An overview of automated reasoning

John Harrison

Intel Corporation

3rd February 2014 (10:30–11:30)

Talk overview

- What is automated reasoning?
- Early history and taxonomy
- Automation, its scope and limits
- Interactive theorem proving

What is automated reasoning?

Attempting to perform logical reasoning in an automatic and algorithmic way. An old dream!

What is automated reasoning?

Attempting to perform logical reasoning in an automatic and algorithmic way. An old dream!

- Hobbes (1651): "*Reason* ... is nothing but *reckoning* (that is, adding and subtracting) of the consequences of general names agreed upon, for the *marking* and *signifying* of our thoughts."

What is automated reasoning?

Attempting to perform logical reasoning in an automatic and algorithmic way. An old dream!

- Hobbes (1651): "*Reason* ... is nothing but *reckoning* (that is, adding and subtracting) of the consequences of general names agreed upon, for the *marking* and *signifying* of our thoughts."
- Leibniz (1685) "When there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right."

What is automated reasoning?

Attempting to perform logical reasoning in an automatic and algorithmic way. An old dream!

- Hobbes (1651): "*Reason* ... is nothing but *reckoning* (that is, adding and subtracting) of the consequences of general names agreed upon, for the *marking* and *signifying* of our thoughts."
- Leibniz (1685) "When there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right."

Nowadays, by 'automatic and algorithmic' we mean 'using a computer program'.

What does automated reasoning involve?

There are two steps to performing automated reasoning, as anticipated by Leibniz:

What does automated reasoning involve?

There are two steps to performing automated reasoning, as anticipated by Leibniz:

- Express *statement* of theorems in a formal language. (Leibniz's *characteristica universalis*.)

What does automated reasoning involve?

There are two steps to performing automated reasoning, as anticipated by Leibniz:

- Express *statement* of theorems in a formal language. (Leibniz's *characteristica universalis*.)
- Use automated algorithmic manipulations on those formal expressions. (Leibniz's *calculus ratiocinator*).

Theoretical and practical limitations

- Limitative results in logic (Gödel, Tarski, Church-Turing, Matiyasevich) imply that not even elementary number theory can be done completely automatically.

Theoretical and practical limitations

- Limitative results in logic (Gödel, Tarski, Church-Turing, Matiyasevich) imply that not even elementary number theory can be done completely automatically.
- There *are* formal proof systems (e.g. first-order set theory) and semi-decision procedures that will in principle find the proof of anything provable in 'ordinary' mathematics.

Theoretical and practical limitations

- Limitative results in logic (Gödel, Tarski, Church-Turing, Matiyasevich) imply that not even elementary number theory can be done completely automatically.
- There *are* formal proof systems (e.g. first-order set theory) and semi-decision procedures that will in principle find the proof of anything provable in 'ordinary' mathematics.
- In practice, because of time or space limits, these automated procedures are often not useful, and we may prefer either

Theoretical and practical limitations

- Limitative results in logic (Gödel, Tarski, Church-Turing, Matiyasevich) imply that not even elementary number theory can be done completely automatically.
- There *are* formal proof systems (e.g. first-order set theory) and semi-decision procedures that will in principle find the proof of anything provable in 'ordinary' mathematics.
- In practice, because of time or space limits, these automated procedures are often not useful, and we may prefer either
  - Attempts to mimic human intelligence

Theoretical and practical limitations

- Limitative results in logic (Gödel, Tarski, Church-Turing, Matiyasevich) imply that not even elementary number theory can be done completely automatically.
- There *are* formal proof systems (e.g. first-order set theory) and semi-decision procedures that will in principle find the proof of anything provable in 'ordinary' mathematics.
- In practice, because of time or space limits, these automated procedures are often not useful, and we may prefer either
  - Attempts to mimic human intelligence
  - An interactive 'proof assistant' guided by a human

Why automated reasoning?

For general intellectual interest? It is a fascinating field that helps to understand the real nature of mathematical creativity. Or more practically:

Why automated reasoning?

