

ENSIIE/Samovar

Exprimer ses théories en Dedukti, le vérificateur de preuves universel

JFLA 2017

Guillaume Burel

vendredi 6 et samedi 7 janvier 2017

Dedukti



Vérificateur de preuve pour le $\lambda\Pi$ -calcul modulo théorie

<http://dedukti.gforge.inria.fr/>

<http://dedukti.gforge.inria.fr/jfla2017/>

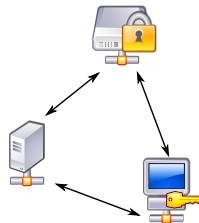
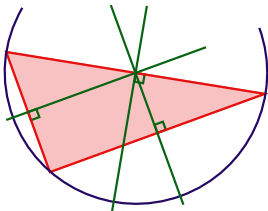
Dedukti : a Logical Framework based on the $\lambda\Pi$ -Calculus Modulo Theory

Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant et Ronan Saillard.

<http://www.lsv.ens-cachan.fr/~dowek/Publi/expressing.pdf>

Sommaire

- Introduction
- Le $\lambda\Pi$ -calcul modulo théorie
- Logiques des prédicats
- Langages de programmation
- Logique d'ordre supérieur
- Calcul des constructions inductives



Les démonstrations sont recherchées dans des **théories** :

- ▶ géométrie, arithmétique
- ▶ structures de données (listes chaînées, tableaux, ...)
- ▶ propriétés du chiffrement
- ▶ ...

Un cadre unique

Un seul cadre logique pour toutes les théories

- ▶ partage des connecteurs (\wedge, \forall, \dots)
- ▶ partage des notions de preuve, de modèle
 - théorème généraux comme complétude prouvés une fois pour toute
- ▶ Notions d'inclusion entre théories (exemple $ZF \subset ZFC$)
- ▶ interopérabilité \mathcal{T} dans \mathcal{L} \mathcal{T}' dans \mathcal{L}' $A \in \mathcal{L} \cap \mathcal{L}'$

$$\begin{array}{l} \mathcal{T} \vdash A \Rightarrow B \quad \mathcal{T}' \vdash A \\ \mathcal{T} \cup \mathcal{T}' \vdash B \end{array}$$

Quel cadre utiliser ?

Logique du premier ordre ?

- ▶ Pour définir une théorie :
 - symboles de fonction et de prédicat
 - axiomes

Mais pas unique en pratique :

- ▶ théorie simple des types (HOL)
- ▶ calcul des constructions inductives

Inconvénient

Pas de possibilité de passer d'un prouveur à un autre

- ▶ CompCert en Isabelle/HOL ?
- ▶ seL4 en Coq ?

Combinaison avec des outils de preuve automatique ?

- ▶ éviter la reconstruction de preuve

Pourquoi ?

Manque en logique du premier ordre :

1. Possibilité de définir des lieux
 - exemple $x \mapsto t$
2. Pas de correspondance preuve/programme — formule/type
3. Pas de prise en compte du calcul
 - simulé par des étapes de déduction
4. Pas de notion universelle de coupure
 - doit être définie pour chaque théorie
5. Cadres différents pour logiques classique et intuitionniste

Résoudre problèmes 1 et 2

Logical Framework LF a.k.a. λP a.k.a. $\lambda\Pi$ [Harper Honsell Plotkin 1993]

Extension

- ▶ du λ -calcul simplement typé avec des types dépendants
- ▶ de la logique des prédicats (minimale) avec des termes de preuve

Résoudre problèmes 3 et 4

Dédution modulo théorie [Dowek Hardin Kirchner 2003]

Règles d'inférence appliquées modulo une congruence \equiv

Système de réécriture sur les termes **et les propositions**

Exemple : $X \in \mathcal{P}(Y) \rightarrow \forall Z. Z \in X \Rightarrow Z \in Y$

$$\frac{\widehat{\vdash} \frac{}{Z \in a \vdash Z \in a}}{\vdash \Rightarrow} \quad \frac{\vdash \frac{}{Z \in a \Rightarrow Z \in a}}{\vdash \forall} \quad a \in \mathcal{P}(a) \equiv \forall Z. Z \in a \Rightarrow Z \in a$$

Résoudre 1, 2, 3 et 4

Combiner LF avec la déduction modulo théorie :

