DTU

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

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

$$\frac{\Vdash p, q, z}{\Vdash \mathrm{Dis}\ p\ q, z}\ \textsc{AlphaDis} \qquad \frac{\Vdash \mathrm{Neg}\ p, z \qquad \Vdash \mathrm{Neg}\ q, z}{\Vdash \mathrm{Neg}\ (\mathrm{Dis}\ p\ q), z}\ \textsc{BetaDis}$$

$$\frac{\Vdash p\,[\mathrm{Var}\ 0/t], z}{\Vdash \mathrm{Exi}\ p, z}\ \textsc{GammaExi}$$

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

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

$$
\begin{array}{c}
\hline
\mathrm{Neg}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0))), \mathrm{Neg}\ P(0), \mathrm{Neg}\ Q(0), \\
\mathrm{Neg}\ P(a), \mathrm{Neg}\ Q(a), P(a) \\
\hline
\mathrm{Neg}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0))), \mathrm{Neg}\ (\mathrm{Con}\ P(0)\ Q(0)), \\
\mathrm{Neg}\ (\mathrm{Con}\ P(a)\ Q(a)), P(a) \\
\hline
\mathrm{Neg}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0))), \mathrm{Neg}\ (\mathrm{Con}\ P(0)\ Q(0)), \\
\mathrm{Neg}\ (\mathrm{Con}\ P(a)\ Q(a)), P(a) \\
\hline
\mathrm{Neg}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0))), P(a) \\
\hline
\mathrm{Neg}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0))), P(a) \\
\hline
\mathrm{Imp}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0)))\ P(a) \\
\hline
\mathrm{Imp}\ (\mathrm{Uni}\ (\mathrm{Con}\ P(0)\ Q(0)))\ P(a)
\end{array}
\quad
\begin{array}{l}
\textsc{Basic} \\
\\
\textsc{AlphaCon} \\
\\
(\alpha) \\
\\
\textsc{GammaUni} \\
(\alpha, \delta, \beta) \\
\textsc{AlphaImp} \\
(\textsc{NegNeg})
\end{array}
$$

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
```

Prover — escape path example

Escape paths

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\text{Neg } P, Q} \qquad \dfrac{}{\text{Neg } P, P}\,\textsc{Basic}}{\text{Neg } P, \text{Con } P\ Q}\,\textsc{BetaCon} \qquad \dfrac{\dfrac{\vdots}{\text{Neg } Q, P} \qquad \dfrac{}{\text{Neg } Q, Q}\,\textsc{Basic}}{\text{Neg } Q, \text{Con } P\ Q}\,\textsc{BetaCon}}{\text{Neg } (\text{Dis } P\ Q), \text{Con } P\ Q}\,\textsc{BetaDis}}{\text{Neg } (\text{Dis } P\ Q), \text{Con } P\ Q}\,(\alpha, \delta, \beta)}{\text{Imp } (\text{Dis } P\ Q)\ (\text{Con } P\ Q)}\,\textsc{AlphaImp}}{\text{Imp } (\text{Dis } P\ Q)\ (\text{Con } P\ Q)}\,(\textsc{NegNeg})
$$

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 $\mathrm{Dis}\ P\ Q, \ldots$, so $P, Q, \ldots$ is a child sequent with a SeCaV proof. We apply the ALPHADIS rule to prove the sequent using the proof of $\Vdash P, Q, \ldots$ (and possibly some reordering).

$$\dfrac{\dfrac{\ldots \qquad \dfrac{\vdots}{\Vdash P, Q, \ldots}\ \textsc{Assumption} \qquad \ldots}{\Vdash \mathrm{Dis}\ P\ Q, \ldots}\ \textsc{AlphaDis}}{\vdots}$$

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 $u, E, F, G \models \mathrm{Uni}\ P$ iff $u, E, F, G \models P(x)$ for all $x \in u$
- We reprove soundness of SeCaV under this (u)semantics

[1] Kunčar and Popescu ITP 2014

Hintikka sets

- We always need at least one term

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

- To quantify over in our Hintikka sets

locale *Hintikka* $=$
  fixes $H :: \textit{fm set}$
  assumes
    Basic: $\textit{Pre } n \ \textit{ts} \in H \Longrightarrow \textit{Neg } (\textit{Pre } n \ \textit{ts}) \notin H$ **and**
    AlphaDis: $\textit{Dis } p\ q \in H \Longrightarrow p \in H \wedge q \in H$ **and**
    BetaDis: $\textit{Neg } (\textit{Dis } p\ q) \in H \Longrightarrow \textit{Neg } p \in H \vee \textit{Neg } q \in H$ **and**
    GammaExi: $\textit{Exi } p \in H \Longrightarrow \forall t \in \textit{terms } H.\ \textit{sub } 0\ t\ p \in H$ **and**
    DeltaExi: $\textit{Neg } (\textit{Exi } p) \in H \Longrightarrow \exists t \in \textit{terms } H.\ \textit{Neg } (\textit{sub } 0\ t\ p) \in H$ **and**
    $\vdots$

Bounded countermodel

- We carefully build the countermodel

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

- *terms* is downwards closed, so members evaluate to themselves

  $t \in \mathit{terms}\ S \Longrightarrow \mathit{semantics\text{-}term}\ (E\ S)\ (F\ S)\ t = t$

- We have a countermodel to any formula in a Hintikka set

  $\mathit{Hintikka}\ S \Longrightarrow (p \in S \longrightarrow \neg\ M\ S\ p) \wedge (\mathit{Neg}\ p \in S \longrightarrow M\ S\ p)$

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

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)