

## Introduction to Proof System Interoperability

Frédéric Blanqui

Deduc⊢eam



September 2022

## Summary of first lecture

Introduction to:

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

#### Outline

#### Introduction

Lambda-Pi-calculus modulo rewriting Lambda-calculus Simple types Dependent types Pure Type Systems Rewriting

Dedukti language

Lambdapi proof assistant

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

Automated Theorem Provers

Intrumenting provers for Dedukti proof production Reconstructing proofs

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  $\top$ ,  $\bot$ ,  $\neg$ ,  $\Rightarrow$ ,  $\land$ ,  $\lor$ ,  $\Leftrightarrow$ ,  $\forall$ ,  $\exists$
- 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  $\Gamma$  (the assumptions) and a proposition *B* (the conclusion) inductively defined from introduction and elimination rules for each connective:

$$(\Rightarrow-intro) \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \quad (\Rightarrow-elim) \frac{\Gamma \vdash A \Rightarrow B}{\Gamma \vdash B}$$
$$(\forall-intro) \frac{\Gamma \vdash A \times \notin \Gamma}{\Gamma \vdash \forall x, A} \quad (\forall-elim) \frac{\Gamma \vdash \forall x, A}{\Gamma \vdash A\{(x, u)\}}$$

. . .

• the set of terms

/ : TYPE

- built from a set of function symbols equipped with an arity

function symbol:  $I \rightarrow \ldots \rightarrow I \rightarrow I$ 

• the set of terms

/ : TYPE

- built from a set of function symbols equipped with an arity function symbol:  $I \rightarrow \ldots \rightarrow I \rightarrow I$ 

- the set of propositions **Prop** : TYPE
  - built from a set of predicate symbols equipped with an arity predicate symbol:  $I \rightarrow \ldots \rightarrow I \rightarrow Prop$

• the set of terms

/ : TYPE

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

- and the logical connectives  $\top$ ,  $\bot$ ,  $\neg$ ,  $\Rightarrow$ ,  $\land$ ,  $\lor$ ,  $\Leftrightarrow$ ,  $\forall$ ,  $\exists$  $\top$ : *Prop*,  $\neg$ : *Prop*  $\rightarrow$  *Prop*,  $\forall$ : ( $I \rightarrow Prop$ )  $\rightarrow$  *Prop*, ... we use  $\lambda$ -calculus to encode quantifiers: we encode  $\forall x, A$  as  $\forall (\lambda x : I, A)$ 

• the set of terms

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

- and the logical connectives  $\top$ ,  $\bot$ ,  $\neg$ ,  $\Rightarrow$ ,  $\land$ ,  $\lor$ ,  $\Leftrightarrow$ ,  $\forall$ ,  $\exists$  $\top$ : *Prop*,  $\neg$ : *Prop*  $\rightarrow$  *Prop*,  $\forall$ : ( $I \rightarrow Prop$ )  $\rightarrow$  *Prop*, ... we use  $\lambda$ -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?

logic	$\lambda$ -calculus
proposition	type
proof	$\lambda$ -term
assumption	variable
$\Rightarrow$	$\rightarrow$
$\Rightarrow$ -intro	abstraction
$\Rightarrow$ -elim	application
$\forall$	П

the Curry-de Bruijn-Howard isomorphism reduces:

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

take the rules of natural deduction

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

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

 $A_1,\ldots,A_n \quad \rightsquigarrow \quad \mathbf{x_1}:A_1,\ldots,\mathbf{x_n}:A_n$ 

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

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  $\lambda$ -term construction the typing rules of  $\lambda\Pi$  correspond to natural deduction rules!

$$(\Rightarrow-intro) \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A, t : A \Rightarrow B}$$
$$(\Rightarrow-elim) \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B}$$
$$(\forall-intro) \frac{\Gamma \vdash t : A \quad x \notin \Gamma}{\Gamma \vdash \lambda x, t : \forall x, A}$$
$$(\forall-elim) \frac{\Gamma \vdash t : \forall x, A}{\Gamma \vdash t u : A\{(x, u)\}}$$

## 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:

 $Prf : Prop \rightarrow TYPE$ 

*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:

 $Prf : Prop \rightarrow TYPE$ 

*Prf* A is the type of proofs of proposition A

but

 $\lambda x$ : Prf A, x : Prf A  $\rightarrow$  Prf A

and

 $\lambda x$ : **Prf** A, x / **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:

 $Prf : Prop \rightarrow TYPE$ 

*Prf* A is the type of proofs of proposition A

but

 $\lambda x$ : Prf A, x : Prf A  $\rightarrow$  Prf A

and

 $\lambda x$ : **Prf** A, x / **Prf** (A  $\Rightarrow$  A)

unless we add the rewrite rule

 $Prf(A \Rightarrow B) \hookrightarrow Prf A \rightarrow Prf B$ 

## Encoding $\forall$

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

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

#### Encoding the other connectives

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

 $Prf(A \land B) \hookrightarrow \Pi \flat : Prop, (Prf A \rightarrow Prf B \rightarrow Prf \flat) \rightarrow Prf \flat$ 

note that introduction and elimination rules can be derived:

 $(\wedge -intro)$ :

 $\lambda a : Prf A, \lambda b : Prf B, \lambda \flat : Prop, \lambda h : Prf A \rightarrow Prf B \rightarrow Prf \flat, hab$ is of type  $Prf A \rightarrow Prf B \rightarrow Prf(A \land B)$ 

 $(\land -elim1)$ :

 $\lambda c : Prf(A \land B), c \land (\lambda a : Prf \land \lambda b : Prf \land B, a)$ is of type  $Prf(A \land B) \rightarrow Prf \land A$ 

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

signature  $\Sigma_{FOL}$ : I : TYPE  $f : I \rightarrow ... \rightarrow I \rightarrow I$  for each function symbol f of arity n Prop : TYPE  $P : I \rightarrow ... \rightarrow I \rightarrow Prop$  for each predicate symbol P of arity n  $\top : Prop, \neg : Prop \rightarrow Prop, \forall : (I \rightarrow Prop) \rightarrow Prop, ...$   $Prf : Prop \rightarrow TYPE$ a : Prf A for each axiom A

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

$$\begin{array}{rcl} \Pr f(A \Rightarrow B) & \hookrightarrow & \Pr f \ A \to \Pr f \ B \\ \Pr f(\forall A) & \hookrightarrow & \Pi x : I, \Pr f(A x) \\ \Pr f(A \land B) & \hookrightarrow & \Pi \flat : \Pr op, (\Pr f \ A \to \Pr f \ B \to \Pr f \ \flat) \to \Pr f \ \flat \\ \Pr f(\bot & \hookrightarrow & \Pi \flat : \Pr op, \Pr f \ \flat \\ \Pr f(\neg A) & \hookrightarrow & \Pr f \ A \to \Pr f \ \bot \end{array}$$

. . .

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

	encoding of propositions:
	$ Pt_1\ldots t_n  = P t_1 \ldots t_n $
encoding of terms:	$ \top  = \top$
x  = x	$ A \wedge B  =  A  \wedge  B $
$ ft_1\ldots t_n =f t_1 \ldots t_n $	$ \forall x, A  = \forall (\lambda x : I,  A )$
	$ \Gamma, A  =  \Gamma , x_{\ \Gamma\ +1} : A$

encoding of proofs:

$$\begin{vmatrix} \frac{\pi_{\Gamma,A\vdash B}}{\Gamma\vdash A\Rightarrow B} (\Rightarrow_i) \end{vmatrix} = \lambda x_{\|\Gamma\|+1} : \Pr [A|, |\pi_{\Gamma,A\vdash B}| \\ \frac{\pi_{\Gamma\vdash A\Rightarrow B} \quad \pi_{\Gamma\vdash A}}{\Gamma\vdash B} (\Rightarrow_e) \end{vmatrix} = |\pi_{\Gamma\vdash A\Rightarrow B}| |\pi_{\Gamma\vdash A}|$$

. . .

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

- a term is mapped to a term of type /
- 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 /
- 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:

 $I_k : \text{TYPE} \\ \forall_k : (I_k \to Prop) \to Prop \\ Prf(\forall_k A) \hookrightarrow \Pi x : I_k, Prf(Ax)$ 

## Polymorphic first-order logic

same trick as Curry-de Bruijn-Howard

Set : TYPE  $EI : Set \rightarrow TYPE$   $\iota : Set$  $\forall : \Pi a : Set, (EI a \rightarrow Prop) \rightarrow Prop$ 

for each sort  $\iota$ 

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

# Higher-order logic

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		
$\omega$	any set		

quantification on functions:

 $\stackrel{\sim}{\rightarrow} : Set \rightarrow Set \rightarrow Set$  $El(a \stackrel{\sim}{\rightarrow} b) \stackrel{\leftarrow}{\rightarrow} El a \rightarrow El b$ 

## Higher-order logic

order	quantification on	
1	elements	
2	sets of elements	
3	sets of sets of elements	
$\omega$	any set	

quantification on functions:

 $\rightsquigarrow: \textit{Set} \rightarrow \textit{Set} \rightarrow \textit{Set}$ 

 $El(a \rightsquigarrow b) \hookrightarrow El a \rightarrow El b$ 

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

o : Set

 $El o \hookrightarrow Prop$ 

## Encoding dependent types

dependent implication:

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

## Encoding dependent types

dependent implication:

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

dependent types:

 $\sim_d$ :  $\Pi a$ : Set, (El  $a \rightarrow Set$ )  $\rightarrow Set$ El( $a \sim_d b$ )  $\hookrightarrow \Pi x$ : El a, El(bx)

#### Encoding dependent types

dependent implication:

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

dependent types:

 $\sim_d$ :  $\Pi a$ : Set, (El  $a \rightarrow$  Set)  $\rightarrow$  Set El( $a \sim_d b$ )  $\hookrightarrow \Pi x$ : El a, El(b x)

proofs in object-terms:

 $\pi : \Pi p : Prop, (Prf \ p \to Set) \to Set$   $El(\pi \ p \ a) \hookrightarrow \Pi x : Prf \ p, El(a \ x)$ example:  $div : El(\iota \hookrightarrow \iota \hookrightarrow_d \lambda y : El \ \iota, \pi(y > 0)(\lambda_-, \iota))$ takes 3 arguments:  $x : El \ \iota, \ y : El \ \iota, \ p : Prf(y > 0)$ and returns a term of type  $El \ \iota$ 

## 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	TYPE, TYPE	$Prf(a \Rightarrow_d b) \hookrightarrow \Pi x : Prf a, Prf(bx)$
polymorphic types	KIND, TYPE	$Prf(\forall ab) \hookrightarrow \Pi x : El a, Prf(bx)$
dependent types	TYPE, KIND	${\it El}(\pi  a  b) \hookrightarrow \Pi x : {\it Prf}  a, {\it El}(b  x)$
type constructors	KIND, KIND	$El(a \sim_d b) \hookrightarrow \Pi x : El a, El(bx)$

# Encoding Functional Pure Type Systems terms and types:

$$t \coloneqq x \mid tt \mid \lambda x : t, t \mid \Pi x : t, t \mid s \in S$$

typing rules:

$$\frac{\overline{\emptyset} \vdash}{\overline{\emptyset} \vdash} \quad \frac{\overline{\Gamma} \vdash A : s}{\overline{\Gamma, x : A \vdash}} \quad \frac{\overline{\Gamma} \vdash (x, A) \in \overline{\Gamma}}{\overline{\Gamma} \vdash x : A} \\
(sort) \frac{\overline{\Gamma} \vdash (s_1, s_2) \in \mathcal{A}}{\overline{\Gamma} \vdash s_1 : s_2} \\
(prod) \frac{\overline{\Gamma} \vdash A : s_1 \quad \overline{\Gamma, x : A \vdash B : s_2} \quad ((s_1, s_2), s_3) \in \mathcal{P}}{\overline{\Gamma} \vdash \overline{\Pi x : A, B : s_3}} \\
\frac{\overline{\Gamma, x : A \vdash t : B} \quad \overline{\Gamma} \vdash \overline{\Pi x : A, B : s}}{\overline{\Gamma} \vdash \lambda x : A, t : \overline{\Pi x : A, B}} \quad \frac{\overline{\Gamma} \vdash t : \overline{\Pi x : A, B} \quad \overline{\Gamma} \vdash u : A}{\overline{\Gamma} \vdash tu : B\{(x, u)\}} \\
\frac{\overline{\Gamma} \vdash t : A \quad A \simeq_{\beta} A' \quad \overline{\Gamma} \vdash A' : s}{\overline{\Gamma} \vdash t : A'}$$

## Encoding Functional Pure Type Systems

(Cousineau & Dowek, 2007)

