EuroProofNet

Introduction to Proof System Interoperability

Frédéric Blanqui

Deduc⊢eam

September 2022

Summary of first lecture

Introduction to:

- logical frameworks
- λ-calculus
- simple types
- dependent types
- rewriting
- $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)
- Dedukti language
- Lambdapi proof assistant

Outline

Encoding logics in $\lambda\Pi/\mathcal{R}$

we have seen what is a theory in the $\lambda\Pi$-calculus modulo rewriting

we are now going to see how to encode logics as $\lambda\Pi/\mathcal{R}$ theories

First-order logic

- the set of terms
  - built from a set of function symbols equipped with an arity
- the set of propositions
  - built from a set of predicate symbols equipped with an arity
  - and the logical connectives $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
- the set of axioms (the actual theory)
- the subset of provable propositions
  - using deduction rules (e.g. natural deduction)

Natural deduction

provability, $\vdash$, is a relation between a sequence of propositions Γ (the assumptions) and a proposition B (the conclusion) inductively defined from introduction and elimination rules for each connective:

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad (\Rightarrow\text{-elim})\ \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash A \quad x \notin \Gamma}{\Gamma \vdash \forall x, A} \qquad (\forall\text{-elim})\ \frac{\Gamma \vdash \forall x, A}{\Gamma \vdash A\{(x, u)\}}$$

...

Encoding of first-order logic

- the set of terms $\qquad \iota : \mathtt{TYPE}$
  - built from a set of function symbols equipped with an arity

    function symbol: $\iota \to \ldots \to \iota \to \iota$

Encoding of first-order logic

- the set of terms $\quad I : \mathtt{TYPE}$
  - built from a set of function symbols equipped with an arity
    function symbol: $I \to \ldots \to I \to I$
- the set of propositions $\quad Prop : \mathtt{TYPE}$
  - built from a set of predicate symbols equipped with an arity
    predicate symbol: $I \to \ldots \to I \to Prop$

Encoding of first-order logic

- the set of terms I : TYPE
  - built from a set of function symbols equipped with an arity
    function symbol: $I \to \ldots \to I \to I$
- the set of propositions $Prop$: TYPE
  - built from a set of predicate symbols equipped with an arity
    predicate symbol: $I \to \ldots \to I \to Prop$
  - and the logical connectives $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
    $\top : Prop$, $\neg : Prop \to Prop$, $\forall : (I \to Prop) \to Prop$, $\ldots$
    we use λ-calculus to encode quantifiers:
    we encode $\forall x, A$ as $\forall(\lambda x : I, A)$

Encoding of first-order logic

- the set of terms $I : \mathtt{TYPE}$
  - built from a set of function symbols equipped with an arity
    function symbol: $I \to \ldots \to I \to I$
- the set of propositions $Prop : \mathtt{TYPE}$
  - built from a set of predicate symbols equipped with an arity
    predicate symbol: $I \to \ldots \to I \to Prop$
  - and the logical connectives $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
    $\top : Prop$, $\neg : Prop \to Prop$, $\forall : (I \to Prop) \to Prop$, $\ldots$
    we use λ-calculus to encode quantifiers:
    we encode $\forall x, A$ as $\forall(\lambda x : I, A)$
- the set of axioms (the actual theory)
- the subset of provable propositions
  - using deduction rules (e.g. natural deduction)
    but how to encode proofs?

Using λ-terms to represent proofs (Curry-de Bruijn-Howard isomorphism)

logic	λ-calculus
proposition	type
proof	λ-term
assumption	variable
$\Rightarrow$	$\rightarrow$
$\Rightarrow$-intro	abstraction
$\Rightarrow$-elim	application
$\forall$	Π
...	...

the Curry-de Bruijn-Howard isomorphism reduces:

- proof-checking to type-checking
- provability to type inhabitation

Using λ-terms to represent proofs (Curry-de Bruijn-Howard isomorphism)

take the rules of natural deduction

$$(\Rightarrow\text{-intro})\ \frac{\Gamma,\quad A \vdash \quad B}{\Gamma \vdash \qquad\qquad A \Rightarrow B}$$

$$(\Rightarrow\text{-elim})\ \frac{\Gamma \vdash \quad A \Rightarrow B \quad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash \quad A \qquad x \notin \Gamma}{\Gamma \vdash \qquad \forall x, A}$$

$$(\forall\text{-elim})\ \frac{\Gamma \vdash \quad \forall x, A}{\Gamma \vdash \qquad A\{(x, u)\}}$$

Using λ-terms to represent proofs (Curry-de Bruijn-Howard isomorphism)

take the rules of natural deduction
by giving a name to every assumption, we get a typing environment

$$A_1, \ldots, A_n \quad \rightsquigarrow \quad x_1 : A_1, \ldots, x_n : A_n$$

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \qquad\qquad A \Rightarrow B}$$

$$(\Rightarrow\text{-elim})\ \frac{\Gamma \vdash \quad A \Rightarrow B \quad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash \quad A \qquad x \notin \Gamma}{\Gamma \vdash \qquad\qquad \forall x, A}$$

$$(\forall\text{-elim})\ \frac{\Gamma \vdash \quad \forall x, A}{\Gamma \vdash \qquad A\{(x, u)\}}$$

Using λ-terms to represent proofs (Curry-de Bruijn-Howard isomorphism)

take the rules of natural deduction
by giving a name to every assumption, we get a typing environment

$$A_1, \ldots, A_n \quad \rightsquigarrow \quad x_1 : A_1, \ldots, x_n : A_n$$

by mapping every deduction rule to a λ-term construction
the typing rules of $\lambda\Pi$ correspond to natural deduction rules!

