

A survey on Interactive Theorem Proving

Andrea Asperti

Department of Computer Science, University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
asperti@cs.unibo.it

Talk given at the Tata Institute of Technology, Mumbai (INDIA)

January 2009

Content

Interactive theorem proving

ITP goals

Recent achievements

ITP systems

First generation ITPs

LCF: the procedural style

Mizar: the declarative style

Automath: a logical framework

Second generation ITPs

Types and Type Theory

Higher Order Logic

Conclusion

Content

Interactive theorem proving

- ITP goals

- Recent achievements

- ITP systems

First generation ITPs

- LCF: the procedural style

- Mizar: the declarative style

- Automath: a logical framework

Second generation ITPs

- Types and Type Theory

- Higher Order Logic

Conclusion

Content

Interactive theorem proving

- ITP goals

- Recent achievements

- ITP systems

First generation ITPs

- LCF: the procedural style

- Mizar: the declarative style

- Automath: a logical framework

Second generation ITPs

- Types and Type Theory

- Higher Order Logic

Conclusion

Content

Interactive theorem proving

- ITP goals

- Recent achievements

- ITP systems

First generation ITPs

- LCF: the procedural style

- Mizar: the declarative style

- Automath: a logical framework

Second generation ITPs

- Types and Type Theory

- Higher Order Logic

Conclusion

Interactive vs. automated proving

ATP proving of mathematical theorems by a computer program.

ITP developing formal proofs by man-machine collaboration.

Different activities with different problems and different communities.

Interactive vs. automated proving

ATP proving of mathematical theorems by a computer program.

ITP developing formal proofs by man-machine collaboration.

Different activities with different problems and different communities.

ITP goals

Invent a new way of doing mathematics “in front of a computer”.

Wang - Towards mechanical mathematics, 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 than Principia Mathematica, from technical papers to textbooks, or from abstracts to technical papers.

ITP goals

Invent a new way of doing mathematics “in front of a computer”.

Wang - Towards mechanical mathematics, 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 than Principia Mathematica, from technical papers to textbooks, or from abstracts to technical papers.

ITP goals

Invent a new way of doing mathematics “in front of a computer”.

Wang - Towards mechanical mathematics, 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 than Principia Mathematica, from technical papers to textbooks, or from abstracts to technical papers.

ITP goals

The machine must be aware of the mathematical content (the logic) of expressions (passing from a machine readable to a machine understandable representation of mathematics).

In proof editing, the user must be relieved of trivial steps, concentrating on the creative choices of the proof.

The system must support automatic proof checking; build large repositories of trusted mathematical knowledge.

ITP goals

The machine must be aware of the mathematical content (the logic) of expressions (passing from a machine readable to a machine understandable representation of mathematics).

In proof editing, the user must be relieved of trivial steps, concentrating on the creative choices of the proof.

The system must support automatic proof checking; build large repositories of trusted mathematical knowledge.

ITP goals

The machine must be aware of the mathematical content (the logic) of expressions (passing from a machine readable to a machine understandable representation of mathematics).

In proof editing, the user must be relieved of trivial steps, concentrating on the creative choices of the proof.

The system must support automatic proof checking; build large repositories of trusted mathematical knowledge.

ITP goals

The machine must be aware of the mathematical content (the logic) of expressions (passing from a machine readable to a machine understandable representation of mathematics).

In proof editing, the user must be relieved of trivial steps, concentrating on the creative choices of the proof.

The system must support automatic proof checking; build large repositories of trusted mathematical knowledge.

ITP goals

The machine must be aware of the mathematical content (the logic) of expressions (passing from a machine readable to a machine understandable representation of mathematics).

In proof editing, the user must be relieved of trivial steps, concentrating on the creative choices of the proof.

The system must support automatic proof checking; build large repositories of trusted mathematical knowledge.

Automatic Proof Checking

Coquand 2008

The history of mathematics has stories about false results that went undetected for long periods of time. However, it is generally believed that if a published mathematical argument is not valid, it will be eventually detected as such. While the process of finding a proof may require creative insight, the activity of checking a given mathematical argument is an objective activity; mathematical correctness should not be decided by a social process.

Automatic Proof Checking

Harrison 2007

A book written 70 years ago by Lescat gave 130 pages of errors made by major mathematicians up to 1900. With the abundance of theorems being published today, often emanating from writers who are not trained mathematicians, one fears that a project like Lescat would be practically impossible, or at least would demand a journal to itself.

Recent achievements

- ▶ Proof of the prime number theorem (J.Avigad et. al. at the Carnegie Mellon University in Pittsburgh using the Isabelle system)

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{\ln(x)} = 1$$

30.000 lines, 43 files.

- ▶ Proof of the four color theorem (G.Gonthier at the Microsoft Research Center in Cambridge using the Coq system).
60.000 lines, 132 files.

