FIRST-ORDER LOGIC: SOFTWARE FOR PROVING

Course “Computational Logic”

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Software for Proving

We will give examples for two classes of such software.

- Automated Theorem Provers
  - Fully automatic execution (until successful termination or timeout/abortion).
  - Logics with complete inference systems (mostly).
    - Typically: first-order logic (FOL) with equality.
  - Example: Vampire (others: E, Prover9, Waldmeister, SPASS, Theorema, ...)
- Proof Assistants
  - User-guided proof elaboration.
    - Supported by automatic proof tactics, decision procedures, external provers.
  - Also logics without complete inference systems.
    - Typically: higher-order logic (HOL).
  - Example: Isabelle/HOL (others: Coq, HOL4, HOL Light, ACL2, PVS, ...)

While automated theorem provers have become very strong, complex problems still require the application of proof assistants.

Vampire

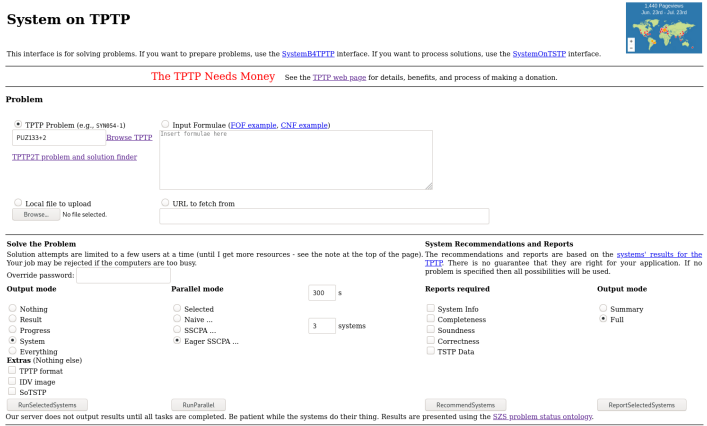
Software: https://vprover.github.io/
Tutorial: First-Order Theorem Proving and Vampire (Laura Kovács and Andrei Voronkov)

- Automated theorem prover
  - A. Voronkov (Univ. Manchester), K. Hoder, L. Kovács, I. Dragan, and others.
  - First implementation in 1993, current implementation started in 2009.
- First-order logic with equality
  - Based on the inference calculi of resolution and superposition.
  - Also supports various theories such as linear integer arithmetic.
- Very efficient algorithms and fast C++ implementation.
  - Regular winner of the CASC categories FOF and TFA.
    - CASC (CADE ATP System Competition): https://tptp.org/CASC
- TPTP and SMTLIB languages supported.
  - Established standards in automated theorem proving.

Vampire represents the state of the art in automated first-order proving.

TPTP: Thousands of Problems for Theorem Provers

https://tptp.org https://tptp.org/cgi-bin/SystemOnTPTP



A huge library of test problems and a web interface to numerous provers.

The TPTP Language

https://tptp.org/TPTP/QuickGuide https://tptp.org/TPTP/SyntaxBNF.html

```
<TPTP_file>     ::= <fof_annotated>*
<fof_annotated> ::= fof(<name>,<formula_role>,<fof_formula> <annotations>).
<formula_role>  :== axiom | conjecture | negated_conjecture | ...
...
<fof_quantified_formula> ::= <fof_quantifier> [<fof_variable_list>] : <fof_unit_formula>
<fof_quantifier>         ::= ! | ?
<fof_variable_list>      ::= <variable> | <variable>,<fof_variable_list>
...
<connective>    ::= ~ | & | [|] | => | <= | <=> |  ~& | ~[|] | <~>
...
<fof_plain_atomic_formula> :== <atomic_word> | <atomic_word>(<fof_arguments>)
<fof_term>                 ::= <variable> | <atomic_word> | <atomic_word>(<fof_arguments>)
<fof_arguments>            ::= <fof_term> | <fof_term>,<fof_arguments>
...
<variable>                 ::= <upper_word>
<atomic_word>              ::= <lower_word> | <single_quoted>
```

FOF: first-order form (TFF: typed first-order form).

Syntactic Pitfalls

WARNING: several common errors give formulas unintended meanings.

- Missing Parentheses: precedence among binary connectives is *undefined*.

```
(~..) (..&..) (..|..) (..=>..) (..<=>..) (![X]:..) (?[X]:..)
```

- Variables Not Capitalized: treated as fresh constants.

```
![X,Y]:(..x..Y..)
```

- Constants Capitalized: treated as universally quantified variables.

```
.. Zero ..
```

- Misspellings: introduce uninterpreted constants, functions, predicates.

```
.. bijective_function_f .. bijectiv_function_f ..
```

In FOF, identifiers are “implicitly declared”, no static checks prevent misspellings (which generally lead to failed proofs or to successful but meaningless proofs)!