For general intellectual interest? It is a fascinating field that helps to understand the real nature of mathematical creativity. Or more practically:

- To check the correctness of proofs in mathematics, supplementing or even replacing the existing ‘social process’ of peer review etc. with a more objective criterion.

Why automated reasoning?

For general intellectual interest? It is a fascinating field that helps to understand the real nature of mathematical creativity. Or more practically:

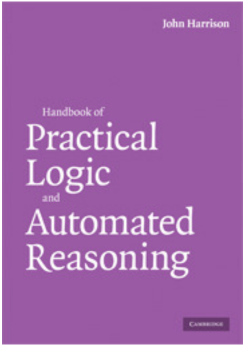
- To check the correctness of proofs in mathematics, supplementing or even replacing the existing 'social process' of peer review etc. with a more objective criterion.
- To extend rigorous proof from pure mathematics to the verification of computer systems (programs, hardware systems, protocols etc.), supplementing or replacing the usual testing process.

Just to fix notation

English	Our notation	Other common notations
false	$\bot$	0, F
true	$\top$	1, T
not p	$\neg p$	$\overline{p}$, $-p$, $\sim p$
p and q	$p \wedge q$	pq, $p \& q$, $p \cdot q$
p or q	$p \vee q$	$p + q$, $p \mid q$, p *or* q
p implies q	$p \Rightarrow q$	$p \leq q$, $p \rightarrow q$, $p \supset q$
p iff q	$p \Leftrightarrow q$	$p = q$, $p \equiv q$, $p \sim q$
for all x, p	$\forall x.\ p$	$(\forall x)p$, $(x)p$
there exists x such that p	$\exists x.\ p$	$(\exists x)p$, $(Ex)p$

For more details

An introductory survey of many central results in automated reasoning, together with actual OCaml model implementations
`http://www.cl.cam.ac.uk/~jrh13/atp/index.html`



Early history and taxonomy

Early research in automated reasoning

Most early theorem provers were fully automatic, with two main styles:

Early research in automated reasoning

Most early theorem provers were fully automatic, with two main styles:

- Human-oriented AI style approaches (Newell-Simon, Gelerntner)

Early research in automated reasoning

Most early theorem provers were fully automatic, with two main styles:

- Human-oriented AI style approaches (Newell-Simon, Gelerntner)
- Machine-oriented algorithmic approaches (Davis, Gilmore, Wang, Prawitz)

Early research in automated reasoning

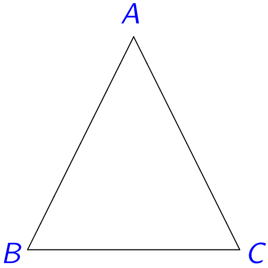
Most early theorem provers were fully automatic, with two main styles:

- Human-oriented AI style approaches (Newell-Simon, Gelerntner)
- Machine-oriented algorithmic approaches (Davis, Gilmore, Wang, Prawitz)

Modern work is dominated by machine-oriented approach but there have been some successes for the AI approach.

A theorem in geometry (1)

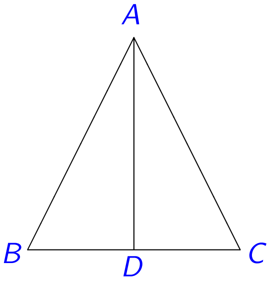
Example of AI approach in action:



If the sides AB and AC are equal (i.e. the triangle is isosceles), then the angles ABC and ACB are equal.

A theorem in geometry (2)

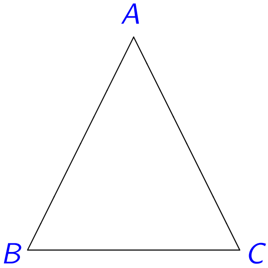
Pick bisector D of the line BC:



and then use the fact that the triangles ABD and ACD are congruent.

A theorem in geometry (3)

Originally found by Pappus but not in many books:



Simply, the triangles ABC and ACB are congruent.