A typical proof requiring case inspection over a large number of cases. In Coq the computation is integrated inside the logic, hence it can be trusted as the other logical steps.

Recent achievements

- ▶ Proof of the prime number theorem (J.Avigad et. al. at the Carnegie Mellon University in Pittsburgh using the Isabelle system)

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{x} = \frac{1}{\ln(x)}$$

30.000 lines, 43 files.

- ▶ Proof of the four color theorem (G.Gonthier at the Microsoft Research Center in Cambridge using the Coq system).
60.000 lines, 132 files.

A typical proof requiring case inspection over a large number of cases. In Coq the computation is integrated inside the logic, hence it can be trusted as the other logical steps.

Recent achievements

- ▶ Proof of the prime number theorem (J.Avigad et. al. at the Carnegie Mellon University in Pittsburgh using the Isabelle system)

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{x} = \frac{1}{\ln(x)}$$

30.000 lines, 43 files.

- ▶ Proof of the four color theorem (G.Gonthier at the Microsoft Research Center in Cambridge using the Coq system).
60.000 lines, 132 files.

A typical proof requiring case inspection over a large number of cases. In Coq the computation is integrated inside the logic, hence it can be trusted as the other logical steps.

Recent achievements

- ▶ Proof of the prime number theorem (J.Avigad et. al. at the Carnegie Mellon University in Pittsburgh using the Isabelle system)

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{x} = \frac{1}{\ln(x)}$$

30.000 lines, 43 files.

- ▶ Proof of the four color theorem (G.Gonthier at the Microsoft Research Center in Cambridge using the Coq system).
60.000 lines, 132 files.

A typical proof requiring case inspection over a large number of cases. In Coq the computation is integrated inside the logic, hence it can be trusted as the other logical steps.

Recent achievements

- ▶ Proof of the Jordan curve theorem (proved by Tom Hales at the University of Pittsburgh using HOL light).
75000 lines,15 files.
(later proved by in Mizar).

In 1998 Hales claimed to have a proof of Kepler conjecture (the optimal way of packing spheres is the way one normally stacks oranges) that however relied on a huge number of inequalities checked by means of a computer program.

The work was rejected in the Annals of Mathematics and Hales decided to formalize his proof using an ITP (Flyspeck project, still running). As a training exercise he proved the Jordan curve theorem.

Recent achievements

- ▶ Proof of the Jordan curve theorem (proved by Tom Hales at the University of Pittsburgh using HOL light).
75000 lines, 15 files.
(later proved by in Mizar).

In 1998 Hales claimed to have a proof of Kepler conjecture (the optimal way of packing spheres is the way one normally stacks oranges) that however relied on a huge number of inequalities checked by means of a computer program.

The work was rejected in the Annals of Mathematics and Hales decided to formalize his proof using an ITP (Flyspeck project, still running). As a training exercise he proved the Jordan curve theorem.

Recent achievements

- ▶ Proof of the Jordan curve theorem (proved by Tom Hales at the University of Pittsburgh using HOL light).
75000 lines, 15 files.
(later proved by in Mizar).

In 1998 Hales claimed to have a proof of Kepler conjecture (the optimal way of packing spheres is the way one normally stacks oranges) that however relied on a huge number of inequalities checked by means of a computer program.

The work was rejected in the Annals of Mathematics and Hales decided to formalize his proof using an ITP (Flyspeck project, still running). As a training exercise he proved the Jordan curve theorem.

Recent achievements

- ▶ Proof of the Jordan curve theorem (proved by Tom Hales at the University of Pittsburgh using HOL light).
75000 lines, 15 files.
(later proved by in Mizar).

In 1998 Hales claimed to have a proof of Kepler conjecture (the optimal way of packing spheres is the way one normally stacks oranges) that however relied on a huge number of inequalities checked by means of a computer program.

The work was rejected in the Annals of Mathematics and Hales decided to formalize his proof using an ITP (Flyspeck project, still running). As a training exercise he proved the Jordan curve theorem.

Characteristic components of ITP systems

1) An interactive proof editor (proof assistant/proof checker)

- ▶ interactive editing of proofs, formulas, and terms in a formal theory of mathematics
- ▶ creation of definitions, theorems, theories and libraries
- ▶ management of the mathematical repository
- ▶ formal correctness check of the database

Constable et al. 1986

... our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is *active*, whereas paper, for instance, is not.

Characteristic components of ITP systems

- 1) An interactive proof editor (proof assistant/proof checker)
 - ▶ interactive editing of proofs, formulas, and terms in a formal theory of mathematics
 - ▶ creation of definitions, theorems, theories and libraries
 - ▶ management of the mathematical repository
 - ▶ formal correctness check of the database

Constable et al. 1986

... our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is *active*, whereas paper, for instance, is not.