A First-Order Problem

```
% file fol6a.p: a simple first-order problem

% the assumptions (note the parentheses around the body of a quantified formula!)
fof(a1, axiom, p(a) | q(b)).
fof(a2, axiom, ![X]: (p(X) => r(X))).
fof(a3, axiom, ![X]: (q(X) => r(f(x)))).

% the goal to prove
fof(g, conjecture, ?[X]: (r(X))).

debian10!1> vampire -om smtcomp fol6a.p
unsat
```

The conjunction of the axioms and the negation of the conjecture is unsatisfiable, thus the conjecture is a logical consequence of the axioms.

Semantic Pitfalls

The axioms might be inconsistent and thus prove every goal.

```
% file fol6a.p: a simple first-order problem

% the assumptions (note the parentheses around the body of a quantified formula!)
fof(a1, axiom, p(a) | q(b)).
fof(a2, axiom, ![X]: (p(X) => r(X))).
fof(a3, axiom, ![X]: (q(X) => r(f(x)))).

% the goal to prove (commented out)
% fof(g, conjecture, ?[X]: (r(X))).

debian10!1> vampire -om smtcomp fol6a.p
sat
```

After the successful proof of a conjecture, better check that this success is not just achieved from inconsistent axioms.

Proof Statistics for the Problem

```
debian10!1> vampire --proof off fol6a.p
% Refutation found. Thanks to Tanya!
% SZS status Theorem for fol6a
% ------------------------------
% Version: Vampire 4.5.1 (commit 57a6f78c on 2020-07-15 11:59:04 +0200)
% Termination reason: Refutation

% Memory used [KB]: 4861
% Time elapsed: 0.028 s
% ------------------------------
% ------------------------------
```

“Refutation found”: the system could prove the unsatisfiability of the conjunction of the axioms and the negation of the conjecture.

A Proof of the Problem

```
debian10!1> vampire fol6a.p
% Refutation found. Thanks to Tanya!
% SZS status Theorem for fol6a
% SZS output start Proof for fol6a
1. q(b) | p(a) [input]
2. ! [X0] : (p(X0) => r(X0)) [input]
3. ! [X0] : (q(X0) => r(f(x))) [input]
4. ? [X0] : r(X0) [input]
5. ~? [X0] : r(X0) [negated conjecture 4]
6. ! [X0] : (r(X0) | ~p(X0)) [ennf transformation 2]
7. ! [X0] : (r(f(x)) | ~q(X0)) [ennf transformation 3]
8. ! [X0] : ~r(X0) [ennf transformation 5]
9. q(b) | p(a) [cnf transformation 1]
10. ~p(X0) | r(X0) [cnf transformation 6]
11. r(f(x)) | ~q(X0) [cnf transformation 7]
12. ~r(X0) [cnf transformation 8]
14. 1 <=> p(a) [avatar definition]
16. p(a) <- (1) [avatar component clause 14]
18. 2 <=> q(b) [avatar definition]
20. q(b) <- (2) [avatar component clause 18]
21. 1 | 2 [avatar split clause 9,18,14]
23. 3 <=> ! [X0] : ~q(X0) [avatar definition]
24. ~q(X0) <- (3) [avatar component clause 23]
26. 4 <=> r(f(x)) [avatar definition]
28. r(f(x)) <- (4) [avatar component clause 26]
29. 3 | 4 [avatar split clause 11,26,23]
30. $false <- (2, 3) [subsumption resolution 20,24]
31. ~2 | ~3 [avatar contradiction clause 30]
32. r(a) <- (1) [resolution 10,16]
33. $false <- (1) [subsumption resolution 32,12]
34. ~1 [avatar contradiction clause 33]
35. $false <- (4) [subsumption resolution 28,12]
36. ~4 [avatar contradiction clause 35]
37. $false [avatar sat refutation 21,29,31,34,36]
% SZS output end Proof for fol6a
% ------------------------------
% Version: Vampire 4.5.1 (commit 57a6f78c on ...)
% Termination reason: Refutation

% Memory used [KB]: 4861
% Time elapsed: 0.033 s
% ------------------------------
% ------------------------------
```

A detailed description of every transformation step and every inference step in a proof based on the principle of “resolution”.