#### signature:

 $\begin{array}{ll} U_{s}: \texttt{TYPE} & \text{for each sort } s \in \mathcal{S} \\ El_{s}: U_{s} \to \texttt{TYPE} & \\ s_{1}: U_{s_{2}} & \text{for every } (s_{1}, s_{2}) \in \mathcal{A} \\ \pi_{s_{1}, s_{2}}: \Pi a: U_{s_{1}}, (El_{s_{1}} a \to U_{s_{2}}) \to U_{s_{3}} & \text{for every } (s_{1}, s_{2}, s_{3}) \in \mathcal{P} \end{array}$ 

#### rules:

$\mathit{El}_{s_2}  s_1 \hookrightarrow \mathit{U}_{s_1}$	for every $(\mathit{s}_1, \mathit{s}_2) \in \mathcal{A}$
$\mathit{El}_{s_3}(\pi_{s_1,s_2}  a  b) \hookrightarrow \Pi x : \mathit{El}_{s_1}  a, \mathit{El}_{s_2}(b  x)$	for every $(\mathit{s}_1, \mathit{s}_2, \mathit{s}_3) \in \mathcal{P}$

#### encoding:

$$\begin{aligned} |x|_{\Gamma} &= x \\ |s|_{\Gamma} &= s \\ |\lambda x : A, t|_{\Gamma} &= \lambda x : El_{s}|A|_{\Gamma}, |t|_{\Gamma, x:A} & \text{if } \Gamma \vdash A : s \\ |tu|_{\Gamma} &= |t|_{\Gamma}|u|_{\Gamma} \\ |\Pi x : A, B|_{\Gamma} &= \pi_{s_{1}, s_{2}}|A|_{\Gamma}(\lambda x : El_{s_{1}}|A|_{\Gamma}, |B|_{\Gamma, x:A}) \\ & \text{if } \Gamma \vdash A : s_{1} \text{ and } \Gamma, x : A \vdash B : s_{2} \end{aligned}$$

#### 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
- $\eta$ -conversion on function types (Genestier 2020)
- predicate subtyping with proof irrelevance (Hondet 2020)
- co-inductive objects and co-recursion (Felicissimo 2021)
### Outline

#### Introduction

Lambda-Pi-calculus modulo rewriting Lambda-calculus Simple types Dependent types Pure Type Systems Rewriting

Dedukti language

Lambdapi proof assistant

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

Automated Theorem Provers Intrumenting provers for Dedukti proof production Reconstructing proofs 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





# 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

Problem	Instrumented		Proof
.p	ATP	,	.dk

### 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.  $\ensuremath{\mathsf{FOL}}$ 

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  $L_1 \lor \ldots \lor L_m \ (m \ge 0)$ Problem: set/conjunction of clauses  $C_1 \land \ldots \land C_k$ 

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  $(C_1 \land \ldots \land C_k) \Rightarrow \bot$  $(C_1 \land \ldots \land C_k) \Rightarrow \bot$  is equivalent to  $(C_1 \Rightarrow \bot) \lor \ldots \lor (C_k \Rightarrow \bot)$  $(L_1 \lor \ldots \lor L_m) \Rightarrow \bot$  is equivalent to  $(L_1 \Rightarrow \bot) \land \ldots \land (L_m \Rightarrow \bot)$ 

 $C = \{L_1, \ldots, L_m\}$  which corresponds to  $\forall x_1, \ldots, \forall x_p, L_1 \lor \ldots \lor L_m$ , where  $x_1, \ldots, x_p$  are the free variables of  $L_1, \ldots, L_m$ , 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  $|A|_{\flat} = PrfA \rightarrow Prf\flat$  and  $|\neg A|_{\flat} = (PrfA \rightarrow Prf\flat) \rightarrow Prf\flat$ 

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

# 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 
$$c : |A, L_1, \dots, L_m|$$
  
 $= \square b : Prop, |A|_b \rightarrow |L_1|_b \rightarrow \dots \rightarrow |L_m|_b \rightarrow Prf b$   
and  $d : |\neg A, L_{m+1}, \dots, L_n|$   
 $= \square b : Prop, (|A|_b \rightarrow Prf b) \rightarrow |L_{m+1}|_b \rightarrow \dots \rightarrow |L_n|_b \rightarrow Prf b$ 

we obtain

$$e: |L_1, \dots, L_n| = \Pi \flat : \operatorname{Prop}, |L_1|_{\flat} \to \dots \to |L_n|_{\flat} \to \operatorname{Prf} \flat$$
  
by taking  
$$e = \lambda \flat, \lambda \overline{I}_1, \dots, \lambda \overline{I}_n, c \flat (\lambda a, d \flat (\lambda \overline{a}, \overline{a}a) \overline{I}_{m+1} \dots \overline{I}_n) \overline{I}_1 \dots \overline{I}_m$$