Characteristic components of ITP systems

- 1) An interactive proof editor (proof assistant/proof checker)
 - ▶ interactive editing of proofs, formulas, and terms in a formal theory of mathematics
 - ▶ creation of definitions, theorems, theories and libraries
 - ▶ management of the mathematical repository
 - ▶ formal correctness check of the database

Constable et al. 1986

... our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is *active*, whereas paper, for instance, is not.

Characteristic components of ITP systems

1) An interactive proof editor (proof assistant/proof checker)

- ▶ interactive editing of proofs, formulas, and terms in a formal theory of mathematics
- ▶ creation of definitions, theorems, theories and libraries
- ▶ management of the mathematical repository
- ▶ formal correctness check of the database

Constable et al. 1986

... our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is *active*, whereas paper, for instance, is not.

Characteristic components of ITP systems

- 1) An interactive proof editor (proof assistant/proof checker)
 - ▶ interactive editing of proofs, formulas, and terms in a formal theory of mathematics
 - ▶ creation of definitions, theorems, theories and libraries
 - ▶ management of the mathematical repository
 - ▶ formal correctness check of the database

Constable et al. 1986

... our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is *active*, whereas paper, for instance, is not.

Characteristic components of ITP systems

- 1) An interactive proof editor (proof assistant/proof checker)
 - ▶ interactive editing of proofs, formulas, and terms in a formal theory of mathematics
 - ▶ creation of definitions, theorems, theories and libraries
 - ▶ management of the mathematical repository
 - ▶ formal correctness check of the database

Constable et al. 1986

... our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is *active*, whereas paper, for instance, is not.

Characteristic components of ITP systems

2) A large library of results

- ▶ a measure of the success of the system
- ▶ an essential requirement for achieving complex results

Wiedijk 2003

... a good library is more important than a user friendly system

Characteristic components of ITP systems

2) A large library of results

- ▶ a measure of the success of the system
- ▶ an essential requirement for achieving complex results

Wiedijk 2003

... a good library is more important than a user friendly system

Characteristic components of ITP systems

2) A large library of results

- ▶ a measure of the success of the system
- ▶ an essential requirement for achieving complex results

Wiedijk 2003

... a good library is more important than a user friendly system

Characteristic components of ITP systems

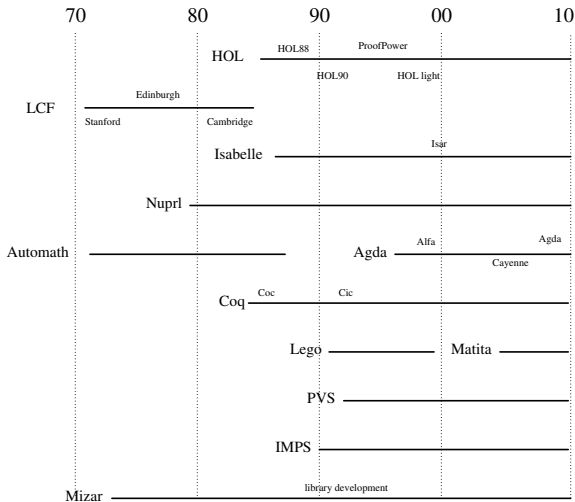
2) A large library of results

- ▶ a measure of the success of the system
- ▶ an essential requirement for achieving complex results

Wiedijk 2003

... a good library is more important than a user friendly system

- ▶ Automath [Eindhoven] (De Bruijn)
- ▶ the HOL family [Cambridge] - deriving from LCF (R.Milner)
 - ▶ HOL4, HOL88 (M.Gordon), HOL90 (K.Slind)
 - ▶ HOL lite (J.Harrison)
 - ▶ Proof Power (ICL Ltd)
- ▶ Isabelle/Isar (L.Paulson,T.Nipkow) [Cambridge,Munich]
- ▶ NuPRL (Constable), MetaPearl [Cornelle]
- ▶ The COQ family
 - ▶ Coq (Huet,Coquand,Paulin-Mohring) [INRIA-France]
 - ▶ Agda (Coquand) [Chalmers]
 - ▶ Lego (Pollack) [Edinburgh]
 - ▶ Matita (Asperti,Sacerdoti Coen) [Bologna]
- ▶ PVS (N.Shankar) [Stanford]
- ▶ IMPS (W.Farmer) [McMaster]
- ▶ Mizar (A.Trybulec) [Bialystok]



Stanford LCF, Mizar, Automath

Not truly interactive but batch-oriented proof checkers.

Harrison (2007)

... each of them in its way has been profoundly influential. Many of the most successful interactive theorem provers around today are directly descended from one of these.

