



Formally Verifying a Theorem Prover for First-Order Logic

Asta Halkjær From Frederik Krogsdal Jacobsen

Technical University of Denmark

Introduction

Automatic theorem provers have many success stories:

- Ada/SPARK (Rolls Royce, Lockheed Martin, EuroFighter, Collins, ...)
- Dafny
- TLA+ (Microsoft, Intel, AWS, ...)
- Mathematics
- Hardware verification

Introduction

Automatic theorem provers have many success stories:

- Ada/SPARK (Rolls Royce, Lockheed Martin, EuroFighter, Collins, ...)
- Dafny
- TLA+ (Microsoft, Intel, AWS, ...)
- Mathematics
- Hardware verification

... but we are not eating our own dog food!

A few dog treats

Examples of formally verified automatic theorem provers:

- Propositional logic (tableaux), Blanchette et al.
- Clausal first-order logic (ordered resolution), Schlichtkrull et al.
- First-order logic (implicit sequent calculus), Ridge et al.
- First-order logic (implicit sequent calculus), Villadsen et al.
- SAT solver, Fleury
- SAT solver, Maríc

What's missing?

- Optimized proof search procedures
- Heuristics
- Realistic logics
- Proof certificates

Overview of results

- A sound and complete prover for first-order logic with functions
- Based on a sequent calculus
- All proofs are formally verified in Isabelle/HOL
- Human-readable proof certificates

Why did we do this?

- Formalized metatheory for non-trivial sequent calculus provers
- · Formal verification of an executable prover
- Novel analytic proof technique for completeness
- · Verifiable and human-readable proof certificates
- A prover for the SeCaV system

Sample SeCaV Proof Rules

$$\frac{\operatorname{Neg} p \in z}{\Vdash p, z} \operatorname{Basic} \qquad \frac{\Vdash z \quad z \subseteq y}{\Vdash y} \operatorname{Ext} \qquad \frac{\Vdash p, z}{\Vdash \operatorname{Neg} (\operatorname{Neg} p), z} \operatorname{NegNeg}$$

$$\frac{\Vdash p, q, z}{\Vdash \operatorname{Dis} p q, z} \operatorname{ALPHADIS} \qquad \frac{\Vdash \operatorname{Neg} p, z \quad \Vdash \operatorname{Neg} q, z}{\Vdash \operatorname{Neg} (\operatorname{Dis} p q), z} \operatorname{BetaDis}$$

$$\frac{\Vdash p[\operatorname{Var} 0/t], z}{\Vdash \operatorname{Exi} p, z} \operatorname{GAMMAExi}$$

$$\frac{\Vdash \operatorname{Neg} (p[\operatorname{Var} 0/\operatorname{Fun} i []]), z \quad i \operatorname{fresh}}{\Vdash \operatorname{Neg} (\operatorname{Exi} p), z} \operatorname{DeltaExi}$$

Prover I

- SeCaV rules affect one formula at a time
- Our prover rules affect every applicable formula at once
- We copy Gamma formulas and remember all terms on the branch
- So no formula or instantiation is forgotten

Prover I

- SeCaV rules affect one formula at a time
- Our prover rules affect every applicable formula at once
- We copy Gamma formulas and remember all terms on the branch
- So no formula or instantiation is forgotten
- Rules affect disjoint formulas
- So we can apply them in any order

Prover I

- SeCaV rules affect one formula at a time
- Our prover rules affect every applicable formula at once
- We copy Gamma formulas and remember all terms on the branch
- So no formula or instantiation is forgotten
- Rules affect disjoint formulas
- So we can apply them in any order
- We apply rules *fairly* and repeatedly
- So we never miss out on a proof



Prover II

- We rely on the abstract completeness framework by Blanchette, Popescu and Traytel
- We need to fix a stream of rules from the beginning
- Proof attempts are *coinductive trees* grown by applying these rules
- If a tree cannot be grown further, we found a proof
- A function gives the *child sequents* representing the subgoals left after applying a rule
- We export code to Haskell to obtain an executable prover



