a random instance is unsatisfiable with high probability and if $m/n < O(1)\cdot 2^k/k$ a random expression is satisfiable with high probability. So, it was clear that a crossover from probably unsatisfiable to probably satisfiable occurs as density is changed. Early on it was speculated that "hardness" is directly related to the nature of crossover and this motivated the study of Satisfiability thresholds.

Actually, thresholds can apply to a multitude of properties, not just the property of a formula being satisfiable, so we give a general definition of threshold here. Let $X = \{x_1, ..., x_e\}$ be a set of e elements. Let A_X, a subset of the power set of X (denoted 2^X), be called a *property*. Call A_X a *monotone property* if for any $s \in A_X$, if $s \subset s'$, then $s' \in A_X$. Typically, a monotone property follows from a high-level description which applies to an infinite family of sets X. For example, let $X = \mathcal{C}_{k,n}$ be the set of all non-tautological, width k clauses that can be constructed from n variables. Thus $e = 2^k\binom{n}{k}$ and any $s \in 2^{\mathcal{C}_{k,n}}$ is a k-CNF expression. Let $\mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ denote the property that a k-CNF expression constructed from n variables is unsatisfiable. That is, any $s \in \mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ has no model and any $s \in 2^{\mathcal{C}_{k,n}} \setminus \mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ has a model. If $s \in \mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ and $c \in \mathcal{C}_{k,n}$ such that $c \notin s$, then $s \cup \{c\} \in \mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ so the property $\mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ is monotone for $k < n$.

For any $0 \le p \le 1$ and any monotone property $A_X \subset 2^X$ define

$$\mu_p(A_X) = \sum_{s\in A_X} p^{|s|}(1-p)^{e-|s|}$$

to be the probability that a random set has the monotone property. For the property $\mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ (among others), s is a set of clauses, hence this probability measure does not match that for what we call random k-CNF expressions but comes very close with $p = m/(2^k\binom{n}{k}) \approx m\cdot k!/(2n)^k$.

Observe that $\mu_p(A_X)$ is an increasing function of p.[5] Let $p_c(X)$ denote that value of p for which $\mu_p(A_X) = c$. The values of $p_c(X)$ change as the size of X increases. There are two threshold types.

[5] By illustration using $\mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ this reflects the fact that, as p increases, the average number of clauses increases, so the probability that an expression has the $\mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ property increases.

A_X is said to have a *sharp threshold* if, for any small, positive ϵ,

$$\lim_{|X|\to\infty}(p_{1-\epsilon}(X) - p_\epsilon(X))/p_{1/2}(X) = 0.$$

A_X is said to have a *coarse threshold* if, for any small, positive ϵ,

$$\lim_{|X|\to\infty}(p_{1-\epsilon}(X) - p_\epsilon(X))/p_{1/2}(X) > 0.$$

From experiments based on the constant width distribution it appeared that $\mathcal{UNSAT}_{\mathcal{C}_{k,n}}$ has a sharp threshold and that the density at which the crossover occurs is a function of k. The so-called *satisfiability threshold* was therefore denoted by r_k. In addition, as m/n is reduced significantly below (the experimentally determined) r_k, algorithms of various types were observed to perform better and expressions were more likely to be members of a polynomial time solvable class (see Section 1.18 for a description - an analysis of these classes is given later in this section), and as m/n is increased significantly above r_k various algorithms were observed to perform better. But, in the neighborhood of r_k random expressions seemed to reach a point of maximum difficulty, at least for well-known, well-tried algorithms. Hence, the study of satisfiability thresholds was motivated by a possible connection between the nature of hardness and the nature and location of thresholds. The constant width distribution has dominated this study, perhaps because expressions generated when parameters are set near r_k tend to be extremely difficult for known algorithms.

The conjecture of a sharp threshold can be traced to a paper by Mitchell, Selman, and Levesque [208] (1992) where the “easy-hard-easy” phenomenon is pointed out. A possible explanation is given by Cheeseman in [55] (1991) where long “backbones” or chains of inference are stated to be a likely cause as the high density of well-separated “near-solutions” induced by backbones leads to thrashing in search algorithms.

The satisfiability threshold for random $(n, m, 2)$-CNF expressions was found by Chvátal and Reed to be sharp with $r_2 = 1$ [58] (1992) (it is historically correct to note that de la Vega [92] and Goerdt [113] independently achieved the same result in the same year). The fact that $(n, m, 2)$-CNF expressions can be solved in polynomial time [62] means that there is a *simple* characterization for those instances which are unsatisfiable. Both [58] and [113] make use of this characterization by focusing on the emergence of the “most likely” unsatisfiable random $(n, m, 2)$-CNF expressions.

For $k > 2$ the situation was much more difficult. The results of Section 1.20, some of which were bounded probability results, can be regarded as a history of lower bounds for the satisfiability threshold. Upper bounds were improved at a consistent rate for a while, the most intense investigation focusing on the case $k = 3$. Results due to Frieze and Suen [103] (1996), Kamath, Motwani, Palem, Spirakis [154] (1995), de la Vega [93] (1997), Dubois, Boufkhad [86] (1997), Kirousis, Kranakis, Krizanc [162] (1996), Kirousis, Kranakis, Krizanc, Stamatiou [163] (1998), Dubois [87] (2001), and Dubois, Boufkhad, Mandler [88] (2002), containing many beautiful ideas, have brought the upper bound down to just over 4.5 for $k = 3$. From Section 1.20 the lower bound for $k = 3$ is 3.52. From experiments, we expect $r_3 \approx 4.25$.

Friedgut [102] (1999) proved that sharp satisfiability thresholds exist for random (n, m, k)-CNF expressions, thereby confirming the conjecture of [208]. This result immediately lifted all previously known and future constant probability bounds, to almost surely bounds. Friedgut's result has left open the possibility that the satisfiability threshold is a function of both k and n and it is still not known whether the satisfiability threshold depends on n, as weak as that dependency must be.

Monasson [210, 211] (1999) and others conjectured that there is a strong connection between the "order" of threshold sharpness, that is whether the transition is smooth or discontinuous, and hardness. Consistent with this, using the characterization of unsatisfiable $(n, m, 2)$-CNF expressions mentioned above, Bollobas et al. [35] (2001) completely determined the "scaling window" for random $(n, m, 2)$-CNF expressions, showing that the transition from satisfiability to unsatisfiability occurs for $m = n + \lambda n^{2/3}$ as λ goes from $-\infty$ to $+\infty$. For some time scaling windows for various problems were consistent with Monasson's conjecture (*e.g.* [71, 72]). But eventually the conjecture was disproved in [10] (2001) where it was found that the order of threshold sharpness for the 1-in-k SAT problem, which is $\mathcal{NP}$-complete, is the same as that of 2-SAT.

On the other hand, the work of Creignou and Daudé [70] (2002), [73] (2004) revealed the importance of *minimal monotonic structures* to the existence of sharp transitions. An element $s \in A_X$ is said to be *minimal* if for all $s' \subset s$, $s' \in 2^X \setminus A_X$. Those results have been used, for example, to show the limitations of most succinctly defined polynomial-time solvable classes of expressions, such as those mentioned in Section 1.18. Using density m/n of (n, m, k)-CNF distributions as a measure, thresholds for some classes of Section 1.18 have been determined: for example, a random (n, m, k)-CNF expression, $k \geq 3$, is q-Horn with probability tending to 1 if $m/n < O(1)/(k^2 - k)$ and with probability tending to 0 if $m/n > O(1)/(k^2 - k)$ [100, 74]. Except for the matched class, the monotonic structure analysis shows why the classes of Section 1.18, including q-Horn, SLUR, renameable Horn and many others, are weak: they are "vulnerable" to cyclic clause structures, the presence of any of these in an expression prevents it from having the polynomial-time solveable property. A random expression is matched with probability tending to 1 if $m/n < 0.64$ [100]: a result that adds perspective to the scope of most polynomial-time solveable classes.

Also of historical importance are results on $2 + p$ mixed random expressions: random expressions with width 2 and width 3 clauses, the p being the fraction of width 3 clauses. This distribution was introduced in [165] (1994) to help understand where random expressions get hard during search: if a search algorithm that embodies unit propagation is presented with a $(n, m, 3)$-CNF expression, every search node represents such a mixed expression with a particular value of p so hardness for some range of p could translate to hardness for search. Experimental evidence suggested that for some p_c, figured to be around 0.417, if $p < p_c$ the width 3 clauses were effectively irrelevant to the satisfiability of a random expression but if $p > p_c$ the transition behavior was more like that of a random width 3 expression. In [7] (2001) it was shown that $0.4 < p_c < 0.696$ and conjectured that $p_c = 0.4$. In [9] (2004) it was shown that a random $2+p$ mixed expression has a minimum exponential size resolution refutation (that includes any DPLL algorithm) with probability $1-o(1)$ when the number of width 2 clauses is less than ρn, $\rho < 1$, and p is any constant. Actually, the results of this paper are quite far-ranging and provide new insights into the behavior of DPLL algorithms as they visit search nodes that represent $2+p$ expressions.

1.22 Stochastic Local Search

Stochastic local search (SLS) is one of the most successfully and widely used general strategies for solving hard combinatorial problems. Early applications to optimisation problems date back to the to the 1950s, and the Lin-Kernighan algorithm for the Traveling Salesman problem [189] (1973) is still among the most widely known problem-specific SLS algorithms. Two of the most prominent general SLS methods are Simulated Annealing [164] (1983) and Evolutionary Algorithms [94, 137, 243] (1966–1981).

SLS algorithms for SAT were first presented in 1992 by Gu [122] and Selman et al. [248] (following earlier applications to Constraint Satisfaction [207] and MAX-SAT [127]). Interestingly, both Gu and Selman et al. were apparently unaware of the MAX-SAT work, and Hansen and Jaumard and Minton et al. appear to have been unaware of each other's work. The success of Selman et al.'s GSAT algorithm in solving various types of SAT instances more effectively than DPLL variants of the day sparked considerable interest in the AI community, giving rise to a fruitful and active branch of SAT research. GSAT is based on a simple iterative best improvement method with static restarts; in each local search step, it flips the truth value of one propositional variable such that the number of unsatisfied clauses in the given CNF formula is maximally reduced.

Within a year, the original GSAT algorithm was succeeded by a number of variants. These include HSAT [108], which uses a very limited form of search memory to avoid unproductive cycling of the search process; GSAT with Clause Weighting [249], which achieves a similar goal using dynamically changing weights associated with the clauses of the given CNF formula; and GSAT with Random Walk (GWSAT) [249], which hybridizes the "greedy" search method underlying GSAT with a simple random walk procedure (which had previously been shown by Papadimitriou [226] to solve satisfiable 2-CNF formulae almost certainly in $O(n^2)$ steps).

Two relatively subtle modifications of GWSAT lead to the prominent (basic) WalkSAT algorithm [250], which is based on the idea of selecting a currently unsatisfied clause in each step and satisfying that clause by flipping the value assigned to one of its variables. Basic WalkSAT (also known as WalkSAT/SKC) was shown empirically to outperform GWSAT and most other GSAT variants for a broad range of CNF formulae; it is also somewhat easier to implement. Variants of WalkSAT that additionally use search memory, in particular WalkSAT/Tabu [198] (1997) and Novelty$^+$ [139, 140] (1998) – an extension of the earlier Novelty algorithm of McAllester et al. [198] – typically achieve even better performance. Novelty+, which has been proven to solve satisfiable CNF formulae with arbitrarily high probability given sufficient time, was later extended with an adaptive noise mechanism [141] (2002), and the resulting Adaptive Novelty+ algorithm is still one of the most effective SLS algorithms for SAT currently known.

