A Logical Revolution

Moshe Y. Vardi

Rice University

Logic in Computer Science

During the past fifty years there has been extensive, continuous, and growing interaction between logic and computer science. In many respects, logic provides computer science with both a unifying foundational framework and a tool for modeling computational systems. In fact, logic has been called "the calculus of computer science". The argument is that logic plays a fundamental role in computer science, similar to that played by calculus in the physical sciences and traditional engineering disciplines. Indeed, logic plays an important role in areas of computer science as disparate as machine architecture, computer-aided design, programming languages, databases, artificial intelligence, algorithms, and computability and complexity.

Why on Earth?

Basic Question: What on earth does an obscure, old intellectual discipline have to do with the youngest intellectual discipline?

M. Davis (1988): *Influences of Mathematical Logic on Computer Science*: "When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization."

Question: Why on earth?

The Fundamental Method of Physical Sciences

The basic method of physical science:

- Construct a mathematical model for physical reality.
- Analyze mathematical model.
- Compare analysis results with physical reality.

Basic Question: What is a mathematical model?

What is The Language of Mathematics?

Friedrich Ludwig Gottlob Frege, *Begriffsschrift*, 1879: a universal mathematical language – *first-order logic* – *logicism*

- Objects, e.g., 2
- Predicates (relationships), e.g., $2 < 3$
- Operations (functions), e.g., $2 + 3$
- Boolean operations, e.g., "and" ($\wedge$), "or" ($\vee$), "implies" ($\rightarrow$)
- Quantifiers, e.g., "for all" ($\forall$), "exists" ($\exists$)

Example: Aristotle's Syllogisms

- "All men are mortal"
- "For all x, if x is a man, then x is mortal"
- $(\forall x)(Man(x) \rightarrow Mortal(x))$

First-Order Logic

A formalism for specifying properties of mathematical structures, such as *graphs*, *partial orders*, *groups*, *rings*, *fields*, . . .

Mathematical Structure: [Tarski, 1933]

$$\mathbf{A} = (D, R_1, \ldots, R_k, f_1, \ldots, f_l),$$

- D is a non-empty set – *universe*
- R_i is an m-ary *relation* on A, for some m (that is, $R_i \subseteq A^m$)
- f_j is an n-ary *function* on A, for some n (that is, $f_i : A^n \to D$)

Examples

- **Graph**: $\mathbf{G} = (V, E)$

– V: nodes

– $E \subseteq V^2$: edges

– $(\forall x)(\exists y)E(x, y)$

- **Group**: $\mathbf{G} = (V, \cdot)$

– V: elements

– $\cdot : V^2 \to V$: product

– $(\forall x)(\forall y)x \cdot y = y \cdot x$

First-Order Logic on Graphs

Syntax:

- First-order variables: x, y, z, . . . (range over nodes)
- Atomic formulas: $E(x, y)$, $x = y$
- Formulas: Atomic Formulas + Boolean Connectives + First-Order Quantifiers $\exists x$, $\forall x$, . . .

Examples:

- "node x has at least two distinct neighbors"

$$(\exists y)(\exists z)(\neg(y = z) \wedge E(x, y) \wedge E(x, z))$$

Concept: x is *free* in the above formula, which expresses a property of nodes. Truth definition requires a *binding* α. $G \models_\alpha \varphi(x)$

- "each node has at least two distinct neighbors"

$$(\forall x)(\exists y)(\exists z)(\neg(y = z) \wedge E(x, y) \wedge E(x, z))$$

Concept: The above is a *sentence*, that is, a formula with no free variables; it expresses a property of graphs. $G \models \varphi$

Russell's Letter

A letter from Russell to Frege, June 16, 1902:

"I find myself in agreement with you in all essentials . . . I find in your work discussions, distinctions, and definitions that one seeks in vain in the work of other logicians . . . There is just one point where I have encountered a difficulty."

Appendix to Frege's 1903 volume:

"There is nothing worse that can happen to a scientist than to have the foundation collapse just as the work is finished. I have been placed in this position by a letter from Mr. Bertrand Russell."

