Embedding logics in Dedukti

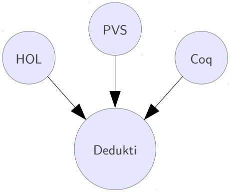
2014 edition

Ali Assaf

2nd KWARC-Deducteam workshop
May 26, 2014

Outline

Universal proof checker



Source: HOL, Coq, ...

- Rich type systems
- Reconstruction, proof search, ...

Target: Dedukti

- $\lambda\Pi$-calculus modulo
- Proof checking (no reconstruction, no proof search, ...)

Outline

Pure type systems

- A general class of type systems for the λ-calculus
  - $\lambda\Pi$-calculus, System F, Calculus of constructions, ...
- Basis for several proof systems
  - HOL, Coq, ...

Pure type systems

Pure type systems in the $\lambda\Pi$-calculus modulo

- Translation of *functional* PTS
- Correctness

Pure type systems

Pure type systems in the $\lambda\Pi$-calculus modulo

- Translation of *functional* PTS
- Correctness

What's new?

- Conservativity
- Normalization

Translation

- As a *type*:

$$\begin{aligned} [\![s]\!] &= s \\ [\![\Pi x : A.B]\!] &= \Pi x : [\![A]\!] \,.\, [\![B]\!] \end{aligned}$$

- As a *term*:

$$\begin{aligned} [s] &= \dot{s} \\ [\Pi x : A.B] &= \dot{\pi}\, [A]\, (\lambda x : [\![A]\!] \,.\, [B]) \end{aligned}$$

Decoding function

- $[A]$ is a code for $[\![A]\!]$
- Decoding function ε

$$\varepsilon\,([A]) \quad \equiv \quad [\![A]\!]$$

- With rewrite rules:

$$\begin{aligned} \varepsilon\,(\dot{s}) &\quad\longrightarrow\quad s \\ \varepsilon\,(\dot{\pi}\,A\,B) &\quad\longrightarrow\quad \Pi x : \varepsilon\,(A).\,\varepsilon\,(B\,x) \end{aligned}$$

Example

Example

Polymorphic identity in CC

$$\lambda A : \text{type}.\lambda x : A.x \quad : \quad \Pi A : \text{type}.A \to A$$

Translation

$$[\lambda A : \text{type}.\lambda x : A.x] \quad : \quad [\![\Pi A : \text{type}.A \to A]\!]$$

Example

Example

Polymorphic identity in CC

$$\lambda A : \mathsf{type}.\lambda x : A.x \quad : \quad \Pi A : \mathsf{type}.A \to A$$

Translation

$$\lambda A : \mathsf{type}.\lambda x : \varepsilon\,(A).x \quad : \quad \Pi A : \mathsf{type}.\,\varepsilon\,(A) \to \varepsilon\,(A)$$

Correctness

- Suppose M is well typed in the PTS
- Is $[M]$ well typed in $\lambda\Pi \simeq$?

Correctness

- Suppose M is well typed in the PTS
- Is $[M]$ well typed in $\lambda\Pi \simeq$?

Theorem (Cousineau & Dowek 2007)

If $\Gamma \vdash_{PTS} M : A$ *then* $[\![\Gamma]\!] \vdash_{\lambda\Pi\simeq} [M] : [\![A]\!]$.

Proof.

By induction on the derivation of $\Gamma \vdash_{PTS} M : A$. □

Conservativity

- Suppose $[\![A]\!]$ is provable in $\lambda\Pi \simeq$
- Is A provable in the PTS?

Conservativity

- Suppose $[\![A]\!]$ is provable in $\lambda\Pi\simeq$
- Is A provable in the PTS?

- Suppose $[\![\Gamma]\!] \vdash_{\lambda\Pi\simeq} M : [\![A]\!]$
- Define an erasure $|M|$ such that $|[M]| = M$

$$
\begin{array}{rcl}
|\dot{s}| & = & s \\
|\dot{\pi}\, A\, B| & = & \Pi x : |A| \,.\, |B\, x| \\
\|\varepsilon\,(A)\| & = & |A|
\end{array}
$$

- Does $\Gamma \vdash_{PTS} |M| : A$ hold?