Based on the same fundamental idea of dynamically changing clause weights as GSAT with Clause Weighting, Wah et al. developed an approach known as Discrete Lagrangian Method [255] (1998), whose later variants were shown to be highly effective, particularly for solving structured SAT instances [280, 281]. A conceptually related approach, which uses multiplicatively modified clause weights has been developed by Schuurmans et al. [253] (2001) and later improved by Hutter et al. [143] (2002), whose SAPS algorithm was, until recently, one of the state-of-the-art SLS algorithms for SAT.

A detailed survey of SLS algorithms for SAT up to 2004 can be found in Chapter 6 of the book by Hoos and Stützle [142]. Since then, a number of new algorithms have been developed, including Li and Huang's G^{2}WSAT [187] (2005), which combines ideas from the GSAT and WalkSAT family of algorithms, and Ishtaiwi et al.'s DDFW algorithm [145] (2006), which uses a dynamic clause weight redistribution. In another line of work, Anbulagan et al. have reported results suggesting that by using a resolution-based preprocessing method, the performance of several state-of-the-art SLS algorithms for SAT can be further improved [16](2005).

SLS algorithms for SAT have also played an important role in theoretical work on upper bounds on worst-case time complexity for solving SAT on k-CNF formulae; this includes the previously mentioned random walk algorithm by Papadimitriou [226] (1991) and later extensions by Schöning [244] (1999) and Schuler et al. [247] (2001).

1.23 Non-linear formulations

The nonlinear formulations for SAT are based on the application of the fundamental concept of liftand project for constructing tractable continuous relaxations of hard binary (or equivalently, Boolean) optimization problems. The application of a continuous relaxation to a binary optimization dates back at least to Lovász's introduction of the so-called theta function as a bound for the stability number of a graph a [190]. More generally, the idea of liftings for binary optimization problems has been proposed by several researchers, and has led to different general-purpose frameworks. Hierarchies based on linear programming relaxations include the lift-and-project method of Balas, Ceria and Cornuéjols [25], the reformulation-linearization technique of Sherali and Adams [257], and the matrix-cuts approach of Lovász and Schrijver [191]. Researchers in the SAT community have studied the complexity of applying some of these techniques, and generalizations thereof, to specific classes of SAT problems (see the recent papers [50, 119, 120]).

While the aforementioned techniques use linear programming relaxations, the recent Lasserre hierarchy is based on semidefinite programming relaxations [181, 182]. (Semidefinite constraints may also be employed in the Lovász-Schrijver matrix-cuts approach, but in a different manner from that of the a Lasserre paradigm.) A detailed analysis of the connections between the Sherali-Adams, Lovász-Schrijver, and Lasserre frameworks was done by Laurent [183]. In particular, Laurent showed that the Lasserre framework is the tightest among the three.

Semidefinite programming (SDP) refers to the class of optimization problems where a linear function of a symmetric matrix variable X is optimized subject to linear constraints on the elements of X and the additional constraint that X must be positive semidefinite. This includes linear programming problems as a special case, namely when all the matrices involved are diagonal. The fact that SDP problems can be solved in polynomial time to within a given accuracy follows from the complexity analysis of the ellipsoid algorithm (see [121]). A variety of polynomial time interior-point algorithms for solving SDPs have been proposed in the literature, and several excellent solvers for SDP are now available. The SDP webpage [130] and the books [168, 277] provide a thorough coverage of the theory and algorithms in this area, as well as a discussion of several application areas where semidefinite programming researchers have made significant contributions. In particular, SDP has been very successfully applied in the development of approximation algorithms for several classes of hard combinatorial optimization problems, including maximum-satisfiability (MAX-SAT) problems.

A σ-approximation algorithm for MAX-SAT is a polynomial-time algorithm that computes a truth assignment such that at least a proportion σ of the clauses in the MAX-SAT instance are satisfied. The number σ is the approximation ratio or guarantee. For instance, the first approximation algorithm for MAX-SAT is a $\frac{1}{2}$-approximation algorithm due to Johnson [152]: given n values $\pi_i \in [0,1]$, the algorithm sets the i^{th} Boolean variable to 1 independently and randomly with probability π_i; the resulting total expected weight of the satisfied clauses is $\frac{1}{2}$. Unless $\mathcal{P}$=$\mathcal{NP}$, there is a limit to the approximation guarantees that can be obtained. Indeed, Hästad [129] proved that unless $\mathcal{P}$=$\mathcal{NP}$, for any $\epsilon > 0$, there is no $(\frac{21}{22}+\epsilon)$-approximation algorithm for MAX-2-SAT, and no $(\frac{7}{8}+\epsilon)$-approximation algorithm for MAX-SAT.

The most significant breakthrough was achieved by Goemans and Williamson [112] who proposed an SDP-based 0.87856-approximation algorithm for MAX-2-SAT. The key to their analysis is the ingenious use of a randomly generated hyperplane to extract a binary solution from the set of n-dimensional vectors defined by the solution of the SDP relaxation. The randomized hyperplane rounding procedure can be formally de-randomized using the techniques in [196].

A further significant improvement was achieved by Feige and Goemans [91] who proposed a 0.931-approximation algorithm for MAX-2-SAT. There are two key innovations introduced by Feige and Goemans. The first one is that they tighten the SDP relaxation of Goemans and Williamson by adding the $\binom{n}{3}$ triangle inequalities (to be explained in the URL). From the optimal solution of this strengthened SDP relaxation, they similarly obtain a set of vectors, but instead of applying the random hyperplane rounding technique to these vectors directly, they use them to generate a set of rotated vectors to which they then apply the hyperplane rounding (to be explained in the URL).

Karloff and Zwick [158] proposed a general construction of SDP relaxations for MAX-k-SAT. Halperin and Zwick [126] consider strengthened SDP relaxations for MAX-k-SAT, and specifically for MAX-4-SAT, they studied several rounding schemes, and obtained approximation algorithms that almost attain the theoretical upper bound of $\frac{7}{8}$. Most recently, Asano and Williamson [21] have combined ideas from several of the aforementioned approaches and obtained a 0.7846-approximation algorithm for general MAX-SAT.

For the decision version of SAT, the first SDP-based approach is the Gap relaxation of de Klerk, van Maaren, and Warners [167, 169]. This SDP relaxation was inspired by the work of Goemans and Williamson as well as by the concept of elliptic approximations for SAT instances. These approximations were first proposed in [192] and were applied to obtain effective branching rules as well as to recognize certain polynomially solvable classes of SAT instances. The idea behind the elliptic approximations is to reformulate a SAT formula on n boolean variables as the problem of finding a ± 1 (hence binary) n-vector in an intersection of ellipsoids in $\mathcal{R}^n$. Although it is difficult to work directly with intersections of ellipsoids, it is possible to relax the formulation to an SDP problem. The resulting SDP relaxation is called the Gap relaxation. This relaxation characterizes unsatisfiability for 2-SAT problems [167].

More interestingly, it also characterizes satisfiability for a class of covering problems, such as mutilated chessboard and pigeonhole instances. Rounding schemes and approximation guarantees for the Gap relaxation, as well as its behaviour on $(2+p)$-SAT problems, are studied in [169].

An elliptic approximation uses a quadratic representation of SAT formulas. More powerful relaxations can be obtained by considering higher-degree polynomial representations of SAT formulas. The starting point is to define for each clause a polynomial in ± 1 variables that equals 0 if and only if the clause is satisfied by the truth assignment represented by the values of the binary variables. Thus, testing satisfiability of a SAT formula is reduced to testing whether there are values $x_1, \ldots, x_n \in \{-1, 1\}$ such that for every clause in the instance, the corresponding polynomial evaluated at these values equals zero.

We present two ways that SDP can be used to attempt to answer this question. One of them applies the Lasserre hierarchy mentioned above as follows. The Gap relaxation has its matrix variable in the space of $(n+1) \times (n+1)$ symmetric matrices, and is thus a first lifting. To generalize this operation, we allow the rows and columns of the SDP relaxations to be indexed by subsets of the discrete variables in the formulation. These larger matrices can be interpreted as higher liftings. Applying directly the Lasserre approach to SAT, we would use the SDP relaxations Q_{K-1} (as defined in [181]) for $K = 1, 2, \ldots, n$ where the matrix variable of Q_{K-1} has rows and columns indexed by all the subsets I with $\|I\| \leq K$ (hence for $K = 1$, we obtain the matrix variable of the Gap relaxation). The results in [182] imply that for $K = n$, the resulting SDP relaxation characterizes satisfiability for every instance of SAT. However, this SDP has dimension exponential in n. Indeed, the SDP problems quickly become far too large for practical computation. This limitation motivated the study of partial higher liftings, where we consider SDP relaxations which have a much smaller matrix variable, as well as fewer linear constraints. The construction of such partial liftings for SAT becomes particularly interesting if we let the structure of the SAT instance specify the structure of the SDP relaxation.

One of these partial liftings was proposed in [18]. This construction considers all the monomials $\prod_i x_i$ that appear in the instance's satisfiability conditions. An appropriate SDP relaxation is then defined where each row and column of the matrix variable correspond to one of these terms. The resulting matrix is highly structured, and hence the SDP relaxation can be strengthened by adding some constraints that capture this structure. The tradeoff involved in adding such constraints to the SDP problem is that as the number of constraints increases, the SDP problems become increasingly more demanding computationally. Anjos [18] defines the SDP relaxation R_3 by proposing to add a relatively small number of these constraints, judiciously chosen so that it is possible to prove the following result: if R_3 is infeasible, then the SAT instance is unsatisfiable; while if R_3 is feasible, and Y is a feasible matrix such that $rank(Y) \leq 3$, then the SAT instance is satisfiable, and a model can be extracted from Y . Thus the SDP relaxation can prove either satisfiability or unsatisfiability of the given SAT instance. A more compact relaxation is obtained by defining the columns of the matrix variable using only the sets of odd cardinality. This yields the SDP relaxation R_2 [130], an intermediate relaxation between the Gap relaxation (call it R_1) and R_3. The names of the relaxations reflect their increasing strength in the following sense: For $k = 1, 2, 3$, any feasible solution to the relaxation R_k with rank at most k proves satisfiability of the corresponding SAT instance. Furthermore, the increasing values of k also reflect an improving ability to detect unsatisfiability, and an increasing computational time for solving the relaxation.

From the computational point of view, it is only possible to tackle relatively small SAT instances (regardless of the choice of SDP relaxation) if branching is needed. However, when it does not require branching, the SDP approach can be competitive. For instance, the SDP approach can successfully prove (without branching) the unsatisfiability of the `hgen8` instances, one of which was the smallest unsatisfiable instance that remained unsolved during the SAT competitions of 2003 and 2004.