The Robbins Conjecture (1)

Huntington (1933) presented the following axioms for a Boolean algebra:

$$\begin{aligned} x + y &= y + x \\ (x + y) + z &= x + (y + z) \\ n(n(x) + y) + n(n(x) + n(y)) &= x \end{aligned}$$

Herbert Robbins conjectured that the Huntington equation can be replaced by a simpler one:

$$n(n(x + y) + n(x + n(y))) = x$$

The Robbins Conjecture (2)

This conjecture went unproved for more than 50 years, despite being studied by many mathematicians, even including Tarski.
It because a popular target for researchers in automated reasoning.

The Robbins Conjecture (2)

This conjecture went unproved for more than 50 years, despite being studied by many mathematicians, even including Tarski.

It because a popular target for researchers in automated reasoning.

In October 1996, a (key lemma leading to) a proof was found by McCune's program EQP.

The successful search took about 8 days on an RS/6000 processor and used about 30 megabytes of memory.

The scope of automation

What can be automated?

- Validity/satisfiability in propositional logic is decidable (SAT).

What can be automated?

- Validity/satisfiability in propositional logic is decidable (SAT).
- Validity/satisfiability in many temporal logics is decidable.

What can be automated?

- Validity/satisfiability in propositional logic is decidable (SAT).
- Validity/satisfiability in many temporal logics is decidable.
- Validity in first-order logic is *semidecidable*, i.e. there are complete proof procedures that may run forever on invalid formulas

What can be automated?

- Validity/satisfiability in propositional logic is decidable (SAT).
- Validity/satisfiability in many temporal logics is decidable.
- Validity in first-order logic is *semidecidable*, i.e. there are complete proof procedures that may run forever on invalid formulas
- Validity in higher-order logic is not even *semidecidable* (or anywhere in the arithmetical hierarchy).

Some specific theories

People usually use extensive background in set theory, arithmetic, algebra or geometry. How does that affect decidability?

Some specific theories

People usually use extensive background in set theory, arithmetic, algebra or geometry. How does that affect decidability?

- Linear theory of $\mathbb{N}$ or $\mathbb{Z}$ is decidable. Nonlinear theory not even semidecidable.

Some specific theories

People usually use extensive background in set theory, arithmetic, algebra or geometry. How does that affect decidability?

- Linear theory of $\mathbb{N}$ or $\mathbb{Z}$ is decidable. Nonlinear theory not even semidecidable.
- Linear and nonlinear theory of $\mathbb{R}$ is decidable, though complexity is very bad in the nonlinear case.

Some specific theories

People usually use extensive background in set theory, arithmetic, algebra or geometry. How does that affect decidability?

- Linear theory of $\mathbb{N}$ or $\mathbb{Z}$ is decidable. Nonlinear theory not even semidecidable.
- Linear and nonlinear theory of $\mathbb{R}$ is decidable, though complexity is very bad in the nonlinear case.
- Linear and nonlinear theory of $\mathbb{C}$ is decidable. Commonly used in geometry.

Some specific theories

People usually use extensive background in set theory, arithmetic, algebra or geometry. How does that affect decidability?

- Linear theory of $\mathbb{N}$ or $\mathbb{Z}$ is decidable. Nonlinear theory not even semidecidable.
- Linear and nonlinear theory of $\mathbb{R}$ is decidable, though complexity is very bad in the nonlinear case.
- Linear and nonlinear theory of $\mathbb{C}$ is decidable. Commonly used in geometry.

Many of these naturally generalize known algorithms like linear/integer programming and Sturm's theorem.

Quantifier elimination

Many decision methods based on quantifier elimination, e.g.

- $\mathbb{C} \models (\exists x.\, x^2 + 1 = 0) \Leftrightarrow \top$
- $\mathbb{R} \models (\exists x.\, ax^2 + bx + c = 0) \Leftrightarrow a \neq 0 \wedge b^2 \geq 4ac \vee a = 0 \wedge (b \neq 0 \vee c = 0)$

If we can decide variable-free formulas, quantifier elimination implies completeness.

The Big Four

Arguably these are the most important of the traditional automated theorem provers:

The Big Four

Arguably these are the most important of the traditional automated theorem provers:

- Propositional satisfiability / tautology checking (SAT), e.g. MiniSAT, zchaff.

