Introduction
We do proof checking:

Is a proof (produced by a prover) to be trusted?
Theorem proving and resolution

We do proof checking:

Is a proof (produced by a prover) to be trusted?

Many high-performance provers use variants of resolution refutation

⇒ Relies on unification
Theorem proving and resolution

We do proof checking:

Is a proof (produced by a prover) to be trusted?

Many high-performance provers use variants of resolution refutation

⇒ Relies on unification

With quantifier alternation, some variables different than others!

A method for removing quantifier alternation is necessary
### Skolemization

Guarantees equi-provability of any formula and a version with no universal quantifiers in an extended language.

For example:

\[
\exists x \forall y P(x, y) \mapsto \exists x P(x, f(x))
\]

This comes from the Skolem theorem in Model Theory.

The language is modified by adding the new symbol \( f \)!

(Note: this is sometimes also called Herbrandization)
An example

\[ A = \exists x. \forall y. \neg p(x) \lor p(y) \]

Skolemized to

\[ A' = \exists x. \neg p(x) \lor p(f(x)) \]

A possible proof in the sequent calculus \( LK \):

\[
\begin{align*}
& \vdash \neg p(c), p(f(c)), \neg p(f(c)), p(f(f(c))) & \text{init} \\
& \vdash \neg p(c) \lor p(f(c)), \neg p(f(c)) \lor p(f(f(c))) & \lor \\
& \vdash \exists x. \neg p(x) \lor p(f(x)), \exists x. \neg p(x) \lor p(f(x)) & \exists c, f(c) \\
& \vdash \exists x. \neg p(x) \lor p(f(x)) & \text{contr.}
\end{align*}
\]

The new Skolem symbol \( f \) appears in the proof!
Certification of proofs involving skolemization

The proof contains the Skolem symbols from the extended language

The original formula contains universally quantified scopes

How can the proof be used as evidence for the original formula?

A deskolemization procedure is needed!
Certification of proofs involving skolemization

Usual procedures to certify proofs with skolemization depend on

- The original, model-theoretic justification
- $\epsilon$-terms
- Other choice axioms

These are not satisfactory:

- Choice axioms need complex foundations $\Rightarrow$ less portability
- Richer metatheory for $LK$ proofs is needed

We adopt an approach aiming for simple foundations
Foundational Proof Certificates (FPC) [JAR 2016]
The **kernel** should be such that anybody can reimplement it

The **clients** are the provers, giving a proof evidence to the kernel
The **kernel** should be such that anybody can reimplement it.

The **clients** are the provers, giving a proof evidence to the kernel.

We chose Gentzen’s *LK*: the kernel uses *LK* rules and eigenvariables.

\[ \implies \text{It builds an } LK \text{ proof, based on the client’s proof evidence} \]

Proof certification = *(LK)* proof reconstruction.
Proof checking in the sequent calculus

But: reconstructing an $L K$ proof is too unconstrained!

At each step, there are too many choices and nondeterminism

\[
\begin{align*}
? & \quad \frac{\vdash p \lor \exists x. q(x), \forall x. q(x)}{}
\end{align*}
\]

What rule should we apply next? On which formula?

Do we really need clients to communicate all that information?
The proof-theoretic technique of **Focusing** improves the situation

- Determines what formula to work on next
- Vastly reduces search space for next rule

Divide $LK$ rules into **invertible** and **non invertible** rules:

- When handling a non invertible rule we query the certificate
- When the rule is invertible, proceed eagerly until told to stop
Focusing and proof checking

*Foundational proof certificates* are a framework for proof checking

- Based on **focusing** to control LK
- Interpret this as a **protocol**:
  - invertible rules are controlled by automatic *clerks*;
  - non-invertible rules ask for the help of *experts*

The **client** (prover)

- Defines the meaning of its proofs by defining clerks and experts
- Provides a proof certificate as evidence for a proof

The **kernel** reconstructs a full *LK* proofs based on this
Our calculus: $LKF^a$, $LK$ with focusing and certificate annotations

Invertible and non invertible judgements in LK...

