Informatique et preuve

Une brève histoire du raisonnement automatisé

Charles Pecheur
Université catholique de Louvain

Séminaire *fondements et notions fondamentales* – 12 mars 2012

Replacing Scholars by Programs?

From  to 

?

Paul Erdõs

HAL 9000

Computer Proofs?

- Can “creativity” be “automated”?

Computer Proofs?

- Can “creativity” be “automated”?
- Can **reasoning** be reduced to **computation**?

Computer Proofs?

- Can “creativity” be “automated”?
- Can **reasoning** be reduced to **computation**?

- Intuition: **NO**, reasoning is genuinely human
  - “Computers are **stupid**, they only blindly execute their program”
  - “Computers can **compute** but they cannot really **reason**”

Computer Proofs?

- Can “creativity” be “automated”?
- Can **reasoning** be reduced to **computation**?

- Intuition: **NO**, reasoning is genuinely human
  - “Computers are **stupid**, they only blindly execute their program”
  - “Computers can **compute** but they cannot really **reason**”
- Reality: **YES**, to a large extent : **Automated Reasoning** (**AR**)
  - A well-established field of **Artificial Intelligence** (50+ years)
  - Rich gamut of approaches, books, tools, applications, results

Computer Proofs?

- Can “creativity” be “automated”?
- Can **reasoning** be reduced to **computation**?

- Intuition: **NO**, reasoning is genuinely human
  - “Computers are **stupid**, they only blindly execute their program”
  - “Computers can **compute** but they cannot really **reason**”
- Reality: **YES**, to a large extent : **Automated Reasoning** (**AR**)
  - A well-established field of **Artificial Intelligence** (50+ years)
  - Rich gamut of approaches, books, tools, applications, results
- . . . Reasoning **can** be reduced to computation (to some extent)

Why Do I Care?

- **Who I am**
  - Professor at UCL / SST / EPL (engineering school)
  - Researcher at UCL / SST / ICTEAM / INGI (computer science)

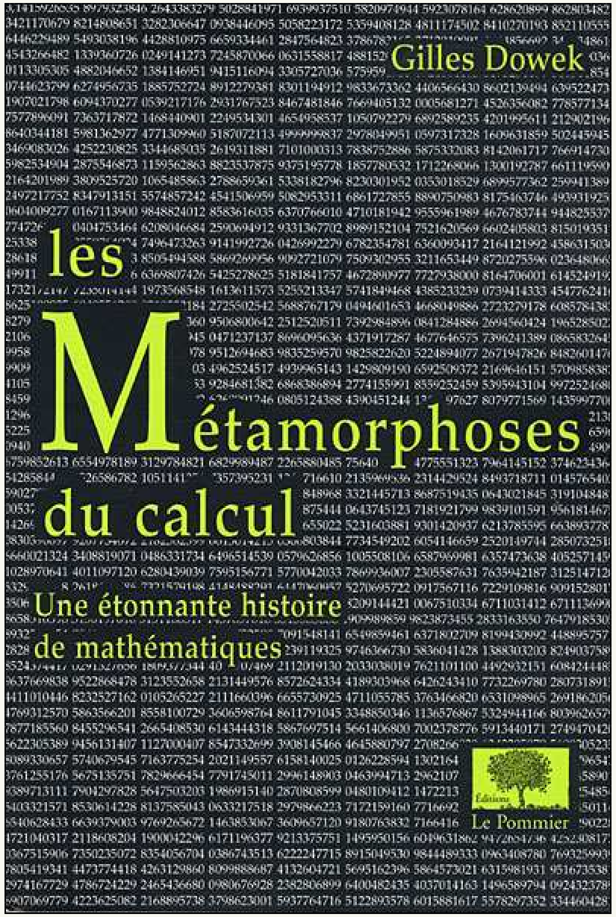
Why Do I Care?

- **Who I am**
  - Professor at UCL / SST / EPL (engineering school)
  - Researcher at UCL / SST / ICTEAM / INGI (computer science)
- **What I study**
  - Verifying computer systems
  - Proving correctness or (more often) finding bugs
  - Model-checking (mostly), solvers (as tools)

Why Do I Care?

- **Who I am**
  - Professor at UCL / SST / EPL (engineering school)
  - Researcher at UCL / SST / ICTEAM / INGI (computer science)
- **What I study**
  - Verifying computer systems
  - Proving correctness or (more often) finding bugs
  - Model-checking (mostly), solvers (as tools)
- **What I teach**
  - Beginner programming (Java), system modelling and analysis,
  - **(automated) program proofs**, **automated reasoning**

Inspiring Reading

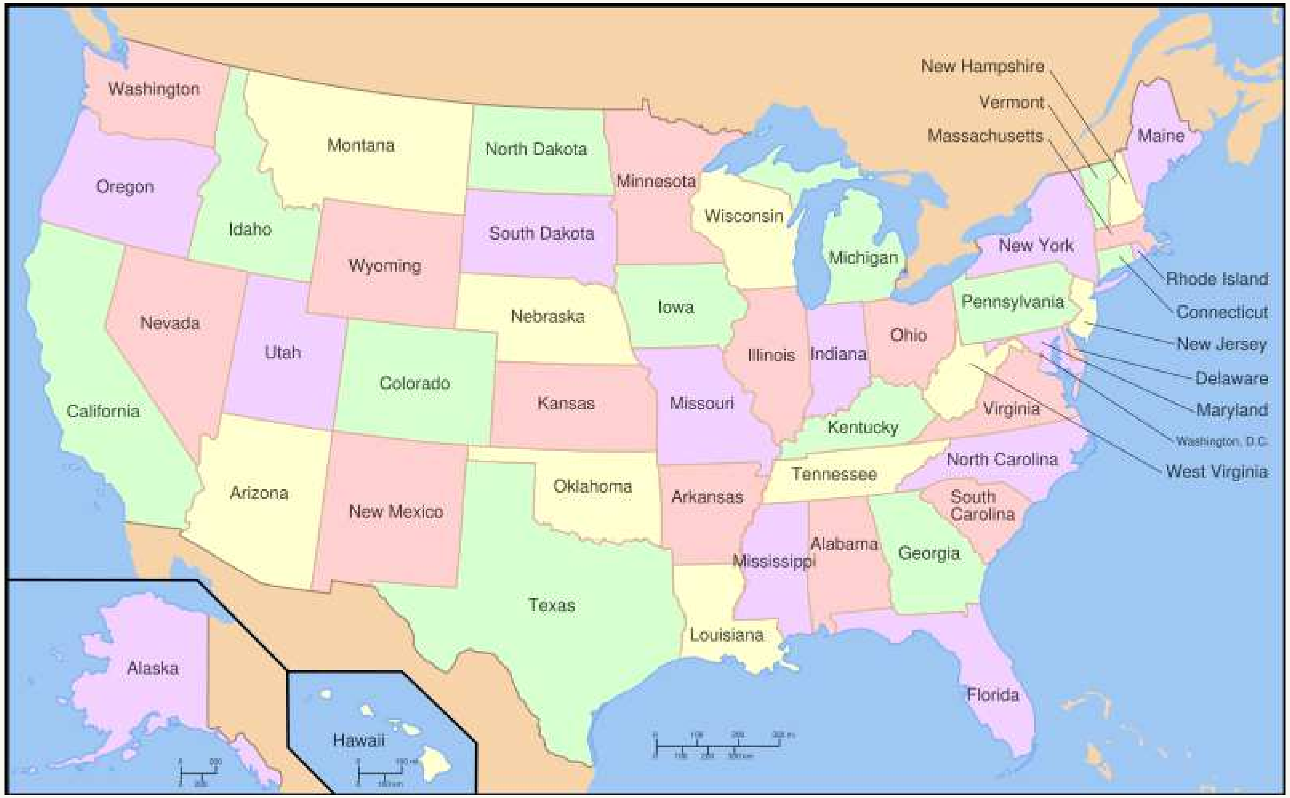


Gilles Dowek
Les métamorphoses du calcul
Une étonnante histoire de mathématiques
Le Pommier, 2007