Hilbert's Program

Hilbert's Program (1922-1930): Formalize mathematics and establish that:

- Mathematics is *consistent*: a mathematical statement and its negation cannot ever both be proved.
- Mathematics is *complete*: all true mathematical statements can be proved.
- Mathematics is *decidable*: there is a mechanical way to determine whether a given mathematical statement is true or false.

The Demise of Hilbert's Program

- Gödel (1930-3):
  - *Incompleteness* of ordinary arithmetic – There is no systematic way of resolving all mathematical questions.
  - *Impossibility* of proving consistency of mathematics

  Gödel (1930): "This sentence is not provable."

- Church and Turing (1936):

  Unsolvability of first-order logic:

  - The set of *valid* first-order sentences is *not computable*.

Entscheidungsproblem

Entscheidungsproblem (**The Decision Problem**) [Hilbert-Ackermann, 1928]: Decide if a given first-order sentence is *valid* (dually, *satisfiable*).

Church-Turing Theorem, 1936: The Decision Problem is unsolvable.

Turing, 1936:

- Defined computability in terms of *Turing machines* (TMs)
- Proved that the *termination problem* for TMs is unsolvable ("this machine terminates iff it does not terminate")
- Reduced termination to Entscheidungsproblem.

Mathematical Logic – 1936

Logic as Foundations of Mathematics:

- Incomplete (example: *Continuum Hypothesis*)
- Cannot prove its own consistency
- Unsolvable decision problem

Can we get some positive results?

- Focus on special cases!

The Fragment-Classification Project

Idea: Identify decidable quantificational fragments of first-order logic – (1915-1983)

- *Bernays-Schönfinkel Class* ($\exists^*\forall^*$)
- *Ackermann Class* ($\exists^*\forall\exists^*$)
- Gödel Class (w/o $=$) ($\exists^*\forall\forall\exists^*$)

Outcome: Very weak classes! What good is first-order logic?

Logic of Integer Addition

Integer Addition:
- Domain: **N** (natural numbers)
- Dyadic predicate: $\leq$
- Addition function: $+$

- $y = 2x$: $y = x + x$
- $x = 0$: $(\forall y)(x \leq y)$
- $x = 1$: $x \neq 0 \wedge (\forall y)(y = 0 \vee x \leq y)$
- $y = x + 1$: $(\exists z)(z = 1 \wedge y = x + z)$

Bottom Line: Theory of Integer Addition can express Integer Programming (integer inequalities) and much more.

Presburger Arithmetics

Mojżesz Presburger, 1929:

- *Sound and complete* axiomatization of integer addition
- *Decidability*: There exists an algorithm that decides whether a given first-order sentence in integer-addition theory is true or false.
  - Decidability is shown using quantifier elimination, supplemented by reasoning about arithmetical congruences.
  - Decidability can also be shown using automata-theoretic techniques.

Complexity of Presburger Arithmetics

Complexity Bounds:

- Oppen, 1978: $TIME(2^{2^{2^{poly}}})$ upper bound
- Fischer&Rabin, 1974: $TIME(2^{2^{lin}})$ lower bound

Rabin, 1974: "Theoretical Impediments to Artificial Intelligence": "the complexity results point to a need for a careful examination of the goals and methods in AI".

Monadic Logic

Monadic Class: First-order logic with *monadic predicates* – captures *syllogisms*.

- $(\forall x)P(x), (\forall x)(P(x) \rightarrow Q(x)) \models (\forall x)Q(x)$

[Löwenheim, 1915]: The Monadic Class is decidable.

- *Proof*: Bounded-model property – if a sentence is satisfiable, it is satisfiable in a structure of bounded size.
- *Proof technique*: quantifier elimination.

