Strategic Principles in the Design of Isabelle

Lawrence C. Paulson

Computer Laboratory

University of Cambridge

Research supported by the EPSRC and ESPRIT
Proof Assistants: A Strategic View

Strength over the long term

• automation: essential in an interactive tool

• flexibility: for the differing needs of users
  – control over syntax
  – a choice of logical formalisms (logical framework!)
  – a toolkit for proof strategies

• soundness needs a small trusted kernel
Automation & Flexibility... How?

- higher-order syntax
- logical variables and unification
- search primitives based on lazy lists

(Can logical frameworks really work?)

a sort of higher-order Prolog (like Dale Miller's $\lambda$Prolog)
Higher-Order Syntax: A Must!

Flexibility: users can define new variable binders

\[
\text{least } n \cdot P(n) \quad \{x \in A \mid P(x)\} \bigcup_{x \in A} B(x)
\]

case \(l\) of \([\]
\(\Rightarrow\) \(z\) \mid x \# l' \(\Rightarrow\) \(f(x, l')\)

Doesn’t require higher-order logic

Alternatives?? Combinators or auxiliary functions
Logical Variables

- don’t know subterms can be left unspecified ...
- ... until unification completes them
- helpful for proof procedures
- declarative representation of rules

rare in higher-order proof tools
Declarative Rules

Define the quantifier $\forall x \in A \ P(x)$ to be $\forall x \ [x \in A \rightarrow P(x)]$

Derive the rule

$$\frac{\forall x \in A \ P(x) \quad a \in A}{P(a)}$$

Can be displayed and transformed and combined (resolution!)

Alternative representations: code, or higher-order formula
Higher-Order + Logical Variables = ?

Higher-order unification (Huet, 1975)

In the worst case...

- infinitely many unifiers
- semi-decidable
- complicated algorithm

Pattern unification handles the easy cases (Miller’s $L_\lambda$)
Tactics Based on Lazy Lists

Tactics describe the search space

- proof state $\rightarrow$ list of proof states
- result is a lazy list

Tacticals explore the search space

- tactic $\rightarrow$ tactic
- strategies: depth-first, best-first, iterative deepening, …

Strategies are easily combined
Tableaux-style provers for intuitionistic and classical FOL

The MESON proof procedure (world’s slowest!)

A generic classical reasoner (here, in ZF set theory):

\[ C \neq \emptyset \rightarrow \bigcap_{x \in C} [A(x) \cap B(x)] = \left( \bigcap_{x \in C} A(x) \right) \cap \left( \bigcap_{x \in C} B(x) \right) \]

1/2 second on Pentium
More Automation: Inductive Definitions

To formalize

- **operational semantics**: languages, type theories, ...
- **proof systems**
- **security**

Induction rules proved, not assumed

Proofs generated using tactics & tacticals

Keep the trusted kernel small
Some Applications

- temporal reasoning: UNITY, TLA, ... (TUM and Cambridge)
- combinations of non-classical logics (MPI-Saarbrücken)
- verification of cryptographic protocols (Cambridge)
- Java type safety (TUM)
Compiler
Java JVM
Bali BVM
Operational Semantics
Type System
Bytecode Verifier
Type Safety?
Correctness?

Operational Semantics
Type System
Bali
Java

Operational Semantics
Bytecode Verifier
Correctness?
Compiler
BVM
JVM
Bali: a large subset of Java

- class, interface, field & method
- inheritance, overriding, & hiding
- overloading, dynamic binding, exceptions...

Bali Virtual Machine

- OO concepts (as above)
- integers & arrays
- predefined exceptions
Cornelia Pusch: Isabelle proof of

\[ ok(\text{bytecode}) \Rightarrow \text{no runtime type error} \]

Bali

Formalization: 1200 lines 5 weeks
Proof of type safety: 2400 lines 10 weeks

BVM

Formalization BVM: 1100 lines 7 weeks
Formalization BV: 600 lines 5 weeks
Proof of type safety: 3000 lines 8 weeks
Can Cryptography Make Networks Secure?

Goals of security protocols:

- **Authenticity**: who sent this message?
- **Secrecy**: who can receive my message?

Threats:

- **Active** attacker
- Careless & compromised agents  … NO code-breaking
The Needham-Schroeder Protocol (1978)

1. \( A \rightarrow B : \{ Na, A \}_Kb \)

Alice sends Bob an encrypted nonce

2. \( B \rightarrow A : \{ Na, Nb \}_Ka \)

Bob returns \( Na \) with a nonce of his own

3. \( A \rightarrow B : \{ Nb \}_Kb \)

Alice returns Bob’s nonce
A Middle-Person Attack (1995)

Villain Charlie can masquerade as Alice to Bob

\[
\begin{align*}
A &\xrightarrow{\{A,Na\}K_c} C \xleftarrow{\{Nb\}K_c} B \\
A &\xrightarrow{\{A,Na\}K_b} C \xleftarrow{\{Nb\}K_b} B
\end{align*}
\]

Gavin Lowe found this attack 17 years later!
Verification Methods

- Logics of belief (BAN, 1989)
  - Allows short, abstract proofs but misses many flaws

- State enumeration
  - Automatically finds attacks but requires strong assumptions

- Inductive protocol verification
  - Trace model of agents
  - Proofs mechanized using Isabelle/HOL
Protocol Verification: Results

- industrial protocols analyzed (TLS, Kerberos, …)
- minutes CPU time, weeks human time per protocol
- the power of
  - inductive definitions
  - the simplifier
  - the classical reasoner
- substantial proofs found automatically
Conclusions

- **logical frameworks** can be practical
- **lazy lists** give the needed **flexibility**
- **higher-order syntax** can be combined with **logical variables**
- **ATP techniques** can be used in an interactive tool

... plus a lot of **hard work** to make it go!