Conservativity

Example

$$
\begin{array}{lllll}
\vdash_{\lambda\Pi\simeq} & (\lambda A : \mathsf{type}.\lambda x : \varepsilon\,(A).x)\,\mathsf{bool} & : & \varepsilon\,(\mathsf{bool}) \to \varepsilon\,(\mathsf{bool}) \\
\nvdash_{STLC} & (\lambda A : \mathsf{type}.\lambda x : A.x)\,\mathsf{bool} & : & \mathsf{bool} \to \mathsf{bool}
\end{array}
$$

Conservativity

Example

$$\begin{array}{llll} \vdash_{\lambda\Pi\simeq} & (\lambda A : \mathrm{type}.\lambda x : \varepsilon(A).x)\,\mathrm{bool} & : & \varepsilon(\mathrm{bool}) \to \varepsilon(\mathrm{bool}) \\ \not\vdash_{STLC} & (\lambda A : \mathrm{type}.\lambda x : A.x)\,\mathrm{bool} & : & \mathrm{bool} \to \mathrm{bool} \end{array}$$

- The term $(\lambda A : \mathrm{type}.\lambda x : A.x)\,\mathrm{bool}$ is not well typed in the PTS
  - It uses "illegal" abstractions (here polymorphism)
- But it reduces to $\lambda x : \mathrm{bool}.x$ which is well-typed!

Conservativity

Example

$$\begin{array}{llcl} \vdash_{\lambda\Pi\simeq} & (\lambda A : \text{type}.\lambda x : \varepsilon(A).x)\,\text{bool} & : & \varepsilon(\text{bool}) \to \varepsilon(\text{bool}) \\ \nvdash_{STLC} & (\lambda A : \text{type}.\lambda x : A.x)\,\text{bool} & : & \text{bool} \to \text{bool} \end{array}$$

- The term $(\lambda A : \text{type}.\lambda x : A.x)\,\text{bool}$ is not well typed in the PTS
  - It uses "illegal" abstractions (here polymorphism)
- But it reduces to $\lambda x : \text{bool}.x$ which is well-typed!
- Key insight: this is always the case!

Conservativity

Theorem (Assaf 2013)

If $[\![\Gamma]\!] \vdash_{\lambda\Pi\simeq} M : [\![A]\!]$ *then* $\Gamma \vdash_{PTS} M' : A$.

Proof.

1. Define an erasure $|M|$ such that $|[M]| = M$.
2. Prove using induction that $\Gamma \vdash_{PTS^*} |M| : A$.
3. Prove using reducibility that $|M| \longrightarrow^* M'$ such that $\Gamma \vdash_{PTS} M' : A$.

$\square$

Normalization

Theorem (Cousineau & Dowek 2007)

If $\lambda\Pi\simeq$ is SN, then PTS is SN.

Proof.

Translation $[M]$ preserves β-reduction. $\square$

Normalization

Theorem (Cousineau & Dowek 2007)

If $\lambda\Pi \simeq$ is SN, then PTS is SN.

Proof.

Translation $[M]$ preserves β-reduction. $\square$

Theorem (Dowek 2014)

For some PTS in SN, $\lambda\Pi \simeq$ is SN.

Proof.

Using reducibility candidates to define a super-consistency criterion for $\lambda\Pi \simeq$. $\square$

Outline

HOL

- A family of theorem provers
  - HOL Light, HOL4, ProofPower, ...
- Based on higher order logic
- Large formalizations
  - Flyspeck project (Kepler's conjecture)

Holide

Holide: HOL in Dedukti

- Developed by Ali Assaf
- Avalailable at: https://gforge.inria.fr/projects/holide/

Last time:

- Proof retrieval
- Proof sharing
- Standard library benchmark

What's new?

- Term sharing
- Improved benchmark results
- Multiple target languages

Proof retrieval

LCF architecture = no proof trace

OpenTheory project [Hurd 2011]

- A standard for exporting and exchanging HOL proofs
- A well-defined standard library

The OpenTheory article format

Instructions that are executed to reconstruct the theorems.