## 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

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

Example of proof:

$$\frac{\neg (P \Rightarrow (P \land P))}{P} \alpha_{\neg \Rightarrow}$$

$$\frac{\neg (P \land P)}{\neg (P \land P)} \beta_{\neg \land}$$

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

Deep embedding of proof calculus

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

symbol Rax p : Prf p  $\rightarrow$  Prf ( $\neg$  p)  $\rightarrow$  Prf  $\bot$ ;

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

 $\begin{array}{l} \texttt{symbol} \ \texttt{R} \Longrightarrow \texttt{a} \ \texttt{b} \ : \\ (\texttt{Prf} \ \texttt{a} \to \texttt{Prf}(\neg \ \texttt{b}) \to \texttt{Prf} \ \bot) \to \texttt{Prf} \ (\neg(\texttt{a} \Rightarrow \texttt{b})) \to \texttt{Prf} \ \bot; \end{array}$ 

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

 $\begin{array}{l} \texttt{symbol} \ \mathbb{R} \neg \land \ \texttt{a} \ \texttt{b} \ : \ (\texttt{Prf} \ (\neg \ \texttt{a}) \ \rightarrow \ \texttt{Prf} \ \bot) \\ \rightarrow \ (\texttt{Prf}(\neg \ \texttt{b}) \ \rightarrow \ \texttt{Prf} \ \bot) \ \rightarrow \ \texttt{Prf} \ (\neg(\texttt{a} \ \land \ \texttt{b})) \ \rightarrow \ \texttt{Prf} \ \bot; \end{array}$ 

Deep translation of the example

$$\frac{\neg (P \Rightarrow (P \land P))}{P} \alpha_{\neg \Rightarrow}$$

$$\frac{\neg (P \land P)}{\neg (P \land P)} \beta_{\neg \land}$$

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

opaque	symbo	<mark>l</mark> goal	:	Prf ¬	(p =	⇒ (p /	∧ p))	ightarrow Prf	: ⊥ ≔
R→⇒ p	• (p ∧	p) (λ	$\pi$	, $R \neg \land$	рр	(Rax	p π)	(Rax	p π));

## Making the embedding more shallow

by reducing it to Natural Deduction:

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

Natural Deduction in Lambdapi:

Defining Tableau rules in ND

rule Rax  $\hookrightarrow \lambda$  p h  $\pi$ ,  $\neg$ E p  $\pi$  h; rule R $\neg \land \hookrightarrow \lambda$  p q h1 h2 h3, h1 ( $\neg$ I p ( $\lambda$  h5, h2 ( $\neg$ I q ( $\lambda$  h6,  $\neg$ E (p  $\land$  q) h3 ( $\land$ I p q h5 h6))))); rule R $\rightarrow \hookrightarrow \lambda$  p q h1 h2,  $\neg$ E (p  $\Rightarrow$  q) h2 ( $\Rightarrow$ I p q ( $\lambda$  h3,  $\bot$ E (h1 h3 ( $\neg$ I q ( $\lambda$  h4,  $\neg$ E (p  $\Rightarrow$  q) h2 ( $\Rightarrow$ I p q ( $\lambda_{-}$ , h4))))) q));

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

compute goal; assert  $\vdash$  goal  $\equiv \lambda$  h2,  $\neg E$  (p  $\Rightarrow$  (p  $\land$  p)) h2 ( $\Rightarrow$ I p (p  $\land$  p) ( $\lambda$  h3,  $\perp E$  ( $\neg E$  (p  $\Rightarrow$  (p  $\land$  p)) h2 ( $\Rightarrow$ I p (p  $\land$  p) ( $\lambda$  \_,  $\land$ I p p h3 h3))) (p  $\land$  p)));