A second way to test whether there is a set of ±1 values for which the clause polynomials all equal zero was proposed by van Maaren and van Norden [194]. They consider (among others) the aggregate polynomial obtained by summing all the polynomials arising from clauses. This polynomial turns out to be non-negative on $\{-1, 1\}^n$, and for $x \in \{-1, 1\}^n$ it equals the number of unsatisfied clauses. (Hence, MAX-SAT is equivalent to the minimization of this polynomial over $\{-1, 1\}^n$.) An SDP relaxation is obtained as follows. Suppose we are given a column vector β of monomials in the variables $x_1, \ldots, x_n$ and a polynomial $p(x)$. Then $p(x)$ can be written as a sum-of-squares (SOS) in terms of the elements of β if and only if there exists a matrix $S \succeq 0$ such that $\beta^T S \beta = p$ [220]. If S is symmetric positive semidefinite, then $S = W^T W$ for some matrix W, and hence we have an explicit decomposition of p as an SOS: $\beta^T S \beta = p \Rightarrow ||W\beta||_2^2 = p$. The resulting SDP problem is

$$\begin{array}{rl} \max & g \\ \text{s.t.} & F_\Phi^{\mathcal{B}}(x) - g \equiv \beta^T S \beta \text{ modulo } I_{\mathcal{B}} \\ & S \succeq 0 \end{array}$$

where $I_{\mathcal{B}}$ denotes the ideal generated by the polynomials $x_k^2 - 1, k = 1, \ldots, n$. (The fact that each k polynomial that is non-negative on $\{-1, 1\}^n$ can be expressed as an SOS modulo $I_{\mathcal{B}}$ follows from the work of Putinar [222].) Note that since β is fixed, the equation $F(x) - g = \beta^T S \beta$ is linear in S and g, and hence this is an SDP problem. The SOS approach can thus be applied to obtain proofs of (un)satisfiability. For instance, it is straightforward to prove that if there exists a monomial basis β and an $\epsilon > 0$ such that $F^{\mathcal{B}}(x) - \epsilon$ is a

SOS modulo $I_{\mathcal{B}}$, then the underlying SAT formula is unsatisfiable.

For the SOS approach, different choices of the basis β result in different SDP relaxations. Among the choices considered by van Maaren and van Norden are the following: SOS GW is the relaxation obtained using the basis containing $1, x_1, \ldots, x_n$; SOS$_p$ is obtained using the basis containing $1, x_1, \ldots, x_n$, plus the monomial $x_{k_1}x_{k_2}$ for each pair of variables that appear together in a clause; SOS$_{ap}$ is obtained using the basis containing $1, x_1, \ldots, x_n$, plus the monomials $x_{k_1}x_{k_2}$ for all pairs of variables; SOS$_t$ is obtained using the basis containing $1, x_1, \ldots, x_n$, plus the monomial $x_{k_1}x_{k_2}x_{k_3}$ for each triple of variables that appear together in a clause; and SOS$_{pt}$ is obtained using the basis containing $1, x_1, \ldots, x_n$, plus the monomial $x_{k_1}x_{k_2}$ for each pair of variables that appear together in a clause, plus $x_{k_1}x_{k_2}x_{k_3}$ for each triple of variables that appear together in a clause.

The notation SOS$_{GW}$ is justified by the fact that SOS$_{GW}$ is precisely the dual of the SDP relaxation used by Goemans and Williamson in their seminal paper [112]. van Maaren and van Norden prove that SOS$_{GW}$ gives the same upper bound for MAX-2-SAT as the relaxation of Goemans and Williamson. They also show that for each triple $x_{k_1}x_{k_2}x_{k_3}$, adding the monomials $x_{k_1}x_{k_2}$, $x_{k_1}x_{k_3}$, and $x_{k_2}x_{k_3}$ gives an SDP relaxation at least as tight as that obtained by adding the corresponding triangle inequality to the Goemans-Williamson relaxation. Furthermore, they prove that the SDP relaxation SOS$_{ap}$ is at least as tight as the Feige-Goemans relaxation, and that for every instance of MAX-3-SAT, the SDP relaxation SOS_{pt} provides a bound at least as tight as the Karloff-Zwick relaxation.

From the computational point of view, van Maaren and van Norden provide computational results comparing several of these relaxations on instances of varying sizes and varying ratios of number of clauses to number of variables. They propose rounding schemes for MAX-2-SAT and MAX-3-SAT based on SOS$_p$ and SOS$_t$ respectively, and present preliminary results comparing their performance with the rounding schemes mentioned above. They also compare the performance of the R_3 relaxation with the SOS approach using either SOS$_t$ or SOS$_{pt}$. Their preliminary results suggest that SOS$_{pt}$ offers the best performance.

The most recent result about the nonlinear approach is that the SDP approach can explicitly characterize unsatisfiability for the well-known Tseitin instances on toroidal grid graphs. Consider a $p \times q$ toroidal grid graph and for each node (i, j), set the parameter $t(i, j) = 0$ or 1. Introduce a Boolean variable for each edge, and for each node (i, j), define 8 clauses on the four variables adjacent to it as follows: if $t(i, j) = 0$, add all clauses with an odd number of negations; and if $t(i, j) = 1$, add all clauses with an even number of negations. It is clear that the SAT instance is unsatisfiable if and only if $\sum_{(i,j)} t(i, j)$ is odd. It is shown in [19] how to construct an SDP relaxation with matrix variable of dimension $14pq$ and with $23pq-1$ linear equality constraints such that the SDP problem is infeasible if and only if the SAT instance is unsatisfiable. Therefore, for these instances, the SDP-based approach provides, in theory, an explicit certificate of (un)satisfiability, and therefore makes it possible to numerically compute such a certificate to within a given precision in polynomial time.

1.24 Quantified Boolean formulas

The concept of quantified Boolean formulas (QBF) is an extension of propositional logic that allows existential ($\exists$) and universal ($\forall$) quantifiers. The intended semantics of quantified Boolean formulas, without free variables, is that a universally quantified formula $\psi = \forall x \phi$ is true if and only if for every assignment of the truth values 1 and 0 to the variable x, ψ is true. For existentially quantified formulas $\psi = \exists x \phi$, ψ is true if and only if there is an assignment to x for which ϕ is true. In the case of free variables, ψ is called *satisfiable* if there is a truth assignment to the free variables such that ψ is true. The satisfiability problem for quantified Boolean formulas is often denoted as QSAT.

Most of the research has been done for formulas in prenex form, that is, formulas of the form $Q_1 x_1 \dots Q_n x_n \phi$, where $Q_i \in \{\exists, \forall\}, x_1, \dots, x_n$ are variables, and ϕ is a propositional formula called the *matrix*. By standard techniques, every quantified Boolean formula can be transformed into a logically equivalent formula in prenex form. In the same way, by the well-known procedure for propositional formulas, logically equivalent formulas with a CNF matrix or 3-CNF matrix can be obtained. These classes are denoted as QCNF and Q3-CNF.

Quantified Boolean formulas with free variables are logically equivalent to Boolean functions. But, clearly, there is no polynomial p such that every n-ary Boolean function can be represented as a quantified Boolean formula of length $p(n)$. However, for various applications, QBFs lead to shorter formulas in comparison to propositional formulas. See, for example, [153].

The first papers on QBFs were motivated by questions arising from computational complexity. Like SAT for NP, it has been shown in [203] that QSAT is one of the prominent $\mathcal{PSPACE}$-complete problems. In a more detailed analysis, a strong relationship was shown between the satisfiability problem of formulas with a fixed number of alternations of quantifiers and the polynomial–time hierarchy [202] where the polynomial–time hierarchy is defined as follows ($k \geq 0$):

$$\Delta_0^P := \Sigma_0^P := \Pi_0^P := P$$

$$\Sigma_{k+1}^P := NP^{\Sigma_k^P}, \ \Pi_{k+1}^P := co\Sigma_{k+1}^P, \ \Delta_{k+1}^P := P^{\Sigma_k^P}$$

For quantified Boolean formulas in prenex form, the prefix type is defined as follows:

1. The prefix type of a propositional formula is $\Sigma_0 = \Pi_0$.
2. Let Φ be a formula with prefix type Σ_n (Π_n respectively), then the formula $\forall x_1 \ldots \forall x_n \Phi$ ($\exists x_1 \ldots \exists x_n \Phi$ respectively) is of prefix type Π_{n+1} (Σ_{n+1} respectively).

It has been proved that for $k \geq 1$, the satisfiability problem for formulas with prefix type Σ_k (Π_k respectively) is Σ_k^P-complete (Π_k^P-complete respectively) [223, 279]. Since $\mathcal{PSPACE} = \mathcal{NPSPACE}$ [241], almost all quantified Boolean formula problems are solvable in $\mathcal{PSPACE}$. For example, in propositional logic, the equivalence problem is $\rfloor\wr\mathcal{NP}$-complete, whereas for quantified Boolean formulas, the problem remains $\mathcal{PSPACE}$-complete.

In addition to propositional formulas, a dichotomy theorem for quantified Boolean formulas has been established in [242]. The idea was to classify classes of quantified Boolean formulas by means of a finite set of constraints, where the constraints are Boolean functions. The dichotomy theorem says that if the Boolean functions are equivalent to Horn formulas (anti-Horn formulas, 2-CNF formulas, XOR-CNF formulas respectively), then the satisfiability problems for the quantified classes are in P, otherwise they are $\mathcal{PSPACE}$-complete. As a consequence, the solvability of the satisfiability problem for Q2-CNF and QHORN follows, where QHORN is the set of formulas, whose matrix is a Horn formula. Detailed proofs and extensions of the dichotomy theorem can be found, for example, in [76, 75]. For formulas with fixed prefix type, further results have been shown in [131].
For Q2-CNF, the first linear-time algorithm for solving the satifiability problem has been presented in [22]. For QHORN, the best known algorithm can be found in [173]. The latter algorithm requires not more than $O(r \cdot n)$ steps, where r is the number of universal variables and n is the length of the formula.

Not all the problems solvable in polynomial time for propositional formulas remain polynomial-time solvable for quantified formulas. For example, in contrast to the polynomial-time solvability of the equivalence problem for Horn formulas, the equivalence problem for quantified Horn formulas is $\rfloor\wr\mathcal{NP}$-complete [174].

Instead of restrictions on the form of clauses, classes of quantified formulas satisfying some graph properties have been investigated. Examples are quantified versions of ordered binary decision diagrams (OBDDs) and free binary decision diagrams (FBDDs) (see, for example, [65]). For instance, the satisfiability problem for quantified FBDDs is $\mathcal{PSPACE}$-complete.

Q-resolution is an extension of the resolution calculus for propositional formulas to quantified formulas. Q-unit-resolution was introduced in [160] and was then generalized to Q-resolution in [173]. Here, a Q-unit clause contains at most one free or existentially quantified literal and arbitrarily many universal literals. The idea of Q-resolution is to resolve only over complementary pairs of existential or free variables, combined with a careful handling of the universal literals. Q-resolution is refutation complete and sound for QCNF. Similar to propositional Horn formulas, Q-unit-resolution is refutation complete for QHORN, and also for the class QEHORN. That is the set of formulas for which, after deleting the universal literals in the matrix, the remaining matrix is a Horn formula. The satisfiability problem for that class remains $\mathcal{PSPACE}$-complete [161].

A more functional view of the valuation of quantified Boolean formulas is the observation that a formula is true if and only if for every existential variable y there is a Boolean function $f_y(x_1, \ldots, x_m)$ depending on the dominating universal variables $x_1, \ldots, x_m$, such that after replacing the existential variables y by the associated functions $f_y(x_1, \ldots, x_m)$ the formula is true. The set of such functions is called a satsifiability model. For some classes of quantified Boolean formulas, the structure of the satisfiability models has been investigated. For example, satisfiable quantified Horn formulas have satisfiability models which consist of the constants *true* and *false* or conjunctions of variables. For Q2-CNF, the models are constants or a literal [175]. Instead of satisfiability models, one can ask for Boolean functions, such that after replacing the existentially quantified variables with the functions, the formula is logically equivalent to the initial formula. These functions are called equivalence model. Equivalence models describe in a certain sense the internal dependencies of the formula. For example, quantified Horn formulas have equivalence models consisting of monotone functions. By means of this result, it has been shown that every quantified Horn formula can be transformed into an equivalent existentially quantified Horn formula in time $O(r \cdot n)$, where r is the number of universal variables and n is the length of the formula [48].