Contents

AR Examples

The Four Colour Theorem



- *The vertices of every planar graph can be colored with at most four colors so that no two adjacent vertices receive the same color*
- Or equivalently, *any map may be colored using no more than four colors in such a way that no two adjacent regions receive the same color*

The Four Colour Theorem: Proof

Wikipedia: Four color theorem

- **Conjectured** in 1852 (Guthrie)
  - Bogus proofs in 1879, 1880

The Four Colour Theorem: Proof

Wikipedia: Four color theorem

- **Conjectured** in 1852 (Guthrie)
  - Bogus proofs in 1879, 1880
- Theoretical **progress** until the 60's–70's
  - But still **no proof**

The Four Colour Theorem: Proof

Wikipedia: Four color theorem

- **Conjectured** in 1852 (Guthrie)
  - Bogus proofs in 1879, 1880
- Theoretical **progress** until the 60's–70's
  - But still **no proof**
- **Proof** in 1976 (*Appel*, *Haken*)
  - Problem reduced to **1936 possible configurations**
  - Each checked one by one **by computer** (specific program)
  - Still need to **trust the program**!

The Four Colour Theorem: Proof

Wikipedia: Four color theorem

- **Conjectured** in 1852 (Guthrie)
  - Bogus proofs in 1879, 1880
- Theoretical **progress** until the 60's–70's
  - But still **no proof**
- **Proof** in 1976 (*Appel*, *Haken*)
  - Problem reduced to **1936 possible configurations**
  - Each checked one by one **by computer** (specific program)
  - Still need to **trust the program**!
- Proof in **Coq** in 2004 (*Werner, Gonthier*)
  - General-purpose theorem prover
  - Still need to **trust Coq**...

Robbins Algebra are Boolean

- **Robbins algebra**: $(A, \vee, \neg)$ satisfying
  $a \vee (b \vee c) = (a \vee b) \vee c$ (associativity)
  $a \vee b = b \vee a$ (commutativity)
  $\neg(\neg(a \vee b) \vee \neg(a \vee \neg b)) = a$ (Robbins's axiom)

- **Boolean algebra**: $(A, \vee, \wedge, \neg, 0, 1)$ satisfying
  $a \vee (b \vee c) = (a \vee b) \vee c$ (associativity)
  $a \vee b = b \vee a$ (commutativity)
  $a \vee (a \wedge b) = a$ (absorption)
  $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ (distributivity)
  $a \vee \neg a = 1$ (complements)
  …and their duals wrt. $\wedge/\vee$, $0/1$

Robbins Algebra are Boolean

- **Robbins algebra**: $(A, \vee, \neg)$ satisfying
  $a \vee (b \vee c) = (a \vee b) \vee c$ (associativity)
  $a \vee b = b \vee a$ (commutativity)
  $\neg(\neg(a \vee b) \vee \neg(a \vee \neg b)) = a$ (Robbins's axiom)

- **Boolean algebra**: $(A, \vee, \wedge, \neg, 0, 1)$ satisfying
  $a \vee (b \vee c) = (a \vee b) \vee c$ (associativity)
  $a \vee b = b \vee a$ (commutativity)
  $a \vee (a \wedge b) = a$ (absorption)
  $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ (distributivity)
  $a \vee \neg a = 1$ (complements)
  ...and their duals wrt. $\wedge/\vee$, $0/1$

- **Conjecture**: all Robbins algebra are Boolean

Robbins Algebra are Boolean: Proof

W. McCune. Solution of the Robbins Problem. JAR 19(3), pp. 263–276, 1997.

- **Problem posed** around 1933 (*Robbins*)
  - as a conjectured variant of another axiom set (*Huntington*)

Robbins Algebra are Boolean: Proof

W. McCune. Solution of the Robbins Problem. JAR 19(3), pp. 263–276, 1997.

- **Problem posed** around 1933 (*Robbins*)
  - as a conjectured variant of another axiom set (*Huntington*)
- **Work** on the problem (*Huntington, Robbins, Tarski*) but no solution
  - became a favorite of Tarski

Robbins Algebra are Boolean: Proof

W. McCune. Solution of the Robbins Problem. JAR 19(3), pp. 263–276, 1997.

- **Problem posed** around 1933 (*Robbins*)
  - as a conjectured variant of another axiom set (*Huntington*)
- **Work** on the problem (*Huntington, Robbins, Tarski*) but no solution
  - became a favorite of Tarski
- **First attempts** using automated reasoning in 1979 (*Winker*)
  - using the Argonne Theorem Prover ($\rightarrow$ Otter $\rightarrow$ Prover9)
  - proved useful lemmas (by hand), still not solved

Robbins Algebra are Boolean: Proof

W. McCune. Solution of the Robbins Problem. JAR 19(3), pp. 263–276, 1997.

- **Problem posed** around 1933 (*Robbins*)
  - as a conjectured variant of another axiom set (*Huntington*)
- **Work** on the problem (*Huntington, Robbins, Tarski*) but no solution
  - became a favorite of Tarski
- **First attempts** using automated reasoning in 1979 (*Winker*)
  - using the Argonne Theorem Prover ($\rightarrow$ Otter $\rightarrow$ Prover9)
  - proved useful lemmas (by hand), still not solved
- **Solution** using automated reasoning in 1997 (*McCune*)
  - using EQP = automated prover for equational logic
  - found proof of the missing lemma
  - after 14 attempts totaling five weeks of CPU time

Paris Métro Ligne 14

- Platform screen doors control software
  - Starting/stopping trains, opening/closing train and platform doors
  - Parts on-board, on wayside, at control center

Paris Métro Ligne 14: Proof

T. Lecomte, T. Servat, G. Pouzancre. Formal Methods in Satefy Critical Railway Systems. SBMF 2007.

- Safety-critical code written in B
  - Includes **formal safety properties**
  - Supports **formal refinement** (from design to implementation)

Paris Métro Ligne 14: Proof

T. Lecomte, T. Servat, G. Pouzancre. Formal Methods in Satefy Critical Railway Systems. SBMF 2007.

- Safety-critical code written in B
  - Includes **formal safety properties**
  - Supports **formal refinement** (from design to implementation)
- Large project
  - 115,000 lines of B
  - **1,000 proof** obligations, 92% fully automatic

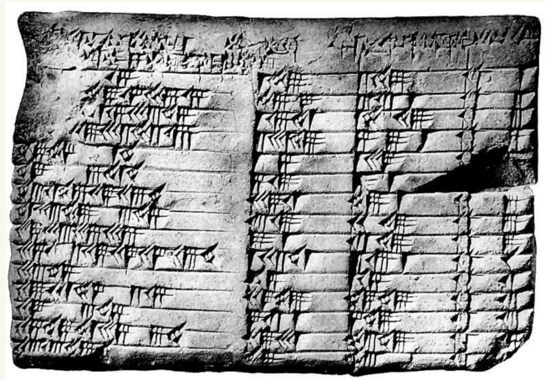
Paris Métro Ligne 14: Proof

T. Lecomte, T. Servat, G. Pouzancre. Formal Methods in Satefy Critical Railway Systems. SBMF 2007.

- Safety-critical code written in B
  - Includes **formal safety properties**
  - Supports **formal refinement** (from design to implementation)
- Large project
  - 115,000 lines of B
  - **1,000 proof** obligations, 92% fully automatic
- Seems to work!
  - **No bug found** after 9 years of operation

Before AR

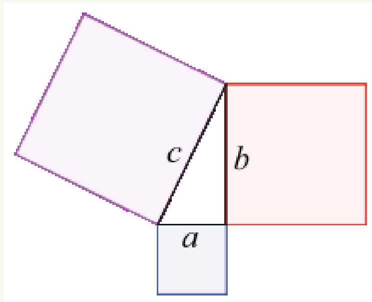
The Early Days



- Mesopotamia, since 2500 BC
  - Add, multiply, divide, area of rectangles, triangles, disks, . . .
  - With **given** numbers: **computing**

The Early Days

- Mesopotamia, since 2500 BC
  - Add, multiply, divide, area of rectangles, triangles, disks, ...
  - With **given** numbers: **computing**



- Pythagoras, 500 BC:
  - **For all** rectangle triangles (a, b, c): $a^2 + b^2 = c^2$
  - **Infinitely many** (a, b, c): **reasoning**

(images from Wikipedia)

And Then Logics

- Aristote, 350 BC:

All men are mortal.
Socrates is a man.
Therefore, Socrates is mortal.