### Making it even more shallow

Reduce Natural Deduction thanks to the shallow encoding of FOL

rule  $\Rightarrow$ I  $\hookrightarrow \lambda$  p q  $\pi$ ,  $\pi$ ; rule  $\Rightarrow$ E  $\hookrightarrow \lambda$  p q  $\pi$ ,  $\pi$ ; rule  $\wedge$ I  $\hookrightarrow \lambda$  p q  $\pi$ p  $\pi$ q r  $\pi$ p $\Rightarrow$ q $\Rightarrow$ r,  $\pi$ p $\Rightarrow$ q $\Rightarrow$ r  $\pi$ p  $\pi$ q; rule  $\wedge$ El  $\hookrightarrow \lambda$  p q  $\pi$ p $\wedge$ q,  $\pi$ p $\wedge$ q p ( $\lambda$  x \_, x); rule  $\wedge$ Er  $\hookrightarrow \lambda$  p q  $\pi$ p $\wedge$ q,  $\pi$ p $\wedge$ q q ( $\lambda$  \_ x, x);

```
compute goal;

assert \vdash goal \equiv

\lambda h2, h2 (\lambda h3, h2 (\lambda _ _ \pi, \pi h3 h3) (p \land 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

$\bullet \ \operatorname{sol}(\mathcal{L}) \subseteq \mathcal{L}.$	Superposition into asystics literals.	<ul> <li>Rewriting of positive literals<sup>2</sup>;</li> </ul>	Deletion of deploate latende
• If $\operatorname{ard}(\mathcal{L}) \cap \mathcal{L}^{-} = \emptyset$ , then $\operatorname{ard}(\mathcal{L}) = \emptyset$ .	if a sumplify the state of the		and and y &
We say that a literal $\mathcal{L}$ is solved (with respect to a given eduction function) in a classe $C$ if $\mathcal{L} \in ool(\mathcal{L})$ .	$(SK) \xrightarrow{s:d \vee S} u g v \vee Q \\ \overline{\sigma(u)v \leftarrow l(d = v \vee S \vee K)} \qquad \qquad \begin{array}{c} \sigma(0, \sigma(u), f(v), \sigma(v=i)) \\ is \ digible for parametrizia. \\ u g u v \vee S \vee K \\ \hline \end{array}$	(RP) $\xrightarrow{x \mapsto x - x \to x \to x} \frac{R}{x \mapsto x \to x \to x}$ and if $x \to x \to x \to x \to x \to x \to x \to x$ $x \mapsto x \to $	(DD) start R
We will use two kinds of seriet/stime on deducing new channes. One induced by ordering monotraints and the other by selection functions. We combine these in the notion of eligible ktewals.	$\label{eq:resolution} \text{ modeling, and } u_0 \notin V.$ • Superposition into positive idensity	Classe subscription:	$\frac{1}{R} \frac{1}{R} \frac{1}$
Definition 2.1.2 (Flighle literals) Let $C = L \lor R$ be a classe, let $v$ be a substitution and let $vl$ be a solvetion function.	$\frac{d \sigma = m\mu(u _{1}, i, \sigma _{2}) \times \sigma}{\sigma(1, \sigma _{2}) \times \sigma(1, \sigma _{2}) \times \sigma} = \frac{d \sigma = m\mu(u _{1}, i, \sigma _{2}) \times \sigma}{\sigma(1, \sigma _{2}) \times \sigma(1, \sigma _{2}) \times \sigma}$ (37) $\frac{d \sigma = m\mu(u _{2}, i, \sigma _{2}) \times \sigma}{\sigma(1, \sigma _{2}) \times \sigma(1, \sigma _{2}) \times \sigma} = \frac{d \sigma}{\sigma}$ (37)	(C8) $\frac{C - a(C \lor R)}{C}$ being the constraints of the amplitude dimension of the amplitude dimension of the amplitude dimension.	Destruction equality modulism:
<ul> <li>We say σ(L) is slipilly for resolution if either</li> </ul>	tion, $\sigma(v = v)$ is signific for resolution, and $u _{\mathcal{L}} \notin V$ .	• Apparty remarkant	$(DE) = \frac{\sigma \cdot p \cdot q + \alpha}{\sigma(E)}$ if $\sigma, g \in V, \sigma = mpr(\sigma, g)$
$-\infty(\mathcal{L}) = \emptyset$ and $\sigma(\mathcal{L})$ is >1-maximal in $\sigma(\mathcal{L})$ or $-\infty(\mathcal{L}) \neq \emptyset$ and $\sigma(\mathcal{L})$ is >2-maximal in $\sigma(\sigma(\mathcal{L}) \cap \mathcal{L}^-)$ or	<ul> <li>Simultaneous superposition into separate intends</li> </ul>	$\frac{ \bar{x} \times     x \rightarrow q  x}{1 + \epsilon}$ (23)	Contestual lateral suffinge
$-\operatorname{sel}(\mathcal{L})\neq \emptyset \text{ and } \sigma(\mathcal{L}) \text{ is } >_{L^*} \text{maximal in } \sigma(\operatorname{sel}(\mathcal{L})\cap \mathcal{L}^n).$	$il \sigma = mp(a s, s , \sigma s) \not\leq$ $s = r f(s) = g $ $\sigma(t), \sigma(s) \not\leq \sigma(s), \sigma(s = t)$	<ul> <li>Paritive simplify reflect<sup>1</sup>.</li> </ul>	$(CLC) = \frac{\sigma(C \vee R \vee sid)}{C \vee sid}$ where $\overline{sid}$ is the segation of
<ul> <li>σ(L) is slightly for parametricalities if L is positive, sel(C) = 0 and σ(L) is strictly &gt;<sub>L</sub>-maximal in σ(C).</li> </ul>	(88N) $\overline{\sigma(S \vee (w \neq v \vee \bar{K})   u_{\phi} = 1)}$ is eligible for parametrization. tion, $\sigma(w \neq v)$ is eligible for modulo of $V$	$(PS) = \frac{s + t - a[p + \sigma(s)]) s a[p + \sigma(t)] \vee R}{s}$	$\sigma(C \lor R) = C \lor vol and \sigma$ is a substitution. This rule is also known as subsamption resolution or classed simplification.
The calculation represented in the form of inference onles. For concerning, we	This inference rule is an alternative to $(8N)$ that performs better in practice.	s of R • Negative simplify reflect	• Condensing: $\frac{1}{2} ( \vee I_2 \vee R)$ If $\sigma(I_1) = \sigma(I_2)$ and $\sigma(I_2 \vee R)$