- ▶ LCF: the procedural style
- ▶ Mizar: the declarative style (Mizar mode)
- ▶ Automath: first logical framework

procedural vs. declarative

$$\begin{array}{c} P_1 \quad \dots \quad P_n \\ \boxed{\text{logical rule}} \\ C \end{array}$$

procedural vs. declarative

procedural

$P_1 \dots P_n$

logical rule

C

procedural vs. declarative

declarative

$$\begin{array}{c} P_1 \quad \dots \quad P_n \\ \boxed{\text{logical rule}} \\ C \end{array}$$

procedural vs. declarative

- procedural
 - ▶ backward reasoning
 - ▶ compact
 - ▶ unreadable (each command is interpreted in a context that depends from the previous history)
- declarative
 - ▶ forward reasoning
 - ▶ verbose
 - ▶ more readable

Stanford LCF

LCF (Milner, 1972): Logic for Computable Functions (a logic devised by Dana Scott for reasoning about recursive functions).

Procedural proofstyle: Proofs are conducted via a progressive refinement of the goal into simpler subgoals (backward reasoning), by means of a fixed set of commands (tactics).

Milner 1972

The user's task is alleviated by two features: a subgoaling facility and a powerful simplification mechanism.

Edinburgh LCF

Two problems with Stanford LCF:

- ▶ the system created data structures for formal proofs (proof objects); in the seventies this caused memory problems.
In Edinburgh LCF: an abstract data type with axioms as values and logical rules as operations.
Warning: an explicit representation of proofs may be convenient for systems with types dependent on proofs (dependent types)
- ▶ Stanford LCF had a fixed set of commands.
To extend the set of commands Milner designed a new declarative, strongly typed programming language: ML (Meta Language).

Edinburgh LCF

Two problems with Stanford LCF:

- ▶ the system created data structures for formal proofs (proof objects); in the seventies this caused memory problems. In Edinburgh LCF: an abstract data type with axioms as values and logical rules as operations.

Warning: an explicit representation of proofs may be convenient for systems with types dependent on proofs (dependent types)

- ▶ Stanford LCF had a fixed set of commands. To extend the set of commands Milner designed a new declarative, strongly typed programming language: ML (Meta Language).

Edinburgh LCF

Two problems with Stanford LCF:

- ▶ the system created data structures for formal proofs (proof objects); in the seventies this caused memory problems. In Edinburgh LCF: an abstract data type with axioms as values and logical rules as operations. Warning: an explicit representation of proofs may be convenient for systems with types dependent on proofs (dependent types)
- ▶ Stanford LCF had a fixed set of commands. To extend the set of commands Milner designed a new declarative, strongly typed programming language: ML (Meta Language).

Edinburgh LCF

Two problems with Stanford LCF:

- ▶ the system created data structures for formal proofs (proof objects); in the seventies this caused memory problems. In Edinburgh LCF: an abstract data type with axioms as values and logical rules as operations.

Warning: an explicit representation of proofs may be convenient for systems with types dependent on proofs (dependent types)

- ▶ Stanford LCF had a fixed set of commands.

To extend the set of commands Milner designed a new declarative, strongly typed programming language: ML (Meta Language).

Edinburgh LCF

Two problems with Stanford LCF:

- ▶ the system created data structures for formal proofs (proof objects); in the seventies this caused memory problems. In Edinburgh LCF: an abstract data type with axioms as values and logical rules as operations. Warning: an explicit representation of proofs may be convenient for systems with types dependent on proofs (dependent types)
- ▶ Stanford LCF had a fixed set of commands. To extend the set of commands Milner designed a new declarative, strongly typed programming language: ML (Meta Language).

ML

ML is particularly suited for implementing tactics

- ▶ ML is a polymorphic, strongly typed language (the result of a function may be constrained to be of type **proof**)
- ▶ ML has a mechanism for raising and handling exceptions (convenient for handling tactic failures and backtracking)
- ▶ ML is higher-order; functions are first class objects, allowing tactics(functions) to be combined by means of tacticals (functions over functions)

ML - continued

In practice, one only used a limited set of functions for composing tactics, called tacticals.

- ▶ then, try-else, repeat, ...
- ▶ functions to examine and build terms

ML is too powerful as a tactical language...
but it proved to be an excellent general purpose language.

ML - continued

In practice, one only used a limited set of functions for composing tactics, called tacticals.

- ▶ then, try-else, repeat, ...
- ▶ functions to examine and build terms

ML is too powerful as a tactical language...

but it proved to be an excellent general purpose language.

ML - continued

In practice, one only used a limited set of functions for composing tactics, called tacticals.

- ▶ then, try-else, repeat, ...
- ▶ functions to examine and build terms

ML is too powerful as a tactical language...
but it proved to be an excellent general purpose language.

Mizar: the declarative style

Rudnicki - An overview of the Mizar Project (1992)

A. Trybulec, the leader of the project, has designed a language for writing mathematics. The logical structure of the language is based on a natural deduction system developed by Jaśkowski. The text written in the language are called Mizar article and are organized in a database. The Tarski-Grothendieck set theory forms the basis of doing mathematics in Mizar.

Since 1989, the main focus is on the development of the Mizar Mathematical Library (about 50.000 theorems).