The Big Four

Arguably these are the most important of the traditional automated theorem provers:

- Propositional satisfiability / tautology checking (SAT), e.g. MiniSAT, zchaff.
- First-order logic (resolution, tableaux, model elimination ...), e.g. Vampire, prover9, Spass.

The Big Four

Arguably these are the most important of the traditional automated theorem provers:

- Propositional satisfiability / tautology checking (SAT), e.g. MiniSAT, zchaff.
- First-order logic (resolution, tableaux, model elimination . . .), e.g. Vampire, prover9, Spass.
- Equational reasoning (Knuth-Bendix completion etc.), e.g. Waldmeister.

The Big Four

Arguably these are the most important of the traditional automated theorem provers:

- Propositional satisfiability / tautology checking (SAT), e.g. MiniSAT, zchaff.
- First-order logic (resolution, tableaux, model elimination ...), e.g. Vampire, prover9, Spass.
- Equational reasoning (Knuth-Bendix completion etc.), e.g. Waldmeister.
- Combined decision procedures (SMT), e.g. Z3, MathSat.

The Big Four

Arguably these are the most important of the traditional automated theorem provers:

- Propositional satisfiability / tautology checking (SAT), e.g. MiniSAT, zchaff.
- First-order logic (resolution, tableaux, model elimination ...), e.g. Vampire, prover9, Spass.
- Equational reasoning (Knuth-Bendix completion etc.), e.g. Waldmeister.
- Combined decision procedures (SMT), e.g. Z3, MathSat.

There are also many more specialized symbolic algorithms, some of which can produce proofs or other certificates.

SAT

Traditionally, propositional logic has been regarded as fairly boring.

SAT

Traditionally, propositional logic has been regarded as fairly boring.

- There are severe limitations to what can be said with propositional logic.

SAT

Traditionally, propositional logic has been regarded as fairly boring.

- There are severe limitations to what can be said with propositional logic.
- Propositional logic is trivially decidable in theory.

SAT

Traditionally, propositional logic has been regarded as fairly boring.

- There are severe limitations to what can be said with propositional logic.
- Propositional logic is trivially decidable in theory.
- Propositional satisfiability (SAT) is the original NP-complete problem, so seems intractable in practice.

SAT

Traditionally, propositional logic has been regarded as fairly boring.

- There are severe limitations to what can be said with propositional logic.
- Propositional logic is trivially decidable in theory.
- Propositional satisfiability (SAT) is the original NP-complete problem, so seems intractable in practice.

However, modern fast algorithms (distantly descended from the Davis-Putnam algorithm from the 60s) are surprisingly effective on big problems.

FOL and equational reasoning

Two major threads in theorem proving research, especially in the 60s and 70s:

FOL and equational reasoning

Two major threads in theorem proving research, especially in the 60s and 70s:

- First-order proof search using a variety of algorithms (tableaux, resolution / inverse method, model elimination, ...)

FOL and equational reasoning

Two major threads in theorem proving research, especially in the 60s and 70s:

- First-order proof search using a variety of algorithms (tableaux, resolution / inverse method, model elimination, . . .)
- Optimized treatment of equations based on rewriting and orderings (Knuth-Bendix completion, demodulation, paramodulation etc.)

FOL and equational reasoning

Two major threads in theorem proving research, especially in the 60s and 70s:

- First-order proof search using a variety of algorithms (tableaux, resolution / inverse method, model elimination, ...)
- Optimized treatment of equations based on rewriting and orderings (Knuth-Bendix completion, demodulation, paramodulation etc.)

Many first-order algorithms and their implementations blend the key ideas from both threads (e.g. superposition).

FOL example

This is Łoś's well-known 'non-obvious' fact:

$$
\begin{aligned}
&(\forall x\ y\ z.\ P(x,y) \land P(y,z) \Rightarrow P(x,z)) \land \\
&(\forall x\ y\ z.\ Q(x,y) \land Q(y,z) \Rightarrow Q(x,z)) \land \\
&(\forall x\ y.\ Q(x,y) \Rightarrow Q(y,x)) \land \\
&(\forall x\ y.\ P(x,y) \lor Q(x,y)) \\
&\Rightarrow (\forall x\ y.\ P(x,y)) \lor (\forall x\ y.\ Q(x,y))
\end{aligned}
$$

Most people take more time to solve this than automated first-order provers.