1.25 Maximum Satisfiability

The problem of finding a truth assignment to the variables of a CNF expression that satisfies the maximum number of clauses possible is known as Maximum Satisfiability or MAX-SAT. If clauses have at most two literals each, the problem is known as MAX-2-SAT. The decision version of MAX-SAT and even MAX-2-SAT is $\mathcal{NP}$-complete. Unfortunately, there is no polynomial time approximation scheme for MAX-SAT unless $\mathcal{P} = \mathcal{NP}$ [20]. Because the MAX-SAT problem is fundamental to many practical problems in Computer Science [127] and Electrical Engineering [282], efficient methods that can solve a large set of instances of MAX-SAT are eagerly sought.

MAX-SAT: Decision Algorithms

Many of the proposed methods for MAX-SAT are based on approximation algorithms [78] (2002); some of them are based on branch-and-bound methods [127] (1990), [36] (1999), [26] (1999), [136] (2000), [217] (2000), [110] (2003); and some of them are based on transforming MAX-SAT into SAT [282] (2002), [15] (2002).

Worst-case upper bounds have been obtained with respect to three parameters: the length L of the input formula (*i.e.*, the number of literals in the input), the number m of the input's clauses, and the number n of distinct variables occurring in the input. The best known bounds for MAX-SAT are $O(L2^{m/2.36})$ and $O(L2^{L/6.89})$ [26] (1999). The question of whether there exist exact algorithms with complexity bounds down to $O(L2^n)$ has been open and of great interest (see [216] (1998), [13] (2000), [118] (2003)) since an algorithm which enumerates all the 2^n assignments and then counts the number of true clauses in each assignment would take time $O(L2^n)$. Recently, it has been shown that a branch-and-bound algorithm can achieve $O(b2^n)$ complexity where b is the maximum number of occurrences of any variable in the input. Typically, $b \simeq L/n$.

The operation of the best branch-and-bound algorithms for MAX-SAT is similar to that of DPLL. Notable implementations are due to Wallace and Freuder (implemented in Lisp) [274] (1996), Gramm [117] (1999), Borchers and Furman [36] (1999 - implemented in C and publicly available), Zhang, Shen, and Manyà [283] (2003), and Zhao and Zhang [284] (2004).

MAX-2-SAT: Decision Algorithms

MAX-2-SAT is important because a number of other $\mathcal{NP}$-complete problems can be reduced to it, for example graph problems such as Maximum Cut and Independent Set [56] (1996), [195]. For MAX-2-SAT, the best bounds have been improved from $O(m2^{m/3.44})$ [26] (1999), to $O(m2^{m/2.88})$ [217] (2000), and recently to $O(m2^{m/5})$ [118] (2003). The recent branch-and-bound algorithm cited above results in a bound of $O(n2^n)$ since $b \leq 2n$. When $m = 4n^2$ the bound is $O(\sqrt{m}1.414^{\sqrt{m}})$, which is substantially better than the result reported in [118].

For random 2-CNF formulas satisfiability thresholds have been found as follows:

Theorem [64] :

1. For $c < 1$, $K(n, cn) = \Theta(1/n)$.

2. For c large,

$$(0.25c - 0.343859\sqrt{c} + O(1))n \succsim K(n, cn) \succsim (0.25c - 0.509833\sqrt{c})n$$

3. For any fixed $\epsilon > 0$, $\frac{1}{3}\epsilon^3 n \succsim K(n, (1+\epsilon)n)$.

In the above theorem, $\succsim$ is a standard asymptotic notation: $f(n) \succsim g(n)$ means that f is greater than or equal to g *asymptotically*, that is, $f(n)/g(n) \geq$

1 when n goes to infinity, although it may be that $f(n) < g(n)$ even for arbitrarily large values of n.

1.26 Pseudo-Boolean functions

1.27 Survey propagation

1.28 SAT solvers become important tools

GRASP, zChaff, miniSAT, Berkmin etc. Yearly SAT competitions.

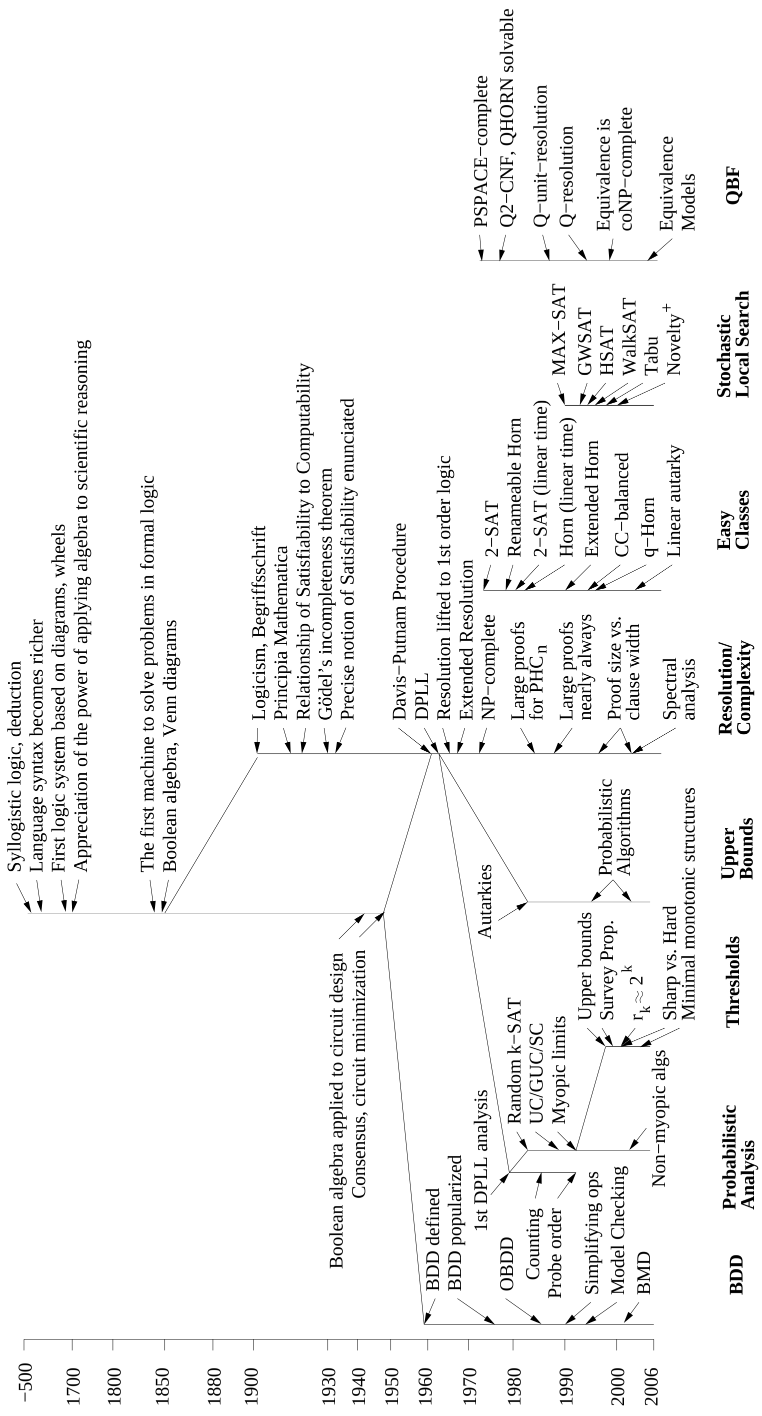

Figure 1.2: **Timeline**: Some nodes on the history tree. The intention is to provide some perspective on the level of activity in a paticular area during a particular period. This figure is not yet complete, suggestions and comments are invited.

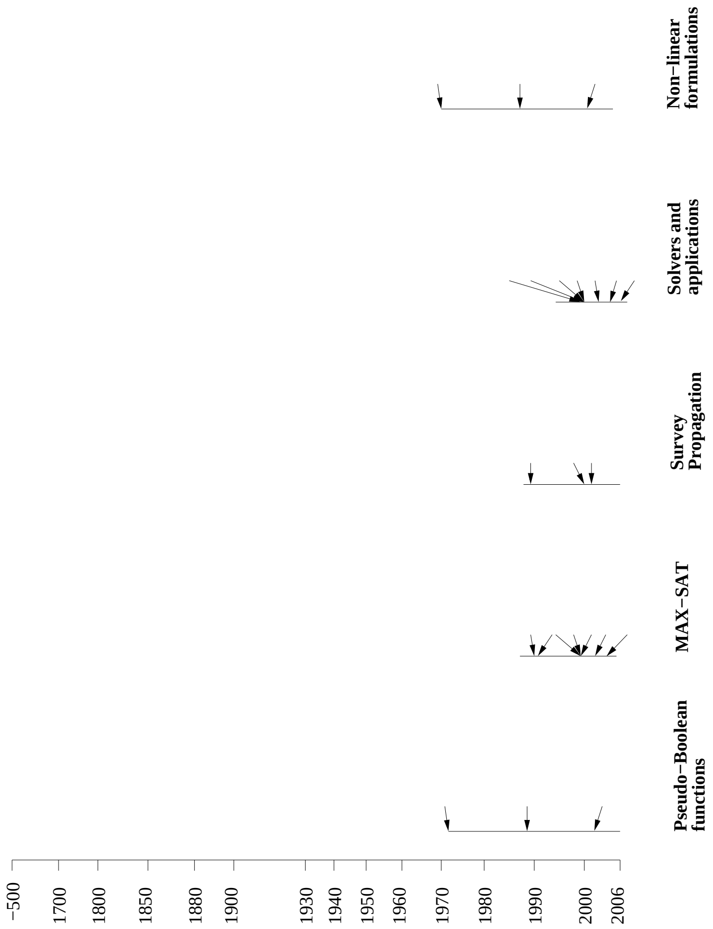

Figure 1.3: **Timeline**: Some nodes on the history tree. The intention is to provide some perspective on the level of activity in a paticular area during a particular period. Suggestions and comments are invited.

Bibliography

[1] D. Achlioptas. Lower bounds for random 3-SAT via differential equations. *Theoretical Computer Science*, **265**:159–185, 2001.

[2] D. Achlioptas. Setting 2 variables at a time yields a new lower bound for random 3-sat. *32nd ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 2000.

[3] D. Achlioptas, and G. Sorkin. Optimal myopic algorithms for random 3-SAT. *41st Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 2000.

[4] D. Achlioptas, and C. Moore. The asymptotic order of the random k-SAT thresholds. *43rd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 2002.

[5] D. Achlioptas, and Y. Peres. The threshold for random k-SAT is $2^k(\ln 2 + o(1))$. *Journal of the American Mathematical Society*, **17**:947–973, 2004.

[6] D. Achlioptas, L.M. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. Stamatiou. Random constraint satisfaction: a more accurate picture. *3rd Conference on the Principles and Practice of Constraint Programming* (Linz, Austria), LNCS, 1330:107–120, Springer, 1997.

[7] D. Achlioptas, L.M. Kirousis, E. Kranakis, D. Krizanc. Rigorous results for random (2+p)-SAT. *Theoretical Computer Science*, **265**(1-2):109–129, 2001.

[8] D. Achlioptas, and M. Molloy. The analysis of a list-coloring algorithm on a random graph. *38th Annual Symposium on Foundations of Computer Science* (Miami, Florida), 204–212, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[9] D. Achlioptas, P. Beame, and M. Molloy. A sharp threshold in proof complexity yields lower bounds for satisfiability search. *Journal of Computer and System Sciences*, **68**:238–268, 2004.