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A, t : A \Rightarrow B}$$

$$(\Rightarrow\text{-elim})\ \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash t : A \qquad x \notin \Gamma}{\Gamma \vdash \lambda x, t : \forall x, A}$$

$$(\forall\text{-elim})\ \frac{\Gamma \vdash t : \forall x, A}{\Gamma \vdash tu\ : A\{(x, u)\}}$$

Encoding the Curry-de Bruijn-Howard isomorphism

terms of type *Prop* are not types...

but we can interpret a proposition as a type by taking:

$$\mathit{Prf} : \mathit{Prop} \to \mathtt{TYPE}$$

Prf A is the type of proofs of proposition A

Encoding the Curry-de Bruijn-Howard isomorphism

terms of type *Prop* are not types...

but we can interpret a proposition as a type by taking:

$$Prf : Prop \to \mathtt{TYPE}$$

Prf A is the type of proofs of proposition A

but

$$\lambda x : Prf\ A, x \quad : \quad Prf\ A \to Prf\ A$$

and

$$\lambda x : Prf\ A, x \quad \not: \quad Prf(A \Rightarrow A)$$

Encoding the Curry-de Bruijn-Howard isomorphism

terms of type *Prop* are not types...

but we can interpret a proposition as a type by taking:

$$Prf : Prop \to \mathtt{TYPE}$$

Prf A is the type of proofs of proposition A

but

$$\lambda x : Prf\ A, x \quad : \quad Prf\ A \to Prf\ A$$

and

$$\lambda x : Prf\ A, x \quad \not: \quad Prf(A \Rightarrow A)$$

unless we add the rewrite rule

$$Prf(A \Rightarrow B) \quad \hookrightarrow \quad Prf\ A \to Prf\ B$$

Encoding $\forall$

we can do something similar for $\forall : (I \to Prop) \to Prop$ by taking:

$$Prf(\forall A) \quad \hookrightarrow \quad \Pi x : I, Prf(A\,x)$$

Encoding the other connectives

the other connectives can be defined by using a meta-level quantification on propositions:

$$\mathit{Prf}(A \wedge B) \quad \hookrightarrow \quad \Pi\flat : \mathit{Prop}, (\mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}\ \flat) \to \mathit{Prf}\ \flat$$

note that introduction and elimination rules can be derived:

($\wedge$-intro):

$$\lambda a : \mathit{Prf}\ A, \lambda b : \mathit{Prf}\ B, \lambda\flat : \mathit{Prop}, \lambda h : \mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}\ \flat, hab$$

is of type

$$\mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}(A \wedge B)$$

($\wedge$-elim1):

$$\lambda c : \mathit{Prf}(A \wedge B), c\, A\, (\lambda a : \mathit{Prf}\ A, \lambda b : \mathit{Prf}\ B, a)$$

is of type

$$\mathit{Prf}(A \wedge B) \to \mathit{Prf}\ A$$

To summarize: $\lambda\Pi/\mathcal{R}$-theory *FOL* for first-order logic

signature Σ_{FOL}:

I : TYPE
$f : I \to \ldots \to I \to I$ for each function symbol f of arity n
$Prop$: TYPE
$P : I \to \ldots \to I \to Prop$ for each predicate symbol P of arity n
$\top : Prop,\ \neg : Prop \to Prop,\ \forall : (I \to Prop) \to Prop, \ldots$
$Prf : Prop \to$ TYPE
$a : Prf\ A$ for each axiom A

rules $\mathcal{R}_{FOL}$:

$$
\begin{aligned}
Prf(A \Rightarrow B) &\hookrightarrow Prf\ A \to Prf\ B \\
Prf(\forall A) &\hookrightarrow \Pi x : I, Prf(A\,x) \\
Prf(A \wedge B) &\hookrightarrow \Pi\flat : Prop, (Prf\ A \to Prf\ B \to Prf\ \flat) \to Prf\ \flat \\
Prf\,\bot &\hookrightarrow \Pi\flat : Prop, Prf\ \flat \\
Prf(\neg A) &\hookrightarrow Prf\,A \to Prf\,\bot
\end{aligned}
$$

...