Finite Words – Deterministic Finite Automata, I

$A = (\Sigma, S, s_0, \rho, F)$

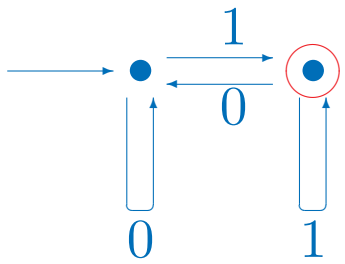
- *Alphabet*: Σ
- *States*: S
- *Initial state*: $s_0 \in S$
- *Transition function*:
  $\rho : S \times \Sigma \to S$
- *Accepting states*: $F \subseteq S$

Finite Words – Deterministic Finite Automata, II

Input word: $a_0, a_1, \ldots, a_{n-1}$
Run: $s_0, s_1, \ldots, s_n$
- $s_{i+1} = \rho(s_i, a_i)$ for $i \geq 0$

Acceptance: $s_n \in F$
Recognition: $L(A)$ – words accepted by A.

Example:



– ends with 1's

Fact: DFAs define the class *Reg* of regular languages.

Logic of Finite Words

View finite word $w = a_0, \ldots, a_{n-1}$ over alphabet Σ as a mathematical structure:

- Domain: $0, \ldots, n-1$
- Dyadic predicate: $<$
- Monadic predicates: $\{P_a : a \in \Sigma\}$

Monadic Second-Order Logic (MSO):

- Monadic atomic formulas: $P_a(x)$ $(a \in \Sigma)$
- Dyadic atomic formulas: $x < y$
- Set quantifiers: $\exists P, \forall P$

Example: $(\exists x)((\forall y)(\neg(x < y)) \wedge P_1(x))$ – last letter is 1.

Automata and Logic

Theorem [Büchi, Elgot, Trakhtenbrot, 1957-8 (independently)]: MSO $\equiv$ DFA
- Both MSO and DFA define the class Reg.

Proof: Effective

- From DFA to MSO ($A \mapsto \varphi_A$)
- From MSO to DFA ($\varphi \mapsto A_\varphi$)

DFA Nonemptiness

Nonemptiness: $L(A) \neq \emptyset$

Nonemptiness Problem: Decide if given A is nonempty.

Directed Graph $G_A = (S, E)$ of DFA $A = (\Sigma, S, s_0, \rho, F)$:

- *Nodes*: S
- Edges: $E = \{(s,t) : t = \rho(s,a) \text{ for some } a \in \Sigma\}$

Lemma: A is nonempty iff there is a path in G_A from s_0 to F.

- Decidable in time linear in size of A, using *breadth-first search* or *depth-first search*.

MSO Satisfiability – Finite Words

Satisfiability: $models(\psi) \neq \emptyset$

Satisfiability Problem: Decide if given ψ is satisfiable.

Lemma: ψ is satisfiable iff A_ψ is nonnempty.

Corollary: MSO satisfiability is decidable.

- Translate ψ to A_ψ.
- Check nonemptiness of A_ψ.

Complexity Barrier

Computational Complexity:

- *Naive Upper Bound*: Nonelementary Growth

$$2^{\cdot^{\cdot^{\cdot^{2^n}}}}$$

(tower of height $O(n)$)

- *Lower Bound* [Stockmeyer, 1974]: Satisfiability of FO over finite words is nonelementary (no bounded-height tower).

