DTU

Verifying a Sequent Calculus Prover

Asta Halkjær From   Frederik Krogsdal Jacobsen

Introduction

- 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

Background

- 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

Isabelle/HOL

- Assistant for writing formal proofs
- HOL = Higher-Order Logic =
  “Functional Programming + Logic”
- Much more precise than formal English
- Mechanically checks every argument
- We can export some definitions to code

Sample SeCaV Proof Rules

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

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

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

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

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 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

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

Prover — escape path example

Escape paths

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

Soundness I

- If the children of a sequent all have SeCaV proofs, so does the sequent
- Framework: finite, well formed proof trees represent SeCaV proofs
- The SeCaV proof system is sound
- ...so the prover is sound

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: $P, Q, \ldots$ is a child sequent, so we can 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
- The 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
- The 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)

Results and future work

- Verified soundness and completeness in Isabelle/HOL
- Verification helped find actual bugs in our implementation
- Very limited performance, but optimizations are possible
- Generation of proof certificates is not verified
- Consider extensions to the logic