Automath

Automath Archive

The ideas of Automath started around 1967 and gave rise to a large-scale project (roughly 1970-1975). It was the first big enterprise for automated verification of mathematics. The system was tested by treating a full text book (E.Landau's Grundlagen der Analysis), which could be accomplished successfully in spite of computer power limitations and in spite of almost complete lack of software support and tex editing facilities.

Automath was never widely publicized at the time, never reached widespread use, and moderately influenced later systems.

Automath

Automath Archive

The ideas of Automath started around 1967 and gave rise to a large-scale project (roughly 1970-1975). It was the first big enterprise for automated verification of mathematics. The system was tested by treating a full text book (E.Landau's Grundlagen der Analysis), which could be accomplished successfully in spite of computer power limitations and in spite of almost complete lack of software support and tex editing facilities.

Automath was never widely publicized at the time, never reached widespread use, and moderately influenced later systems.

Logical frameworks

A logical framework is a system supporting a large class of logics. The object-logics are formalized within the primitive meta-logic of the system.

De Bruijn - Memories of the Automath Project

Don't put logic into the system; let the user start his book with it.
Don't put induction and recursion in the system; consider it as book material, even when that might be slightly clumsier.

Apparently a very good idea

In practice, for the need to develop a coherent library, one ends up to push a single object-logic.

Logical frameworks

A logical framework is a system supporting a large class of logics. The object-logics are formalized within the primitive meta-logic of the system.

De Bruijn - Memories of the Automath Project

Don't put logic into the system; let the user start his book with it.
Don't put induction and recursion in the system; consider it as book material, even when that might be slightly clumsier.

Apparently a very good idea

In practice, for the need to develop a coherent library, one ends up to push a single object-logic.

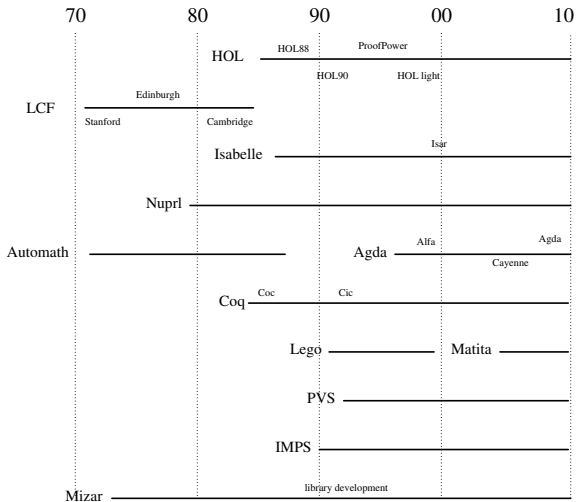
Logical frameworks

A logical framework is a system supporting a large class of logics. The object-logics are formalized within the primitive meta-logic of the system.

De Bruijn - Memories of the Automath Project

Don't put logic into the system; let the user start his book with it.
Don't put induction and recursion in the system; consider it as book material, even when that might be slightly clumsier.

Apparently a very good idea
In practice, for the need to develop a coherent library, one ends up to push a single object-logic.



In the middle of the 80's, there is a sudden proliferation of systems.

Trybulec (attributed to)

While in Mizar we have a library of mathematics, people in Type Theory has a library of systems.

L.C.Paulson - The next seven hundred provers (1988)

Logics are proliferating at an alarming rate; there are seven theorem provers descended from Edinburgh LCF. With Isabelle, you need only specify the logic's syntax and rules. To go beyond proof checking you can implement search procedures using built-in tools.

Paulson's dream mainly resulted in ...
a flurry of new Logical Frameworks (ALF, Elf, Twelf, ...).

In the middle of the 80's, there is a sudden proliferation of systems.

Trybulec (attributed to)

While in Mizar we have a library of mathematics, people in Type Theory has a library of systems.

L.C.Paulson - The next seven hundred provers (1988)

Logics are proliferating at an alarming rate; there are seven theorem provers descended from Edinburgh LCF. With Isabelle, you need only specify the logic's syntax and rules. To go beyond proof checking you can implement search procedures using built-in tools.

Paulson's dream mainly resulted in ...
a flurry of new Logical Frameworks (ALF, Elf, Twelf, ...).

In the middle of the 80's, there is a sudden proliferation of systems.

Trybulec (attributed to)

While in Mizar we have a library of mathematics, people in Type Theory has a library of systems.

L.C.Paulson - The next seven hundred provers (1988)

Logics are proliferating at an alarming rate; there are seven theorem provers descended from Edinburgh LCF. With Isabelle, you need only specify the logic's syntax and rules. To go beyond proof checking you can implement search procedures using built-in tools.

Paulson's dream mainly resulted in ...
a flurry of new Logical Frameworks (ALF, Elf, Twelf, ...).

In the middle of the 80's, there is a sudden proliferation of systems.

Trybulec (attributed to)

While in Mizar we have a library of mathematics, people in Type Theory has a library of systems.