Nonelementary Growth

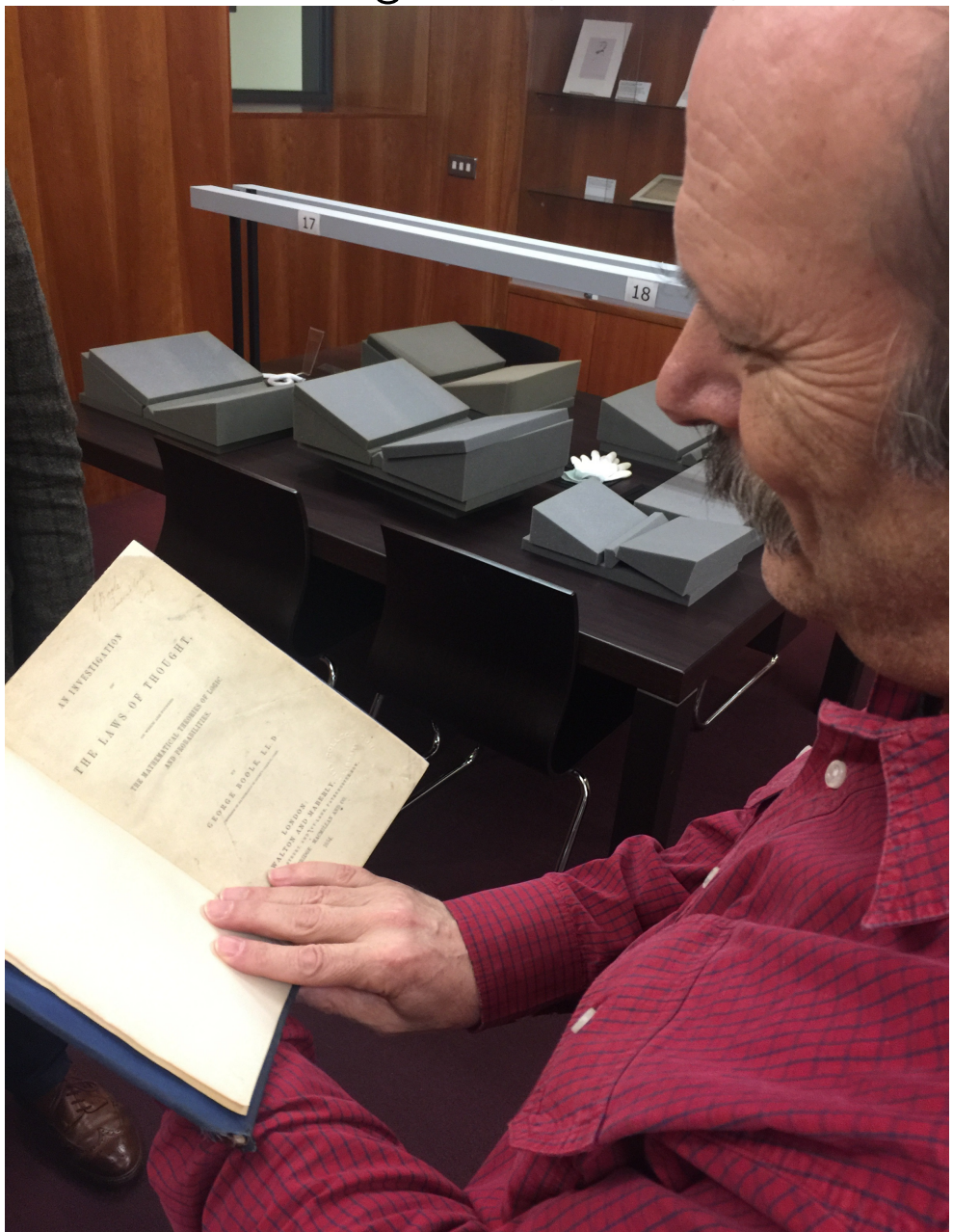
- $2^2 = 4$
- $2^{2^3} = 256$
- $2^{2^{2^4}} = 4,294,967,296$
- $2^{2^{2^{2^5}}} \approx 10^{10^9}$

Boole's Symbolic Logic

Boole's insight: Aristotle's syllogisms are about *classes* of objects, which can be treated *algebraically*.

"If an adjective, as 'good', is employed as a term of description, let us represent by a letter, as y, all things to which the description 'good' is applicable, i.e., 'all good things', or the class of 'good things'. Let it further be agreed that by the combination xy shall be represented that class of things to which the name or description represented by x and y are simultaneously applicable. Thus, if x alone stands for 'white' things and y for 'sheep', let xy stand for 'white sheep'.

Vardi at Univ. College Cork, Ireland, March 2017



Boolean Satisfiability

Boolean Satisfiability (SAT); Given a Boolean expression, using "and" ($\wedge$) "or", ($\vee$) and "not" ($\neg$), *is there a satisfying solution* (an assignment of 0's and 1's to the variables that makes the expression equal 1)?