Encoding of first-order logic in $\lambda\Pi/FOL$

encoding of terms:

$|x| = x$
$|ft_1 \dots t_n| = f|t_1| \dots |t_n|$

encoding of propositions:

$|Pt_1 \dots t_n| = P|t_1| \dots |t_n|$
$|\top| = \top$
$|A \wedge B| = |A| \wedge |B|$
$|\forall x, A| = \forall(\lambda x : I, |A|)$
$\dots$
$|\Gamma, A| = |\Gamma|, x_{\|\Gamma\|+1} : A$

encoding of proofs:

$$\left| \frac{\pi_{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} (\Rightarrow_i) \right| = \lambda x_{\|\Gamma\|+1} : Prf\ |A|, |\pi_{\Gamma, A \vdash B}|$$

$$\left| \frac{\pi_{\Gamma \vdash A \Rightarrow B} \quad \pi_{\Gamma \vdash A}}{\Gamma \vdash B} (\Rightarrow_e) \right| = |\pi_{\Gamma \vdash A \Rightarrow B}|\ |\pi_{\Gamma \vdash A}|$$

$\dots$

Properties of the encoding in $\lambda\Pi/FOL$

- a term is mapped to a term of type I
- a proposition is mapped to a term of type $Prop$
- a proof of A is mapped to a term of type $Prf\ |A|$

Properties of the encoding in $\lambda\Pi/FOL$

- a term is mapped to a term of type I
- a proposition is mapped to a term of type $Prop$
- a proof of A is mapped to a term of type $Prf\ |A|$

but, if we find a term t of type $Prf\ |A|$, can we deduce that A is provable ?

Properties of the encoding in $\lambda\Pi/FOL$

- a term is mapped to a term of type I
- a proposition is mapped to a term of type $Prop$
- a proof of A is mapped to a term of type $Prf\ |A|$

but, if we find a term t of type $Prf\ |A|$, can we deduce that A is provable ?

- yes, the encoding is conservative: if $Prf\ |A|$ is inhabited then A is provable

proof sketch: because $\hookrightarrow_\beta$ terminates and is confluent, t has a normal form, and terms in normal form can be easily translated back in first-order logic and natural deduction

Multi-sorted first-order logic

for each sort I_k (e.g. point, line, circle), add:

I_k : $\texttt{TYPE}$
$\forall_k : (I_k \to Prop) \to Prop$

$Prf(\forall_k A) \hookrightarrow \Pi x : I_k, Prf(Ax)$

Polymorphic first-order logic

same trick as Curry-de Bruijn-Howard

$Set : \mathtt{TYPE}$
$El : Set \to \mathtt{TYPE}$
$\iota : Set$ for each sort ι
$\forall : \Pi a : Set, (El\ a \to Prop) \to Prop$

$Prf(\forall a p) \hookrightarrow \Pi x : El\ a, Prf(p\, x)$

Higher-order logic

order	quantification on
1	elements
2	sets of elements
3	sets of sets of elements
...	...
ω	any set

Higher-order logic

order	quantification on
1	elements
2	sets of elements
3	sets of sets of elements
...	...
ω	any set

quantification on functions:

$\leadsto : Set \to Set \to Set$

$El(a \leadsto b) \hookrightarrow El\ a \to El\ b$

Higher-order logic

order	quantification on
1	elements
2	sets of elements
3	sets of sets of elements
...	...
ω	any set

quantification on functions:

$\leadsto\ : \mathit{Set} \to \mathit{Set} \to \mathit{Set}$

$\mathit{El}(a \leadsto b) \hookrightarrow \mathit{El}\ a \to \mathit{El}\ b$

quantification on propositions/impredicativity (e.g. $\forall p, p \Rightarrow p$):

$o : \mathit{Set}$

$\mathit{El}\ o \hookrightarrow \mathit{Prop}$

Encoding dependent types

dependent implication:

$\Rightarrow_d \, : \Pi a : Prop, (Prf \; a \to Prop) \to Prop$

$Prf \, (a \Rightarrow_d b) \hookrightarrow \Pi x : Prf \; a, Prf \, (b \, x)$

Encoding dependent types

dependent implication:

$\Rightarrow_d \,:\, \Pi a : Prop, (Prf\ a \to Prop) \to Prop$

$Prf(a \Rightarrow_d b) \hookrightarrow \Pi x : Prf\ a, Prf(b\,x)$

dependent types:

$\leadsto_d \,:\, \Pi a : Set, (El\ a \to Set) \to Set$

$El(a \leadsto_d b) \hookrightarrow \Pi x : El\ a, El(b\,x)$

Encoding dependent types

dependent implication:

$\Rightarrow_d \, : \Pi a : \mathit{Prop}, (\mathit{Prf}\ a \to \mathit{Prop}) \to \mathit{Prop}$

$\mathit{Prf}\,(a \Rightarrow_d b) \hookrightarrow \Pi x : \mathit{Prf}\ a, \mathit{Prf}\,(b\,x)$

dependent types:

$\leadsto_d \, : \Pi a : \mathit{Set}, (\mathit{El}\ a \to \mathit{Set}) \to \mathit{Set}$

$\mathit{El}(a \leadsto_d b) \hookrightarrow \Pi x : \mathit{El}\ a, \mathit{El}(b\,x)$

proofs in object-terms:

$\pi : \Pi p : \mathit{Prop}, (\mathit{Prf}\ p \to \mathit{Set}) \to \mathit{Set}$

$\mathit{El}(\pi\ p\, a) \hookrightarrow \Pi x : \mathit{Prf}\ p, \mathit{El}(a\,x)$

example: $\mathit{div} : \mathit{El}(\iota \leadsto \iota \leadsto_d \lambda y : \mathit{El}\ \iota, \pi(y > 0)(\lambda\_, \iota))$
takes 3 arguments: $x : \mathit{El}\ \iota$, $y : \mathit{El}\ \iota$, $p : \mathit{Prf}(y > 0)$
and returns a term of type $\mathit{El}\ \iota$