L.C.Paulson - The next seven hundred provers (1988)

Logics are proliferating at an alarming rate; there are seven theorem provers descended from Edinburgh LCF. With Isabelle, you need only specify the logic's syntax and rules. To go beyond proof checking you can implement search procedures using built-in tools.

Paulson's dream mainly resulted in ...

a flurry of new Logical Frameworks (ALF, Elf, Twelf, ...).

In the middle of the 80's, there is a sudden proliferation of systems.

Trybulec (attributed to)

While in Mizar we have a library of mathematics, people in Type Theory has a library of systems.

L.C.Paulson - The next seven hundred provers (1988)

Logics are proliferating at an alarming rate; there are seven theorem provers descended from Edinburgh LCF. With Isabelle, you need only specify the logic's syntax and rules. To go beyond proof checking you can implement search procedures using built-in tools.

Paulson's dream mainly resulted in ...
a flurry of new Logical Frameworks (ALF, Elf, Twelf, ...).

system	logic	impl. language
HOL	Higher Order Logic	lisp (HOL88) SML (HOL90) ocaml (HOL light)
Isabelle	MetaLogic: I-HOL ObjectLogics: HOL, ZF, ...	SML
Nuprl	Martin-Löf predicative type theory	lisp
Coq, Lego Matita	Calculus of Inductive Constructions	ocaml
Agda	Martin-Löf predicative type theory with inductive types	haskell
PVS	HOL with subtyping and dependent types	lisp
IMPS	HOL with partial functions	T

ITPs Synopsis

<i>ITP</i>	<i>HOL</i>	<i>Isabelle</i>	<i>Coq</i>	<i>Nuprl</i>	<i>PVS</i>	<i>IMPS</i>
proof style	<i>P/D</i>	<i>P/D</i>	<i>P</i>	<i>P</i>	<i>P</i>	<i>P</i>
small kernel	•	•	•	•		
extensible	•	•	•	•	•	
powerful auto	•	•			•	•
decid. types	•	•	•			
depend. types			•	•	•	
reflection			•			

Types

Shankar and Owre, 1999

1. Types impose a useful discipline on the specification.
2. Types leads to easy and early detection of a large class of syntactic and semantic errors.
3. Type information is useful in mechanized reasoning.

Type Theory

- ▶ A foundational alternative to set theory.
- ▶ Interesting connections with constructive logic (Curry-Howard correspondence, Brouwer-Heyting-Kolmogov interpretation, realizability, categorical logic, ...)

The Curry-Howard correspondence is the relation between computer programs and mathematical proofs (i.e. between programming and proving).

Type \equiv Proposition
 $t:T$ \equiv proof of T

Type Theory

- ▶ A foundational alternative to set theory.
- ▶ Interesting connections with constructive logic (Curry-Howard correspondence, Brouwer-Heyting-Kolmogorov interpretation, realizability, categorical logic, ...)

The Curry-Howard correspondence is the relation between computer programs and mathematical proofs (i.e. between programming and proving).

Type \equiv Proposition
 $t:T$ \equiv proof of T

Type Theory

- ▶ A foundational alternative to set theory.
- ▶ Interesting connections with constructive logic (Curry-Howard correspondence, Brouwer-Heyting-Kolmogorov interpretation, realizability, categorical logic, ...)

The Curry-Howard correspondence is the relation between computer programs and mathematical proofs (i.e. between programming and proving).

$$\begin{array}{lcl} \text{Type} & \equiv & \text{Proposition} \\ t:T & \equiv & \text{proof of } T \end{array}$$

Higher Order Logic (Church)

Extend the simply typed lambda calculus with

- ▶ a basic type $prop$ of proposition
- ▶ a constant $\supset: prop \rightarrow prop \rightarrow prop$ (implication)
- ▶ a constant $\forall_\sigma: (\sigma \rightarrow prop) \rightarrow prop$ (universal quantification)

notation:

$$\forall x: \sigma. \varphi \equiv \forall(\lambda x: \sigma. \varphi)$$

Higher Order Logic (Church)

Extend the simply typed lambda calculus with

- ▶ a basic type $prop$ of proposition
- ▶ a constant $\supset: prop \rightarrow prop \rightarrow prop$ (implication)
- ▶ a constant $\forall_\sigma: (\sigma \rightarrow prop) \rightarrow prop$ (universal quantification)

notation: $\forall x: \sigma. \varphi \equiv \forall(\lambda x: \sigma. \varphi)$

examples

► Induction

$$\forall P : nat \rightarrow prop.$$
$$P(0) \rightarrow (\forall x : nat. P(x) \supset P(S(x))) \rightarrow \\ \forall x : nat. P(x)$$

