paraVerifier: An Automatic Framework for Proving Parameterized Cache Coherence Protocols

Yongjian Li[1,3] Jun Pang[2] Yi Lv[1] Dongrui Fan[4]
Shen Cao[1] Kaiqiang Duan[1]

State Key Laboratory of Computer Science, China

Computer Science and Communications, University of Luxembourg, Luxembourg

College of Information Engineering, Capital Normal University, Beijing, China

Institute of Computing Technology, Chinese Academy of Sciences, China

2016 年 9 月 12 日

Theorem prover(Proof assistant)

“A theorem prover is a computer program used interactively for developing human-readable reliable mathematical documents in a formal language.”

- computer program (working mechanically)
- interacting with people
- a formal proof script as an output

Main Ideas

“A theorem prover is a computer program used interactively for developing human-readable reliable mathematical documents in a formal language.”

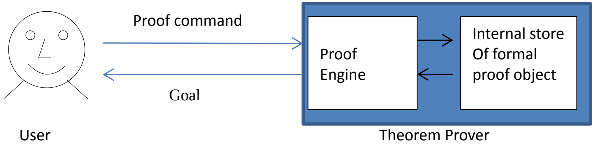
- Formal logical calculus
- Assistant people’s formal logical calculus by a computer

Leibniz's opnion on formula logic

"Leibniz enunciated the principal properties of what we now call conjunction, disjunction, negation, identity, set inclusion, and the empty set. The principles of Leibniz's logic and, arguably, of his whole philosophy, reduce to two:"

- All our ideas are compounded from a very small number of simple ideas, which form the alphabet of human thought.
- Complex ideas proceed from these simple ideas by a uniform and symmetrical combination, analogous to arithmetical multiplication.

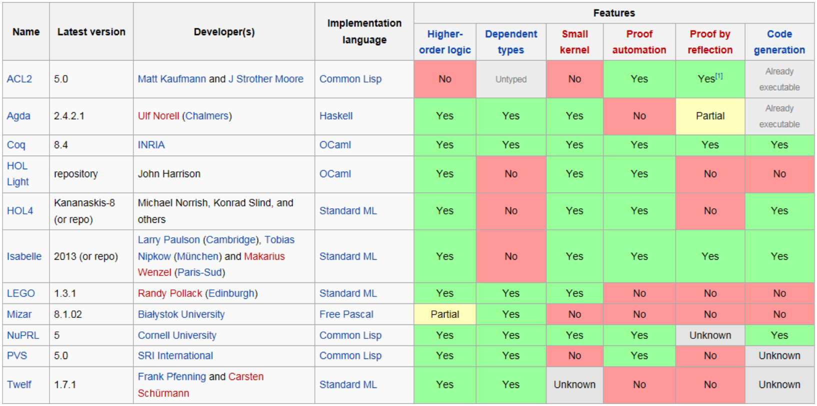
Theorem proving



Main theorem provers

- Isabelle
- Coq
- HOL4
- PVS
- ....

Main theorem provers



Name	Latest version	Developer(s)	Implementation language	Features					
				Higher-order logic	Dependent types	Small kernel	Proof automation	Proof by reflection	Code generation
ACL2	5.0	Matt Kaufmann and J Strother Moore	Common Lisp	No	Untyped	No	Yes	Yes[1]	Already executable
Agda	2.4.2.1	Ulf Norell (Chalmers)	Haskell	Yes	Yes	Yes	No	Partial	Already executable
Coq	8.4	INRIA	OCaml	Yes	Yes	Yes	Yes	Yes	Yes
HOL Light	repository	John Harrison	OCaml	Yes	No	Yes	Yes	No	No
HOL4	Kananaskis-8 (or repo)	Michael Norrish, Konrad Slind, and others	Standard ML	Yes	No	Yes	Yes	No	Yes
Isabelle	2013 (or repo)	Larry Paulson (Cambridge), Tobias Nipkow (München) and Makarius Wenzel (Paris-Sud)	Standard ML	Yes	No	Yes	Yes	Yes	Yes
LEGO	1.3.1	Randy Pollack (Edinburgh)	Standard ML	Yes	Yes	Yes	No	No	No
Mizar	8.1.02	Białystok University	Free Pascal	Partial	Yes	No	No	No	No
NuPRL	5	Cornell University	Common Lisp	Yes	Yes	Yes	Yes	Unknown	Yes
PVS	5.0	SRI International	Common Lisp	Yes	Yes	No	Yes	No	Unknown
Twelf	1.7.1	Frank Pfenning and Carsten Schürmann	Standard ML	Yes	Yes	Unknown	No	No	Unknown