Example:

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_1 \vee x_4)$$

Solution: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$

Complexity of Boolean Reasoning

History:

• William Stanley Jevons, 1835-1882: "I have given much attention, therefore, to lessening both the manual and mental labour of the process, and I shall describe several devices which may be adopted for saving trouble and risk of mistake."

• Ernst Schröder, 1841-1902: "Getting a handle on the consequences of any premises, or at least the fastest method for obtaining these consequences, seems to me to be one of the noblest, if not the ultimate goal of mathematics and logic."

• 1950-60s: *"Perebor"*: Futile effort to show hardness of search problems.

• Cook, 1971, Levin, 1973: Boolean Satisfiability is NP-complete.

NP-Completeness

Complexity Class NP: problems where solutions can be *checked* in polynomial time.

- Factoring numbers
- Hamiltonian Cycle

Significance: *Tens of thousands* of optimization problems are in NP!!!

- CAD, flight scheduling, chip layout, protein folding, ...

NP-complete: *Hardest* problems in NP.

- If *one* NP-complete problem can be solved in polynomial time, then *all* NP-complete problems can be solved in polynomial time.

Is $P = NP$?

S. Aaronson, MIT: "If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in 'creative leaps,' no fundamental gap between solving a problem and recognizing the solution once it is found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss."

Common Belief: Boolean SAT *cannot* be solved in polynomial time.

Program Verification

The Dream – Hoare, 1969: "When the correctness of a program, its compiler, and the hardware of the computer have all been established with mathematical certainty, it will be possible to place great reliance on the results of the program."

The Nightmare – De Millo, Lipton, and Perlis, 1979: "We believe that ... program verification is bound to fail. We can't see how it's going to be able to affect anyone's confidence about programs."

- **Basic Argument**: Acceptance of proof is a *social process*!

Logic in Computer Science: c. 1980

Status: Logic in CS is not too useful!

- Unsolvable termination problem.
- First-order logic is undecidable.
- The decidable fragments are either too weak or too intractable.
- Even Boolean logic is intractable.
- Program verification is hopeless.

Post 1980: From Irrelevance to Relevance

A Logical Revolution:

- Relational databases
- Boolean reasoning
- Model checking
- Termination checking
- ...

From Model Theory to Relational Databases

- A sentence ψ is either true or false on a given graph $\mathbf{G}$. In particular, sentences specify classes of graphs: $\text{models}(\psi) = \{\mathbf{G} : \mathbf{G} \models \psi\}$

  Model Theory: Logic provides a metatheory for mathematical modeling.

- E.F. Codd, 1970: Formulas $\varphi(x_1, \ldots, x_k)$ define *queries*:
  $\varphi(\mathbf{G}) = \{\langle \alpha(x_1), \ldots, \alpha(x_k) \rangle : \mathbf{G} \models_\alpha \varphi(x_1, \ldots, x_k)\}$

  Example: $(\exists y)(\exists z)(\neg(y = z) \wedge E(x, y) \wedge E(x, z))$ – "List nodes that have at least two distinct neighbors"

  Turing Awards: Codd, 1981, Stonebraker, 2014
  Relational Databases: \$30B+ industry

Relational Databases

Codd's Two Fundamental Ideas:

- *Tables are relations*: a *row* in a table is just a *tuple* in a relation; order of rows/tuples does not matter!

- *Formulas are queries*: they specified the *what* rather then the *how* – declarative programming!

Reduction to Practice:

- *System R*, IBM, 1976
- Ingress, UC Berkeley, 1976
- *Oracle*, Relational Software, 1979
- DB2, IBM, 1982

Algorithmic Problems in First-Order Logic

Satisfiability Problem: Given a first-order formula ψ, *is there* a graph $\mathbf{G}$ and binding α, such that $\mathbf{G} \models_\alpha \psi$?

Truth-Evaluation Problem: Given a first-order formula $\varphi(x_1, \ldots, x_k)$, a *finite* graph $\mathbf{G}$, and a binding α, does $\mathbf{G} \models_\alpha \varphi(x_1, \ldots, x_k)$?