A Proof for External Checking

```
debian10!1> vampire --proof proofcheck fol6a.p
% Refutation found. Thanks to Tanya!
% SZS status Theorem for fol6a
% SZS output start Proof for fol6a
fof(r30,conjecture, $false ). %subsumption resolution
fof(pr20,axiom, q(b) ).
fof(pr24,axiom, ( ! [X0] : (~q(X0)) ) ).
%#
fof(r32,conjecture, r(a) ). %resolution
fof(pr10,axiom, ( ! [X0] : (~p(X0) | r(X0)) ) ).
fof(pr16,axiom, p(a) ).
%#
fof(r33,conjecture, $false ). %subsumption resolution
fof(pr32,axiom, r(a) ).
fof(pr12,axiom, ( ! [X0] : (~r(X0)) ) ).
%#
fof(r35,conjecture, $false ). %subsumption resolution
fof(pr28,axiom, r(f(x)) ).
fof(pr12,axiom, ( ! [X0] : (~r(X0)) ) ).
%#
% ...
```

The resolution proof as a sequence of TPTP problems for external checking.

A Problem in First-Order Logic with Equality

```
% fol6b.p: the group theory example from the tutorial paper.
% "if all elements in a group have order 2, then the group is commutative"

% the group axioms
fof(leftidentity,axiom, ![X]:(mult(e,X) = X)).
fof(leftinverse,axiom, ![X]:(mult(inverse(X),X) = e)).
fof(associativity,axiom, ![X,Y,Z]:(mult(mult(X,Y),Z) = mult(X,mult(Y,Z)))).

% all elements have order two
fof(groupoforder2,hypothesis, ![X]:(mult(X,X) = e)).

% the group is commutative
fof(commutativity,conjecture, ![X]:(mult(X,Y) = mult(Y,X))).

debian10!1> vampire -om smtcomp fol6b.p
unsat
```

The atomic predicate = has the fixed interpretation “equality”.

A Proof of the Problem

```
debian10!1> vampire  fol6b.p
% Refutation found. Thanks to Tanya!
% SZS status Theorem for fol6b
% SZS output start Proof for fol6b
1. ! [X0] : mult(e,X0) = X0 [input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))
  [input]
4. ! [X0] : e = mult(X0,X0) [input]
5. ! [X0] : mult(X0,X1) = mult(X1,X0) [input]
6. ~! [X1] : ! [X0] : mult(X0,X1) = mult(X1,X0)
  [negated conjecture 5]
7. ~! [X0] : ! [X1] : mult(X0,X1) = mult(X1,X0) [rectify 6]
8. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [flattening 7]
9. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 8]
10. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) => mult(sK0,sK1) !=
  mult(sK1,sK0) [choice axiom]
11. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 9,10]
12. mult(e,X0) = X0 [cnf transformation 1]
13. e = mult(inverse(X0),X0) [cnf transformation 2]
14. mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))
  [cnf transformation 3]
15. e = mult(X0,X0) [cnf transformation 4]
16. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 11]
17. mult(X0,mult(X0,X1)) = mult(e,X1) [superposition 14,15]
19. mult(inverse(X4),mult(X4,X5)) = mult(e,X5)
  [superposition 14,13]
21. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 14,15]
23. mult(X0,mult(X0,X1)) = X1 [forward demodulation 17,12]
24. mult(inverse(X4),mult(X4,X5)) = X5
  [forward demodulation 19,12]
29. mult(inverse(X2),e) = X2 [superposition 23,13]
77. mult(X4,mult(X3,X4)) = mult(inverse(X3),e)
  [superposition 24,21]
92. mult(X4,mult(X3,X4)) = X3 [forward demodulation 77,29]
127. mult(X2,X3) = mult(X3,X2) [superposition 23,92]
271. mult(sK0,sK1) != mult(sK0,sK1) [superposition 16,127]
272. $false [trivial inequality removal 271]
% SZS output end Proof for fol6b
% ------------------------------
% Version: Vampire 4.5.1 (commit 57a6f78c on ...)
% Termination reason: Refutation

% Memory used [KB]: 5117
% Time elapsed: 0.037 s
% ------------------------------
% ------------------------------
```

Equality reasoning based on the principle of “superposition”.

An Incorrect Conjecture

```
% file fol6c.p: an incorrect conjecture
fof(a1, axiom, p(a) | q(b)).
fof(a2, axiom, ![X]: (p(X) => r(X))).
fof(g, conjecture, ?[X]: (r(X))).

debian10!1> vampire fol6c.p
% SZS status CounterSatisfiable for fol6c
% # SZS output start Saturation.
% # SZS output end Saturation.
% ------------------------------
% Version: Vampire 4.5.1 (commit 57a6f78c on 2020-07-15 11:59:04 +0200)
% Termination reason: Satisfiable

% Memory used [KB]: 4861
% Time elapsed: 0.029 s
% ------------------------------
% ------------------------------
```

The system *may* be also able to show that the conjecture is *incorrect* by a model that satisfies the conjunction of the axioms and of the negated conjecture.