[10] D. Achlioptas, A. Chtcherba, G. Istrate, and C. Moore. The phase transition in 1-in-k SAT and NAE 3-SAT. In *Proceedings of the 12th ACM-IEEE Symposium on Discrete Algorithms*, 721–722, 2001.

[11] R. Aharoni and N. Linial. Minimal Non-Two-Colorable Hypergraphs and Minimal Unsatisfiable Formulas. *Journal of Combinatorial Theory, Series A*, **43**:196–204, 1986.

[12] S.B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, **C-27**(6):509–516, 1978.

[13] J. Alber, J. Gramm, R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics*, **229**(1-3):3–27, 2001.

[14] N. Alon, and J. Spencer. *The Probabilistic Method* (2nd Edition). Wiley, 2000.

[15] F.A. Aloul, A. Ramani, I.L. Markov, K.A. Sakallah. Generic ILP versus specialized 0-1 ILP: An update. In *Proceedings of the 2002 International Conference on Computer-Aided Design* (ICCAD '02), 450–457, IEEE Computer Society Press, Los Alamitos, CA, 2002.

[16] Anbulagan, D.N. Pham, J. Slaney, and A. Sattar. Old resolution meets modern SLS. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 2005.

[17] M.F. Anjos. On semidefinite programming relaxations for the satisfiability problem. *Mathematical Methods in Operations Research*, **60**(3), 2004.

[18] M.F. Anjos. An improved semidefinite programming relaxation for the satisfiability problem. *Mathematical Programming*, **102**(3):589–608, 2005.

[19] M.F. Anjos. An explicit semidefinite characterization of satisfiability for Tseitin instances on toroidal grid graphs. *Annals of Mathematics and Artificial Intelligence*, to appear.

[20] S. Arora, C. Lund. Hardness of approximation. In *Approximation algorithms for NP-hard problems*, D. Hochbaum (ed.), Chapter 10, 399–446. PWS Publishing Company, Boston, 1997.

[21] T. Asano, and D.P. Williamson. Improved approximation algorithms for MAX SAT. *Journal of Algorithms*, **42**(1):173–202, 2002.

[22] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–132, 1979.

[23] B. Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms* **1**:97–103, 1980.

[24] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the International Conference on Computer-Aided Design*, 188–191, IEEE Computer Society Press, Los Alamitos, CA, 1993.

[25] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, **58**(3,Ser. A):295–324, 1993.

[26] N. Bansal, V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of 10th Annual conference on Algorithms and Computation*, ISSAC'99, Aggarwal and Rangan (eds.), Lecture Notes in Computer Science, **1741**:247–258, Springer-Verlag, 1999.

[27] M.E. Baron. A note on the historical development of logic diagrams. *The Mathematical Gazette: The Journal of the Mathematical Association*, **LIII**(383), 1969.

[28] P. Beame, and T. Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science* (Burlington, VT), 274–282, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[29] P. Beame, R.M. Karp, T. Pitassi, and M. Saks. On the complexity of unsatisfiability proofs for random k-CNF formulas. In *Proceedings of the 30th Annual Symposium on the Theory of Computing* (Dallas, TX), 561–571, 1998.

[30] P. Beame, H. Kautz, and A. Sabharwal. On the power of clause learning. In *Proceedings of the 18th International Joint Conference in Artificial Intelligence*, 94–99, Acapulco, Mexico, 2003.

[31] E. Ben-Sasson, and A. Wigderson. Short proofs are narrow - resolution made simple. *Journal of the Association for Computing Machinery*, 48:149–169, 2001.

[32] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near Optimal Separation of Tree-like and General Resolution. *Combinatorica*, 585–603, 2004.

[33] A. Biere, A. Cimatti, E. Clarke, Y. Zhu. Symbolic Model Checking without BDDs. *Lecture Notes in Computer Science*, **1579**:193–207, 1999.

[34] A. Blake. Canonical Expressions in Boolean Algebra. Ph.D. Dissertation, Department of Mathematics, University of Chicago, 1937.

[35] B. Bollobás, C. Borgs, J. Chayes, J.H. Kim, and D.B. Wilson. The scaling window of the 2-SAT transition. *Random Structures and Algorithms*, **18**:201–256, 2001.

[36] B. Borchers, J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, **2**(4):299–306, 1999.

[37] E. Boros, Y. Crama, and P.L. Hammer. Polynomial-time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, **1**:21–32, 1990.

[38] E. Boros, P.L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics*, **55**:1–13, 1994.

[39] E. Boros, Y. Crama, P.L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journal on Computing*, **23**:45–49, 1994.

[40] R.T. Boute. The Binary Decision Machine as a programmable controller. *EUROMICRO Newsletter*, **1**(2):16–22, 1976.

[41] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 40–45, IEEE Computer Society Press, 1990.

[42] A.Z. Broder, A.M. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. *4th Annual ACM-SIAM Symposium on Discrete Algorithms* (Austin, TX), 322–330. Association for Ccomputing Machinery, New York, 1993.

[43] T. Brueggemann, and W. Kern. An improved local search algorithm for 3-SAT. *Theoretical Computer Science*, **329**:1–3, 303–313, 2004.

[44] F.M. Brown. Boolean Reasoning. Dover Publications, Mineola, New York, 2003.

[45] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, **C-35**(8):677–691, 1986.

[46] R.E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, **24**(3):293–318, 1992.

[47] R.E. Bryant, and Y.-A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, 535–541, IEEE Computer Society Press, 1995.

[48] U. Bubeck, H. Kleine Büning, X. Zhao. Quantifier rewriting and equivalence models for quantified Horn formulas. *Lecture Notes in Computer Science*, **3569**:386–392, Springer, 2005.

[49] J.R. Burch, E.M. Clarke, and K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, **98**:142–170, 1992.

[50] J. Buresh-Oppenheim, N. Galesi, S. Hoory, A. Magen, and T. Pitassi. Rank bounds and integrality gaps for cutting plane procedures. *44th Annual Symposium on Foundations of Computer Science*, 318–327, IEEE Computer Society Press, Los Alamitos, CA, 2003.

[51] R. Chandrasekaran. Integer programming problems for which a simple rounding type of algorithm works. In W. Pulleyblank, ed. *Progress in Combinatorial Optimization.* Academic Press Canada, Toronto, Ontario, Canada, 101–106, 1984.

[52] V. Chandru, and J.N. Hooker. Extended Horn sets in propositional logic. *Journal of the Association for Computing Machinery*, **38**:205–221, 1991.

[53] M.-T. Chao, and J. Franco. Probabilistic analysis of two heuristics for the 3-Satisfiability problem. *SIAM Journal on Computing*, 15:1106–1118, 1986.

[54] M.-T. Chao, and J. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k-satisfiability problem. *Information Sciences*, 51:289–314, 1990.

[55] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 331–340, Morgan Kaufmann, 1991.

[56] J. Cheriyan, W.H. Cunningnham, L. Tuncel, Y. Wang. A linear programming and rounding approach to Max 2-Sat. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **26**:395–414, 1996.

[57] A.S. Church. Formal definitions in the theory of ordinal numbers. *Fundamental Mathematics*, **28**:11-21, 1936.

[58] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). *33th Annual Symposium on Foundations of Computer Science* (Pittsburgh, Pennsylvania), 620–627, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[59] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *Journal of the Association for Computing Machinery*, **35**:759–768, 1988.

[60] A. Cimatti, E. Giunchiglia, P. Giunchiglia, and P. Traverso. Planning via model checking: a decision procedure for AR. *Lecture Notes in Artificial Intelligence*, **1348**:130–142, Springer-Verlag, New York, 1997.

[61] M. Conforti, G. Cornuéjols, A. Kapoor, K. Vušković, and M.R. Rao. Balanced Matrices. Mathematical Programming: State of the Art. J.R. Birge and K.G. Murty, eds. Braun-Brumfield, United States. Produced in association with the 15th International Symposium on Mathematical Programming, University of Michigan, 1994.

[62] S.A. Cook. The complexity of theorem-proving procedures. *3rd Annual ACM Symposium on Theory of Computing* (Shaker Heights, OH), 151–158, ACM, New York, 1971.

[63] S.A. Cook and R.A. Reckhow. Corrections for "On the lengths of proofs in the propositional calculus preliminary version." *SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 6:15-22, 1974.

[64] D. Coppersmith, D. Gamarnik, M. Hajiaghay, G.B. Sorkin. Random MAX SAT, random MAX CUT, and their phase transitions. In *Proceedings of the 14^{th} Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA), 2003.

[65] S. Coste-Marquis, D. Le Berre, F. Letombe, and P. Marquis. Complexity results for quantified Boolean formulae based on complete propositional languages. *Journal on Satisfiability*, Boolean Modeling and Computation **1**:61–88, 2006.

[66] O. Coudert, C. Berthet and J.C. Madre. Verification of synchronous sequential machines based on symbolic execution. *Lecture Notes in Computer Science*, **407**:365–373, Springer, 1990.

[67] O. Coudert, and J.C. Madre. A unified framework for the formal verification of sequential circuits. In *Proceedings of the 1990 International Conference on Computer-Aided Design* (ICCAD '90), 126–129, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[68] O. Coudert. Two-level logic minimization: an overview. *Integration*, **17**(2):97–140, 1994.

[69] O. Coudert, J.C. Madre, H. Fraisse, H. Touati. Implicit prime cover computation: an overview. In *Proceedings of the First Workshop on Synthesis And System Integration of Mixed Information technologies*, 1993.

[70] N. Creignou and H. Daudé: Generalized satisfiability problems: Minimal elements and Phase transitions. Theoretical Computer Science **302(1-3)** (2003) 417–430 (Appeared in an earlier version in: Proceedings of the 5th International Symposium on Theory and Applications of Satisfiability Testing, SAT'2002 Cincinatti (2002) 17–26).

[71] N. Creignou and H. Daudé. Satisfiability threshold for random XOR-CNF formulas. *Discrete Applied Mathematics*, **96-97**:41–53, 1999.

[72] N. Creignou and H. Daudé. Smooth and sharp thresholds for random k-XOR-CNF satisfiability. *Theoretical Informatics and Applications*, **37**(2):127–148, 2003.

[73] N. Creignou and H. Daudé. Combinatorial sharpness criterion and phase transition classification for random CSPs. *Information and Computation*, **190**(2):220–238, 2004.

[74] N. Creignou, H. Daudé, and J. Franco. A sharp threshold for the renameable Horn and q-Horn properties. *Discrete Applied Mathematics*, **153**:48–57, 2005.

[75] N. Creignou, S. Khanna, and M. Sudan. Complexity classifications of Boolean constraint satisfaction problems. *Monographs on Discrete Applied Mathematics*, SIAM, 2001.

[76] V. Dalmau. Some dichotomy theorems on constant free Boolean formulas. Technical Report TR-LSI-97-43-R, Universitat Polytechnica de Catalunya, 1997.

[77] R. Damiano, and J. Kukula. Checking satisfiability of a conjunction of BDDs. In *Proceedings of the 40^{th} ACM/IEEE Design Automation Conference*, 818–823, IEEE Computer Society Press, 2003.

[78] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k–SAT based on local search. *Theoretical Computer Science*, **289**:69–83, 2002.

[79] E. Dantsin, and A. Wolpert. Derandomization of Schuler's algorithm for SAT. In *Lecture Notes in Computer Science*, **2919**:69–75, Springer, New York, 2004.