  - **Syllogisms**: First general **reasoning rules**

And Then Logics

- Aristote, 350 BC:

  All men are mortal.
  Socrates is a man.
  Therefore, Socrates is mortal.

  - **Syllogisms**: First general **reasoning rules**

- Stoïcians 300 BC:

  If Socrates is a man, then Socrates is mortal.
  Socrates is a man.
  Therefore, Socrates is mortal.

  - **Modus ponens**: roots of **propositional logic**

And Then Logics

- Aristote, 350 BC:

  All men are mortal.
  Socrates is a man.
  Therefore, Socrates is mortal.

  - **Syllogisms**: First general **reasoning rules**

- Stoïcians 300 BC:

  If Socrates is a man, then Socrates is mortal.
  Socrates is a man.
  Therefore, Socrates is mortal.

  - **Modus ponens**: roots of **propositional logic**

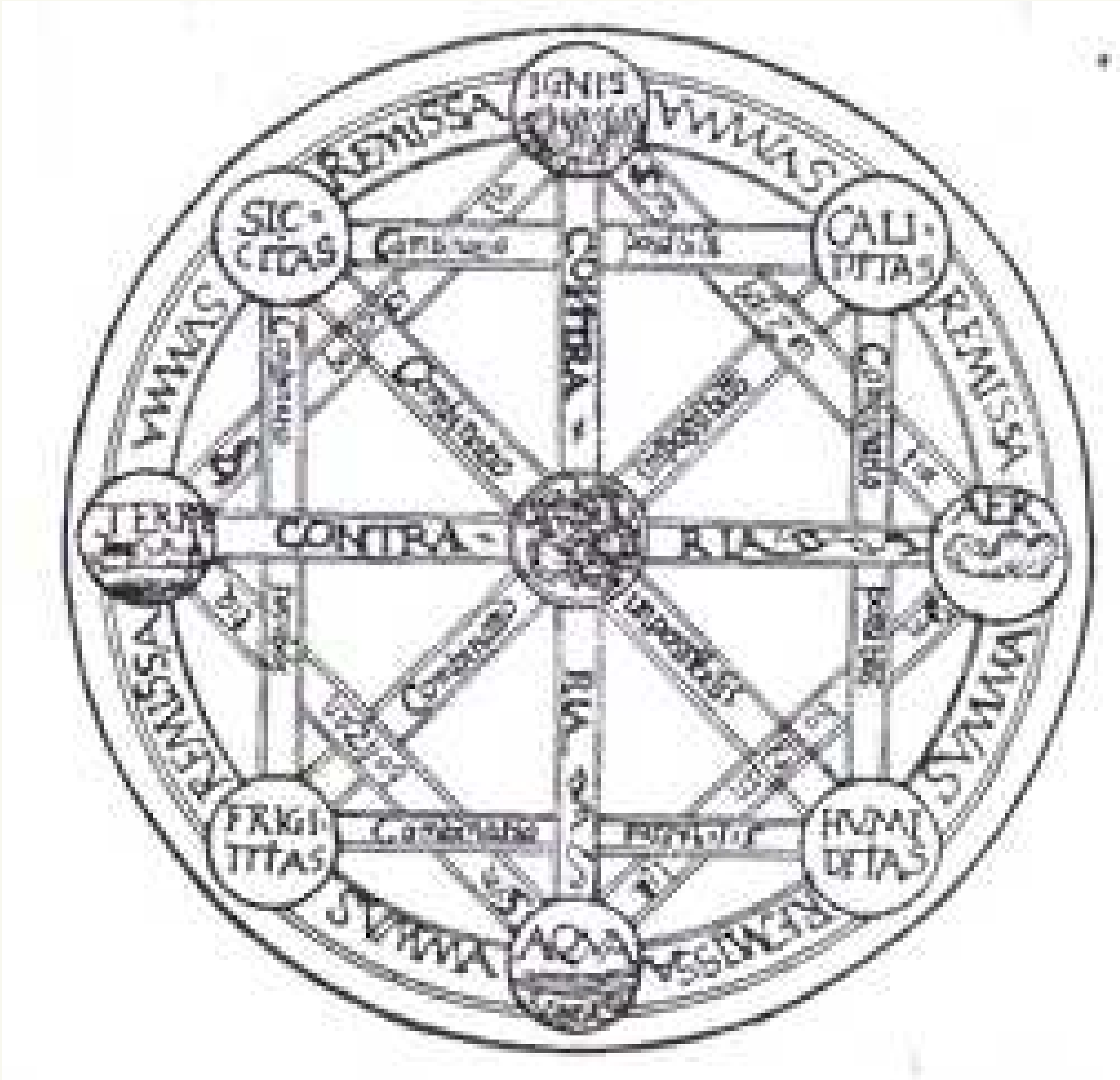
- Seen as **philosophy**, not mathematics!
  - Euclid's Elements did not (explicitly) use them!
  - Too crude: needs functions, predicates

Reasoning as Computing?

- Reducing **reasoning** to **computing** is an **old idea**
  - “Reason [. . .] is nothing but reckoning [= calculating]” (T. Hobbes, 1651)

Reasoning as Computing?

- Reducing **reasoning** to **computing** is an **old idea**
  - "Reason [. . .] is nothing but reckoning [= calculating]" (T. Hobbes, 1651)
- *Characteristica Universalis* (Leibniz, 1646–1716)
  - An (unrealized) universal **language** to express mathematical, scientific, and philosophic concepts
  - *Calculus ratiocinator* (calculus of reasoning): an (unrealized) universal logical **calculation**



(image from Wikipedia)

Formalizing Logics

- *Calculus of logic* (Boole, 1815–1864)
  - **Propositional** (Boolean!) logic, **set-theoretic** reasoning
  - **Formal rules** without interpretation

Formalizing Logics

- *Calculus of logic* (Boole, 1815–1864)
  - **Propositional** (Boolean!) logic, **set-theoretic** reasoning
  - **Formal rules** without interpretation
- *Begriffsschrift* (Frege, 1879)
  - “A formula language, modelled on that of arithmetic, of pure thought”
  - First-order logic, **Quantifiers**, sets
  - Russell’s paradox $(\{x \mid x \notin x\})$

Formalizing Logics

- *Calculus of logic* (Boole, 1815–1864)
  - **Propositional** (Boolean!) logic, **set-theoretic** reasoning
  - **Formal rules** without interpretation
- *Begriffsschrift* (Frege, 1879)
  - “A formula language, modelled on that of arithmetic, of pure thought”
  - First-order logic, **Quantifiers**, sets
  - Russell’s paradox $(\{x \mid x \notin x\})$
- *Principia Mathematica* (Whitehead and Russell, 1910)
  - **Type** theory
  - Formal foundations of **mathematics**

Frege's Begriffsschrift

Basic concept	Frege's notation	Modern notations
Judging	$\vdash A$, $\Vdash A$	$p(A) = 1$ $p(A) = i$
Negation	A	$\neg A$; $\sim A$
Conditional (implication)	A B	$B \rightarrow A$ $B \supset A$
Universal quantification	$\mathfrak{u}$ $\Phi(\mathfrak{u})$	$\forall y$: $\Phi(y)$
Existential quantification	$\mathfrak{u}$ $\Phi(\mathfrak{u})$	$\exists y$: $\Phi(y)$
Content identity (equal sign)	$A \equiv B$	$A = B$

(image from Wikipedia)

Reasoning as Computing. . . or Not?