An Incorrect Conjecture

Sometimes the counterexample model is reasonably intuitive.

```
debian10!1> vampire --mode casc_sat fol6c.p
...
% SZS status CounterSatisfiable for fol6c
% # SZS output start Saturation.
cnf(u7,negated_conjecture,
    ~r(X0)).

cnf(u11,negated_conjecture,
    q(b)).

cnf(u10,negated_conjecture,
    ~p(X0)).
...
```

A counterexample model where $q(b)$ is true and $p(x)$ and $r(x)$ are false for all x.

Problem: Does Superman Exist?

Consider the following assumptions:

1. If Superman were able and willing to prevent evil, he would do so.
2. If Superman were unable to to prevent evil, he would be impotent.
3. If Superman were unwilling to prevent evil, he would be malevolent.
4. Superman does not prevent evil.
5. If Superman exists, he is neither impotent nor malevolent.

Prove the following conjecture: “Superman does not exist”.

```
% file fol6f.p: does superman exist?
fof(a1, axiom, ![X]:(superman(X) => ((able(X) & willing(X)) => prevent(X)))).
fof(a2, axiom, ![X]:(superman(X) => ((~able(X)) => impotent(X)))).
fof(a3, axiom, ![X]:(superman(X) => ((~willing(X)) => malevolent(X)))).
fof(a4, axiom, ![X]:(superman(X) => (~prevent(X)))).
fof(a5, axiom, ![X]:(superman(X) => ((~impotent(X)) & (~malevolent(X))))).
fof(g, conjecture, ~?[X]:(superman(X))).

debian10!1> vampire -om smtcomp fol6f.p
unsat
```

Problem: Who Killed Agatha?

Pelletier Problem 55 (Francis Jeffry Pelletier, 1986):

> Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler.

Prove the following conjecture: “Agatha killed herself”.

Solution

```
% fol6d.p Pelletier problem 55 (TPTP problem PUZ001+1)
fof(pel55_1,   axiom, ?[X]:(lives(X) & killed(X,agatha))).
fof(pel55_2_1, axiom, lives(agatha)).
fof(pel55_2_2, axiom, lives(butler)).
fof(pel55_2_3, axiom, lives(charles)).
fof(pel55_3,   axiom, ![X]:(lives(X) => (X = agatha | X = butler | X = charles))).
fof(pel55_4,   axiom, ![X,Y]:(killed(X,Y) => hates(X,Y))).
fof(pel55_5,   axiom, ![X,Y]:(killed(X,Y) => ~richer(X,Y))).
fof(pel55_6,   axiom, ![X]:(hates(agatha,X) => ~hates(charles,X))).
fof(pel55_7,   axiom, ![X]:(X != butler => hates(agatha,X))).
fof(pel55_8,   axiom, ![X]:(~richer(X,agatha) => hates(butler,X))).
fof(pel55_9,   axiom, ![X]:(hates(agatha,X) => hates(butler,X))).
fof(pel55_10,  axiom, ![X]:(?[Y]:(~hates(X,Y)))).
fof(pel55_11,  axiom, agatha != butler).
fof(pel55, conjecture, killed(agatha,agatha)).

debian10!^> vampire -om smtcomp fol6d.p
unsat
```

Problem: Composition of Bijective Functions

Prove the following theorem:

> Let $f\colon D \to D$ and $g\colon D \to D$ be functions on some domain D. If f and g are bijective, then also their composition is.

```
% file fol6h.p: if f and j are bijective, then also their composition is.
fof(goal, conjecture, (bijective_f & bijective_g & composition_f_g_h) => bijective_h).
fof(a1, axiom, bijective_f <=> (
  (![Y]:(?[X]:(f(X)=Y))) &
  (![X1,X2,Y]:((f(X1)=Y & f(X2)=Y) => X1=X2)))).
fof(a2, axiom, bijective_g <=> (
  (![Y]:(?[X]:(g(X)=Y))) &
  (![X1,X2,Y]:((g(X1)=Y & g(X2)=Y) => X1=X2)))).
fof(a3, axiom, composition_f_g_h <=>
  (![X]: (h(X) = g(f(X))))).
fof(a4, axiom, bijective_h <=> (
  (![Y]:(?[X]:(h(X)=Y))) &
  (![X1,X2,Y]:((h(X1)=Y & h(X2)=Y) => X1=X2)))).
```

```
debian10!1> vampire -om smtcomp fol6h.p
unsat
```

Problem: Inversion of a Bijective Function

Let $f \subseteq D \times D$. Prove that, if $f\colon D \to D$ is a bijective function, also its inverse is.