Example

Reflexivity on a variable x of type A:

$$\frac{}{\vdash x^A = x^A}\ \mathsf{refl}$$

OpenTheory article file

```
"A"
varType
"x"
var
varTerm
refl
```

OCaml execution

```
let A = varType("A") in
let x = varTerm(var("x", A)) in
refl x
```

Proof size

LCF architecture = huge proof trees

Example

A proof of $t + t = u + u$:

```
let p = ... (* A very large proof of t = u *) in
appThm (appThm (refl f) p p)
```

$$\dfrac{\dfrac{\dfrac{}{(+) = (+)}\,\text{refl} \qquad \dfrac{\pi}{t = u}}{(+)\,t = (+)\,u}\,\text{appThm} \qquad \dfrac{\pi}{t = u}}{(+)\,t\,t = (+)\,u\,u}\,\text{appThm}$$

Need proof sharing!

Proof sharing

- Share common subproofs

```
step1 : proof (t = u) := ...
step2 : proof ((+) = (+)) := refl q.
step3 : proof ((+) t = (+) u) := appThm step2 step1.
step4 : proof ((+) t t = (+) u u) := appThm step3 step1.
```

- Need lambda lifting

Term sharing

- No implicit arguments in Dedukti
  - Terms are annotated by their types
  - Proofs are annotated by their terms *and* types

Example

```
step1 : proof (f x = g y) := appThm p q.
```

Term sharing

- No implicit arguments in Dedukti
  - Terms are annotated by their types
  - Proofs are annotated by their terms *and* types

Example

```
step1 : proof (f x (= a b) g x) := appThm a b f g x y p q.
```

- Size is at least $O(n^2)$!
- Need term sharing

Term sharing

- Lambda-lift terms and types to top-level definitions
- During translation, keep *both* the name and the body
  - Need name for referencing
  - Need body for analysis

Example

In appThm p q, we need to know that the statement of p is $f = g$.

- Memoization

Results

Package	Size (in kB)		Verification time (in s)	Percentage verified
	OpenTheory	Dedukti		
unit	26	309	0	100%
function	89	1,301	3	100%
pair	195	4,943	15	100%
bool	305	4,258	7	100%
sum	502	20,988	99	100%
option	520	23,815	77	100%
relation	971	42,572	350	100%
list	1,377	68,031	182	100%
real	1,754	68,508	1	1%
natural	1,952	130,111	496	100%
set	2,329	90,819	431	100%
Total	10,020	455,656	1,661	85%

Results

Package	Size (in kB)		Verification time (in s)	Percentage verified
	OpenTheory	Dedukti		
unit	26	116	0.04	100%
function	89	595	0.2	100%
pair	195	1,469	0.73	100%
bool	305	1,824	0.53	100%
sum	502	3,754	1.23	100%
option	520	4,027	1.29	100%
relation	971	8,113	2.99	100%
list	1,377	10,128	3.39	100%
real	1,754	116,758	3.31	100%
natural	1,952	12,542	3.48	100%
set	2,329	18,055	6.17	100%
Total	10,020	72,303	23.36	100%

Target languages

LF modulo embedding

- Dedukti
- Coq?