- Hilbert's program (Hilbert, 1922)
  - (Science program, not computer!)
  - Goal: **formalize** all of mathematics
  - Goal: **prove** completeness, consistency, . . .
  - **Reduce** everything (integers, reals, functions, integration, geometry, . . .) to logic with (few) axioms

Reasoning as Computing. . . or Not?

- Hilbert's program (Hilbert, 1922)
  - (Science program, not computer!)
  - Goal: **formalize** all of mathematics
  - Goal: **prove** completeness, consistency, . . .
  - **Reduce** everything (integers, reals, functions, integration, geometry, . . .) to logic with (few) axioms
- The incompleteness theorems (Gödel, 1931)
  - Any "rich enough" formal system is **incomplete**
  - i.e. some valid statements cannot be proven
  - Essential **limit** to Hilbert's goal

Deciding is Computing

- Formalization of computation = **decidability**
  - . . . before creation of computers!
  - **Turing machines** (Turing, 1936)
  - λ**-calculus** (Church, 1936)
  - **Halting problem** is not decidable
  - **First-order logic** is not decidable

Deciding is Computing

- Formalization of computation = **decidability**
  - ... before creation of computers!
  - **Turing machines** (Turing, 1936)
  - λ**-calculus** (Church, 1936)
  - **Halting problem** is not decidable
  - **First-order logic** is not decidable
- Then came the **computers** (1940's, WWII)
  - ... and the first attempts to **compute proofs**
  - **Artificial intelligence** (McCarthy, 1956)
  - Lisp (1956), Prolog (1972)

The AR Problem

What's **logic**?

- **Facts**: logic **formulae** ϕ (syntax)

$$\forall a, b, c, n \in \mathbb{N} : n \geq 3 \Rightarrow a^n + b^n \neq c^n$$

- **Reasoning**: logic **proofs** $\phi_1, \ldots, \phi_n \vdash \phi$
  - Generally from an initial set of **axioms** Ax (aka theory)
  - A **theorem** is a ϕ such that $Ax \vdash \phi$

Logics

What's **logic**?

- **Facts**: logic **formulae** ϕ (syntax)

$$\forall a, b, c, n \in \mathbb{N} : n \geq 3 \Rightarrow a^n + b^n \neq c^n$$

- **Reasoning**: logic **proofs** $\phi_1, \ldots, \phi_n \vdash \phi$
  - Generally from an initial set of **axioms** Ax (aka theory)
  - A **theorem** is a ϕ such that $Ax \vdash \phi$
- A **proof system** defines allowable proofs
  - Using rules, tableaux, truth tables, …
  - **Synthetic** (from Ax to ϕ) or **analytic** (from ϕ to Ax)
  - Many allowed **choices**: which rule, axiom, lemma, …
  - Needs **strategies**, may stray away

Logics

What's **logic**?

- **Facts**: logic **formulae** ϕ (syntax)

$$\forall a, b, c, n \in \mathbb{N} : n \geq 3 \Rightarrow a^n + b^n \neq c^n$$

- **Reasoning**: logic **proofs** $\phi_1, \ldots, \phi_n \vdash \phi$
  - Generally from an initial set of **axioms** Ax (aka theory)
  - A **theorem** is a ϕ such that $Ax \vdash \phi$
- A **proof system** defines allowable proofs
  - Using rules, tableaux, truth tables, ...
  - **Synthetic** (from Ax to ϕ) or **analytic** (from ϕ to Ax)
  - Many allowed **choices**: which rule, axiom, lemma, ...
  - Needs **strategies**, may stray away
- **Proof = Rules + Strategy = Computing + Reasoning**

Models

What's a **useful** logic?

- **Means** something: **interpretations** M (aka models)
  - Propositions, predicates, functions, sets, numbers, programs, ...
  - **Semantics**: $M \models \phi$ if ϕ is true in/about/for M
  - **Consequence**: $\phi_1, \ldots, \phi_n \models \phi$
  - **Validity**: $Ax \models \phi$
  - **Satisfiability**: $Ax \not\models \neg\phi$
- Reasons **properly**
  - **Soundness**: all **proofs** are **valid**

$$Ax \vdash \phi \quad \Rightarrow \quad Ax \models \phi$$

  - **Completeness**: all **valid facts** can be **proven**

$$Ax \models \phi \quad \Rightarrow \quad Ax \vdash \phi$$

Computing

What's **computing**?

- An **effective** way to produce **outputs** from **inputs**
- Many models: Turing machines, Lambda calculus, recursive functions, . . .
  - All **equivalent** (Turing-complete)
  - Nothing better (Church thesis)
  - Also Lisp, C, Java, Mathlab, ...

Computing

What's **computing**?

- An **effective** way to produce **outputs** from **inputs**
- Many models: Turing machines, Lambda calculus, recursive functions, . . .
  - All **equivalent** (Turing-complete)
  - Nothing better (Church thesis)
  - Also Lisp, C, Java, Mathlab, ...

What's **deciding** a problem?

- **Computing** a **yes-or-no** answer to (any instance of) the problem
- Some things are **undecidable**
  - Does a program terminate?
  - Is a (context-free) grammar unambiguous?
  - Does a Diophantine equation have solutions?
  - **Is a logic formula valid**? (Entscheidungsproblem)

Computing Proofs

- Proofs systems can be used to **enumerate** proofs
  - E.g.: all proofs of length 0 (axioms), then length 1, etc.
  - Fair: will find a proof if there is one. . .
  - . . . but will go forever if there isn't
  - Very dumb and inefficient, but we can be smarter
- We have at least a **semi-decision** procedure
  (for **theorems** at least, for **validity** if **complete**)

Computing Proofs

- Proofs systems can be used to **enumerate** proofs
  - E.g.: all proofs of length 0 (axioms), then length 1, etc.
  - Fair: will find a proof if there is one. . .
  - . . . but will go forever if there isn't
  - Very dumb and inefficient, but we can be smarter
- We have at least a **semi-decision** procedure
  (for **theorems** at least, for **validity** if **complete**)
- Common approaches
  - Reduce formulae to **normal forms** (easier for computing)
  - Part of the theory "built-in" the method (e.g. equality), the rest provided as ordinary formulae Ax
  - Proof by **refutation**: (un)**satisfiability** of $Ax \wedge \neg\phi$

Some Decidability Results

- **Propositional logic** is **decidable**
  - Finitely many cases (exponentially many: NP-complete)
  - SAT solvers

Some Decidability Results

- **Propositional logic** is **decidable**
  - Finitely many cases (exponentially many: NP-complete)
  - SAT solvers
- **First-order logic** is only **semi-decidable**
  - Related to halting problem (Church, 1936; Turing, 1937)

Some Decidability Results

- **Propositional logic** is **decidable**
  - Finitely many cases (exponentially many: NP-complete)
  - SAT solvers
- **First-order logic** is only **semi-decidable**
  - Related to halting problem (Church, 1936; Turing, 1937)
- **Arithmetics** (on integers) is **not decidable**
  - No complete, consistent, effective proof system (Gödel, 1931)
  - Can't even enumerate valid facts
  - **Inductive** reasoning can't be effectively mechanized
  - Arithmetics on **reals** is **decidable**!