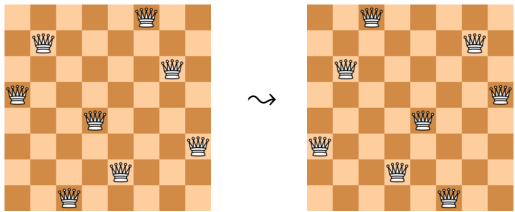
```
% file fol6g.p: if f is a bijective function, also its inverse is.
fof(goal, conjecture, (bijective_function_f & inverse_f_g) => bijective_function_g).

% atomic predicate apply(f,X,Y): "tuple (X,Y) is in binary relation f" (aka "f(X)=Y")
fof(a1, axiom, bijective_function_f <=> (
  (![X]:(?[Y]:(apply(f,X,Y)))) &
  (![X,Y1,X2]:((apply(f,X,Y1) & apply(f,X,Y2)) => Y1=Y2)) &
  (![Y]:(?[X]:(apply(f,X,Y)))) &
  (![X1,X2,Y]:((apply(f,X1,Y) & apply(f,X2,Y)) => X1=X2)))).
fof(a2, axiom, inverse_f_g <=>
  ![X,Y]:(apply(f,X,Y) <=> apply(g,Y,X))).
fof(a3, axiom, bijective_function_g <=> (
  (![X]:(?[Y]:(apply(g,X,Y)))) &
  (![X,Y1,X2]:((apply(g,X,Y1) & apply(g,X,Y2)) => Y1=Y2)) &
  (![Y]:(?[X]:(apply(g,X,Y)))) &
  (![X1,X2,Y]:((apply(g,X1,Y) & apply(g,X2,Y)) => X1=X2)))).

debian10!1> vampire -om smtcomp fol6g.p
unsat
```

N Queens Problem

Prove that every “flipped” solution to the N queens problem is also a solution:



$$queens(p) \land symmetric(p,q) \Rightarrow queens(q)$$

$$queens(p) :\Leftrightarrow$$
$$\quad p\colon \{1,\dots,N\} \to \{1,\dots,N\} \land$$
$$\quad \forall i \in \mathbb{Z}, j \in \mathbb{Z}.\ 1 \le i \land i < j \land j \le N \Rightarrow$$
$$\qquad p(i) \neq p(j) \land p(i)+i \neq p(j)+j \land p(i)-i \neq p(j)-j$$

$$symmetric(p,q) :\Leftrightarrow \forall i \in \mathbb{Z}.\ 1 \le i \land i \le N \Rightarrow q(i) = p(N+1-i)$$

Arithmetic with constant 1, functions $+$ and $-$, and predicate $\le$.

First Solution: zero, succ, plus, minus, leq

```
% file fol6e-leq.p: solution with "zero" "succ" "plus" "minus" and "leq"
fof(goal, conjecture, (queens_p & symmetric_pq) => queens_q).

% p and q are solutions to the N queens problem
fof(queens_p, axiom, queens_p <=>
  ![I,J]:((leq(succ(zero),I) & leq(succ(I),J) & leq(J,n)) =>
    (p(I) != p(J) & plus(p(I),I) != plus(p(J),J) & minus(p(I),I) != minus(p(J),J)))).
fof(queens_q, axiom, queens_q <=>
  ![I,J]:((leq(succ(zero),I) & leq(succ(I),J) & leq(J,n)) =>
    (q(I) != q(J) & plus(q(I),I) != plus(q(J),J) & minus(q(I),I) != minus(q(J),J)))).

% q is symmetric to p: q(i) = p(pos(i)) = p(n+1-i)
fof(symmetric, axiom, symmetric_pq <=>
  ![I]:((leq(succ(zero),I) & leq(I,n)) => q(I)=p(pos(I)))).

% the definition of pos is actually not helpful, the properties below are required
% fof(symmetric, axiom, pos <=> ![I]:(pos(I) = minus(succ(n),I))).
...
```

First Solution: plus, succ, zero, leq, minus

```
% file fol6e-leq.p: solution with "zero" "succ" "plus" "minus" and "leq"
...

% property required from minus: i+j = k+l <=> i-k = l-j
fof(plus_minus, axiom, ![I,J,K,L]:(plus(I,J) = plus(K,L) <=> minus(I,K) = minus(L,J))).

% properties required from leq: i <= i+1 and (i <= j & j <= k => i <= k)
fof(le_succ, axiom, ![I]:(leq(I,succ(I)))).
fof(le_tran, axiom, ![I,J,K]:((leq(I,J) & leq(J,K)) => leq(I,K))).

% properties required from pos
fof(pos1, axiom, % 1 <= i & i <= n => 1 <= pos(i) & pos(i) <= n
  ![I]:((leq(succ(zero),I) & leq(I,n)) => (leq(succ(zero),pos(I)) & leq(pos(I),n)))).
fof(pos2, axiom, % i-j = pos(j)-pos(i)
  ![J,I]:(minus(I,J) = minus(pos(J),pos(I)))).
fof(pos3, axiom, % 1 <= i & i < j & j <= n => pos(j) < pos(i)
  ![J,I]:((leq(succ(zero),I) & leq(succ(I),J) & leq(J,n)) => leq(succ(pos(J)),pos(I)))).
```