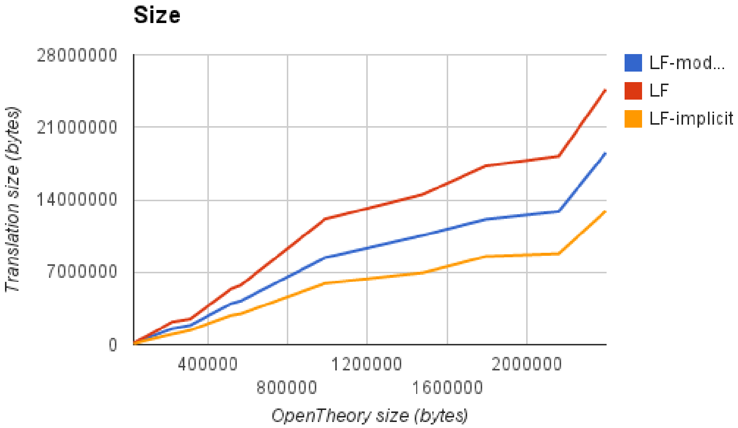
LF embedding

- Dedukti
- Twelf
- Coq

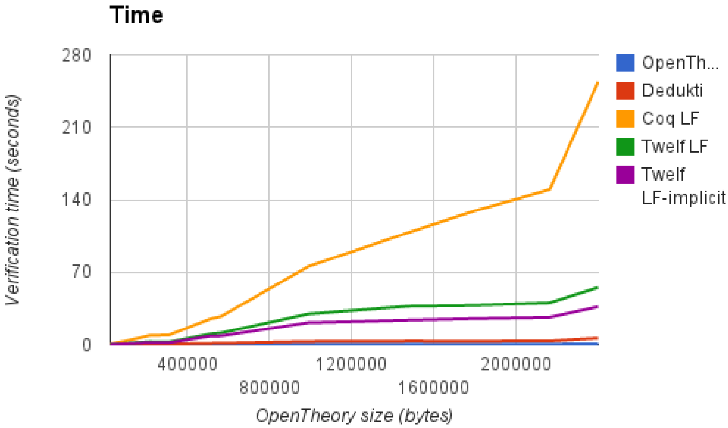
LF embedding with implicit arguments

- Twelf
- Coq?

Target languages



Target languages



Future work

- Use smarter sharing
  - Avoid unnecessary lambda lifting
  - Use caching methods (Kaliszyk & Krauss 2013)
- Translate larger formalizations
  - Flyspeck? (Kaliszyk & Krauss 2013)

Outline

Coq

- Proof system based on the calculus of inductive constructions
  - Infinite hierarchy of universes type_i
  - Subtyping $\mathsf{type}_i \subseteq \mathsf{type}_{i+1}$
  - Floating universes
  - Inductive types
  - Co-inductive types
  - Modules
  - ...
- Large formalizations
  - 4 color theorem, Feit–Thompson theorem, ...

Coqine

Coqine: Coq in Dedukti

- Developped by Ali Assaf and Guillaume Burel
- Available at: https://gforge.inria.fr/projects/coqine/

Last time:

- Proof retrieval
- Universe hierarchy
- Inductive types
- Modules

What's new?

- Plugin architecture
- Universe subtyping

New architecture

- A Coq plugin that is loaded at runtime

Example

```
Require Coqine Logic Arith.

Dedukti Export Library Logic Arith.
```

- Pros
  - Directly access the contents of `.vo` files
  - No need to reimplement features
- Cons
  - Still rely on the Coq implementation

Universe subtyping in Coq

- Infinite hierarchy

$$\mathsf{Prop}, \mathsf{Type}_0 : \mathsf{Type}_1 : \mathsf{Type}_2 : \ldots$$

- Cumulative

$$\mathsf{Prop} \subseteq \mathsf{Type}_0 \subseteq \mathsf{Type}_1 \subseteq \mathsf{Type}_2 : \ldots$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash A : \mathsf{Type}_{i+1}}$$

Universe subtyping in Dedukti

- Type uniqueness

$$\mathsf{type}_i : \mathsf{Type}_{i+1}$$

- Explicit coercions

$$\uparrow_i : \mathsf{Type}_i \to \mathsf{Type}_{i+1}$$

Multiple representations

- Different typing derivations yield different terms

$$\dfrac{\dfrac{A : \mathrm{Type}_0 \qquad x : A \vdash B : \mathrm{Type}_0}{\Pi x : A.B : \mathrm{Type}_0}}{\Pi x : A.B : \mathrm{Type}_1} \qquad {\color{blue}\uparrow_0} \, \dot{\pi}_0 \; a \; b$$

$$\dfrac{\dfrac{A : \mathrm{Type}_0}{A : \mathrm{Type}_1} \qquad \dfrac{x : A \vdash B : \mathrm{Type}_0}{x : A \vdash B : \mathrm{Type}_1}}{\Pi x : A.B : \mathrm{Type}_1} \qquad \dot{\pi}_1 \; ({\color{blue}\uparrow_0} \, a) \; ({\color{blue}\uparrow_0} \, b)$$

Problem with dependent types

In the context

$$
\begin{aligned}
a, b &: \mathrm{Type}_0 \\
p, q &: \mathrm{Type}_1 \to \mathrm{Type}_1 \\
f &: \Pi a, b : \mathrm{Type}_1 .\, p\,(\Pi x : a.b) \\
g &: \Pi c : \mathrm{Type}_0 .\, p\, c \to q\, c
\end{aligned}
$$

we have

$$
g\,(\Pi x : a.b)\,(f\; a\; b) \quad : \quad q\,(\Pi x : a.b)
$$

Problem with dependent types

In the context

$$\begin{array}{rcl} a, b & : & \mathsf{Type}_0 \\ p, q & : & \mathsf{Type}_1 \to \mathsf{Type}_1 \\ f & : & \Pi a, b : \mathsf{Type}_1 .\, \varepsilon_1\,(p\,(\dot{\pi}_1\; a\; b)) \\ g & : & \Pi c : \mathsf{Type}_0 .\, \varepsilon_1\,(p\,(\uparrow_0 c) \to q\,(\uparrow_0 c)) \end{array}$$

we have

$$g\,(\dot{\pi}_0\; a\; b)\,(f\,(\uparrow_0 a)\,(\uparrow_0 b)) \quad : \quad \varepsilon_1\,(q\,(\uparrow_0 (\dot{\pi}_0\; a\; b)))$$