Some Decidability Results

- **Propositional logic** is **decidable**
  - Finitely many cases (exponentially many: NP-complete)
  - SAT solvers
- **First-order logic** is only **semi-decidable**
  - Related to halting problem (Church, 1936; Turing, 1937)
- **Arithmetics** (on integers) is **not decidable**
  - No complete, consistent, effective proof system (Gödel, 1931)
  - Can't even enumerate valid facts
  - **Inductive** reasoning can't be effectively mechanized
  - Arithmetics on **reals** is **decidable**!
- Many **quantifier-free fragments** are **decidable**
  - Enough for many applications

Decidability and Complexity of Some Theories

Theory	full	CQFF
propositional	NP-comp.	$\Theta(n)$
first-order	no	$\Theta(n)$
equality (uninterpreted fct.)	no	$O(n \log n)$
$\mathbb{N}, +, \times$ (Peano)	no	no
$\mathbb{N}, +$ (Pressburger)	$O(2^{2^{2^{kn}}})$	NP-comp.
$\mathbb{R}, +, \times$	$O(2^{2^{kn}})$	$O(2^{2^{kn}})$
$\mathbb{R}, +$ (or $\mathbb{Q}, +$)	$O(2^{2^{kn}})$	PTIME
recursive data structures	no	$O(n \log n)$
acyclic recursive data struct.	not elementary	$\Theta(n)$
arrays	no	NP-comp.

(CQFF = **conjunctive** quantifier-free formulae)

Using Computed Proofs

- **Finding** mathematical proofs
  - Is this conjecture a theorem?
  - Compute the mundane parts, guide strategic choices

Using Computed Proofs

- **Finding** mathematical proofs
  - Is this conjecture a theorem?
  - Compute the mundane parts, guide strategic choices
- **Checking** existing proofs
  - Detect human mistakes, document, re-organize, simplify
  - Experimental mathematics

Using Computed Proofs

- **Finding** mathematical proofs
  - Is this conjecture a theorem?
  - Compute the mundane parts, guide strategic choices
- **Checking** existing proofs
  - Detect human mistakes, document, re-organize, simplify
  - Experimental mathematics
- **Verifying** artifacts
  - Ax models the artifact, ϕ the specification

Using Computed Proofs

- **Finding** mathematical proofs
  - Is this conjecture a theorem?
  - Compute the mundane parts, guide strategic choices
- **Checking** existing proofs
  - Detect human mistakes, document, re-organize, simplify
  - Experimental mathematics
- **Verifying** artifacts
  - Ax models the artifact, ϕ the specification
- **Synthesizing** artifacts
  - Constructive proof of $\exists x.\phi(x)$

AR Milestones

Before Computers

- Deciding linear arithmetics (Presburger 1929)
  - Decision algorithm for first-order formulae over $(\mathbb{N}, +)$
  - By **quantifier elimination**
  - Very inefficient! ($O(2^{2^{2^{cn}}})$)

Before Computers

- Deciding linear arithmetics (Presburger 1929)
  - Decision algorithm for first-order formulae over $(\mathbb{N}, +)$
  - By **quantifier elimination**
  - Very inefficient! ($O(2^{2^{2^{cn}}})$)
- Along the same lines:
  - Decision algorithm for $(\mathbb{N}, \times)$ (Skolem 1930)
  - Decision algorithm for $(\mathbb{R}, +, \times)$ (Tarski 1931)
  - NB: Euclidean geometry reducible to $(\mathbb{R}, +, \times)$
  - NB: $(\mathbb{N}, +, \times)$ (Peano) is not decidable (Gödel 1931)

Before Computers

- Deciding linear arithmetics (Presburger 1929)
  - Decision algorithm for first-order formulae over $(\mathbb{N}, +)$
  - By **quantifier elimination**
  - Very inefficient! ($O(2^{2^{2^{cn}}})$)
- Along the same lines:
  - Decision algorithm for $(\mathbb{N}, \times)$ (Skolem 1930)
  - Decision algorithm for $(\mathbb{R}, +, \times)$ (Tarski 1931)
  - NB: Euclidean geometry reducible to $(\mathbb{R}, +, \times)$
  - NB: $(\mathbb{N}, +, \times)$ (Peano) is not decidable (Gödel 1931)
- **Reasoning** reduced to **computing**!

Computer Proofs: First Steps

- **Logic Theory Machine** (Newell, Shaw, Simon 1957)
  - Proofs from Principia Mathematica
  - Natural deduction in propositional logic, heuristic
  - (though propositional logic is decidable!)

Computer Proofs: First Steps

- **Logic Theory Machine** (Newell, Shaw, Simon 1957)
  - Proofs from Principia Mathematica
  - Natural deduction in propositional logic, heuristic
  - (though propositional logic is decidable!)
- **Geometry Machine** (Gelertner 1963)
  - Proofs for elementary geometry
  - Similar approach
  - (decidable but impractical)

Computer Proofs: First Steps

- **Logic Theory Machine** (Newell, Shaw, Simon 1957)
  - Proofs from Principia Mathematica
  - Natural deduction in propositional logic, heuristic
  - (though propositional logic is decidable!)
- **Geometry Machine** (Gelertner 1963)
  - Proofs for elementary geometry
  - Similar approach
  - (decidable but impractical)
- **Symbolic Integrator** (Slagle 1963)
  - Symbolic resolution of integrals
  - First “expert system”

Computer Proofs: First Steps

- **Logic Theory Machine** (Newell, Shaw, Simon 1957)
  - Proofs from Principia Mathematica
  - Natural deduction in propositional logic, heuristic
  - (though propositional logic is decidable!)
- **Geometry Machine** (Gelertner 1963)
  - Proofs for elementary geometry
  - Similar approach
  - (decidable but impractical)
- **Symbolic Integrator** (Slagle 1963)
  - Symbolic resolution of integrals
  - First “expert system”
- **Human-like** proofs!

SAT Solving

- Solving **propositional logic** satisfiability (SAT)
  - Computationally hard (NP-complete)
  - The heart of proof search

SAT Solving

- Solving **propositional logic** satisfiability (SAT)
  - Computationally hard (NP-complete)
  - The heart of proof search
- **Davis-Putnam-Logemann-Loveland** (**DPLL**) algorithm (1962)

SAT Solving

- Solving **propositional logic** satisfiability (SAT)
  - Computationally hard (NP-complete)
  - The heart of proof search
- **Davis-Putnam-Logemann-Loveland** (**DPLL**) algorithm (1962)
- Basic principle:
  - Put problem in **clausal form** (CNF) $\ell_1 \vee \ldots \vee \ell_n$
  - While possible, apply **Boolean Constraint Propagation**:

$$\frac{\boxed{\ell} \quad \boxed{\neg\ell} \vee \ell_1 \vee \ldots \vee \ell_n}{\ell_1 \vee \ldots \vee \ell_n}$$

  - Otherwise, choose a literal ℓ and try ℓ then $\neg\ell$ (**case-split**)