[80] E. Dantsin, and A. Wolpert. An improved upper bound for SAT. In *Lecture Notes in Computer Science*, **3569**:400–407, Springer, New York, 2005.

[81] M. Davis. The early history of Automated Deduction. In *Handbook of Automated Deduction*, A. Robinson and A. Voronkov, eds., MIT Press, 2001.

[82] M. Davis, and H. Putnam. Computational methods in the propositional calculus. Unpublished report, Rensselaer Polytechnic Institute, 1958.

[83] M. Davis, and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, **7**(3):201–215, 1960.

[84] M. Davis, G. Logemann, D. Loveland. A machine program for theorem proving. *Communications of the ACM*, **5**:394–397, 1962.

[85] W.F. Dowling, and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, **1**:267–284, 1984.

[86] O. Dubois, and Y. Boufkhad. A general upper bound for the satisfiability threshold of random r-SAT formulae. *Journal of Algorithms*, **24**:395–420, 1997.

[87] O. Dubois. Upper bounds on the satisfiability threshold. *Theoretical Computer Science*, **265**:187–197, 2001.

[88] O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In *Proceedings 11th ACM-SIAM Symposium on Discrete Algorithms*, (San Francisco, CA), 124–126, ACM, New York, 2000. Also available from http://arxiv.org/abs/cs/0211036 (2002).

[89] A. El Maftouhi, and W. Fernandez de la Vega. On random 3-SAT. *Combinatorics, Probability, and Computing*, 4:189-195, 1995.

[90] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, **5**:691–703, 1976.

[91] U. Feige, and M. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the* 3^{rd} *Israel Symposium on the Theory of Computing and Systems*, 182–189, 1995.

[92] W. Fernandez de la Vega. On random 2-sat. Manuscript, 1992.

[93] A. El Maftouhi, and W. F. de la Vega. On Random 3-SAT. Manuscript, Laboratoire de Recherche en Informatique, Université Paris-Sud, France. 1997.

[94] L.J. Fogel, A.J. Owens, and M.J. Walsh. Artificial Intelligence Through Simulated Evolution. John Wiley & Sons, New York, NY, USA, 1966.

[95] J. Franco. On the probabilistic performance of algorithms for the satisfiability problem. *Information Processing Letters*, 23:103–106, 1986.

[96] J. Franco. Elimination of infrequent variables improves average case performance of satisfiability algorithms. *SIAM Journal on Computing*, 20:1119–1127, 1991.

[97] J. Franco. On the occurrence of null clauses in random instances of satisfiability. *Discrete Applied Mathematics*, **41**:203–209, 1993.

[98] J. Franco and Y.C. Ho. Probabilistic performance of heuristic for the satisfiability problem. *Discrete Applied Mathematics*, 22:35–51, 1988/89.

[99] J. Franco, and M. Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[100] J. Franco, and A. Van Gelder. A Perspective on Certain Polynomial Time Solvable Classes of Satisfiability. *Discrete Applied Mathematics*, **125**(2-3):177–214, 2003.

[101] J. Franco, M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W.M. Vanfleet. SBSAT: a state-based, BDD-based satisfiability solver. *Lecture Notes in Computer Science*, **2919**:398–410, 2004.

[102] E. Friedgut, and an appendix by J. Bourgain. Sharp thresholds of graph properties, and the k-sat problem. *Journal of the American Mathematical Society*, **12**(4):1017–1054, 1999.

[103] A.M. Frieze, and S. Suen. Analysis of two simple heuristics on a random instance of k-SAT. *Journal of Algorithms*, **20**:312–355, 1996.

[104] X. Fu. On the complexity of proof systems. Ph.D. Thesis, University of Toronto, 1995.

[105] Z. Galil. On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science*, 4:23–46, 1977.

[106] M. Gardner. Logic Machines and Diagrams. McGraw-Hill, New York, 1958.

[107] H. Gelernter. Realization of a geometry-theorem proving machine. In *Proceedings of the International Conference on Information Processing*, UNESCO House, 273–282, 1959.

[108] I.P. Gent, and T. Walsh. An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research*, **1**:47–59, 1993.

[109] P. Gilmore. A proof method for quantification theory: its justification and realization. *IBM Journal of Research and Development*, **4**:28–35, 1960.

[110] S. de Givry, J. Larrosa, P. Meseguer, T. Schiex. Solving MAX-SAT as weighted CSP. *Lecture Notes in Computer Science*, **2833**:363–376, Springer, 2003.

[111] K. Gödel. On Formally Undecidable Propositions of *Principia Mathmatica* and Related Systems. 1931.

[112] M.X. Goemans, and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, **42**(6):1115–1145, 1995.

[113] A. Goerdt. A threshold for unsatisfiability. *Journal of Computer System Science*, **53**:469–486, 1996.

[114] A. Goerdt, and M. Krivelevich. Efficient recognition of random unsatisfiable k-SAT instances by spectral methods. In *Lecture Notes in Computer Science*, 2010:294–, 2001.

[115] A. Goldberg. On the complexity of the satisfiability problem. *4th Workshop on Automated Deduction* (Austin, TX), 1–6, 1979.

[116] A. Goldberg, P.W. Purdom, and C. Brown. Average time analysis of simplified Davis-Putnam procedures. *Information Processing Letters*, 15:72–75, 1982.

[117] J. Gramm. Exact algorithms for Max2Sat and their applications. Diplomarbeit, Universität Tübingen, October 1999.

[118] J. Gramm, E.A. Hirsch, R. Niedermeier, P. Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Discrete Applied Mathematics*, **130**(2):139–155, 2003.

[119] D. Grigoriev, E.A. Hirsch, and D.V. Pasechnik. Complexity of semialgebraic proofs. *Moscow Mathematics Journal*, **2**(4):647–679, 2002.

[120] D. Grigoriev, E.A. Hirsch, and D.V. Pasechnik. Exponential lower bound for static semi-algebraic proofs. *Lecture Notes in Computer Science*, **2380**:257–268, Springer, Berlin, 2002.

[121] M. Grötschel, L. Lovsz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization, volume 2 of Algorithms and Combinatorics*, Springer-Verlag, Berlin, second edition, 1993.

[122] J. Gu. Efficient local search for very large scale satisfiability problems. *SIGART Bulletin*, **3**(1):8–12, 1992.

[123] M.T. Hajiaghayi, and G.B. Sorkin. The satisfiability threshold of random 3-SAT is at least 3.52.
Available from http://arxiv.org/pdf/math.CO/0310193.

[124] A. Haken. The intractability of resolution. *Theoretical Computer Science*, **39**:297–308, 1985.

[125] P. Hall. On representatives of subsets. *Journal of London Mathematical Society*, **10**:26–30, 1935.

[126] E. Halperin, and U. Zwick. Approximation algorithms for MAX 4-SAT and rounding procedures for semidefinite programs. *Journal of Algorithms*, **40**(2):184–211, 2001.

[127] P. Hansen, B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, **44**:279–303, 1990.

[128] P. Hansen, B. Jaumard, and G. Plateau. An extension of nested satisfiability. Les Cahiers du GERAD, G-93-27, 1993.

[129] J. Hästad. Some optimal inapproximability results. *Journal of the Association for Computing Machinery*, **48**:798–859, 2001.

[130] C. Helmberg. `http://www-user.tu-chemnitz.de/ helmberg/semidef.html.`

[131] E. Hemaspaandra. Dichotomy theorems for alternation-bound quantified Boolean formulas. *ACM Computing Research Repository*, Technical Report cs.CC/0406006, June 2004.

[132] L. Henkin. The Completeness of the First-Order Functional Calculus. *Journal of Symbolic Logic*, **14**:159–166, 1949.

[133] J. Herbrand. Sur la Théorie de la Démonstration. *Comptes Rendus des Séances de la Sociéte des Sciences et des Lettres de Varsovie*, Classe III **24**:12–56, 1931.

[134] C. Herlinus, and C. Dasypodius. Analyseis Geometricae Sex Liborum Euclidis. J. Rihel, Strausbourg, 1566.

[135] E.A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, **24**(4):397–420, 2000.

[136] E.A. Hirsch. A new algorithm for MAX-2-SAT. In *Proceedings of 17th International Symposium on Theoretical Aspects of Computer Science*, STACS 2000, Lecture Notes in Computer Science, **1770**:65–73, Springer-Verlag, 2000.

[137] J.H. Holland. Adaption in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI, 1975.

[138] Y. Hong, P.A. Beerel, J.R. Burch, and K.L. McMillan. Safe BDD minimization using don't cares. In *Proceedings of the 34th ACM/IEEE Design Automation Conference*, 208–213, IEEE Computer Society Press, 1997,

[139] H.H. Hoos. Stochastic local search - methods, models, applications. Ph.D. Thesis, TU Darmstadt, FB Informatik, Darmstadt, Germany, 1998.

[140] H.H. Hoos. On the run-time behavior of stochastic local search algorithms for SAT. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 661–666, AAAI Press/The MIT Press, Menlo Park, CA, 1999.

[141] H.H. Hoos. A mixture model for the behaviour of SLS algorithms for SAT. In *Proceedings of the 18th National Conference on Artificial Intelligence*, 661–667, AAAI Press/The MIT Press, Menlo Park, CA, 2002.

[142] H.H. Hoos, and T. Stützle. Stochastic Local Search. Elsevier, Amsterdam, 2005.

[143] F. Hutter, D.A.D. Tompkins, and H.H. Hoos. Scaling and probabilistic smoothing: efficient dynamic local search for SAT. *Lecture Notes in Computer Science*, **2470**:233-248, Springer Verlag, Berlin, Germany, 2002.

[144] J. Huang, and A. Darwiche. Toward good elimination orders for symbolic SAT solving. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, 2004.

[145] A. Ishtaiwi, J.R. Thornton, A.S. Anbulagan, and D.N. Pham. (2006). Adaptive clause weight redistribution. In *Proceedings of the 12th International Conference on the Principles and Practice of Constraint Programming*, CP-2006, Nantes, France, 229–243, 2006.

[146] A. Itai, and J. Makowsky. On the complexity of Herbrand's theorem. Working paper 243, Department of Computer Science, Israel Institute of Technology, 1982.

[147] Iwama, K., "CNF satisfiability test by counting and polynomial average time," *SIAM Journal on Computing* **18**:385–391, 1989.

[148] K. Iwama, and S. Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the* 15th *annual ACM-SIAM Symposium on Discrete Algorithms*, 328–328, Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2004.

[149] J. Jacob, A. Mishchenko. Unate decomposition of Boolean functions. In *Proceedings of the 10th International Workshop on Logic and Synthesis*, 2001.

[150] W.S. Jevons. On the mechanical performance of logic inference. *Philosophical Transactions of the Royal Society of London*, **160**:497, 1870.

[151] H.-S. Jin, and F. Somenzi. CirCUs: A hybrid satisfiability solver. *Lecture Notes in Computer Science*, **3542**:211–223, 2005.

[152] D.S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and Systems Sciences*, **9**:256–278, 1974.

[153] T. Jussilla, and A. Biere. Compressing bmc encodings with QBF. In *Proceedings of the 44th International Workshop on Bounded Model Checking*, 27–39, 2006.

[154] A. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail bounds for occupancy and the satisfiability conjecture. *Random Structures and Algorithms*, **7**:59–80, 1995.

[155] A.C. Kaporis, L.M. Kirousis, and E.G. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. In em 10th Annual European Symposium on Algorithms, (Rome, Italy), 2002.

[156] A.C. Kaporis, L.M. Kirousis, and E.G. Lalas. Selecting complementary pairs of literals. *Electronic Notes in Discrete Mathematics*, **16**(1):1-24, 2004.