Problem with dependent types

In the context

$$\begin{array}{rcl} a, b & : & \mathsf{Type}_0 \\ p, q & : & \mathsf{Type}_1 \to \mathsf{Type}_1 \\ f & : & \Pi a, b : \mathsf{Type}_1 .\, \varepsilon_1\,(p\,(\dot{\pi}_1\; a\, b)) \\ g & : & \Pi c : \mathsf{Type}_0 .\, \varepsilon_1\,(p\,({\uparrow_0}\; c) \to q\,({\uparrow_0}\; c)) \end{array}$$

we have

$$\begin{array}{rcl} g\,(\dot{\pi}_0\; a\, b)\,(f\,({\uparrow_0}\; a)\,({\uparrow_0}\; b)) & \not: & \varepsilon_1\,(q\,({\uparrow_0}\,(\dot{\pi}_0\; a\, b))) \quad \times \\ f\,({\uparrow_0}\; a)\,({\uparrow_0}\; b) & : & \varepsilon_1\,(p\,(\dot{\pi}_1\,({\uparrow_0}\; a)\,({\uparrow_0}\; b))) \end{array}$$

Solution: Reflecting equalities

- Add equation

$$\uparrow_i (\dot{\pi}_i \; a \; b) \quad \equiv \quad \dot{\pi}_{i+1} \, (\uparrow_i a) \, (\uparrow_i b)$$

Solution: Reflecting equalities

- Add equation

$$\uparrow_i (\dot{\pi}_i \, a \, b) \quad \equiv \quad \dot{\pi}_{i+1} (\uparrow_i a) (\uparrow_i b)$$

- How does this help?

$$\begin{array}{rcl} a, b & : & \mathrm{Type}_0 \\ p, q & : & \mathrm{Type}_1 \to \mathrm{Type}_1 \\ f & : & \Pi a, b : \mathrm{Type}_1 . \, \varepsilon_1 (p (\dot{\pi}_1 \, a \, b)) \\ g & : & \Pi c : \mathrm{Type}_0 . \, \varepsilon_1 (p (\uparrow_0 c) \to q (\uparrow_0 c)) \end{array}$$

$$\begin{array}{rcl} g (\dot{\pi}_0 \, a \, b) (f (\uparrow_0 a) (\uparrow_0 b)) & : & \varepsilon_1 (q (\uparrow_0 (\dot{\pi}_0 \, a \, b))) \\ f (\uparrow_0 a) (\uparrow_0 b) & : & \varepsilon_1 (p (\dot{\pi}_1 (\uparrow_0 a) (\uparrow_0 b))) \end{array}$$

Solution: Reflecting equalities

- Add equation

$$\uparrow_i (\dot{\pi}_i\, a\, b) \quad \equiv \quad \dot{\pi}_{i+1} (\uparrow_i a) (\uparrow_i b)$$

- How does this help?

$$\begin{array}{rcl} a, b & : & \mathrm{Type}_0 \\ p, q & : & \mathrm{Type}_1 \to \mathrm{Type}_1 \\ f & : & \Pi a, b : \mathrm{Type}_1 .\, \varepsilon_1 (p\, (\dot{\pi}_1\, a\, b)) \\ g & : & \Pi c : \mathrm{Type}_0 .\, \varepsilon_1 (p\, (\uparrow_0 c) \to q\, (\uparrow_0 c)) \end{array}$$

$$\begin{array}{rcl} g\, (\dot{\pi}_0\, a\, b)\, (f\, (\uparrow_0 a)\, (\uparrow_0 b)) & : & \varepsilon_1 (q\, (\uparrow_0 (\dot{\pi}_0\, a\, b))) \quad \checkmark \\ f\, (\uparrow_0 a)\, (\uparrow_0 b) & : & \varepsilon_1 (p\, (\uparrow_0 (\dot{\pi}_0\, a\, b))) \end{array}$$

- Exercise: Do the same for Prop.