- ▶ $\lambda\Pi$ -calcul modulo théorie

Extension de la règle de conversion

- ▶ sans modulo : uniquement β
- ▶ avec : β + système de réécriture

Problème 5

Un connecteur \triangleright pour passer d'atome à formule

Deux versions des connecteurs : une constructive, une classique

Dedukti

$\lambda\Pi$ -calcul modulo théorie pas seulement expressif en théorie

Implémentation efficace :

- Dedukti

vérificateur de preuve du $\lambda\Pi$ -calcul modulo théorie

M. Boespflug 2008 \rightsquigarrow ... \rightsquigarrow Ronan Saillard 2015

Démo

Outils autour de Dedukti

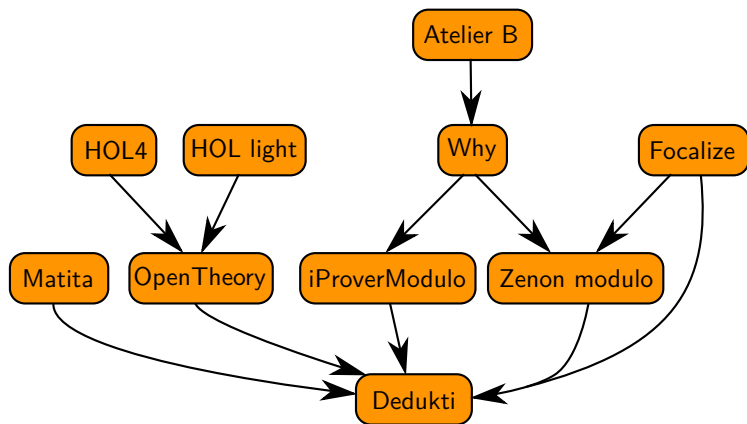
Traducteurs de preuves vers Dedukti

- ▶ HOL
- ▶ CIC

Backend vers Dedukti

- ▶ prouveurs automatiques iProverModulo, Zenon modulo
- ▶ focalize

Panorama actuel



En cours : Coq, PVS, VeriT, ...

Plongement profond vs. plongement superficiel

Superficiel : réutilisation des fonctionnalités du langage cible :

- ▶ contextes
- ▶ connecteurs et leur
- ▶ calcul
- ▶ ...

Intérêt des plongements superficiels

- ▶ traductions plus légères
- ▶ plus efficace
- ▶ rend l'interopérabilité plus simple

Outline

- Introduction
- Le $\lambda\Pi$ -calcul modulo théorie
- Logiques des prédicats
- Langages de programmation
- Logique d'ordre supérieur
- Calcul des constructions inductives

Le $\lambda\Pi$ -calcul
$$t, u ::= \textit{Type} \mid \textit{Kind} \mid x \mid \lambda x : t. u \mid t \ u \mid \Pi x : t. u$$

niveaux : *Kind*

<i>Type</i>	$\Pi x : \textit{nat}. \textit{Type}$	types des (familles de) types
<i>nat</i>	$\Pi x : \textit{nat}. \textit{vect } x$	(familles) de types
0	<i>S</i> <i>nil</i> <i>cons</i>	objets

Types dépendants

$x : \text{nat} \rightarrow \text{nat}$ ✓

$x : \text{nat} \rightarrow \text{vect } x$ ✓

$x : \text{nat} \rightarrow \text{Type}$ ✓

$A : \text{Type} \rightarrow \text{list } A$ ✗

$A : \text{Type} \rightarrow \text{Type}$ ✗

$\lambda\Pi$ -calcul modulo théorie

En plus des déclarations de variables

On va rajouter des règles de réécriture dans les contextes

$$l \longrightarrow^{\Delta} r$$

Δ : contexte local à la règle, ne contient que des déclarations de variables objets

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash B : Type \quad \Gamma \vdash t : A}{\Gamma \vdash t : B} A \equiv_{\beta\Gamma} B$$

On peut réécrire avec les règles dans Γ

Bonnes hypothèses

Quelles hypothèses pour :

- ▶ réduction du sujet
- ▶ décidabilité de la vérification de type ?