Encoding the calculus of constructions

we now have all the ingredients to encode
the calculus of constructions:

system	PTS rule	$\lambda\Pi/\mathcal{R}$ rule
simple types	$\mathtt{TYPE}, \mathtt{TYPE}$	$\mathit{Prf}\,(a \Rightarrow_d b) \hookrightarrow \Pi x : \mathit{Prf}\ a, \mathit{Prf}\,(b\,x)$
polymorphic types	$\mathtt{KIND}, \mathtt{TYPE}$	$\mathit{Prf}\,(\forall ab) \hookrightarrow \Pi x : \mathit{El}\ a, \mathit{Prf}\,(b\,x)$
dependent types	$\mathtt{TYPE}, \mathtt{KIND}$	$\mathit{El}(\pi\ a\ b) \hookrightarrow \Pi x : \mathit{Prf}\ a, \mathit{El}(b\,x)$
type constructors	$\mathtt{KIND}, \mathtt{KIND}$	$\mathit{El}(a \leadsto_d b) \hookrightarrow \Pi x : \mathit{El}\ a, \mathit{El}(b\,x)$

Encoding Functional Pure Type Systems

terms and types:

$$t := x \mid tt \mid \lambda x : t, t \mid \Pi x : t, t \mid s \in \mathcal{S}$$

typing rules:

$$\frac{}{\emptyset \vdash} \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash} \qquad \frac{\Gamma \vdash \quad (x, A) \in \Gamma}{\Gamma \vdash x : A}$$

$$(sort)\ \frac{\Gamma \vdash \quad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2}$$

$$(prod)\ \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad ((s_1, s_2), s_3) \in \mathcal{P}}{\Gamma \vdash \Pi x : A, B : s_3}$$

$$\frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash \Pi x : A, B : s}{\Gamma \vdash \lambda x : A, t : \Pi x : A, B} \qquad \frac{\Gamma \vdash t : \Pi x : A, B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B\{(x, u)\}}$$

$$\frac{\Gamma \vdash t : A \quad A \simeq_\beta A' \quad \Gamma \vdash A' : s}{\Gamma \vdash t : A'}$$

Encoding Functional Pure Type Systems

(Cousineau & Dowek, 2007)

signature:

$U_s : \mathtt{TYPE}$ for each sort $s \in \mathcal{S}$

$El_s : U_s \to \mathtt{TYPE}$

$s_1 : U_{s_2}$ for every $(s_1, s_2) \in \mathcal{A}$

$\pi_{s_1,s_2} : \Pi a : U_{s_1}, (El_{s_1}\ a \to U_{s_2}) \to U_{s_3}$ for every $(s_1, s_2, s_3) \in \mathcal{P}$

rules:

$El_{s_2}\ s_1 \hookrightarrow U_{s_1}$ for every $(s_1, s_2) \in \mathcal{A}$

$El_{s_3}(\pi_{s_1,s_2}\ a\ b) \hookrightarrow \Pi x : El_{s_1}\ a, El_{s_2}(b\, x)$ for every $(s_1, s_2, s_3) \in \mathcal{P}$

encoding:

$|x|_\Gamma = x$

$|s|_\Gamma = s$

$|\lambda x : A, t|_\Gamma = \lambda x : El_s |A|_\Gamma, |t|_{\Gamma,x:A}$ if $\Gamma \vdash A : s$

$|tu|_\Gamma = |t|_\Gamma |u|_\Gamma$

$|\Pi x : A, B|_\Gamma = \pi_{s_1,s_2} |A|_\Gamma (\lambda x : El_{s_1} |A|_\Gamma, |B|_{\Gamma,x:A})$
if $\Gamma \vdash A : s_1$ and $\Gamma, x : A \vdash B : s_2$

Encoding other features

- recursive functions (Assaf 2015, Cauderlier 2016, Férey 2021)
  - different approaches, no general theory
  - encoding in recursors (ongoing work by Felicissimo & Cockx)
- universe polymorphism (Genestier 2020)
  - requires rewriting with matching modulo AC
    or rewriting on AC canonical forms
- η-conversion on function types (Genestier 2020)
- predicate subtyping with proof irrelevance (Hondet 2020)
- co-inductive objects and co-recursion (Felicissimo 2021)

Outline

from slides by Guillaume Burel at the Dedukti school (June 2022)

ITP vs ATP

Limitations of interactive theorem provers (ITP):

- lack of automation
- need for specially trained experts
- bottleneck for widespread use

Limitations of automated theorem provers (ATP):

- lack of confidence
- highly optimized tools
- code too complex to be certified

Cooperation

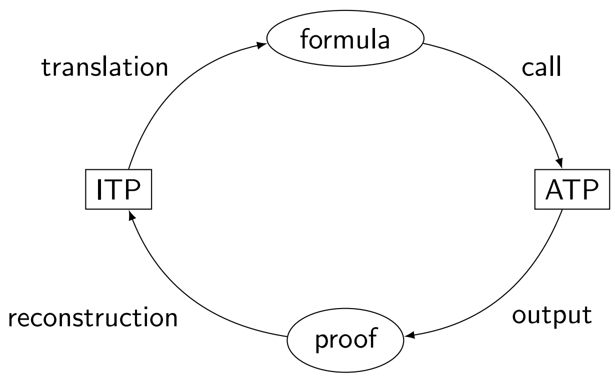
ITP:

- use ATPs to discharge some proof obligations
  e.g. Sledgehammer, SMTCoq

ATP:

- Export proofs that can be independently checked
- Ideally, checkable by a well known tool

Ideal goal


formula
translation
call
ITP
ATP
reconstruction
output
proof

From Lambdapi to ATPs

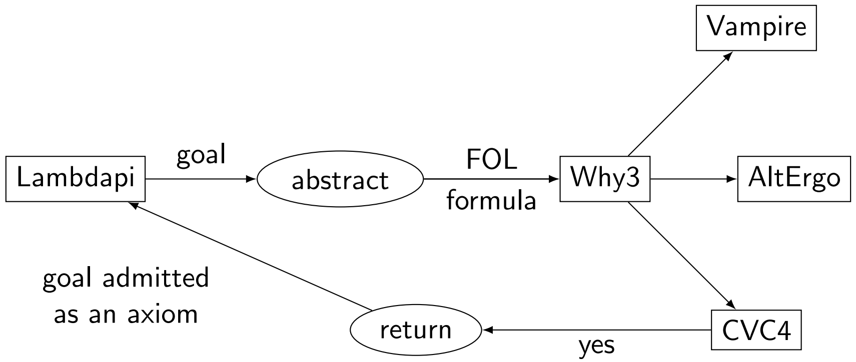
Why3:

- platform for deductive program verification
- able to delegate proofs to many provers
- `https://why3.lri.fr/`

Calling provers within Lambdapi:

- Tactic `why3`

Current `why3` tactic



Trusting ATPs

ATP:

- quite big piece of software
- complex proof calculi
- finely tuned, optimization hacks

Trust?

- Originally, only answer "yes"/"no" (more often, "maybe")
- More and more, produce proof traces/big steps proofs

Trusting ATPs

To increase confidence:

- either build a certified proof checker for proof traces
  e.g. Coq certified checker for DRAT proof traces of SAT solvers

- or directly produce a proof checkable by your favorite assistant

$$\begin{array}{c}\text{Problem}\\ \texttt{.p}\end{array} \longrightarrow \boxed{\begin{array}{c}\text{Instrumented}\\ \text{ATP}\end{array}} \longrightarrow \begin{array}{c}\text{Proof}\\ \texttt{.dk}\end{array}$$

Instrumenting a prover to produce proofs

Pros:

- Access to all needed informations

Cons:

- Needs to embed the calculus of the prover into Dedukti
- Needs to know precisely the code of the prover

more or less easy depending complexity of code/proof calculus
easier if proof output designed from the start (e.g. Zenon)

$\Rightarrow$ can only be done for a few provers

Provers outputing Dedukti proofs

- iProverModulo:
  extension of iProver for Deduction Modulo Theory
  `https://github.com/gburel/iProverModulo.git`
- ZenonModulo:
  extension of Zenon for Deduction Modulo Theory + Arithmetic
  `https://github.com/Deducteam/zenon_modulo.git`
- ArchSAT:
  SMT solver
  `https://github.com/Gbury/archsat`

Translating proofs

First, need to carefully choose in which theory we are working
e.g. FOL

Then, two approaches:

- Directly translate proofs into Dedukti, e.g. iProverModulo
- Embedding the proof calculus into Dedukti, e.g. ZenonModulo

iProverModulo (Burel 2011)

Patch to iProver (Korovin 2008)

iProver: Combination of two proof procedures:

- Inst-Gen
- Ordered resolution

iProverModulo: add support for Deduction Modulo Theory

Resolution Calculus

Literal: atom A or negation of atom $\neg A$
Clause: set/disjunction of literals $L_1 \vee \ldots \vee L_m$ ($m \geq 0$)
Problem: set/conjunction of clauses $C_1 \wedge \ldots \wedge C_k$

Derive new clauses using

$$\frac{A, C \quad \neg B, D}{C\sigma, D\sigma} \quad \sigma = mgu(A, B)$$

until the empty clause is produced

Translation of clauses

we want to prove $(C_1 \wedge \ldots \wedge C_k) \Rightarrow \bot$

$(C_1 \wedge \ldots \wedge C_k) \Rightarrow \bot$ is equivalent to $(C_1 \Rightarrow \bot) \vee \ldots \vee (C_k \Rightarrow \bot)$
$(L_1 \vee \ldots \vee L_m) \Rightarrow \bot$ is equivalent to $(L_1 \Rightarrow \bot) \wedge \ldots \wedge (L_m \Rightarrow \bot)$

$C = \{L_1, \ldots, L_m\}$ which corresponds to $\forall x_1, \ldots, \forall x_p, L_1 \vee \ldots \vee L_m$, where $x_1, .., x_p$ are the free variables of $L_1, .., L_m$, is translated as:

$$\Pi x_1 : I, \ldots \Pi x_p : I, \Pi\flat : Prop, |L_1|_\flat \to \ldots \to |L_m|_\flat \to Prf\,\flat$$

with $|A|_\flat = Prf\,A \to Prf\,\flat$ and $|\neg A|_\flat = (Prf\,A \to Prf\,\flat) \to Prf\,\flat$