\[ \vdash \Gamma, A, B \quad \vdash \Gamma, A \vee B \]

\[ \vdash \Gamma, A_i \quad \vdash \Gamma, A_1 \vee A_2 \]
Our calculus: \( LK^a, LK \) with focusing and certificate annotations

Invertible and non invertible judgements in \( LK \)... 

\[
\begin{align*}
\Xi_1 \vdash \Gamma, A, B \\
\Xi_0 \vdash \Gamma, A \lor B \\
\Xi_1 \vdash \Gamma, A_i \\
\vdash \Gamma, A_1 \lor A_2
\end{align*}
\]

- \( \Xi_i \) are the proof certificates
Our calculus: \( LKF^a \), \( LK \) with focusing and certificate annotations.

Invertible and non invertible judgements in \( LK \)...

\[
\begin{align*}
\Xi_1 \vdash \Gamma, A, B & \quad \lor_c (\Xi_0, \Xi_1) \\
\Xi_0 \vdash \Gamma, A \lor B & \\
\vdash \Gamma, A_i & \\
\vdash \Gamma, A_1 \lor A_2 &
\end{align*}
\]

- \( \Xi_i \) are the *proof certificates*
- \( \lor_c \) is the clerk
Our calculus: \( \text{LKF}^a, \text{LK} \) with focusing and certificate annotations

Invertible and non-invertible judgements in LK...

\[
\Xi_1 \vdash \Gamma, A, B \quad \lor_c(\Xi_0, \Xi_1) \\
\Xi_0 \vdash \Gamma, A \lor B
\]

\[
\Xi_1 \vdash \Gamma, A_i \\
\Xi_0 \vdash \Gamma, A_1 \lor A_2
\]

- \( \Xi_i \) are the proof certificates
- \( \lor_c \) is the clerk
Our calculus: $\text{LKF}^a$, $\text{LK}$ with focusing and certificate annotations

Invertible and non invertible judgements in $\text{LK}$...

\[
\begin{align*}
\Xi_1 & \vdash \Gamma, A, B \quad \lor_c (\Xi_0, \Xi_1) \\
\Xi_0 & \vdash \Gamma, A \lor B \\
\Xi_1 & \vdash \Gamma, A_i \quad \lor_e (\Xi_0, \Xi_1, i) \\
\Xi_0 & \vdash \Gamma, A_1 \lor A_2
\end{align*}
\]

- $\Xi_i$ are the *proof certificates*
- $\lor_c$ is the clerk
- $\lor_e$ is the expert
Our calculus: $LKF^a$, $LK$ with focusing and certificate annotations

Invertible and non invertible judgements in $LK$...and $LKF^a$

\[
\Xi_1 \vdash \Gamma, A, B \quad \forall_c(\Xi_0, \Xi_1) \\
\Xi_0 \vdash \Gamma, A \lor B
\]

\[
\Xi_0 \vdash \Gamma, A_i \quad \forall_e(\Xi_0, \Xi_1, i) \\
\Xi_0 \vdash \Gamma, A_1 \lor A_2
\]

- $\Xi_i$ are the proof certificates
- $\forall_c$ is the clerk
- $\forall_e$ is the expert
We wanted to do $LK$ proofs, but our calculus is $LKF^a$

But $LKF^a$ just adds decorations to $LK$ sequents

If we remove the decoration, we have immediately

**Theorem (Soundness of $LKF^a$)**

*If an $LKF^a$ sequent is derivable, then its underlying sequent is provable in $LK$*
Kernel and client formulas

The distinction into the invertible and non-invertible rules needs to be reflected in formulas.

Therefore we have notions of

- *kernel* formula, with connectives are marked as inv./non-inv.
- *client* formula, with the usual connectives

...is this a hint on how we could treat Skolemization?
Deskolemization
We wish to extend FPCs to handle Skolemized proofs

The crucial observation:

- Skolemized formulas have client-space names (in a namespace extended with Skolem symbols)
- The kernel uses a different namespace, with eigenvariables!