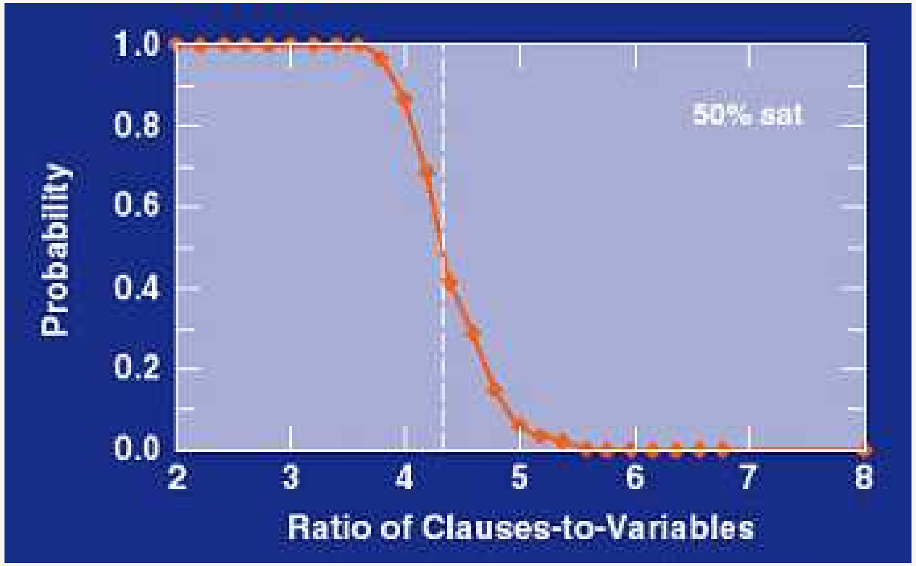
SAT Solving

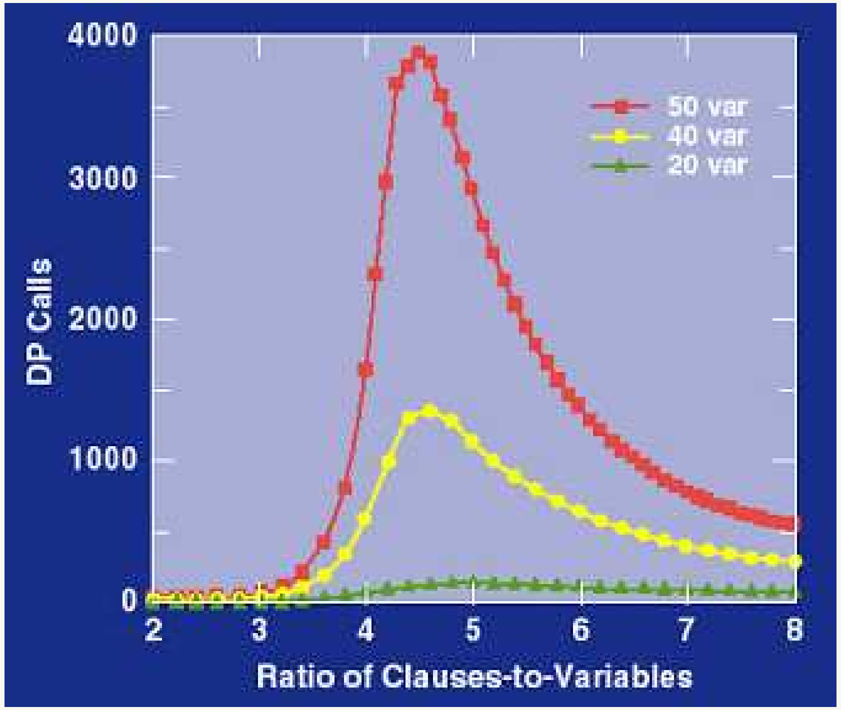
- Solving **propositional logic** satisfiability (SAT)
  - Computationally hard (NP-complete)
  - The heart of proof search
- **Davis-Putnam-Logemann-Loveland** (**DPLL**) algorithm (1962)
- Basic principle:
  - Put problem in **clausal form** (CNF) $\ell_1 \vee \ldots \vee \ell_n$
  - While possible, apply **Boolean Constraint Propagation**:

$$\frac{\boxed{\ell} \qquad \boxed{\neg\ell} \vee \ell_1 \vee \ldots \vee \ell_n}{\ell_1 \vee \ldots \vee \ell_n}$$

  - Otherwise, choose a literal ℓ and try ℓ then $\neg\ell$ (**case-split**)
- **Computer-like** proofs, not intuitive but efficient!

- DPLL-based SAT solvers **widely used** today
  - Lots of improvements, very efficient implementations
  - Berkmin, Chaff, zChaff, Minisat, . . .
  - Inside many applications
  - Often good performance in practice





images from http://www.isi.edu/ szekely/antsebook/ebook/

The Resolution Method

The **Resolution method** (Robinson 1965)

- Key idea: **unification**

$$mgu(x + 0, a^2 + y) = \{x \mapsto a^2, y \mapsto 0)$$

The Resolution Method

The **Resolution method** (Robinson 1965)

- Key idea: **unification**

$$mgu(x + 0, a^2 + y) = \{x \mapsto a^2, y \mapsto 0)$$

- Binary resolution rule:

$$\frac{\ell_1 \vee \ldots \vee \ell_n \vee \boxed{\ell} \quad \boxed{\neg \ell'} \vee \ell'_1 \vee \ldots \vee \ell'_m}{\ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee \ell'_1\sigma \vee \ldots \vee \ell'_m\sigma} \quad \boxed{\sigma = mgu(\ell, \ell')}$$

The Resolution Method

The **Resolution method** (Robinson 1965)

- Key idea: **unification**

$$mgu(x + 0, a^2 + y) = \{x \mapsto a^2, y \mapsto 0)$$

- Binary resolution rule:

$$\frac{\ell_1 \vee \ldots \vee \ell_n \vee \boxed{\ell} \quad \boxed{\neg\ell'} \vee \ell'_1 \vee \ldots \vee \ell'_m}{\ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee \ell'_1\sigma \vee \ldots \vee \ell'_m\sigma} \quad \boxed{\sigma = mgu(\ell, \ell')}$$

- This single rule (+ factoring) provides a **complete** proof method for **first-order logic**!

The Resolution Method

The **Resolution method** (Robinson 1965)

- Key idea: **unification**

$$mgu(x + 0, a^2 + y) = \{x \mapsto a^2, y \mapsto 0)$$

- Binary resolution rule:

$$\frac{\ell_1 \vee \ldots \vee \ell_n \vee \boxed{\ell} \quad \boxed{\neg \ell'} \vee \ell'_1 \vee \ldots \vee \ell'_m}{\ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee \ell'_1\sigma \vee \ldots \vee \ell'_m\sigma} \quad \boxed{\sigma = mgu(\ell, \ell')}$$

- This single rule (+ factoring) provides a **complete** proof method for **first-order logic**!

- Limitations of Resolution
  - Clauses, generic rule $\Rightarrow$ inefficient, lacks guidance
  - Need more: equality, numbers, sets, induction, ...

Equational Reasoning

Paramodulation (Robinson, Wos, 1969) another Robinson!

- For proofs with **equational theories**

e.g.
$$0 + x = x$$
$$(x + y) + z = x + (y + z)$$
$$-x + x = 0$$

- Combines resolution and replacing equals by equals

Paramodulation (Robinson, Wos, 1969)

another Robinson!

- For proofs with **equational theories**

e.g. $0 + x = x$
$(x + y) + z = x + (y + z)$
$-x + x = 0$

- Combines resolution and replacing equals by equals

- **Paramodulation rule**:

$$\frac{\ell_1 \vee \ldots \vee \ell_n \vee \boxed{s = t} \qquad \boxed{\ell'[u]} \vee \ell'_1 \vee \ldots \vee \ell'_m}{\ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee \boxed{\ell'\sigma[t\sigma]} \vee \ell'_1\sigma \vee \ldots \vee \ell'_m\sigma} \quad \boxed{\sigma = mgu(s, u)}$$

Paramodulation (Robinson, Wos, 1969)

another Robinson!