► Transitive closure of a relation R

$$\lambda R : \sigma \rightarrow sigma \rightarrow prop. \lambda x, y : \sigma.$$
$$\forall Q : \sigma \rightarrow \sigma \rightarrow Prop. (trans(Q) \supset (R \subseteq Q) \supset Q \times y$$

of type $(\sigma \rightarrow \sigma \rightarrow Prop) \rightarrow (\sigma \rightarrow \sigma \rightarrow Prop)$

(*axiom*) $\Delta \vdash \varphi$ if $\varphi \in \Delta$

(\supset -*intro*) $\frac{\Delta, \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi}$

(\supset -*elim*) $\frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$

(\forall -*intro*) $\frac{\Delta \vdash \varphi}{\Delta \vdash \forall x : \sigma. \varphi}$ if $x : \sigma \notin FV(\Delta)$

(\forall -*elim*) $\frac{\Delta \vdash \forall x : \sigma. \varphi}{\Delta \vdash \varphi[t/x]}$ if $t : \sigma$

(*conversion*) $\frac{\Delta \vdash \varphi}{\Delta \vdash \psi}$ if $\varphi =_{\beta} \psi$

About the conversion rule

$$\frac{\frac{\Delta \vdash \forall P : \text{nat} \rightarrow \text{Prop} . (\dots Pc \dots)}{\Delta \vdash (\dots (\lambda x : \text{nat} . x > 0) c \dots)}}{\Delta \vdash (\dots (c > 0) \dots)}$$

Poincaré principle

Perform computations in a completely automatic way.

H.Poincaré 1902, talking about the proof of $2 + 2 = 4$

Ce n'est pas une démonstration proprement dite, [...] c'est une vérification.[...]La verification differe precisément del la véritable démonstration, parce qu'elle est purement analytique et parce qu'elle est stérile.

By enhancing the expressive power of the logical system (recursive definitions, small reflection, etc) one can tackle at the logical level, via reduction and conversion, the so called Poincaré principle

(but a similar behaviour may be achieved via rewriting).

Poincaré principle

Perform computations in a completely automatic way.

H.Poincaré 1902, talking about the proof of $2 + 2 = 4$

Ce n'est pas une démonstration proprement dite, [...] c'est une vérification.[...]La verification differe precisément del la véritable démonstration, parce qu'elle est purement analytique et parce qu'elle est stérile.

By enhancing the expressive power of the logical system (recursive definitions, small reflection, etc) one can tackle at the logical level, via reduction and conversion, the so called Poincaré principle

(but a similar behaviour may be achieved via rewriting).

Definability of other connectives

Idea: to define a connective encode its elimination rule.

$$\begin{aligned}\perp &\equiv \forall \alpha : \mathit{prop}.\alpha \\ \varphi \wedge \psi &\equiv \forall \alpha : \mathit{prop} . (\varphi \supset \psi \supset \alpha) \supset \alpha \\ \varphi \vee \psi &\equiv \forall \alpha : \mathit{prop} . (\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha \\ \exists x : \sigma . \varphi &\equiv \forall \alpha : \mathit{prop} . (\forall x : \sigma . \varphi \supset \alpha) \supset \alpha\end{aligned}$$

Leibniz equality

Equality is definable in higher order logic:

Leibniz

Two terms are equal if they share the same properties.

$$x =_{\sigma} y \equiv \forall P : \sigma \rightarrow \text{prop}. (Px \supset Py)$$

This equality is congruence.

Intensionality and extensionality

Leibniz equality is an **intensional** equality. E.g. equality of functions is the equality of their algorithms, and not of their graphs.

The following extensionality rule consistent with HOL

$$\frac{\Delta \vdash \forall x : \sigma. fx =_{\tau} gx}{f =_{\sigma \rightarrow \tau} g}$$

... but type checking becomes undecidable.

Nuprl is based on an extensional type theory.

Intensionality and extensionality

Leibniz equality is an **intensional** equality. E.g. equality of functions is the equality of their algorithms, and not of their graphs.

The following extensionality rule consistent with HOL

$$\frac{\Delta \vdash \forall x : \sigma. fx =_{\tau} gx}{f =_{\sigma \rightarrow \tau} g}$$

... but type checking becomes undecidable.

Nuprl is based on an extensional type theory.

Intensionality and extensionality

Leibniz equality is an **intensional** equality. E.g. equality of functions is the equality of their algorithms, and not of their graphs.

The following extensionality rule consistent with HOL

$$\frac{\Delta \vdash \forall x : \sigma. fx =_{\tau} gx}{f =_{\sigma \rightarrow \tau} g}$$

... but type checking becomes undecidable.

Nuprl is based on an extensional type theory.

Intensionality and extensionality

Leibniz equality is an **intensional** equality. E.g. equality of functions is the equality of their algorithms, and not of their graphs.

The following extensionality rule consistent with HOL

$$\frac{\Delta \vdash \forall x : \sigma. fx =_{\tau} gx}{f =_{\sigma \rightarrow \tau} g}$$

... but type checking becomes undecidable.

Nuprl is based on an extensional type theory.