Use of theorem prover

- Formalizing mathematics and mathematical libraries
  - Mata theories: Set theory, LCF, ZF, ...
  - Some advanced theories: Kepler guess, Four-coloured problems
- Formal verification of system (hardware design, program, algorithm, system)

The Kepler Conjecture

“The Kepler Conjecture says that the ?cannonball packing? (see picture) is a densest packing of 3-dimensional balls of the same size. This was stated as a fact by Kepler in 1611 but only proved by Thomas Hales in 1998. His proof relies on a Java program for generating all (3000) possible counterexamples (all of which are then shown not to be counterexamples). With the help of Isabelle we were able to prove the correctness of a functional implementation of his Java program. Listen to Thomas Hales speaking about the proof (ABC Radio National Science Show, March 11th 2006). A formal proof of the Kepler conjecture was completed in 2014.”

Essentials of formal verification (by a theorem prover)

- Formally model the system (by a formula)
- Formalize the specification (by a formula)
- Prove that the model satisfies the spec (logical deduction)

A case study—Formal verification of anonymity protocols (by a theorem prover)

```
constdefs box::"agent⇒trace⇒trace set⇒  assertOfTrace⇒bool"
 "box A r rs Assert≡  ∀r'.r'∈rs⟶obsEquiv A r r' ⟶(Assert r')"


constdefs diamond::"agent⇒trace⇒trace set⇒  assertOfTrace⇒bool"
 "diamond A r rs Assert≡  ∃r'.r'∈rs ∧obsEquiv A r r'  ∧(Assert r')"
```

Formalization of anonymity properties

```
constdefs senderAnomity::"agent set⇒agent⇒msg⇒
 trace⇒trace set⇒bool"
 "senderAnomity AS B m r rs≡ (∀X.X∈AS⟶  r ⊨ ◇B rs (originates X m))"

constdefs unlinkability::"agent set⇒agent⇒msg⇒
 trace⇒trace set⇒bool"
 "unlinkability AS A m r rs≡
  (let P= λX m' r.  sends X m' r in  (¬ (r ⊨ Spy rs (P A m)) ∧
  (∀X.X∈AS ⟶r ⊨ ◇Spy rs (P A m)))"
```

Modelling Onion Routing Protocols

```
--- Formal inductive definition inductive_set oneOnionSession::"nat⇒agent⇒tra
for k::"nat" and M::"agent" where
 onionNil:  "[]∈ (oneOnionSession k M) "
 | onionCons1:  "tr∈(oneOnionSession k M);XM;YM;
 Nonce n0∉(used tr);Nonce n∉(used tr);
 length tr<k⟹
 Says X M (Crypt (pubK M)
```
$\{Noncen0, AgentY, Crypt(pubKY)(Noncen)\}$)

#tr∈ *oneOnionSessionkM*"

```
 | onionCons2:  "tr∈(oneOnionSession k M);XM;
 Nonce n∉(used tr);length tr<k⟹
 Says X M (Crypt (pubK M) (Nonce n))
```
#tr∈ *oneOnionSessionkM*"

```
 | onionCons3:  "tr∈(oneOnionSession k M);
 length tr≥k;
```