with a single income space of a prevention of a proving measure rate, which a single income space of the set of all channes. For contracting informate rules, written with a double	<ul> <li>Simultaneous superposition into positive ditends</li> <li>If a = second of which d</li> </ul>	$(88) = \frac{s \# s - s(s) \# \sigma(l) \vee R}{R}$	$r(i, i, k) = r(i_1 \vee R)$ submatrix $h \vee h \vee R$
Easy, the result choose are substituted for the choose in the precondition. In the following $n, r, s$ and $t$ are trens, $\sigma$ is a substitution and $R, S$ and $T$ are (partial) choose, $p$ is a position in a term and $\lambda$ is the weaply or topposition.	$(83P) \xrightarrow{a + i + V \cdot S - a + i + v + R}_{s + (i + v + i)} \xrightarrow{a + i + V \cdot S - a + i + v + R}_{s + (i + v + i)} b + i + (i + v + i) + i + (i + v + i) + i + (i + i) + (i +$	Zantology deletore	<ul> <li>Introduce defaulton<sup>*</sup></li> <li>If R and R do not share any RVS</li> <li>Standalle, A C D has not here</li> </ul>
assumed to not share any common variables.	This inference rule is an alternative to (SP) that performs better in proc-	(TD) C is a taskings <sup>4</sup>	(12) $\frac{1}{d \vee R} - d \vee S$ and it is previous definition and $R$ does not contain any result from $D$
Definition 2.1.3 (The inference system SP) Let > be a total simplification ordering (saturated to orderings > <sub>L</sub> and > <sub>U</sub> or linearly and chemath lat of the a solution function, and lat $D$ he a set of	tere. • Equality factoring:	<sup>4</sup> A strenger variant of (3D) is proven to maintain completeness for Unit and Here prob- lems and is generally believed to maintain completeness for the general case as well (300-9). However, the word of consolitouring for the system of new series to be rather involved, as it re- tresses as the series of the second series of the system of the system of the system of the system of the second seco	<ul> <li>Apply definition</li> </ul>
fresh proportional constants. The inference system SP consists of the following inference rules: • Equality Resolutions	$ (\overline{b}T)  \frac{\sin d  \forall  u  \mathrm{in}  v  \forall  \overline{k}}{\sigma(z)  \sin v  \forall  \overline{k}}  \qquad \qquad$	prime a very different clears ordering than the area includent [ROR], and we are not a sense of any mixing proof in the literature. The means rule allows receiving of maximal terms of maximal literals under solution (commutances $H u_{0}^{i} = \sigma(a, \sigma(a) > \sigma))$ and if we were some some $H$ .	$(AD)  \begin{array}{c} w(d \lor R)  R \lor S \\ w(d \lor R)  R \lor S \\ \hline w(d \lor R)  -d \lor S \\ \end{array} \qquad \qquad \begin{array}{c} \text{if } \sigma \text{ in a variable remaining, } R \\ \text{and } X \text{ ds not share any variables } work \\ with any sequence of them D \\ \hline we determine any sequence dense of the set \\ \end{array}$
$\frac{w d v \vee R}{w (\Delta)} = \frac{d v - m p (u, v) \text{ and } p(v \phi)}{w (\Delta)}$ (ER) $\frac{d v d v - m p (u, v) \text{ and } p(v \phi)}{w (\Delta)}$	<ul> <li>Rewriting of negative literals</li> <li>new next with VR</li> </ul>	(HP) (HP) (H) (H) (H) (H) (H) (H) (H) (H) (H) (H	*This rule runs only be implemented approximately, as the problem of ecosystem producting for he only modification for equational angle. Concrete version of Koy to dotest translaging by detailing the grant complete threads hereby a base are of the problem frame,
	$a = a_{\mu} = a_{\mu} a_{\mu} a_{\mu} a_{\mu}$ $a = a_{\mu} = a_{\mu} a_{\mu} a_{\mu} a_{\mu}$	<sup>5</sup> In practice, this rule is only applied if $\sigma(z)$ and $\sigma(z)$ are $>$ incomparable $-$ is all other mass this rule is submanial by (EO) and the distribution of resoluted Brends (DB).	*Else rule is always enhancionly applied to any channe, having a split-off channes and one final link channe of all arguiter propositions.
*		10	

## 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 U ArchSAT	95%	85%

### Percentage of completely reconstructed proofs

Prover	% E TSTP	% VAMPIRE TSTP
ZenonModulo	45%	54%
ArchSAT	56%	74%
ZenonModulo U 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

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

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



### Reducing commuting cuts

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

 $\Rightarrow$  we need to stay at the ND level and add rules to reduce commuting cuts:

rule  $\land$ El \$c \$d ( $\lor$ E \$a \$b \$paorb (\$c  $\land$  \$d) \$pac \$pbc)  $\hookrightarrow \lor$ E \$a \$b \$paorb \$c ( $\lambda$  pa,  $\land$ El \$c \$d (\$pac pa)) ( $\lambda$  pb,  $\land$ El \$c \$d (\$pbc pb)); Example proof with Skolem symbol