(remember that $Prf\,\bot \hookrightarrow \Pi\flat : Prop, Prf\,\flat$)

Translation of propositional resolution

$$\frac{A, L_1, \ldots, L_m \quad \neg A, L_{m+1}, \ldots, L_n}{L_1, \ldots, L_n}$$

given $c : |A, L_1, \ldots, L_m|$

$$= \Pi\flat : \mathit{Prop}, |A|_\flat \to |L_1|_\flat \to \ldots \to |L_m|_\flat \to \mathit{Prf}\, \flat$$

and $d : |\neg A, L_{m+1}, \ldots, L_n|$

$$= \Pi\flat : \mathit{Prop}, (|A|_\flat \to \mathit{Prf}\, \flat) \to |L_{m+1}|_\flat \to \ldots \to |L_n|_\flat \to \mathit{Prf}\, \flat$$

we obtain

$$e : |L_1, \ldots, L_n| = \Pi\flat : \mathit{Prop}, |L_1|_\flat \to \ldots \to |L_n|_\flat \to \mathit{Prf}\, \flat$$

by taking

$$e = \lambda\flat, \lambda\bar{l}_1, \ldots, \lambda\bar{l}_n, c\, \flat\, (\lambda a, d\, \flat\, (\lambda\bar{a}, \bar{a} a)\, \bar{l}_{m+1} \ldots \bar{l}_n)\, \bar{l}_1 \ldots \bar{l}_m$$

Limits

Can handle various simplification rules, rewriting

Can be extended to superposition (E, Vampire, . . .)

But:

- works if the proof uses resolution only (i.e. no Inst-Gen)
- no translation of the transformation into clauses

ZenonModulo

(Delahaye, Doligez, Gilbert, Halmagrand, and Hermant, 2013)

- extension of Zenon to Deduction Modulo Theory
- tableau-based
- polymorphic first-order logic with equality

Tableau proofs

- proofs by contradiction
- roughly bottom-up sequent-calculus with metavariables

$$\frac{P, \neg P}{\odot}\ \odot \qquad \frac{\neg(A \Rightarrow B)}{A, \neg B}\ \alpha_{\neg\Rightarrow} \qquad \frac{\neg(A \land B)}{\neg A \quad \mid \quad \neg B}\ \beta_{\neg\land}$$

Example of proof:

$$\cfrac{\cfrac{\cfrac{\neg(P \Rightarrow (P \land P))}{P}\ \alpha_{\neg\Rightarrow}}{\neg(P \land P)}}{\cfrac{\neg P}{\odot}\ \odot \qquad \cfrac{\neg P}{\odot}\ \odot}\ \beta_{\neg\land}$$

Deep embedding of proof calculus

$$\frac{P, \neg P}{\odot}\ \odot:$$

```
symbol Rax p : Prf p → Prf (¬ p) → Prf ⊥;
```

$$\frac{\neg(A \Rightarrow B)}{A, \neg B}\ \alpha_{\neg\Rightarrow}:$$

```
symbol R¬⇒ a b :
  (Prf a → Prf(¬ b) → Prf ⊥) → Prf (¬(a ⇒ b)) → Prf ⊥;
```

$$\frac{\neg(A \wedge B)}{\neg A \quad \mid \quad \neg B}\ \beta_{\neg\wedge}:$$

```
symbol R¬∧ a b : (Prf (¬ a) → Prf ⊥)
  → (Prf(¬ b) → Prf ⊥) → Prf (¬(a ∧ b)) → Prf ⊥;
```

Deep translation of the example

$$
\begin{array}{c}
\neg(P \Rightarrow (P \wedge P)) \\
\hline
P \\
\hline
\neg(P \wedge P) \\
\hline
\begin{array}{cc}
\dfrac{\neg P}{\odot}\ \odot & \qquad \dfrac{\neg P}{\odot}\ \odot
\end{array}
\end{array}
\begin{array}{l}
\alpha_{\neg\Rightarrow} \\
\\
\beta_{\neg\wedge} \\
\\
\end{array}
$$

```
opaque symbol goal : Prf ¬(p ⇒ (p ∧ p)) → Prf ⊥ ≔
  R¬⇒ p (p ∧ p) (λ π, R¬∧ p p (Rax p π) (Rax p π));
```

Making the embedding more shallow

by reducing it to Natural Deduction:

$$(\wedge I)\ \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \quad (\wedge El)\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad (\wedge Er)\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$$

$$(\Rightarrow I)\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \quad (\Rightarrow E)\ \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

Natural Deduction in Lambdapi:

```
symbol ∧I p q : Prf p → Prf q → Prf (p ∧ q);
symbol ∧El p q : Prf (p ∧ q) → Prf p;
symbol ∧Er p q : Prf (p ∧ q) → Prf q;

symbol ⇒I p q : (Prf p → Prf q) → Prf (p ⇒ q);
symbol ⇒E p q : Prf (p ⇒ q) → Prf p → Prf q;
```