We need to add a mechanism to handle kernel and client side terms!
Handling client terms

Add to the inference rules a relation between client and kernel terms

- All terms in the signature are related to themselves
- The relation is hereditary with respect to function application
- The client might introduce new terms for eigenvariables

We call the relation `copy`. For the signature $a/0, f/1, g/2$ one has

\[
\text{copy } a \ a \ a \\
\text{copy } (f \ X) \ (f \ U) : \text{copy } X \ U \\
\text{copy } (g \ X \ Y) \ (g \ U \ V) : \text{copy } X \ U, \ \text{copy } Y \ V
\]
Handling client terms

When encountering $\forall x. A$:

- Create an eigenvariable $y$
- Continue checking $[y/x]A$
Handling client terms

When encountering $\forall x. A$:

- Create an eigenvariable $y$
- Assume $(\text{copy } t \ y)$ for some Skolem term $t$
- Continue checking $[y/x]A$ under the assumption that $t$ names $y$
Handling client terms

When encountering $\forall x. A$:

- Create an eigenvariable $y$
- Assume (copy $t \ y$) for some Skolem term $t$
- Continue checking $[y/x]A$ under the assumption that $t$ names $y$

When encountering $\exists x. A$:

- **Query** the certificate for a term $t$
- Proceed checking $[t/x]A$
Handling client terms

When encountering $\forall x. A$:

- Create an eigenvariable $y$
- Assume $(\text{copy } t y)$ for some Skolem term $t$
- Continue checking $[y/x]A$ under the assumption that $t$ names $y$

When encountering $\exists x. A$:

- **Query** the certificate for a term $t$
- Infer a **kernel term** $s$ such that $\text{copy } t s$
- Proceed checking $[s/x]A$
Implementations
Extension to a λProlog checker for foundational proof certificates

Advantages:

- Declarative syntax
- Built-in handling of kernel eigenvariables
- Built-in backtracking and unification for proof-search

Therefore each inference rule is implemented with few lines of code
\textbf{λProlog implementation}

\[
\frac{\Sigma \vdash (\text{copy } t \ s) \quad \Xi_1; \Sigma \vdash \Gamma, [s/x]A \quad \exists_e (\Xi_0, \Xi_1, t)}{\Xi_0; \Sigma \vdash \Gamma, \exists x. \ A}
\]

\text{sync } \Xi_0 (\text{some } A) :-
\text{someE } \Xi_0 \Xi_1 T, 
\text{copy } T S, \text{ sync } \Xi_1 (A \ S).

\[
\frac{\Xi_1; \Sigma, (\text{copy } t \ y) \vdash \Gamma, [y/x]A \quad \forall_c (\Xi_0, \Xi_1, t)}{\Xi_0; \Sigma \vdash \Gamma, \forall x. \ A}
\]

\text{async } \Xi_0 (\text{all } A) :-
\text{allCx } \Xi_0 \Xi_1 T, 
\text{pi } w \ \text{copy } T w \Rightarrow \text{async } \Xi_1 (A \ w).
Copy clauses are similarly handled in a natural fashion:

For every constant term, add:

- copy a a.

For every function term, add:

\[
\text{copy } (f \ X) \ (f \ U) :\! - \! \text{ copy } X \ U.
\]

Proof formats defined for the usual FPC checker needed minimal modification in order to support deskolemization.
Towards a Coq implementation

\(\lambda\)Prolog makes our implementation natural and easy to inspect.

But it has a big runtime system! What if I don’t trust it?

We said that the kernel should be easily reimplementable...
Towards a Coq implementation

\[ \lambda \text{Prolog makes our implementation natural and easy to inspect.} \]

But it has a big runtime system! What if I don’t trust it?

We said that the kernel should be easily reimplementable...

But eigenvars, backtraking search, are difficult to have e.g. in Coq

The situation is evolving: ELPI could allow us to easily do this!
Towards a Coq implementation

Idea: since the internal object of the kernel is an $LK$ proof

- Use the runtime as a preprocessor
- Check with an external tool (e.g. Coq)

The exporter:

- In the spirit of FPCs, describe a “pairing certificate” — actually a predicate saying “these two certificates are equivalent”
- Second parameter of the predicate, use a fully explicit kind of certificate, building the entire $LK$ proof

We are building a Coq checker for the fully explicit certificates
Thank you