Prover — proof example

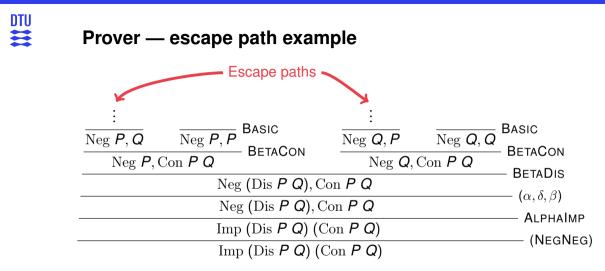
Basic Neg (Uni (Con P(0) Q(0))), Neg P(0), Neg Q(0), Neg P(a), Neg Q(a), P(a)—— AlphaCon Neg (Uni (Con P(0) Q(0))), Neg (Con P(0) Q(0)), Neg (Con P(a) Q(a)), P(a)Neg (Uni (Con P(0) Q(0))), Neg (Con P(0) Q(0)). (α) Neg (Con P(a) Q(a)), P(a)GAMMAUNI Neg (Uni (Con P(0) Q(0))), P(a) (α, δ, β) Neg (Uni (Con P(0) Q(0))), P(a)ALPHAIMP Imp (Uni (Con P(0) Q(0))) P(a)(NEGNEG) Imp (Uni (Con P(0) Q(0))) P(a)



Prover — certificate example

Imp (Uni (Con (P [0]) (Q [0]))) (P [a])
AlphaImp
Neg (Uni (Con (P [0]) (Q [0])))
P [a]
Ext
Neg (Uni (Con (P [0]) (Q [0])))
Neg (Uni (Con (P [0]) (Q [0])))
P [a]
GammaUni[0]
Neg (Con (P [0]) (Q [0]))
Neg (Uni (Con (P [0]) (Q [0])))
P [a]

Ext Neg (Uni (Con (P [0]) (Q [0]))) Neg (Uni (Con (P [0]) (Q [0]))) P [a] Neg (Con (P [0]) (Q [0])) GammaUni[a] Neg (Con (P [a]) (Q [a])) Neg (Uni (Con (P [0]) (Q [0]))) P [a] Neg (Con (P [0]) (Q [0])) Ext Neg (Con (P [0]) (Q [0])) Neg (Con (P [a]) (Q [a])) P [a] Neg (Uni (Con (P [0]) (Q [0]))) AlphaCon Neg (P [0]) Neg (Q [0]) Neg (Con (P [a]) (Q [a])) P [a] Neg (Uni (Con (P [0]) (Q [0]))) Ext Neg (Con (P [a]) (Q [a])) P [a] Neg (Uni (Con (P [0]) (Q [0]))) Neg (P [0]) Neg (Q [0]) AlphaCon Neg (P [a]) Neg (Q [a]) P [a] Neg (Uni (Con (P [0]) (Q [0]))) Neg (P [0]) Neg (Q [0]) Ext P [a] Neg (Uni (Con (P [0]) (Q [0]))) Neg (P [0]) Neg (Q [0]) Neg (P [a]) Neg (Q [a]) Basic



Soundness I

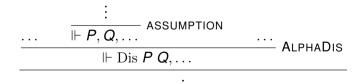
- If our prover returns a proof, we can build a SeCaV proof
- The SeCaV proof system is sound, so the prover is sound
- We use the abstract soundness framework by Blanchette et al.
- If the children of a sequent all have SeCaV proofs, so does the sequent

Soundness II

If the children of a sequent all have SeCaV proofs, so does the sequent:

1 Assume all child sequents have a proof

2 Induction on sequent: use appropriate SeCaV rule for each formula Example: Our sequent looks like $\text{Dis } P \ Q, \dots$, so P, Q, \dots is a child sequent with a SeCaV proof. We apply the ALPHADIS rule to prove the sequent using the proof of $\Vdash P, Q, \dots$ (and possibly some reordering).