Defining Tableau rules in ND

```
rule Rax ↪ λ p h π, ¬E p π h;

rule R¬∧ ↪ λ p q h1 h2 h3,
  h1 (¬I p (λ h5, h2 (¬I q (λ h6,
  ¬E (p ∧ q) h3 (∧I p q h5 h6)))));

rule R¬⇒ ↪ λ p q h1 h2,
  ¬E (p ⇒ q) h2 (⇒I p q (λ h3, ⊥E (h1 h3
  (¬I q (λ h4, ¬E (p ⇒ q) h2 (⇒I p q (λ _, h4))))) q));
```

correctness follows from subject reduction
which is checked automatically by Lambdapi!

```
compute goal;
assert ⊢ goal ≡ λ h2, ¬E (p ⇒ (p ∧ p)) h2 (⇒I p (p ∧ p)
  (λ h3, ⊥E (¬E (p ⇒ (p ∧ p)) h2
    (⇒I p (p ∧ p) (λ _, ∧I p p h3 h3))) (p ∧ p)));
```

Making it even more shallow

Reduce Natural Deduction thanks to the shallow encoding of FOL

```
rule ⇒I ↪ λ p q π, π;
rule ⇒E ↪ λ p q π, π;

rule ∧I ↪ λ p q πp πq r πp⇒q⇒r, πp⇒q⇒r πp πq;
rule ∧El ↪ λ p q πp∧q, πp∧q p (λ x _, x);
rule ∧Er ↪ λ p q πp∧q, πp∧q q (λ _ x, x);
```

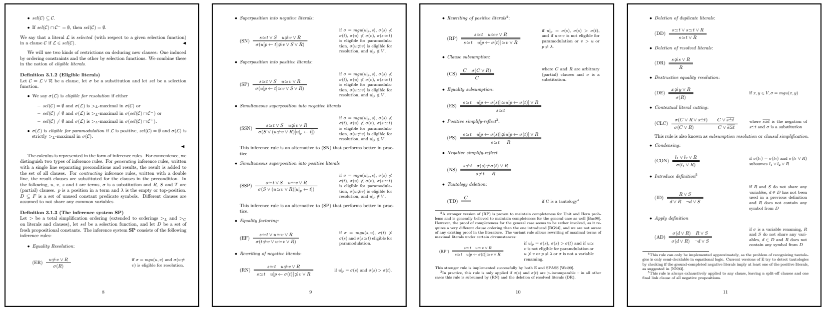
```
compute goal;
assert ⊢ goal ≡
  λ h2, h2 (λ h3, h2 (λ _ _ π, π h3 h3) (p ∧ p));
```

Limits of instrumentation

Provers can be hard to instrument to produce Dedukti proofs

- large piece of software
- developers not expert in $\lambda\Pi$-calculus modulo theory
- non stable and quite big proof calculus

Proof calculus of E



Proof trace

But often, provers produce at least a proof trace:

- list of formulas that were derived to obtain the proof
- sometimes with more information
  - premises
  - name of the inference rules
  - theory
  - ...

Example of trace: TSTP format

Output format of E, Vampire, Zipperposition, ...

- list of formulas
- annotated by an inference tree whose leaves are other formulas

```
cnf(c_0_60,plain,
    ( join(X1,join(X2,X3)) = join(X2,join(X1,X3)) ),
    inference(rw,[status(thm)],
      [inference(spm,[status(thm)],[c_0_30,c_0_18]),
       c_0_30])).
```

Example of trace: TSTP format

Output format of E, Vampire, Zipperposition, ...

- list of formulas
- annotated by an inference tree whose leaves are other formulas

```
cnf(c_0_60,plain,
    ( join(X1,join(X2,X3)) = join(X2,join(X1,X3)) ),
    inference(rw,[status(thm)],
      [inference(spm,[status(thm)],[c_0_30,c_0_18]),
       c_0_30])).
```

Independent of the proof calculus

Proof reconstruction

Use the content of the proof trace to reconstruct a Dedukti proof

Idea:

- Prove each step using a Dedukti producing tool
- Combine those proofs to get a proof of the original formula

Try to be agnostic:

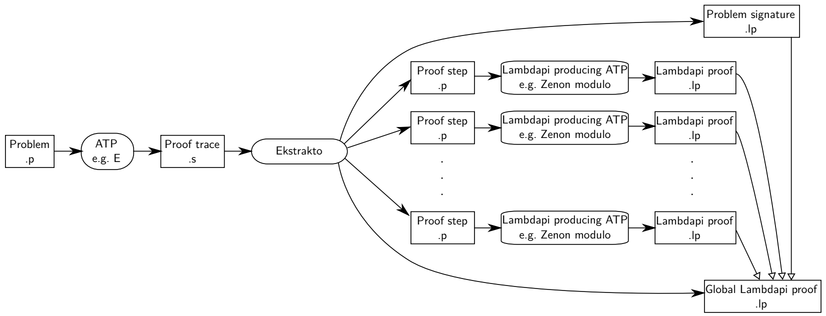
- w.r.t. the prover that produces the trace
- w.r.t. the prover that reproves the steps

Ekstrakto (El Haddad 2021)