[157] A.C. Kaporis, L.M. Kirousis, and Y.C. Stamatiou. How to prove conditional randomness using the principle of deferred decisions. Available from http://www.ceid.upatras.gr/faculty/kirousis/kks-pdd02.ps, 2002.

[158] H. Karloff, and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? 38th *Annual Symposium on the Foundations of Computer Science*, 406–415, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[159] R.M. Karp, and M. Sipser. Maximum matchings in sparse random graphs. In 22nd *Annual Symposium on the Foundations of Computer Science*, (Nashville, Tennessee), 364–375, IEEE Computer Society Press, Los Alamitos, CA, 1981.

[160] M. Karpinski, H. Kleine Büning, and P. Schmitt. On the computational complexity of quantified Horn clauses. In *Proceedings of Compute Science Logic (CSL'88)*, Springer, LNCS **329**:129–137, 1988.

[161] M. Karpinski, H. Kleine Büning, and A. Flögel. Subclasses of quantified Boolean formulas. In *Proceedings of Computer Science Logic (CSL'90)*, Springer LNCS **533**:145–155, 1991.

[162] L. M. Kirousis, E. Kranakis, and D. Krizanc. A better upper bound for the unsatisfiability threshold. In *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, **60**, 1996.

[163] L.M. Kirousis, E. Kranakis, D. Krizanc, and Y.C. Stamatiou. Approximating the unsatisfiability threshold of random formulae. *Random Structures and Algorithms*, **12**:253–269, 1998.

[164] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, **220**:671–680, 1983.

[165] S. Kirkpatrick, B. Selman. Critical behavior in the satisfiability of random formulas. *Science*, **264**:1297–1301, 1994.

[166] S.C. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society* , **53**:41–73, 1943.

[167] E. de Klerk, H. van Maaren, and J.P. Warners. Relaxations of the satisfiability problem using semidefinite programming. *Journal of Automated Reasoning*, **24**(1-2):37–65, 2000.

[168] E. de Klerk. *Aspects of Semidefinite Programming*, Volume 65 of Applied Optimization, Kluwer Academic Publishers, Dordrecht, 2002.

[169] E. de Klerk, and H. van Maaren. On semidefinite programming relaxations of $(2+p)$-SAT. *Annals of Mathematics and Artificial Intelligence*, **37**(3):285–305, 2003.

[170] R. Kowalski, and P.J. Hayes. Semantic Trees in Automatic Theorem-Proving. In *Machine Intelligence*, **4**:87-101, Meltzer and Michie (eds.), Edinburgh Uuniversity Press, Edinburgh, Scotland, 1969.

[171] H. Kleine Büning. An Upper Bound for Minimal Resolution Refutations. *Lecture Notes in Computer Science*, **1584**:171–178, 1999.

[172] H. Kleine Büning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, **107**(1-3):83–98, 2000.

[173] H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, **117**(1):12–18, 1995.

[174] H. Kleine Büning, and T. Lettmann. Propositional Logic: Deduction and Algorithms. Cambridge University Press, 1999.

[175] H. Kleine Büning, K. Subramani, X. Zhao. Boolean functions as models for quantified Boolean formulas. *Journal of Automated Reasoning*, **39**(1):49–75, 2007.

[176] D. Knuth. Nested satisfiability. *Acta Informatica*, **28**:1-6, 1990.

[177] M. Krivelevich, and V.H. Vu. Approximating the independence number and the chromatic number in expected polynomial time. Lecture Notes in Computer Science, 1853:13-, 2000.

[178] O. Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, **107**:99-137, 2000.

[179] O. Kullmann. Lean clause-sets: generalizations of minimally unsatisfiable clause-sets. *Discrete Applied Mathematics*, **130**(2):209–249, 2003.

[180] T.G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7:49–58, 1970.

[181] J.B. Lasserre. Optimality conditions and LMI relaxations for 0-1 programs. Technical report, LAASCNRS, Toulouse, France, 2000.

[182] J.B. Lasserre. An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM Journal on Optimization*, **12**(3):756–769, 2002.

[183] M. Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 a programming. *Mathematics of Operations Research*, **28**(3):470–496, 2003.

[184] S. Lavine. Understanding the Infinite. Harvard University Press, Cambridge, 1994.

[185] C.Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Systems Technical Journal*, **38**:985–999, 1959.

[186] H.R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the Association for Computing Machinery*, **25**:134–135, 1978.

[187] C.M. Li, and W. Huang. Diversification and determinism in local search for Satisfiability. In *Lecture Notes in Computer Science*, **3569**:158–172, Springer, New York, 2005.

[188] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, **11**:329–343, 1982.

[189] S. Lin, and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, **21**(2):498–516, 1973.

[190] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, **25**(1):1–7, 1979.

[191] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, **1**(2):166–190, 1991.

[192] H. van Maaren. Elliptic approximations of propositional formulae. *Discrete Applied Mathematics*, **96/97**:223– 244, 1999.

[193] H. vanMaaren. A short note on some tractable classes of the satisfiability problem. *Information and Computation*, **158**(2):125–130, 2000.

[194] H. van Maaren, and L. van Norden. Sums of squares, satisfiability and maximum satisfiability. *Lecture Notes in Computer Science*, **3569**:293–307, Springer, Berlin, 2005.

[195] M. Mahajan, V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, **31**:335–354, 1999.

[196] S. Mahajan, and H. Ramesh. Derandomizing approximation algorithms based on semidefinite programming. *SIAM Journal on Computing*, **28**(5):1641–1663, 1999.

[197] J.N. Martin. Aristotle's natural deduction reconsidered. *History and Philosophy of Logic*, **18**:1–15, 1997.

[198] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 321–326, AAAI Press/The MIT Press, Monlo Park, CA, 1997.

[199] E.I. McCluskey, Jr. Minimization of Boolean functions. *Bell System Technical Journal*, **35**:1417–1444, 1959.

[200] C.B. McDonald, and R.E. Bryant. Computing logic-stage delays using circuit simulation and symbolic elmore analysis. In *Proceedings of the 38th ACM/IEEE Design Automation Conference*, 283–288, IEEE Computer Society Press, 2001.

[201] K.L. McMillan. Symbolic Model Checking: An Approach to the State Explosion Problem. Kluwer Academic Publishers, 1993.

[202] A.R. Meyer, and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the* 13th *Symposium on Switching and Automata Theory*, 125–129, 1972.

[203] A.R. Meyer, and L.J. Stockmeyer. Word problems requiring exponential time. In *Proceedings of the* 5th *Annual Symposium on the Theory of Computing*, 1–9, 1973.

[204] M. Mézard, and Riccardo Zecchina. The random k-satisfiability problem: from an analytic solution to an efficient algorithm. Technical report available from http://arXiv.org 2002.

[205] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, 272–277, IEEE Computer Society Press, 1993.

[206] S. Minato. Fast factorization method for implicit cube cover representation. *IEEE Transactions on Computer Aided Design*, **15**(4):377–384, 1996.

[207] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the 8th National Conference on Artificial*

Intelligence, 17–24. AAAI Press/The MIT Press, Menlo Park, CA, USA, 1990.

[208] D. Mitchell, B. Selman, H. Levesque. Hard and easy distributions for SAT problems. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence*, 459–465, 1992.

[209] M. Mitzenmacher. Tight thresholds for the pure literal rule. DEC/SRC Technical Note 1997-011, June 1997.

[210] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic "phase transitions." *Nature*, **400**:133–137, 1999.

[211] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. $2+p$-SAT: Relation of typical-case complexity to the nature of the phase transition. *Random Structures and Algorithms*, **15**(3-4):414–435, 1999.

[212] B. Monien, and E. Speckenmeyer. 3-Satisfiability is testable in $O(1.62^r)$ steps. Reihe Informatik, Bericht Nr. 3 (1979), Universität-GH Paderborn

[213] B. Monien, and E. Speckenmeyer. Solving Satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, **10**:287–295, 1985.

[214] D. Motter, and I. Markov. A compressed breadth-first search for satisfiability. *Lecture Notes in Computer Science*, **2409**:29–42, 2002.

[215] A. Newell, J. Shaw, and H. Simon. Empirical explorations with the logic theory machine. In *Proceedings of the Western Joint Computer Conference*, **15**:218–239, 1957.

[216] R. Niedermeier. Some prospects for efficient fixed parameter algorithms. In *Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics* (SOFSEM'98), **1521**:168–185, Springer, 1998.

[217] R. Niedermeier, P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, **36**:63–88, 2000.

[218] V.Y. Pan, and Z.Q. Chen. The complexity of the matrix eigenproblem. *31st ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1999.

[219] G. Pan, and M. Vardi. Search vs. symbolic techniques in satisfiability solving. *Lecture Notes in Computer Science*, **3542**:235–250, 2005.

[220] P.A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, **96**(2, Ser. B):293–320, 2003.

[221] C.I. Lewis. A Survey of Symbolic Logic. Dover, New York, 1960.

[222] M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, **42**(3):969–984, 1993.

[223] L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, **3**:1–22, 1977.

[224] F. Turquette, M. Turquette, and A. Turquette. Peirce's Triadic Logic. *Transactions of the Charles S. Peirce Society*, **11**:71–85, 1966.

[225] J.M. Plotkin, J.W. Rosenthal, and J. Franco. Correction to probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, **17**:295–299, 1987.

[226] C.H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, 163–169, IEEE Computer Society Press, Los Alamitos, CA, USA, 1991.

[227] R. Paturi, P. Pudlak, and F. Zane. Satisfiability coding lemma. In *Proceedings of the* 38th *annual IEEE Symposium on Foundations of Computer Science*, 566–574, IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.

[228] R. Paturi, P. Pudlak, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the* 39th *annual IEEE Symposium on Foundations of Computer Science*, 628–637, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998.

[229] D. Prawitz. An improved proof procedure. *Theoria*, **26**(2):102–139, 1960.

[230] P.W. Purdom and G.N. Haven. Probe order backtracking. *SIAM Journal on Computing*, **26**:456–483, 1997.

[231] P.W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, **21**:117–133, 1983.

[232] P.W. Purdom, and C.A. Brown. The pure literal rule and polynomial average time. *SIAM Journal on Computing*, **14**:943–953, 1985.

[233] W.V.O. Quine. A Way To Simplify Truth Functions. *American Mathematical Monthly*, **62**:627–631, 1955.

[234] W.V.O. Quine. On cores and prime implicants of truth functions. *American Mathematics Monthly*, **66**:755–760, 1959.

[235] J.A. Robinson. Theorem-proving on the computer. *Journal of the ACM*, **10**:163–174, 1963.

[236] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, **12**:23–41, 1965.

[237] J.A. Robinson. The generalized resolution principle. In *Machine Intelligence*, **3**:77–94, Dale and Michie (eds.), American Elsevier, New York, 1968.

[238] J.W. Rosenthal, J.W. Plotkin, and J. Franco. The probability of pure literals. *Journal of Logic and Computation*, **9**:501–513, 1999.

[239] B. Russell. Principles of Mathematics, Section 212. Norton, New York, 1902.

[240] E.W. Samson, B.E. Mills. Circuit minimization: algebra and algorithms for new Boolean canonical expressions. AFCRC Technical Report 54-21, 1954.

[241] W.S. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and Systems Sciences*, **4**:177–192, 1970.

[242] T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the* 10^{th} *ACM Symposium on the Theory of Computing*, 1–9, 1973.

[243] H.-P. Schwefel. Numerical Optimization of Computer Models. John Wiley & Sons, Chichester, UK, 1981.