Conclusions

De Bruijn (2003) - Memories of the AUTOMATH Project

If you can't explain your mathematics to a machine it is an illusion to think you can explain it to a student.

Writing mathematics in a way understandable by a computer is still an **extremely expensive** operation (2 hours per source line).

Conclusions

De Bruijn (2003) - Memories of the AUTOMATH Project

If you can't explain your mathematics to a machine it is an illusion to think you can explain it to a student.

Writing mathematics in a way understandable by a computer is still an **extremely expensive** operation (2 hours per source line).

Conclusions

De Bruijn (2003) - Memories of the AUTOMATH Project

If you can't explain your mathematics to a machine it is an illusion to think you can explain it to a student.

Writing mathematics in a way understandable by a computer is still an **extremely expensive** operation (2 hours per source line).

Future work

Main ITP goal: reduce the cost of formalization and improve the benefit.

Constable et al. 1986

The natural growth path for a system like Nuprl tends toward increased “intelligence”. [...] For example, it is helpful if the system is aware of what is in the library and what users are doing with it. It is good if the user knows when to involve certain tactics, but once we see a pattern to this activity, it is easy and natural to inform the system about it. Hence there is an impetus to give the system more knowledge about itself.

Future work

Main ITP goal: reduce the cost of formalization and improve the benefit.

Constable et al. 1986

The natural growth path for a system like Nuprl tends toward increased “intelligence”. [...] For example, it is helpful if the system is aware of what is in the library and what users are doing with it. It is good if the user knows when to involve certain tactics, but once we see a pattern to this activity, it is easy and natural to inform the system about it. Hence there is an impetus to give the system more knowledge about itself.

Future work

Main ITP goal: reduce the cost of formalization and improve the benefit.

Constable et al. 1986

The natural growth path for a system like Nuprl tends toward increased “intelligence”. [...] For example, it is helpful if the system is aware of what is in the library and what users are doing with it. It is good if the user knows when to involve certain tactics, but once we see a pattern to this activity, it is easy and natural to inform the system about it. Hence there is an impetus to give the system more knowledge about itself.



D.Aspinall. Proof General: a Generic Tool for Proof Development. Proceedings of TACAS'2000, LNCS 1785, Springer Verlag, 2000.



R.L.Constable et al. Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall 1986.



N.G.De Bruijn. Memories of the Automath Project. Invited Lecture at the Mathematics Knowledge Management Symposium, 25-29 November 2003, Heriot-Watt University, Edinburgh, Scotland.



W.M.Farmer, J.D.Guttman, F.J.Thayer. IMPS: An Interactive Mathematical Proof System. J.Autom.Reasoning 11(2), pp.213-248, 1993.



W.M.Farmer. A Partial Functions Version of Church's Simple Theory of Types. The Journal of Symbolic Logic, Vol.55, N.3, 1990.



M.Gordon. From LCF to HOL: a short history. In [7]



D.Griffioen, M.Huisman. A comparison of PVS and Isabelle/HOL. Proceedings of TPHOLS'98, LNCS 1479, 1998.














J.Harrison. A Short Survey of Automated Reasoning. Proceedings of AB 2007, LNCS 4545, Springer Verlag 2007.



G.Huet, G.Plotkin (editors). Logical Frameworks. Cambridge University Press, 1991.



G.Huet, G.Plotkin (editors). Logical Environments. Cambridge University Press, 1993.

-  L.Lamport, L.C.Paulson. Should Your Specification Language Be Typed? SRC Research Report n.147, 1998.
-  M.Lecat. Erreurs des Mathématiciens des origins á nos jours. Ancienne Librairie Castaigne, Brussels, 1935.
-  S.Owre, J.M.Rushby, N.Shankar. PVS: a Prototype Verification System. Proceeding of CADE'92, LNCS 607, Springer Verlag, 1992.
-  S.Owre, N.Shankar. The formal semantics of PVS. Technical Report CSL-97-2R, SRI International.
-  L.C.Paulson. Isabelle: the next seven hundred theorem provers. Proceedings of CADE'88, LNCS 310, Springer Verlag, 1988.
-  L.C.Paulson. The foundation of a generic theorem prover. Journal of Automated Reasoning, V.5, n.3, 1989.
-  G.Plotkin, C.P.Stirling, M.Toft (editors). Proof, Language and Interaction. MIT Press, 2000.
-  P.Rudnicki. An Overview of the Mizar Project. Proceedings of the Workshop on Types for Proofs and Programs, Chalmers University, Bastad, 1992.
-  .Wang. Toward mechanical mathematics. IBM Journal of research and development 4, 1960.
-  F.Wiedijk. Comparing Mathematical Provers. Proceedings of MKM03, LNCS 2594, Spriger Verlag, 2003.
-  F.Wiedijk (editor). The Seventeen Provers of the World. LNCS 3600, Spriger Verlag, 2006.