- Input: TSTP proof trace
- Output: Reconstructed Lambdapi proof

https://github.com/Deducteam/ekstrakto

Ekstrakto architecture



Experimental evaluation

Benchmark:

- CNF problems of TPTP v7.4.0 (8118 files)

Trace producers:

- E and Vampire

Step provers:

- ZenonModulo and ArchSat

Results

Percentage of reconstructed proof steps

Prover	% E	% Vampire
ZenonModulo	87%	60%
ArchSAT	92%	81%
ZenonModulo ∪ *ArchSAT*	95%	85%

Percentage of completely reconstructed proofs

Prover	% E TSTP	% Vampire TSTP
ZenonModulo	45%	54%
ArchSAT	56%	74%
ZenonModulo ∪ *ArchSAT*	69%	83%

Non provable steps

Problem:

- some steps are not provable
  their conclusion is not a logical consequence of their premises
- OK because they preserve provability
- but Ekstrakto cannot work for them

Non provable steps

Problem:

- some steps are not provable
  their conclusion is not a logical consequence of their premises
- OK because they preserve provability
- but Ekstrakto cannot work for them

Main instance: Skolemization

$$\Gamma, \forall \vec{x}, \exists y, A[\vec{x}, y] \vdash B \text{ iff } \Gamma, \forall \vec{x}, A[\vec{x}, f(\vec{x})] \vdash B \text{ for a fresh } f$$

Present in the CNF transformation used by almost all ATPs

Skonverto (El Haddad 2021)

Inputs:

- an axiom and its Skolemized version
- a Lambdapi proof using the latter

Output:

- a Lambdapi proof using the non-Skolemized axiom

Content

Implementation of Dowek & Werner's constructive proof of Skolem theorem (2005) in the context of first-order natural deduction

Problem:

- the proof has to be in normal form
- also w.r.t. so-called commuting cuts

Commuting cuts

$$\cfrac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \wedge D \qquad \Gamma, B \vdash C \wedge D}{\cfrac{\Gamma \vdash C \wedge D}{\Gamma \vdash C}\,\wedge_{EI}}\,\vee_E$$

$$\rightsquigarrow$$

$$\cfrac{\Gamma \vdash A \vee B \qquad \cfrac{\Gamma, A \vdash C \wedge D}{\Gamma, A \vdash C}\,\wedge_{EI} \qquad \cfrac{\Gamma, B \vdash C \wedge D}{\Gamma, B \vdash C}\,\wedge_{EI}}{\Gamma \vdash C}\,\vee_E$$

Reducing commuting cuts

If we work on shallow proofs, these cuts are no longer visible

⇒ we need to stay at the ND level
and add rules to reduce commuting cuts:

```
rule ∧El $c $d (∨E $a $b $paorb ($c ∧ $d) $pac $pbc)
  ↪ ∨E $a $b $paorb $c (λ pa, ∧El $c $d ($pac pa))
    (λ pb, ∧El $c $d ($pbc pb));
```

Example proof with Skolem symbol

```
symbol goal
  (ax_tran : Prf (∀ (λ X1, ∀ (λ X2, ∀ (λ X3,
     (p X1 X2) ⇒ ((p X2 X3) ⇒ (p X1 X3)))))))
  // skolemized version of
  // (ax_step : Prf (∀ (λ X, ∃ (λ Y, (p X (s Y))))))
  (ax_step : Prf (∀ (λ X, (p X (s (f X))))))
  (ax_congr : Prf (∀ (λ X1, ∀ (λ X2,
     (p X1 X2) ⇒ (p (s X1) (s X2))))))
  (ax_goal : Prf (¬ (∃ (λ X, ((p a (s (s X))))))))
  : Prf ⊥
≔ ax_goal (∃I (λ X, p a (s (s X))) (f (f a))
   (ax_tran a (s (f a)) (s (s (f (f a))))
     (ax_step a)
     (ax_congr (f a) (s (f (f a))) (ax_step (f a)))));
```

Example proof without Skolem symbol generated by Skonverto

```
symbol goal
  (ax_tran : Prf (∀ (λ X1, ∀ (λ X2, ∀ (λ X3,
     (p X1 X2) ⇒ ((p X2 X3) ⇒ (p X1 X3)))))))
  (ax_step : Prf (∀ (λ X, ∃ (λ Y, (p X (s Y))))))
  (ax_congr : Prf (∀ (λ X1, ∀ (λ X2,
     (p X1 X2) ⇒ (p (s X1) (s X2))))))
  (ax_goal : Prf (¬ (∃ (λ X4, ((p a (s (s X4))))))))
  : Prf ⊥
≔  ax_goal (λ r h, ∃E (λ z, p a (s z)) (ax_step a) r
        (λ z a1, ∃E (λ z0, p z (s z0)) (ax_step z) r
        (λ z0 a2, h z0 (ax_tran a (s z) (s (s z0)) a1
            (ax_congr z (s z0) a2)))));
```

Conclusion

Instrumenting a prover to produce Dedukti proofs

- good if you start your prover from scratch

Reconstructing proofs

- more adapted for existing provers
- cannot reconstruct all proofs
- useful for proof assistants using provers internally
  e.g. PVS, Atelier B

Putting everything together