[244] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the* 40^{th} *Annual IEEE Symposium on Foundations of Computer Science*, 410-414, IEEE Computer Society Press, Los Alamitos, CA, USA, 1999.

[245] U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, **32**:615-623, 2002.

[246] I. Schiermeyer. Pure literal look ahead: a 3-Satisfiability algorithm. In *Proceedings of the First Workshop on the Satisfiability Problem*, J. Franco, G. Gallo, H. Kleine Büning, E. Speckenmeyer, C. Spera, eds., Report No.96-230, Reihe: Angewandte Mathematik und Informatik, Universität zu Köln, 127–136, 1996.

[247] R. Schuler, U. Schöning, and O. Watanabe. An improved randomized algorithm for 3-SAT. Technical Report TR-C146, Deptartment of Mathematics and Computer Sciences, Tokyo Institute of Technology, Japan, 2001.

[248] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence*, 440–446, AAAI Press/The MIT Press, Menlo Park, CA, 1992.

[249] B. Selman, and H. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 290–295, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[250] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 337–343, AAAI Press/The MIT Press, Menlo Park, CA, 1994.

[251] J.S. Schlipf, F. Annexstein, J. Franco, and R. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, 54:133–137, 1995.

[252] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, **54**(1): 40–44, 2005.

[253] D. Schuurmans, F. Southey, and R.C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 334–341, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001.

[254] M.G. Scutella. A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability. *Journal of Logic Programming*, **8**:265–273, 1990.

[255] Y. Shang, and B.W. Wah. A discrete Lagrangian-based globalsearch method for solving satisfiability problems. *Journal of Global Optimization*, **12**(1):61–100, 1998.

[256] C.E. Shannon. A symbolic analysis of relay and switching circuits. Masters Thesis, Massachusetts Institute of Technology, 1940. Available from http://hdl.handle.net/1721.1/11173.

[257] H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, **3**(3):411–430, 1990.

[258] M.H. Stone. The Theory of Representation for Boolean Algebras. *TAMS*, **40**:37–111, 1936.

[259] R.P. Swaminathan, and D.K. Wagner. The arborescence–realization problem. *Discrete Applied Mathematics*, **59**:267–283, 1995.

[260] S. Szeider. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. In *Proc. 9th COCOON: Annual International Conference on Computing and Combinatorics*, LNCS **2697**:548–558. Springer, 2003.

[261] A. Tarski. The Concept of Truth in Formalized Languages. In *Logic Semantics, Metamathematics*, A. Tarski, ed., Clarendon Press, Oxford, 1956.

[262] A. Tarski. Truth and Proof. *Philosophy and Phenomenological Research*, **4**:341–75, 1944.

[263] A. Tarski. Contributions to the Theory of Models. *Indagationes Mathematicae*, **16**:572–88, 1954.

[264] C.A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, **8**:85–89, 1984.

[265] K. Truemper. Monotone Decomposition of Matrices. Technical Report UTDCS-1-94, University of Texas at Dallas. 1994.

[266] K. Truemper. Effective Logic Computation. Wiley, 1998.

[267] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, 115–125, 1968.

[268] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, Series 2, **42**:230-265, 1936.

[269] T.E. Uribe, and M.E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. *Lecture Notes in Computer Science*, **845**:34–49, 1994.

[270] J.F. Groote, and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete Applied Mathematics*, **130**(2):157–171, 2003.

[271] A. Urquhart. Hard examples for resolution. *Journal of the Association for Computing Machinery*, 34:209–219, 1987.

[272] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Dublin Philosophical Magazine and Journal of Science*, **9**(59):1–18, 1880.

[273] B.W. Wah, and Y. Shang. Discrete Lagragian-based search for solving MAX-SAT problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 378–383, 1998.

[274] R. Wallace, E. Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In *Cliques, Coloring and Satisfiability*, D. Johnson and M. Trick (eds.), **26**:587–615, 1996.

[275] I. Wegener. BDDs - design, analysis, complexity, and applications. *Discrete Applied Mathematics*, **138**(1-2):229–251, 2004.

[276] L. Wittgenstein. Tractatus Logico-Philosophicus. Reprinted by K. Paul, Trench, Trubner, London, 1933.

[277] H. Wolkowicz, R. Saigal, and L. Vandenberghe (eds.), *Handbook of Semidefinite Programming.* Kluwer Academic Publishers, Boston, MA, 2000.

[278] N.C. Wormald. Differential equations for random processes and random graphs. *Annals of Applied Probability*, **5**:1217–1235, 1995.

[279] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, **3**:22–33, 1977.

[280] Z. Wu, and B.W. Wah. Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 673–678, AAAI Press/The MIT Press, Menlo Park, CA, USA, 1999.

[281] Z. Wu, and B.W. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 310-315, AAAI Press/The MIT Press, Menlo Park, CA, USA, 2000.

[282] H. Xu, R.A. Rutenbar, K. Sakallah. sub-SAT: A formulation for related boolean satisfiability with applications in routing. In *Proceedings of the 2002 ACM International Symposium on Physical Design*, 182–187, 2002.

[283] H. Zhang, H. Shen, F. Manyà. Exact algorithms for MAX-SAT. In *Proceedings of the International Workshop on First-order Theorem Proving* (FTP 2003). Available from:
`http://www.elsevier.com/gej-ng/31/29/23/135/23/show/Products/notes/index.htt.`

[284] X. Zhao, W. Zhang. An efficient algorithm for maximum boolean satisfiability based on unit propagation, linear programming, and dynamic weighting. Preprint, Department of Computer Science, Washington University, 2004.

Appendix A

Glossary

algebraic structure: (2)

A set of underlying elements and operations on them obeying defining axioms. See *structure* below for details and *Boolean algebra* below for the definition of Boolean algebra as a structure.

Boolean algebra: (9)

An algebraic structure whose constant set is $\{0, 1\}$, whose operations $\vee$, $\wedge$, $\neg$ obey axioms of distributivity, commutativity, associativity, absorbtion, and where $x \vee \neg x = 1$, $x \wedge \neg x = 0$, $1 \wedge x = x$, and $0 \vee x = x$.

epistemic modifier: (7)

A grammatical element which is neither an argument nor a predicate, but which modifies another element or phrase (e.g. a predicate) to indicate degree of truth. For example, A knows B, A believes B

formal language: (8,13)

An unordered pair $\{\mathbf{A}, \mathbf{F}\}$ where $\mathbf{F}$ is a set of finite-length sequences of elements taken from a set $\mathbf{A}$ of symbols. A first-order logic language $\mathcal{L}$ has the following sets of non-logical symbols in $\mathbf{A}$:

1. $\mathcal{C}$, the set of constant symbols of $\mathcal{L}$.

2. $\mathcal{P}$, the set of predicate symbols of $\mathcal{L}$. For each $P \in \mathcal{P}$, $\alpha(P)$ denotes the arity of P. The symbols in $\mathcal{P}$ are also called *relation* symbols of $\mathcal{L}$.

3. $\mathcal{F}$, the set of function symbols of $\mathcal{L}$. For each $f \in \mathcal{F}$, $\alpha(f)$ denotes the arity of f. The symbols in $\mathcal{F}$ are also called *operation* symbols of $\mathcal{L}$.

4. $\{\forall,\exists\}$, the universal and existential quantifier symbols of $\mathcal{L}$.

homomorphism: (80)

As used here, given structures $\mathcal{A}$ and $\mathcal{B}$ for a common language $\mathcal{L}$, the mapping $h : \mathcal{A} \rightarrow \mathcal{B}$ is a homomorphism if

1. For each constant $c \in \mathcal{C}$, $h(c^{\mathcal{A}}) = c^{\mathcal{B}}$.
2. For each predicate symbol $P \in \mathcal{P}$, if $\alpha(P) = n$, then $P^{\mathcal{B}} = \{h(a_1), ..., h(a_n) | a_1, ..., a_n \in P^{\mathcal{A}}\}$.
3. For each function symbol $f \in \mathcal{F}$, if $\alpha(f) = n$, then for any $a_1, ..., a_n \in A, h(f^{\mathcal{A}}(a_1, ..., a_n)) = f^{\mathcal{B}}(h(a_1), ..., h(a_n))$.

Thus, a homomorphism h between the Boolean algebras $\mathcal{A}$ and $\mathcal{B}$ is a function such that for all $a, b \in \mathbf{A}$:

$$\begin{aligned}
&h(a \vee b) = h(a) \vee h(b)\\
&h(a \wedge b) = h(a) \wedge h(b)\\
&h(0) = 0\\
&h(1) = 1
\end{aligned}$$

interpretation: (2)

A homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$ from the syntax of a formal language viewed as an algebraic structure to semantic values in some other structure. The structure $\mathcal{B}$ is said to be a *model*. In first-order logic the underlying set of $\mathcal{A}$ includes symbols $\mathcal{D}$ representing values that may be assigned to variables. The interpretation supplies mappings for each of the function and predicate symbols of $\mathcal{A}$ to arity consistent functions of domain $\mathcal{D}$ for the former and domain $\{0, 1\}$ for the latter.

modal modifier: (7)

A grammatical element which is neither an argument nor a predicate, but which modifies another element or phrase (e.g. a predicate) to indicate the attitude of the speaker with respect to the truth-value of the proposition expressed. Examples are must, should, maybe, possibly, can.

model: (1)

A model of a theory T consists of a structure in which all sentences of T are true.

prenex form: (20,49)

A formula of the form $Q_1x_1 \dots Q_nx_n\phi$, where $Q_i \in \{\exists, \forall\}, x_1, \dots, x_n$ are variables, and ϕ is a propositional formula called the *matrix*.

structure: (2,80)

For the purposes of this handbook, a structure for a language $\mathcal{L}$ is a pair $\mathcal{A} = \langle \mathbf{D}, R \rangle$, where $\mathbf{D}$ is a non-empty set of entities (objects, concepts, etc.) called the *domain* of $\mathcal{L}$, and R is a function, called the *interpretation*, that assigns to each constant of $\mathcal{L}$ an entity in $\mathbf{D}$, to each predicate symbol of $\mathcal{L}$ a relation among entities in $\mathbf{D}$, and to each function symbol of $\mathcal{L}$ a function among entities in $\mathbf{D}$. A sentence p of $\mathcal{L}$ is said to be true if the entities chosen as the interpretations of the sentence's terms and functors *stand* to the relations chosen as the interpretation of the sentence's predicates.

Alternatively, $\mathcal{A} = \langle \mathbf{D}, \{P^{\mathcal{A}}\}_{P \in \mathcal{P}}, \{f^{\mathcal{A}}\}_{f \in \mathcal{F}}, \{c^{\mathcal{A}}\}_{c \in \mathcal{C}} \rangle$ where $\mathcal{P}$, $\mathcal{F}$, and $\mathcal{C}$ are the constant, predicate, and function symbols for $\mathcal{L}$. The set $P^{\mathcal{A}}$ is a set of predicate symbols, $f^{\mathcal{A}}$ is a set of function symbols, and $c^{\mathcal{A}}$ is a set of constant symbols, such that

1. For each $P \in \mathcal{P}, P^{\mathcal{A}} \subseteq \mathbf{D}^{\alpha(P)}$,
2. For each $f \in \mathcal{F}, f^{\mathcal{A}} : \mathbf{D}^{\alpha(f)} \to \mathbf{D}$.
3. For each $c \in \mathcal{C}, c^{\mathcal{A}} \in \mathbf{D}$,

where α is the arity function.

theory: (1)

A set of sentences in a language L. A theory is said to be closed if the set of sentences is closed under the usual rules of inference.