- For proofs with **equational theories**

e.g. $0 + x = x$
$(x + y) + z = x + (y + z)$
$-x + x = 0$

- Combines resolution and replacing equals by equals

- **Paramodulation rule**:

$$\frac{\ell_1 \vee \ldots \vee \ell_n \vee \boxed{s = t} \qquad \boxed{\ell'[u]} \vee \ell'_1 \vee \ldots \vee \ell'_m}{\ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee \boxed{\ell'\sigma[t\sigma]} \vee \ell'_1\sigma \vee \ldots \vee \ell'_m\sigma} \quad \boxed{\sigma = mgu(s, u)}$$

- Used for proof of Robbins conjecture

- **Term Rewriting**
  - Rules $s \to t$ used to reduce (= rewrite) s into t
  - Repeat until irreducible **normal form** $s{\downarrow}$

e.g. $0 + x \to x$
$(x + y) + z \to x + (y + z)$
$-x + x \to 0$

$\Rightarrow (a + 0) + b$ becomes $a + (0 + b)$ becomes $a + b$

Rewrite Systems

- **Term Rewriting**
  - Rules $s \to t$ used to reduce (= rewrite) s into t
  - Repeat until irreducible **normal form** $s{\downarrow}$

e.g. $0 + x \to x$
$(x + y) + z \to x + (y + z)$
$-x + x \to 0$

$\Rightarrow (a + 0) + b$ becomes $a + (0 + b)$ becomes $a + b$

- Used for reasoning in **equational theories**
  - Turn equations into rewrite rules
  - If the rules are **convergent**,
    then $s = t$ iff $s{\downarrow}$ and $t{\downarrow}$ are identical
  - **Knuth-Bendix** procedure (1970) for checking convergence
- Also at the core of **functional programming**

Prolog (Colmerauer 1972)

```
ancestor(X,X).
ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
parent(albertII,philippe).
parent(philippe,elisabeth).

?- ancestor(albertII,X), ancestor(X,elisabeth).
X = albertII
```

Prolog (Colmerauer 1972)

```
 ancestor(X,X).
ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
parent(albertII,philippe).
parent(philippe,elisabeth).

?- ancestor(albertII,X), ancestor(X,elisabeth).
X = albertII
```

- **Logic clauses** as **program statements**,
  logic reasoning as **program execution**!

Logic Programming

Prolog (Colmerauer 1972)

```
 ancestor(X,X).
ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
parent(albertII,philippe).
parent(philippe,elisabeth).

?- ancestor(albertII,X), ancestor(X,elisabeth).
X = albertII
```

- **Logic clauses** as **program statements**,
  logic reasoning as **program execution**!
- Based on **SLD-resolution** (Kowalski 1973)
  - Resolution specialized on definite clauses
- Prolog adds many programming language features!

Richer Logics

- Higher-Order Logics
  - Functions, sets, relations
- Type systems
  - Numbers, lists, trees, . . .
  - and functions/sets/relations thereof
- Inductive reasoning
- Forces **interactive** approaches = **proof assistants**
  - Most problems are undecidable, huge search spaces
  - Proof tactics and tacticals, proof planning
  - Proof editors and browsers

Some Proof Assistants

- **LCF** (Milner, 1972)
  - Based on functional programming language ML
  - Several descendants: **HOL** (Gordon, 88), **Isabelle** (Paulson, 1989)

Some Proof Assistants

- **LCF** (Milner, 1972)
  - Based on functional programming language ML
  - Several descendants: **HOL** (Gordon, 88), **Isabelle** (Paulson, 1989)
- **Coq** (Coquand, Huet, 1984)
  - Based on constructive logic
  - Used to check the 4-colour theorem (Gonthier, Werner, 2004)

Some Proof Assistants

- **LCF** (Milner, 1972)
  - Based on functional programming language ML
  - Several descendants: **HOL** (Gordon, 88), **Isabelle** (Paulson, 1989)
- **Coq** (Coquand, Huet, 1984)
  - Based on constructive logic
  - Used to check the 4-colour theorem (Gonthier, Werner, 2004)
- **PVS** (Owre, Rushby, Shankar, 1992)
  - Based on sequent calculus

Example: PVS Proof

```
sum_plus :

  |-------
{1}   (FORALL (f: [nat -> nat], g: [nat -> nat], n: nat):
         sum((LAMBDA (n: nat): f(n) + g(n)), n) = sum(f, n) + sum(g, n))

Rule? (skolem!)
Skolemizing,
this simplifies to:
sum_plus :

  |-------
{1}   sum((LAMBDA (n: nat): f!1(n) + g!1(n)), n!1)
          = sum(f!1, n!1) + sum(g!1, n!1)

Rule? (lemma "nat_induction")
Applying nat_induction where
this simplifies to:
sum_plus :

{-1}   (FORALL (p: pred[nat]):
         (p(0) AND (FORALL (j: nat): p(j) IMPLIES p(j + 1)))
             IMPLIES (FORALL (i: nat): p(i)))
  |-------
[1]   sum((LAMBDA (n: nat): f!1(n) + g!1(n)), n!1)
          = sum(f!1, n!1) + sum(g!1, n!1)
```

Decision Procedures

- **Automated** decision procedures (DPs) for specific theories
  - **Quantifier-free** fragments
  - (QF) Linear integers/reals $\Rightarrow$ **simplex algorithm**
  - (QF) Polynomials $\Rightarrow$ **Gröbner bases**
  - (QF) Equality on uninterpreted functions $\Rightarrow$ **congruence closure**
  - (QF) arrays, data structures $\Rightarrow$ reduce to previous case

Decision Procedures

- **Automated** decision procedures (DPs) for specific theories
  - **Quantifier-free** fragments
  - (QF) Linear integers/reals $\Rightarrow$ **simplex algorithm**
  - (QF) Polynomials $\Rightarrow$ **Gröbner bases**
  - (QF) Equality on uninterpreted functions $\Rightarrow$ **congruence closure**
  - (QF) arrays, data structures $\Rightarrow$ reduce to previous case
- **Nelson-Oppem** method (1979)
  - Solve (QF) problems over multiple theories by **combining DPs**
  - Split the problem and coordinate solutions
  - Intuition: **proof** = **logic** (SAT) + **theories** (DP)

Decision Procedures

- **Automated** decision procedures (DPs) for specific theories
  - **Quantifier-free** fragments
  - (QF) Linear integers/reals $\Rightarrow$ **simplex algorithm**
  - (QF) Polynomials $\Rightarrow$ **Gröbner bases**
  - (QF) Equality on uninterpreted functions $\Rightarrow$ **congruence closure**
  - (QF) arrays, data structures $\Rightarrow$ reduce to previous case
- **Nelson-Oppem** method (1979)
  - Solve (QF) problems over multiple theories by **combining DPs**
  - Split the problem and coordinate solutions
  - Intuition: **proof** = **logic** (SAT) + **theories** (DP)
- Inside many tools: **embedded** automated reasoning

Proving Programs

- Principle: reduce **programs** to **logic**
  - Base case: $\{x \times x > 0\}\; y := x \times x \;\{y > 0\}$
  - Program properties reduce to (first-order) **verification conditions**
  - Prove with standard proof tools (solvers)
  - Needs guidance: loop invariants, pre/post conditions, ...