First Solution: plus, zero, succ, leq, minus

```
debian10!1> vampire fol6e-leq.p
% Refutation found. Thanks to Tanya!
...
% Time elapsed: 12.504 s
% ------------------------------
% ------------------------------

debian10!1> vampire --mode casc fol6e-leq.p
% Refutation found. Thanks to Tanya!
...
% ------------------------------
% ------------------------------
% Success in time 0.418 s
```

The proof can be found much faster by trying a portfolio of search strategies.

Second Solution: Typed First-Order Logic

```
% file fol6e-typed.p: typed solution

tff(queens_p_type, type, queens_p: $o).
tff(queens_q_type, type, queens_q: $o).
tff(symmetric_pq_type, type, symmetric_pq: $o).
tff(p_type, type, p: $int > $int).
tff(q_type, type, q: $int > $int).
tff(n_type, type, n: $int).

tff(goal, conjecture, (queens_p & symmetric_pq) => queens_q).
...
```

Actually, Vampire also provides built-in support for the theory of integers.

Second Solution: Typed First-Order Logic

```
% file fol6e-typed.p: typed solution
...
tff(queens_p, axiom, queens_p <=> % much faster with $lesseq and != than with $less
  ![I:$int,J:$int]:(($lesseq(1,I) & $lesseq(I,J) & $lesseq(J,n) & I != J) =>
    (p(I) != p(J) & $sum(p(I),I) != $sum(p(J),J) & $sum(p(I),J) != $sum(p(J),I)))).
tff(queens_q, axiom, queens_q <=> % much faster with $lesseq and != than with $less
  ![I:$int,J:$int]:(($lesseq(1,I) & $lesseq(I,J) & $lesseq(J,n) & I != J) =>
    (q(I) != q(J) & $sum(q(I),I) != $sum(q(J),J) & $sum(q(I),J) != $sum(q(J),I)))).
tff(symmetric, axiom, symmetric_pq <=>
  ![I:$int]:(($lesseq(1,I) & $lesseq(I,n)) => q(I)=p($difference($sum(n,1),I)))).

debian10!1> vampire --mode casc fol6e-typed.p
% Refutation found. Thanks to Tanya!
...
% Success in time 7.42 s
```

Indeed the integer support is sufficient for solving this problem.

Summary

Automated first-order theorem proving has some caveats.

- Problems must be captured by a finite number of first-order axioms.
  - Incompleteness Theorem (Kurt Gödel, 1930): the arithmetic of natural numbers cannot be completely captured by any sound logical calculus.
- For every problem, sufficiently strong axioms have to be provided.
  - TPTP provides a library of axiom sets that are shared by many problems.
- However, the axiomatization should be minimal to facilitate proof search.
  - The size of the search space quickly explodes.
- Ultimately, it requires experience to find adequate problem formalizations.
  - Many formalizations possible, minor changes may have drastic consequences.
- Last but not least, the generated proofs are typically not “human-oriented”.
  - See Theorema: https://www.risc.jku.at/research/theorema/software

Automated first-order provers may also integrate/interact with SMT solvers or be applied for proof search in interactive proof assistants.

Isabelle

Software: https://isabelle.in.tum.de/
Tutorial: Programming and Proving in Isabelle/HOL (Tobias Nipkow)

- Generic proof assistant:
  - Larry Paulson (Univ. Cambridge), Tobias Nipkow (TU Munich), and many others.
  - Initial release in 1986, yearly updates (“Isabelle 2024”).
- Generic: typed meta-logic to encode various object-logics.
  - Polymorphic type system similar to ML-like programming languages.
- Proof assistant: interactive elaboration of procedural/declarative proofs.
  - Procedural: application of proof procedures (“tactics”).
    Declarative: writing of formal proofs in the “Isar” language.
- Automation: combination of various reasoning tools.
  - Term rewriting, tableaux method, resolution, decision procedures, external automated provers and SMT solvers (tool “Sledgehammer”).
  - Counterexample generators Nitpick and Nunchaku.

Many applications in mathematics and computer science.

Archive of Formal Proofs

https://www.isa-afp.org

Archive of Formal Proofs

The Archive of Formal Proofs is a collection of proof libraries, examples, and larger scientific developments, mechanically checked in the theorem prover Isabelle. It is organized in the way of a scientific journal, is indexed by dblp and has an ISSN: 2150-914x. Submissions are refereed. The preferred citation style is available [here]. We encourage companion AFP submissions to conference and journal publications.