Equational logic examples

A simple group theory exercise,

$$
\begin{aligned}
&(\forall x\ y\ z.\ x \cdot (y \cdot z) = (x \cdot y) \cdot z) \wedge \\
&(\forall x.\ 1 \cdot x = x) \wedge \\
&(\forall x.\ \mathsf{inv}(x) \cdot x = 1) \\
&\Rightarrow \mathsf{inv}(\mathsf{inv}(x)) = x
\end{aligned}
$$

Equational logic examples

A simple group theory exercise,

$$
\begin{array}{l}
(\forall x\ y\ z.\ x \cdot (y \cdot z) = (x \cdot y) \cdot z)\wedge \\
(\forall x.\ 1 \cdot x = x)\wedge \\
(\forall x.\ \mathrm{inv}(x) \cdot x = 1) \\
\Rightarrow \mathrm{inv}(\mathrm{inv}(x)) = x
\end{array}
$$

The conceptual core of the Eckmann-Hilton argument that certain homotopy groups are abelian:

$$
\begin{array}{l}
(\forall x.\ 1 \cdot x = x)\wedge \\
(\forall x.\ x \cdot 1 = x)\wedge \\
(\forall x.\ 1 + x = x)\wedge \\
(\forall x.\ x + 1 = x)\wedge \\
(\forall w\ x\ y\ z.\ (w \cdot x) + (y \cdot z) = (w + y) \cdot (x + z)) \\
\Rightarrow \forall x\ y.\ x \cdot y = x + y
\end{array}
$$

SMT

Two major threads of development of combined decision procedures for 'quantifier-free' (purely universal) FOL together with additional theories like linear arithmetic.

SMT

Two major threads of development of combined decision procedures for ‘quantifier-free’ (purely universal) FOL together with additional theories like linear arithmetic.

- Nelson-Oppen

SMT

Two major threads of development of combined decision procedures for ‘quantifier-free’ (purely universal) FOL together with additional theories like linear arithmetic.

- Nelson-Oppen
- Shostak

SMT

Two major threads of development of combined decision procedures for ‘quantifier-free’ (purely universal) FOL together with additional theories like linear arithmetic.

- Nelson-Oppen
- Shostak

Both were used in some early theorem provers and program verification systems and some of the key ideas are still used today.

SMT

Two major threads of development of combined decision procedures for 'quantifier-free' (purely universal) FOL together with additional theories like linear arithmetic.

- Nelson-Oppen
- Shostak

Both were used in some early theorem provers and program verification systems and some of the key ideas are still used today. Modern systems are usually based on SAT as the core, hence 'satisfiability modulo theories' (SMT).

SMT examples

SMT systems can handle purely equational reasoning (without embedded quantifiers)

$$f(f(f(f(f(x))))) = x \wedge f(f(f(x))) = x \Rightarrow f(x) = x$$

SMT examples

SMT systems can handle purely equational reasoning (without embedded quantifiers)

$$f(f(f(f(f(x))))) = x \wedge f(f(f(x))) = x \Rightarrow f(x) = x$$

or with arithmetic theories too:

$$f(v - 1) - 1 = v + 1 \wedge f(u) + 1 = u - 1 \wedge u + 1 = v \Rightarrow \bot$$

Quantifiers and theories?

We could attack a much wider range of problems with both quantifier and theory reasoning together. However, almost all such combinations immediately enter the realm of the undecidable.

Quantifiers and theories?

We could attack a much wider range of problems with both quantifier and theory reasoning together. However, almost all such combinations immediately enter the realm of the undecidable.