Completeness

- Framework: prover either produces a finite, well formed proof tree or an infinite tree with a saturated escape path
- Need to show that root sequent of a saturated escape path is not valid:
 - Formulas on saturated escape paths form Hintikka sets
 - Hintikka sets induce a well formed countermodel
- ... so valid sequents result in finite, well formed proof trees



Completeness

- Framework: prover either produces a finite, well formed proof tree or an infinite tree with a saturated escape path
- Need to show that root sequent of a saturated escape path is not valid:
 - Formulas on saturated escape paths form Hintikka sets
 Hintikka sets induce a well formed countermodel
- ... so valid sequents result in finite, well formed proof trees

HERE BE DRAGONS.

(need to build a bounded countermodel over only the terms in the sequent and ensure functions stay inside its domain)

Bounded semantics

- In a completeness proof for a *calculus* we can assume that Gamma formulas are instantiated with all possible terms
- Thus, we can build a countermodel in the full Herbrand domain
- Our prover only uses terms from the given sequent (and fresh ones)
- So we must build a *bounded* countermodel over this restricted domain
- · We must ensure that our function denotation stays inside this domain

Subtypes fail us

- The SeCaV semantics represents the domain as a type variable.
- We cannot build the subtype of terms from a *local* sequent (yet?¹)
- So we represent the domain as an explicit parameter to the semantics
- We have $u, E, F, G \models \text{Uni } P$ iff $u, E, F, G \models P(x)$ for all $x \in u$
- We reprove soundness of SeCaV under this (u)semantics

Hintikka sets

We always need at least one term

terms $H \equiv if (\bigcup p \in H. set (subtermFm p)) = \{\}$ then {Fun 0 []} else ($\bigcup p \in H. set (subtermFm p)$)

To quantify over in our Hintikka sets

locale Hintikka =

fixes H :: fm set

assumes

.

Basic: Pre $n ts \in H \Longrightarrow Neg (Pre n ts) \notin H$ and AlphaDis: Dis $p q \in H \Longrightarrow p \in H \land q \in H$ and BetaDis: Neg (Dis p q) $\in H \Longrightarrow Neg p \in H \lor Neg q \in H$ and GammaExi: Exi $p \in H \Longrightarrow \forall t \in terms H$. sub 0 $t p \in H$ and DeltaExi: Neg (Exi p) $\in H \Longrightarrow \exists t \in terms H$. Neg (sub 0 t p) $\in H$ and

Bounded countermodel

• We carefully build the countermodel

 $E S n \equiv if Var n \in terms S then Var n else SOME t. t \in terms S$ $F S i I \equiv if Fun i I \in terms S then Fun i I else SOME t. t \in terms S$ $G S n ts \equiv Neg (Pre n ts) \in S$ $M S \equiv usemantics (terms S) (E S) (F S) (G S)$

- terms is downwards closed, so members evaluate to themselves
 t ∈ terms S ⇒ semantics-term (E S) (F S) t = t
- We have a countermodel to any formula in a Hintikka set *Hintikka S* ⇒ (p ∈ S → ¬ M S p) ∧ (Neg p ∈ S → M S p)

Saturated escape paths form Hintikka sets

- Final step is to inspect the saturated escape paths
- · We need to show that the formulas constitute a Hintikka set
- On paper, this follows straightforwardly from our rules
- In practice, it requires fiddly reasoning about the coinductive paths
- In the end: any saturated escape path has a (bounded) countermodel, contradicting the validity of its root sequent

010 ₩ Results

- · We have verified soundness and completeness in Isabelle/HOL
- Verification helped find actual bugs in our implementation
- The performance is limited, but optimizations are possible
- · Generation of proof certificates is not (yet) fully verified

Where can we go from here?

- New frameworks for integrating heuristics
- Fully verified proof certificate generation
- Verified proof certificate compression
- Frameworks for other proof systems
- Even more realistic logics (e.g. with equality)