

**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

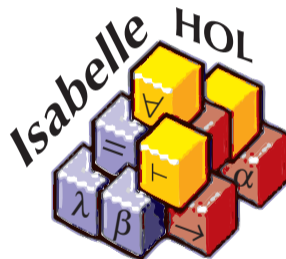
Assistant for writing formal proofs

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

$$\frac{\text{Neg } p \ 2 \ z}{p; z} \text{ BASIC}$$

$$\frac{z \quad z \quad y}{y} \text{ EXT}$$

$$\frac{p; z}{\text{Neg } (\text{Neg } p); z} \text{ NEGNEG}$$

$$\frac{p; q; z}{\text{Dis } p \ q; z} \text{ ALPHADIS}$$

$$\frac{\text{Neg } p; z \quad \text{Neg } q; z}{\text{Neg } (\text{Dis } p \ q); z} \text{ BETADIS}$$

$$\frac{p[\text{Var } 0=t]; z}{\text{Exi } p; z} \text{ GAMMAEXI}$$

$$\frac{\text{Neg } (p[\text{Var } 0=\text{Fun } i \ []]); z \quad i \text{ fresh}}{\text{Neg } (\text{Exi } p); z} \text{ DELTAEXI}$$

# 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

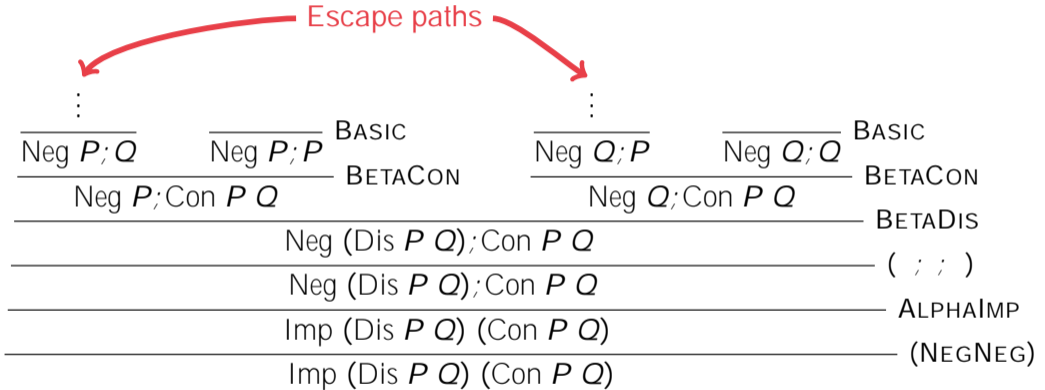
A function gives the *child sequents* representing the subgoals left after applying a rule

We export code to Haskell to obtain an executable prover

# Prover — proof example

$\text{Neg (Uni (Con } P(0) Q(0))\text{)); Neg } P(0)\text{; Neg } Q(0)\text{;}$	BASIC
$\text{Neg } P(a)\text{; Neg } Q(a)\text{; } P(a)$	
$\text{Neg (Uni (Con } P(0) Q(0))\text{)); Neg (Con } P(0) Q(0)\text{);}$	ALPHACON
$\text{Neg (Con } P(a) Q(a)\text{); } P(a)$	
$\text{Neg (Uni (Con } P(0) Q(0))\text{)); Neg (Con } P(0) Q(0)\text{);}$	( )
$\text{Neg (Con } P(a) Q(a)\text{); } P(a)$	
$\text{Neg (Uni (Con } P(0) Q(0))\text{)); } P(a)$	GAMMAUNI
$\text{Neg (Uni (Con } P(0) Q(0))\text{)); } P(a)$	( ; ; )
$\text{Imp (Uni (Con } P(0) Q(0))\text{)) } P(a)$	ALPHAIMP
$\text{Imp (Uni (Con } P(0) Q(0))\text{)) } P(a)$	(NEGNEG)

# Prover — escape path example



# 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:  $P; Q; \dots$  is a child sequent, so we can apply the ALPHADIS rule to prove the sequent using the proof of  $P; Q; \dots$  (and possibly some reordering).

$$\begin{array}{c}
 \vdots \\
 \hline
 \dots \quad P; Q; \dots \quad \dots \quad \text{ALPHADIS} \\
 \hline
 \text{Dis } P \ Q; \dots \\
 \hline
 \vdots
 \end{array}$$

# 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