For example, using linear integer arithmetic with one function symbol, when we can characterize squaring:

$$(\forall n. f(-n) = f(n)) \wedge f(0) = 0 \wedge (\forall n. 0 \leq n \Rightarrow f(n+1) = f(n) + n + n + 1)$$

Quantifiers and theories?

We could attack a much wider range of problems with both quantifier and theory reasoning together. However, almost all such combinations immediately enter the realm of the undecidable.

For example, using linear integer arithmetic with one function symbol, when we can characterize squaring:

$$(\forall n. f(-n) = f(n)) \land f(0) = 0 \land (\forall n. 0 \leq n \Rightarrow f(n+1) = f(n) + n + n + 1)$$

and then multiplication by

$$m = n \cdot p \Leftrightarrow (n + p)^2 = n^2 + p^2 + 2m$$

Quantifiers and theories?

We could attack a much wider range of problems with both quantifier and theory reasoning together. However, almost all such combinations immediately enter the realm of the undecidable.

For example, using linear integer arithmetic with one function symbol, when we can characterize squaring:

$$(\forall n. f(-n) = f(n)) \land f(0) = 0 \land (\forall n. 0 \leq n \Rightarrow f(n+1) = f(n) + n + n + 1)$$

and then multiplication by

$$m = n \cdot p \Leftrightarrow (n + p)^2 = n^2 + p^2 + 2m$$

There are special cases that work (e.g. based on type structure), and many SMT systems have incomplete heuristics for quantifiers.

Interactive theorem proving

Interactive theorem proving (1)

In practice, most interesting problems can't be automated completely:

- They don't fall in a practical decidable subset
- Pure first order proof search is not a feasible approach with, e.g. set theory

Interactive theorem proving (1)

In practice, most interesting problems can't be automated completely:

- They don't fall in a practical decidable subset
- Pure first order proof search is not a feasible approach with, e.g. set theory

In practice, we need an interactive arrangement, where the user and machine work together.
The user can delegate simple subtasks to pure first order proof search or one of the decidable subsets.
However, at the high level, the user must guide the prover.

Interactive theorem proving (2)

The idea of a more 'interactive' approach was already anticipated by pioneers, e.g. Wang (1960):

> *[...] the writer believes that perhaps machines may more quickly become of practical use in mathematical research, not by proving new theorems, but by formalizing and checking outlines of proofs, say, from textbooks to detailed formalizations more rigorous that* Principia *[Mathematica], from technical papers to textbooks, or from abstracts to technical papers.*

However, constructing an effective and programmable combination is not so easy.

SAM

First successful family of interactive provers were the SAM systems:

> *Semi-automated mathematics is an approach to theorem-proving which seeks to combine automatic logic routines with ordinary proof procedures in such a manner that the resulting procedure is both efficient and subject to human intervention in the form of control and guidance. Because it makes the mathematician an essential factor in the quest to establish theorems, this approach is a departure from the usual theorem-proving attempts in which the computer* unaided *seeks to establish proofs.*

SAM V was used to settle an open problem in lattice theory.

Three influential proof checkers

- AUTOMATH (de Bruijn, . . .) — Implementation of type theory, used to check non-trivial mathematics such as Landau's *Grundlagen*

Three influential proof checkers

- AUTOMATH (de Bruijn, . . .) — Implementation of type theory, used to check non-trivial mathematics such as Landau's *Grundlagen*
- Mizar (Trybulec, . . .) — Block-structured natural deduction with 'declarative' justifications, used to formalize large body of mathematics

Three influential proof checkers

- AUTOMATH (de Bruijn, ...) — Implementation of type theory, used to check non-trivial mathematics such as Landau's *Grundlagen*
- Mizar (Trybulec, ...) — Block-structured natural deduction with 'declarative' justifications, used to formalize large body of mathematics
- LCF (Milner et al) — Programmable proof checker for Scott's Logic of Computable Functions written in new functional language ML.

Three influential proof checkers

- AUTOMATH (de Bruijn, . . .) — Implementation of type theory, used to check non-trivial mathematics such as Landau's *Grundlagen*
- Mizar (Trybulec, . . .) — Block-structured natural deduction with 'declarative' justifications, used to formalize large body of mathematics
- LCF (Milner et al) — Programmable proof checker for Scott's Logic of Computable Functions written in new functional language ML.