Contrast:

- Satisfiability is *undecidable* [Church, Turing 1936]
- Truth evaluation, which is query evaluation, is *decidable*.

Algorithmic Boolean Reasoning: Early History

- Newell, Shaw, and Simon, 1955: "Logic Theorist"
- Davis and Putnam, 1958: "Computational Methods in The Propositional calculus", unpublished report to the NSA
- Davis and Putnam, JACM 1960: "A Computing procedure for quantification theory"
- Davis, Logemman, and Loveland, CACM 1962: "A machine program for theorem proving"

DPLL Method: Propositional Satisfiability Test

- Convert formula to conjunctive normal form (CNF)
- Backtracking search for satisfying truth assignment
- Unit-clause preference

Modern SAT Solving

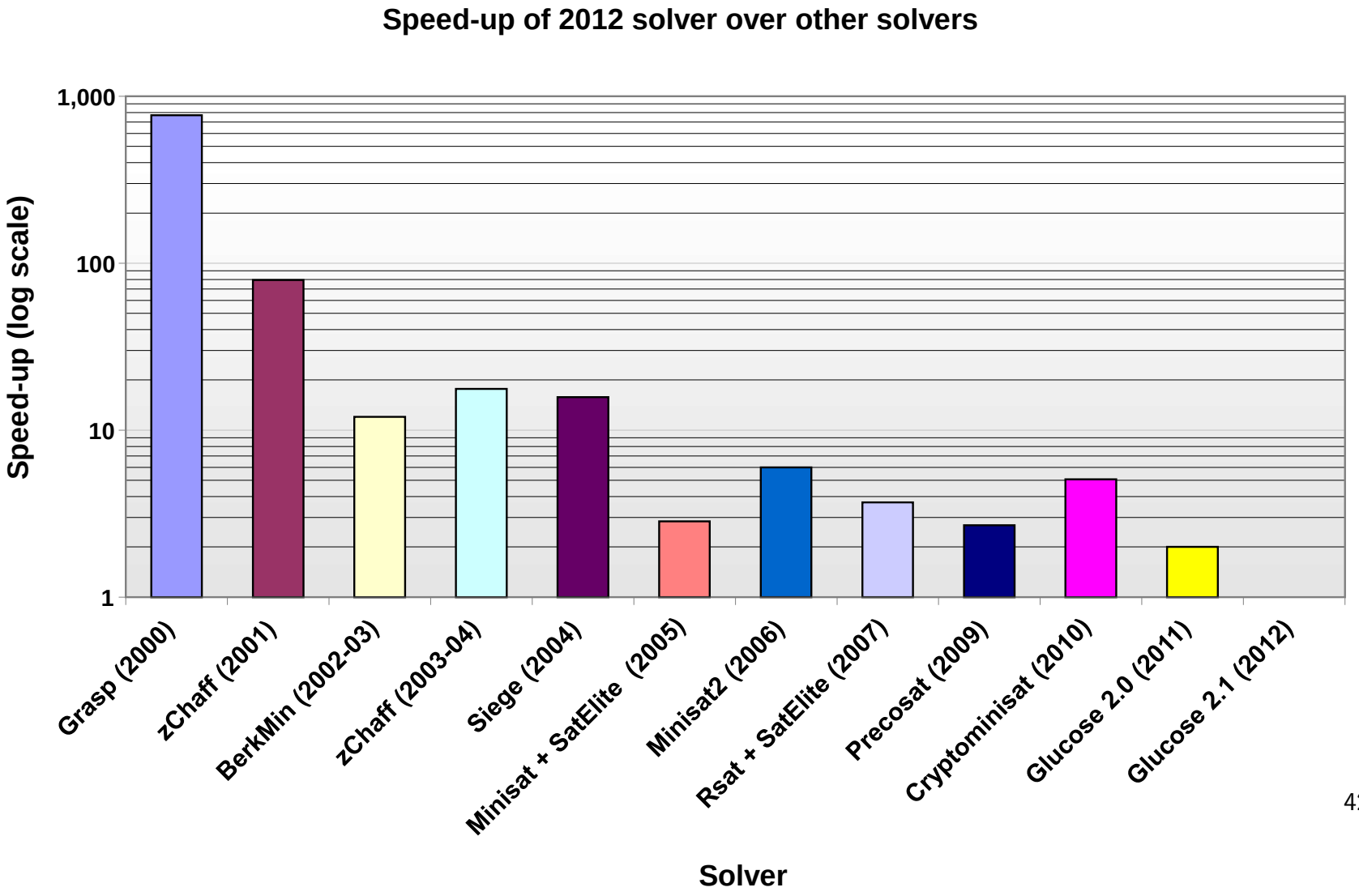
CDCL = conflict-driven clause learning

- Backjumping
- Smart unit-clause preference
- Conflict-driven clause learning
- Smart choice heuristic (brainiac vs speed demon)
- Restarts

Key Tools: GRASP, 1996; Chaff, 2001

Current capacity: *millions* of variables

Some Experience with SAT Solving



Applications of SAT Solving in SW Engineering

Leonardo De Moura+Nikolaj Björner, 2012: applications of Z3 at Microsoft

- Symbolic execution
- Model checking
- Static analysis
- Model-based design
- ...

The Temporal Logic of Programs

Crux: Need to specify ongoing behavior rather than input/output relation!

> **"Temporal logic to the rescue"** [Pnueli, 1977]:
> • Linear temporal logic (LTL) as a logic for the specification of non-terminating programs
> • Model checking via reduction to MSO

But: nonelementary complexity!

Examples

- always not (CS_1 and CS_2): safety
- always (Request implies eventually Grant): liveness
- always (Request implies (Request until Grant)): liveness

Model Checking

"Algorithmic verification" [Clarke & Emerson, 1981, Queille & Sifakis, 1982]: Model checking programs of size m wrt CTL formulas of size n can be done in time mn.