A development version of the archive is available as well.

- Home
- About
- Submission
- Updating Entries
- Using Entries
- Search
- Statistics
- Index
- Download

2021

2021-07-07: Finitely Generated Abelian Groups
Authors: Joseph Thommes and Manuel Eberl

2021-07-01: SpecCheck - Specification-Based Testing for Isabelle/ML
Authors: Kevin Kappelmann, Lukas Bulwahn and Sebastian Willenbrink

2021-06-22: Van der Waerden's Theorem
Authors: Katharina Kreuzer and Manuel Eberl

2021-06-18: MiniSail - A kernel language for the ISA specification language SAIL
Author: Mark Wassell

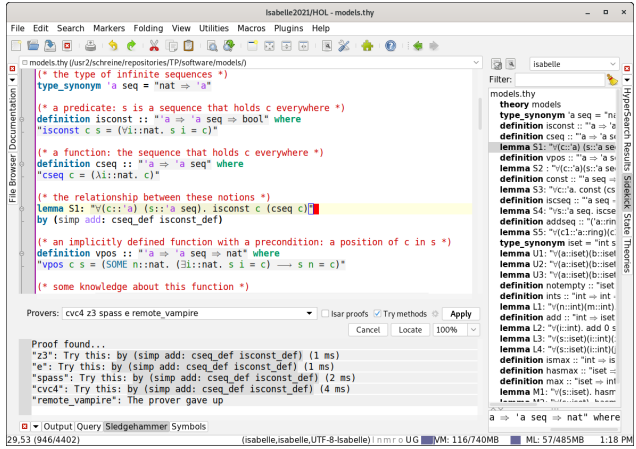
2021-06-17: Public Announcement Logic
Author: Asta Halkjær From

2021-06-04: A Shorter Compiler Correctness Proof for Language IMP
Author: Pasquale Noce

More than 800 articles with 270.000 lemmas and 4 million lines of proof code.

Isabelle/jEdit User Interface

```
debian12!1> /software/Isabelle2022/Isabelle2022 models.thy &
```



Formal proof documents with semantic markup and continuous checking.

Isabelle/HOL Syntax: A Theory of Infinite Sequences

```
theory models (* file models.thy, type \forall to select symbol ∀ from popup window *)
imports Main
begin
type_synonym 'a seq = "nat \<Rightarrow> 'a"

definition isconst :: "'a \<Rightarrow> 'a seq \<Rightarrow> bool" where
"isconst c s = (\<forall>i::nat. s i = c)"

definition cseq :: "'a \<Rightarrow> 'a seq" where
"cseq c = (\<lambda>i::nat. c)"

lemma S1: "\<forall>(c::'a) (s::'a seq). isconst c (cseq c)"
by (simp add: cseq_def isconst_def)
...
end
```

Automatic proof search: move cursor to lemma; press in tab “Sledgehammer” button `Apply` (“search for first-order proof using automatic theorem provers”); output `Proof found...` displays proof command `by (...)`; click on command to insert it into document.

Isabelle/HOL Syntax: A Theory of Infinite Sequences

```
definition vpos :: "'a \<Rightarrow> 'a seq \<Rightarrow> nat" where
"vpos c s = (SOME n::nat. (\<exists>i::nat. s i = c) \<longrightarrow> s n = c)"

lemma S2 : "\<forall>(c::'a)(s::'a seq). \<not>(\<forall>i::nat. s i \<noteq> c)
  \<longrightarrow> s(vpos c s) = c" by (metis (mono_tags, lifting) someI_ex vpos_def)

definition vpos :: "'a \<Rightarrow> 'a seq \<Rightarrow> nat" where
"vpos c s = (SOME n::nat. (\<exists>i::nat. s i = c) \<longrightarrow> s n = c)"

lemma S2 : "\<forall>(c::'a)(s::'a seq). \<not>(\<forall>i::nat. s i \<noteq> c)
  \<longrightarrow> s(vpos c s) = c"
by (metis (mono_tags, lifting) someI_ex vpos_def)

definition const :: "'a seq \<Rightarrow> 'a" where
"const s = (SOME c::'a. isconst c s)"

lemma S3: "\<forall>c::'a. const (cseq c) = c"
by (metis S1 const_def isconst_def someI_ex)
```