Ideas from all these systems are used in present-day systems. (Corbineau's declarative proof mode for Coq . . .)

Who checks the checker?

Why should we believe that a formally checked proof is more reliable than a hand proof or one supported by ad-hoc programs?

Who checks the checker?

Why should we believe that a formally checked proof is more reliable than a hand proof or one supported by ad-hoc programs?

- What if the underlying logic is inconsistent? Many notable logicians (Frege, Curry, Martin-Löf, . . .) have proposed systems that turned out to be inconsistent.

Who checks the checker?

Why should we believe that a formally checked proof is more reliable than a hand proof or one supported by ad-hoc programs?

- What if the underlying logic is inconsistent? Many notable logicians (Frege, Curry, Martin-Löf, ...) have proposed systems that turned out to be inconsistent.
- What if the inference rules of the logic are specified incorrectly? It's easy and common to make mistakes connected with variable capture.

Who checks the checker?

Why should we believe that a formally checked proof is more reliable than a hand proof or one supported by ad-hoc programs?

- What if the underlying logic is inconsistent? Many notable logicians (Frege, Curry, Martin-Löf, ...) have proposed systems that turned out to be inconsistent.
- What if the inference rules of the logic are specified incorrectly? It's easy and common to make mistakes connected with variable capture.
- What if the proof checker has a bug? They are often large and complex pieces of software not developed to high standards of rigour

Prover architecture

The reliability of a theorem prover increases dramatically if its correctness depends only on a small amount of code.

Prover architecture

The reliability of a theorem prover increases dramatically if its correctness depends only on a small amount of code.

- de Bruijn approach — generate proofs that can be certified by a simple, separate checker.

Prover architecture

The reliability of a theorem prover increases dramatically if its correctness depends only on a small amount of code.

- de Bruijn approach — generate proofs that can be certified by a simple, separate checker.
- LCF approach — reduce all rules to sequences of primitive inferences implemented by a small logical kernel.

Prover architecture

The reliability of a theorem prover increases dramatically if its correctness depends only on a small amount of code.

- de Bruijn approach — generate proofs that can be certified by a simple, separate checker.
- LCF approach — reduce all rules to sequences of primitive inferences implemented by a small logical kernel.

The checker or kernel can be much simpler than the prover as a whole.
Nothing is ever certain, but we can potentially achieve very high levels of reliability in this way.

Key ideas behind LCF

- Implement in a strongly-typed functional programming language (usually a variant of ML)
- Make `thm` ('theorem') an abstract data type with only simple primitive inference rules
- Make the implementation language available for arbitrary extensions.

Proof styles

Directly invoking the primitive or derived rules tends to give proofs that are *procedural*.

A *declarative* style (*what* is to be proved, not *how*) can be nicer:

- Easier to write and understand independent of the prover
- Easier to modify
- Less tied to the details of the prover, hence more portable

Mizar pioneered the declarative style of proof.

Recently, several other declarative proof languages have been developed, as well as declarative shells round existing systems like HOL and Isabelle.

Finding the right style is an interesting research topic.

Procedural proof example

```
let NSQRT_2 = prove
 (`!p q. p * p = 2 * q * q ==> q = 0`,
  MATCH_MP_TAC num_WF THEN REWRITE_TAC[RIGHT_IMP_FORALL_THM] THEN
  REPEAT STRIP_TAC THEN FIRST_ASSUM(MP_TAC o AP_TERM `EVEN`) THEN
  REWRITE_TAC[EVEN_MULT; ARITH] THEN REWRITE_TAC[EVEN_EXISTS] THEN
  DISCH_THEN(X_CHOOSE_THEN `m:num` SUBST_ALL_TAC) THEN
  FIRST_X_ASSUM(MP_TAC o SPECL [`q:num`; `m:num`]) THEN
  ASM_REWRITE_TAC[ARITH_RULE
   `q < 2 * m ==> q * q = 2 * m * m ==> m = 0 <=>
    (2 * m) * 2 * m = 2 * q * q ==> 2 * m <= q`] THEN
  ASM_MESON_TAC[LE_MULT2; MULT_EQ_0; ARITH_RULE `2 * x <= x <=> x = 0`]);;
```