Linear-Time Response [Lichtenstein & Pnueli, 1985]: Model checking programs of size m wrt LTL formulas of size n can be done in time $m2^{O(n)}$ (*tableau heuristics*).

Seemingly:

- *Automata*: Nonelementary
- Tableaux: exponential

Back to Automata

Exponential-Compilation Theorem [V. & Wolper, 1983–1986]: Given an LTL formula φ of size n, one can construct an automaton A_φ of size $2^{O(n)}$ such that a trace σ satisfies φ if and only if σ is accepted by A_φ.

Automata-Theoretic Algorithms:
1. *LTL Satisfiability*:
φ is satisfiable iff $L(A_\varphi) \neq \emptyset$ (PSPACE)
2. *LTL Model Checking*:
$M \models \varphi$ iff $L(M \times A_{\neg\varphi}) = \emptyset$ $(m2^{O(n)})$

Today: Widespread industrial usage

- **Industrial Languages**: PSL, SVA (IEEE standards)

Solving the Unsolvable

B. Cook, A. Podelski, and A. Rybalchenko, 2011: "in contrast to popular belief, proving termination is not always impossible"

- The *Terminator* tool can prove termination or divergence of many Microsoft programs.
- Tool is *not guaranteed* to terminate!
- See also `http://termination-portal.org/`.

Explanation:

- Most real-life programs, if they terminate, do so for rather simple reasons.
- Programmers almost never conceive of very deep and sophisticated reasons for termination.

Logic: from failure to success

Key Lessons:

- Algorithms
- Heuristics
- Experimentation
- Tools and systems

Key Insight: Do not be scared of worst-case complexity.

- It barks, but it does not necessarily bite!

Logic vs. Machine Learning

Daniel Kahneman, *Thinking, Fast and Slow*, 2011:

- **Machine Learning**: fast thinking, e.g., "Is this a stop sign?"
- **Logic**: slow thinking, e.g., "Do you stop at a stop sign?"

Grand Challenge: Combine logic with machine learning!