```
symbol goal
(ax_tran : Prf (\forall (\lambda X1, \forall (\lambda X2, \forall (\lambda X3,
  (p X1 X2) \Rightarrow ((p X2 X3) \Rightarrow (p X1 X3)))))))
// skolemized version of
// (ax_step : Prf (\forall (\lambda X, \exists (\lambda Y, (p X (s Y)))))
(ax_step : Prf (\forall (\lambda X, (p X (s (f X))))))
(ax_congr : Prf (\forall (\lambda X1, \forall (\lambda X2,
  (p X1 X2) \Rightarrow (p (s X1) (s X2))))))
(ax_goal : Prf (\neg (\exists (\lambda X, ((p a (s (s X))))))))
: Prf \bot
:= ax_goal (\existsI (\lambda 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 ( $\forall$  ( $\lambda$  X1,  $\forall$  ( $\lambda$  X2,  $\forall$  ( $\lambda$  X3, (p X1 X2)  $\Rightarrow$  ((p X2 X3)  $\Rightarrow$  (p X1 X3)))))) (ax\_step : Prf ( $\forall$  ( $\lambda$  X,  $\exists$  ( $\lambda$  Y, (p X (s Y))))) (ax\_congr : Prf ( $\forall$  ( $\lambda$  X1,  $\forall$  ( $\lambda$  X2, (p X1 X2)  $\Rightarrow$  (p (s X1) (s X2))))) (ax\_goal : Prf ( $\neg$  ( $\exists$  ( $\lambda$  X4, ((p a (s (s X4)))))))) : Prf  $\bot$  $\coloneqq$  ax\_goal ( $\lambda$  r h,  $\exists$ E ( $\lambda$  z0, p z (s z0)) (ax\_step a) r ( $\lambda$  z a1,  $\exists$ E ( $\lambda$  z0, p z (s z0)) (ax\_step z) r ( $\lambda$  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