Isabelle/HOL Syntax: A Theory of Infinite Sequences

```
definition iscseq :: "'a seq \<Rightarrow> bool" where
"iscseq s = (\<exists>c::'a. isconst c s)"

lemma S4: "\<forall>s::'a seq. iscseq s \<longrightarrow> (\<forall>i::nat. s i = const s)"
by (metis const_def isconst_def iscseq_def someI_ex)

definition addseq :: "('a::ring) seq \<Rightarrow> 'a seq \<Rightarrow> 'a seq" where
"addseq s1 s2 = (\<lambda>i::nat. (s1 i) + (s2 i))"

lemma S5: "\<forall>(c1::'a::ring)(c2::'a::ring). iscseq(addseq (cseq c1) (cseq c2))"
by (simp add: addseq_def cseq_def isconst_def iscseq_def)
```

Isabelle/HOL Syntax: A Theory of Sets

```
type_synonym iset = "int set"

lemma U1: "\<forall>(a::iset)(b::iset)(c::iset). a \<union> (b \<inter> c)
  = (a \<union> b) \<inter> (a \<union> c)" by auto
lemma U2: "\<forall>(a::iset)(b::iset). a \<subseteq> b \<longrightarrow> a \<union> b = b"
by auto
lemma U3: "\<forall>(a::iset)(b::iset). a \<inter> b \<noteq> {} \<longrightarrow>
  a - b \<subset> a" by blast

definition notempty :: "iset \<Rightarrow> bool" where
"notempty s = (\<exists>x::int. x \<in> s)"

definition ints :: "int \<Rightarrow> int \<Rightarrow> iset" where
"ints n m = { i::int. n \<le> i \<and> i \<le> m }"

lemma L1: "\<forall>(n::int)(m::int). n \<le> m \<longleftrightarrow> notempty (ints n m)"
using notempty_def ints_def by auto
```

Isabelle/HOL Syntax: A Theory of Sets

```
(* adding an integer to a set *)
definition add :: "int \<Rightarrow> iset \<Rightarrow> iset" where
"add i s = { (i+x) | (x::int). x \<in> s }"

(* some lemmas about adding an integer to a set *)
lemma L2: "\<forall>(i::int). add 0 s = s"
by (simp add: add_def)
lemma L3: "\<forall>(s::iset)(i::int)(x::int). x \<in> (add i s)
  \<longleftrightarrow> x-i \<in> s"
using add_def by force
lemma L4: "\<forall>(s::iset)(i::int)(j::int). add i (add j s) = add (i+j) s"
by (smt Collect_cong L3 add_def mem_Collect_eq)
```

Isabelle/HOL Syntax: A Theory of Sets

```
definition ismax :: "int \<Rightarrow> iset \<Rightarrow> bool" where
"ismax m s = (m \<in> s \<and> (\<forall>x::int. x \<in> s \<longrightarrow> x \<le> m))"
definition hasmax :: "iset \<Rightarrow> bool" where
"hasmax s = (\<exists>m::int. ismax m s)"
definition max :: "iset \<Rightarrow> int" where
"max s = (SOME (m::int). ismax m s)"

lemma M1: "\<forall>(s::iset). hasmax s \<longrightarrow>
  max s \<in> s \<and> (\<forall>x::int. x \<in> s \<longrightarrow> x \<le> max s)"
by (metis hasmax_def ismax_def models.max_def someI_ex)
lemma M2: "\<forall>(s::iset). hasmax s \<longrightarrow>
  (\<forall>x::int. x \<in> s \<and> x \<noteq> max s \<longrightarrow> x < max s)"
using M1 by fastforce
lemma M3: "\<forall>(s::iset). hasmax s \<and> hasmax(s - { max s }) \<longrightarrow>
  max s > max (s - { max s })"
using M1 M2 by blast
```

All proofs can be found by automatic proof search.

Isar Proofs

```
theorem sqrt2_not_rational:
  "sqrt 2 \<notin> \<rat>"
proof
  let ?x = "sqrt 2"
  assume "?x \<in> \<rat>"
  then obtain m n :: nat where
    sqrt_rat: "¦?x¦ = m / n" and lowest_terms: "coprime m n"
    by (rule Rats_abs_nat_div_natE)
  hence "m^2 = ?x^2 * n^2" by (auto simp add: power2_eq_square)
  hence eq: "m^2 = 2 * n^2" using of_nat_eq_iff power2_eq_square by fastforce
  hence "2 dvd m^2" by simp
  hence "2 dvd m" by simp
  have "2 dvd n" proof -
    from <2 dvd m> obtain k where "m = 2 * k" ..
    with eq have "2 * n^2 = 2^2 * k^2" by simp
    hence "2 dvd n^2" by simp
    thus "2 dvd n" by simp
  qed
  with <2 dvd m> have "2 dvd gcd m n" by (rule gcd_greatest)
  with lowest_terms have "2 dvd 1" by simp
  thus False using odd_one by blast
qed
```

A reasonably understandable argument that $\sqrt{2} \notin \mathbb{Q}$, formally checked by Isabelle.