With rewrite rules

- The main challenge is turning the equations into rewrite rules
- Distributing $\uparrow_i$ breaks confluence

$$\begin{aligned} \uparrow_i (\dot{\pi}_i\, A\, B) &\longrightarrow \dot{\pi}_{i+1}\, (\uparrow_i A)\, (\uparrow_i B) \\ \forall_{i+1} x : (\uparrow_i A)\, .B &\longrightarrow \forall_i x : A.B \end{aligned}$$

With rewrite rules

- The main challenge is turning the equations into rewrite rules
- Distributing $\uparrow_i$ breaks confluence

$$\begin{aligned} \uparrow_i (\dot{\pi}_i\, A\, B) &\longrightarrow \dot{\pi}_{i+1}\, (\uparrow_i A)\, (\uparrow_i B) \\ \forall_{i+1} x : (\uparrow_i A)\, .B &\longrightarrow \forall_i x : A.B \end{aligned}$$

- Need to raise $\uparrow_i$ to the top

$$\begin{aligned} \uparrow_i (\dot{\pi}_i\, A\, B) &\longleftarrow \dot{\pi}_{i+1}\, (\uparrow_i A)\, (\uparrow_i B) \\ \forall_i x : A.B &\longleftarrow \forall_{i+1} x : (\uparrow_i A)\, .B \\ \uparrow^{(i)}_{\mathrm{Prop}} (\forall_i x : A.B) &\longleftarrow \dot{\pi}_i\, A \left(\uparrow^{(i)}_{\mathrm{Prop}} B\right) \end{aligned}$$

- Corresponds to minimal typing!

Properties

- Terms *must* have a unique representation

Theorem (Canonicity)

If $M \equiv M'$ *then* $[M] \equiv [M']$.

- Essential for correctness

Theorem (Correctness)

If $\Gamma \vdash_{Coq} M : A$ *then* $[\![\Gamma]\!] \vdash_{\lambda\Pi\simeq} [M] : [\![A]\!]$.

Future work

- Universe polymorphism
- Better translation
  - of inductive types
  - of modules
- Translate the standard library

Outline

Focalize

- An environment to develop certified programs
  - A functional programming language with object-oriented features
- Different from usual setting
  - Objects and inheritance
  - Non-termination
- http://focalize.inria.fr/

Focalize

```
class OrderingNat = {
  rep = nat;
  methods :
    abstract leq : rep -> rep -> bool;
    geq : rep -> rep -> bool;
    geq this n = leq n this;
    lt : rep -> rep -> bool;
    lt this n = (leq this n) && ~~(geq this n);
    gt : rep -> rep -> bool;
    gt this n = lt n this;
    abstract leq_refl : forall n : rep, leq n n;
    abstract leq_asym : forall n, m : rep, ...;
    abstract leq_trans : forall n, m, p : rep, ...;
}
```

Backends

- Built with interoperability in mind
- Proof backends:
    - Zenon
    - Coq

Backends

- Built with interoperability in mind
- Proof backends:
  - Zenon
  - Coq
  - Dedukti

Focalide

Focalide: Focalize in Dedukti

- Developed by Raphaël Cauderlier
- Available in branch `focalide` of Focalize: http://focalize.inria.fr/download/

Current features:

- Objects
- Inheritence
- Specifications

Work in progress:

- Proofs

Results

File	Size (kb)		Factor	Typing
	Original	Translation		
basics	23	4.3	0.19	OK
sets	6.3	56	8.9	OK
products	14	250	18	KO
lattices	22	333	15	OK
orders	7.6	625	83	OK
strict_orders	7.1	120	17	OK
orders_and_lattices	19	740	39	OK
wellfounded	4.5	176	40	KO
sums	20	589	30	KO
quotients	8.4	214	25	OK
fix	24	KO	KO	KO
Total	132	3107	24	71%

Outline

Conclusion

- Theoretical and practical challenges
- Solving these challenges leads to better specifications
- One step closer towards interoperability

Other/future work:

- Proof assistants: PVS, Agda
- Theorem provers: Zenon (modulo), iProver (modulo)