Declarative proof example

```
let NSQRT_2 = prove
 (`!p q. p * p = 2 * q * q ==> q = 0`,
  suffices_to_prove
   `!p. (!m. m < p ==> (!q. m * m = 2 * q * q ==> q = 0))
         ==> (!q. p * p = 2 * q * q ==> q = 0)`
   (wellfounded_induction) THEN
  fix [`p:num`] THEN
  assume("A") `!m. m < p ==> !q. m * m = 2 * q * q ==> q = 0` THEN
  fix [`q:num`] THEN
  assume("B") `p * p = 2 * q * q` THEN
  so have `EVEN(p * p) <=> EVEN(2 * q * q)` (trivial) THEN
  so have `EVEN(p)` (using [ARITH; EVEN_MULT] trivial) THEN
  so consider (`m:num`,"C",`p = 2 * m`) (using [EVEN_EXISTS] trivial) THEN
  cases ("D",`q < p \/ p <= q`) (arithmetic) THENL
   [so have `q * q = 2 * m * m ==> m = 0` (by ["A"] trivial) THEN
    so we're finished (by ["B"; "C"] algebra);
    so have `p * p <= q * q` (using [LE_MULT2] trivial) THEN
    so have `q * q = 0` (by ["B"] arithmetic) THEN
    so we're finished (algebra)]);;
```

The Seventeen Provers of the World (1)

- ACL2 — Highly automated prover for first-order number theory without explicit quantifiers, able to do induction proofs itself.
- Alfa/Agda — Prover for constructive type theory integrated with dependently typed programming language.
- B prover — Prover for first-order set theory designed to support verification and refinement of programs.
- Coq — LCF-like prover for constructive Calculus of Constructions with reflective programming language.
- HOL (HOL Light, HOL4, ProofPower) — Seminal LCF-style prover for classical simply typed higher-order logic.
- IMPS — Interactive prover for an expressive logic supporting partially defined functions.

The Seventeen Provers of the World (2)

- Isabelle/Isar — Generic prover in LCF style with a newer declarative proof style influenced by Mizar.
- Lego — Well-established framework for proof in constructive type theory, with a similar logic to Coq.
- Metamath — Fast proof checker for an exceptionally simple axiomatization of standard ZF set theory.
- Minlog — Prover for minimal logic supporting practical extraction of programs from proofs.
- Mizar — Pioneering system for formalizing mathematics, originating the declarative style of proof.
- Nuprl/MetaPRL — LCF-style prover with powerful graphical interface for Martin-Löf type theory with new constructs.

The Seventeen Provers of the World (3)

- Omega — Unified combination in modular style of several theorem-proving techniques including proof planning.
- Otter/IVY — Powerful automated theorem prover for pure first-order logic plus a proof checker.
- PVS — Prover designed for applications with an expressive classical type theory and powerful automation.
- PhoX — prover for higher-order logic designed to be relatively simple to use in comparison with Coq, HOL etc.
- Theorema — Ambitious integrated framework for theorem proving and computer algebra built inside Mathematica.

For more, see Freek Wiedijk, *The Seventeen Provers of the World*, Springer Lecture Notes in Computer Science vol. 3600, 2006.

Conclusions

- Automated reasoning as a general idea is an old dream, which is finally being realized, at least to the extent it is possible.

Conclusions

- Automated reasoning as a general idea is an old dream, which is finally being realized, at least to the extent it is possible.
- Traditional automated methods have their limitations, but many of these tools are remarkably powerful.

Conclusions

- Automated reasoning as a general idea is an old dream, which is finally being realized, at least to the extent it is possible.
- Traditional automated methods have their limitations, but many of these tools are remarkably powerful.
- There is a rich and diverse group of interactive proof assistants, which are integrating many automated tools in a sound way, and being used for a variety of applications.