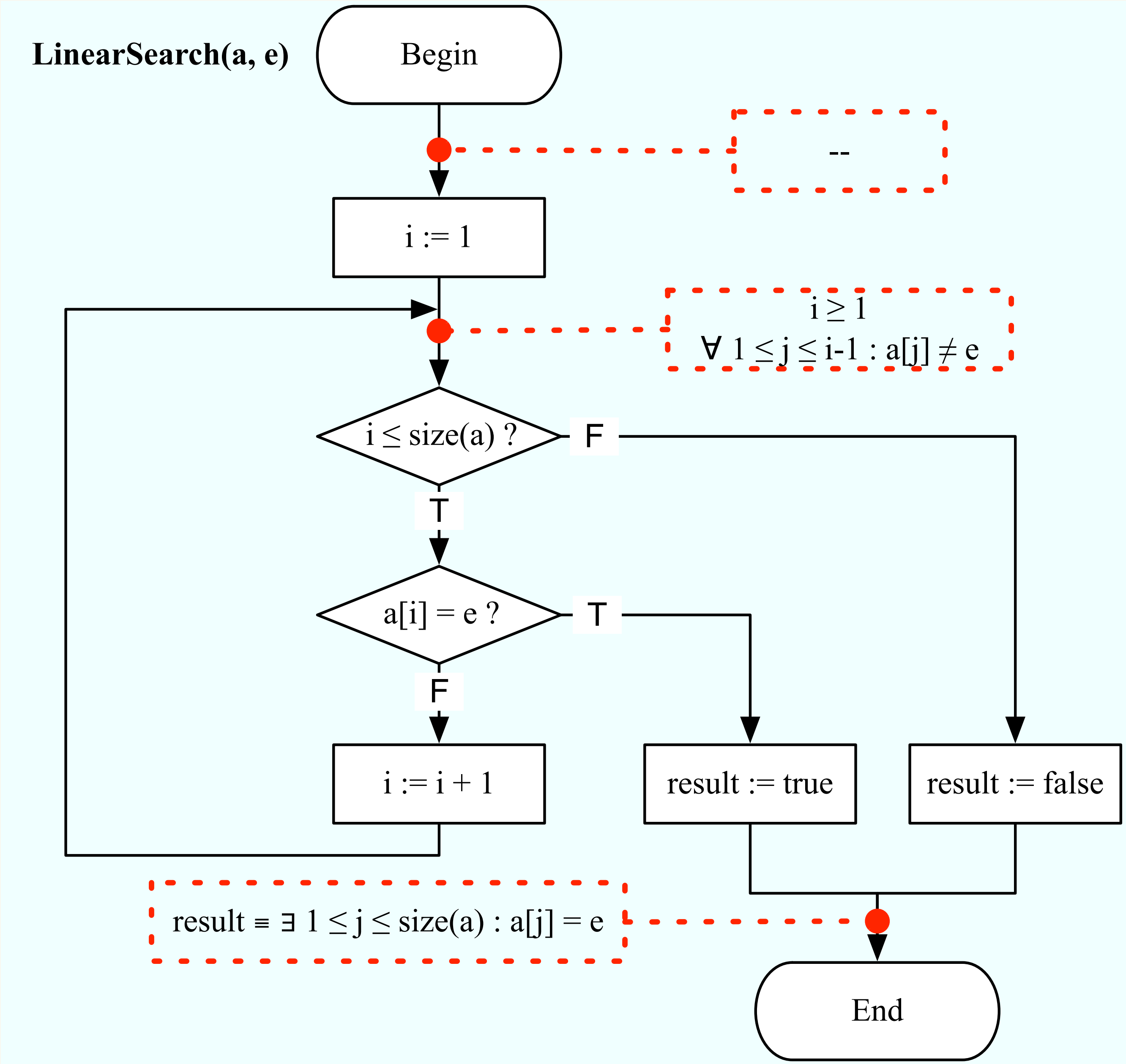
Proving Programs

- Principle: reduce **programs** to **logic**
  - Base case: $\{x \times x > 0\}\, y := x \times x\, \{y > 0\}$
  - Program properties reduce to (first-order) **verification conditions**
  - Prove with standard proof tools (solvers)
  - Needs guidance: loop invariants, pre/post conditions, ...
- Floyd's **inductive assertions** (1967)
  - Decompose a program in sequential **basic paths**
  - Specify **assertions** at connection points
  - Prove that each path preserves the assertions

Proving Programs

- Principle: reduce **programs** to **logic**
  - Base case: $\{x \times x > 0\}\, y := x \times x\, \{y > 0\}$
  - Program properties reduce to (first-order) **verification conditions**
  - Prove with standard proof tools (solvers)
  - Needs guidance: loop invariants, pre/post conditions, ...
- Floyd's **inductive assertions** (1967)
  - Decompose a program in sequential **basic paths**
  - Specify **assertions** at connection points
  - Prove that each path preserves the assertions
- Hard problem: loops, recursion, pointers, objects, concurrency, ...
- Lots of conditions to check (thousands) but "easy" proofs
- Example: B method applied to Paris metro line

Example: Inductive Assertions



Model-Checking

- **Model-Checking**: check $M \models \phi$ for a **given** model M
  - Rather than **validity**: $M \models \phi$ for all M
    or **consequence**: $M \models \phi$ for **all** M such that $M \models Ax$
  - By exhaustive exploration of M: **semantic** approach
  - Fully automatic! (though computation-intensive)

Model-Checking

- **Model-Checking**: check $M \models \phi$ for a **given** model M
  - Rather than **validity**: $M \models \phi$ for all M
    or **consequence**: $M \models \phi$ for **all** M such that $M \models Ax$
  - By exhaustive exploration of M: **semantic** approach
  - Fully automatic! (though computation-intensive)
- Concretely, M = (the state space of) a computer program/system
  - Very large (millions of states), state space explosion
  - Even infinite, with symbolic approaches ($\Rightarrow$ solvers!)
  - Explore all possible **executions**
  - For all parameters, inputs, scheduling, timing
- ϕ = temporal logic

e.g. $\Box\neg(busy_a \wedge busy_b)$
$\Box(send \Rightarrow \Diamond receive)$

AR Perspectives

Some Current Trends

- Richer logics
  - Linear, separation logic (resources, memory)
  - Non-monotonic, default logic (commonsense)
  - Modal logic (time, knowledge, possibility)

Some Current Trends

- Richer logics
  - Linear, separation logic (resources, memory)
  - Non-monotonic, default logic (commonsense)
  - Modal logic (time, knowledge, possibility)
- Meta-reasoning
  - Analyze proof goals, select proof methods
  - Reflection, proof planning

Some Current Trends

- Richer logics
  - Linear, separation logic (resources, memory)
  - Non-monotonic, default logic (commonsense)
  - Modal logic (time, knowledge, possibility)
- Meta-reasoning
  - Analyze proof goals, select proof methods
  - Reflection, proof planning
- Embedded (automated) proving
  - In computer algebra systems
  - In computer/software analysis tools
  - In planning and scheduling

Some Current Trends

- Richer logics
  - Linear, separation logic (resources, memory)
  - Non-monotonic, default logic (commonsense)
  - Modal logic (time, knowledge, possibility)
- Meta-reasoning
  - Analyze proof goals, select proof methods
  - Reflection, proof planning
- Embedded (automated) proving
  - In computer algebra systems
  - In computer/software analysis tools
  - In planning and scheduling
- Algorithmic improvements
  - CASC competition (8 divisions, 20+ categories in 2012)

Parting Thoughts

- Automated reasoning is a **flourishing discipline**

Parting Thoughts

- Automated reasoning is a **flourishing discipline**
- **Assists**, rather than replaces, human proofs
  - Experimental mathematics

Parting Thoughts

- Automated reasoning is a **flourishing discipline**
- **Assists**, rather than replaces, human proofs
    - Experimental mathematics
- Comprehensive, interactive **proof assistants** for rich logics
- Efficient, automatic **decision procedures** for simpler theories

Parting Thoughts

- Automated reasoning is a **flourishing discipline**
- **Assists**, rather than replaces, human proofs
  - Experimental mathematics
- Comprehensive, interactive **proof assistants** for rich logics
- Efficient, automatic **decision procedures** for simpler theories
- Computers can do **a lot of** reasoning
  - By reducing it to computing
  - Is this still reasoning?
  - The AI Effect: *As soon as AI works, it is no longer called AI*

Parting Thoughts

- Automated reasoning is a **flourishing discipline**

- **Assists**, rather than replaces, human proofs
  - Experimental mathematics

- Comprehensive, interactive **proof assistants** for rich logics
- Efficient, automatic **decision procedures** for simpler theories

- Computers can do **a lot of** reasoning
  - By reducing it to computing
  - Is this still reasoning?
  - The AI Effect: *As soon as AI works, it is no longer called AI*

- Will computer provers someday equal, then surpass humans?
  That is the (weak) AI question!

Bibliography

Bibliography

[1] A. Bundy. A Survey of Automated Deduction. Research Report Nr. 1, Division of Informatics, University of Edinburgh, April 1999.

[2] M. Davis. The Early History of Automated Deduction. In: A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Elsevier, 2001.

[3] G. Dowek. Les métamorphoses du calcul : une étonnante histoire de mathématiques. Le Pommier, 2007.

[4] J. Harrison. A Short Survey of Automated Reasoning. in: Algebraic Biology 2007, Lecture Notes in Computer Science 4545, Springer, 2007.