Definition (Bon typage d'une règle)

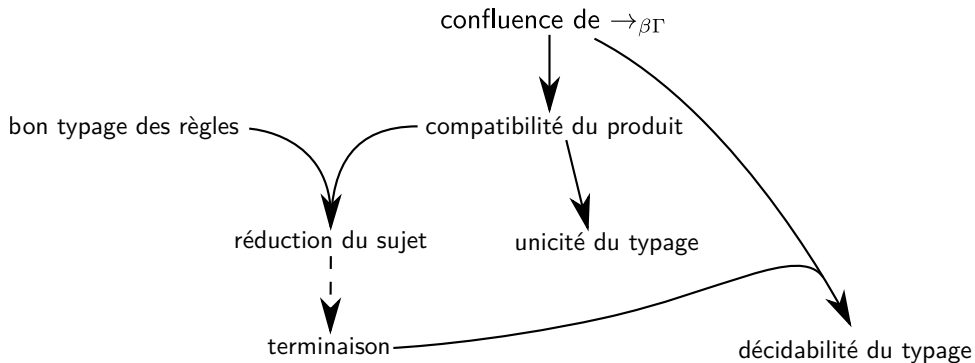
$l \longrightarrow^\Delta r$ est bien typé si :

Pour toute substitution σ des variables de Δ ,
si $\Gamma \vdash \sigma l : T$ alors $\Gamma \vdash \sigma r : T$.

Definition (Compatibilité avec le produit)

si $\Pi x : A_1 \ B_1 \equiv_{\beta\Gamma} \Pi x : A_2 \ B_2$ then $A_1 \equiv_{\beta\Gamma} A_2$ and $B_1 \equiv_{\beta\Gamma} B_2$

Implications



Bon typage des règles

$l \longrightarrow^\Delta r$ bien typé si $\Gamma, \Delta \vdash l : T$ et $\Gamma, \Delta \vdash r : T$

mais ne marche pas pour $tail\ n\ (cons\ m\ a\ v) \longrightarrow^{n,m,a,v} v$ alors que bien typée

Il est suffisant :

- Pour la substitution la plus générale τ permettant de typer τl dans Γ ,
Si $\Gamma, \Delta \vdash \tau l : T$ alors $\Gamma, \Delta \vdash \tau r : T$

Pour calculer τ , besoin de savoir quels symboles sont injectifs

- def

Outline

- Introduction
- Le $\lambda\Pi$ -calcul modulo théorie
- Logiques des prédicats
- Langages de programmation
- Logique d'ordre supérieur
- Calcul des constructions inductives

Logique des prédicats minimale

$$\begin{aligned} t &::= x \mid f(t, \dots, t) \\ A &::= P(t_1, \dots, t_n) \mid A \Rightarrow A \mid \forall_{s_1} x A \mid \dots \mid \forall_{s_n} x A \end{aligned}$$

Logique des prédicats minimale

$$\begin{aligned}
 t &::= x \mid f(t, \dots, t) \\
 A &::= P(t_1, \dots, t_n) \mid A \Rightarrow A \mid \forall_{s_1} x A \mid \dots \mid \forall_{s_n} x A
 \end{aligned}$$

```

s1 : Type.
s2 : Type.
f  : s1 -> s2.
P  : s2 -> s1 -> Type.

```

$$\begin{aligned}
 \|P(t_1, \dots, t_n)\| &= (\mathbf{P} \mid t_1 \mid \dots \mid t_n) \\
 \|A \Rightarrow B\| &= \|A\| \multimap \|B\| \\
 \|\forall_{s_i} x A\| &= \mathbf{x} : \mathbf{s} \multimap \|A\|
 \end{aligned}$$

Logique des prédicats constructive

$$\begin{aligned} t &::= x \mid f(t, \dots, t) \\ A &::= P(t_1, \dots, t_n) \mid A \Rightarrow A \mid \forall_s x A \mid \top \mid \perp \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists_s x A \end{aligned}$$

Logique des prédicats constructive

$$\begin{aligned} t &::= x \mid f(t, \dots, t) \\ A &::= P(t_1, \dots, t_n) \mid A \Rightarrow A \mid \forall_s x A \mid \top \mid \perp \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists_s x A \end{aligned}$$

Étendre le $\lambda\Pi$ -calcul modulo théorie

- ▶ avec de nouveaux connecteurs
- ▶ avec des inductifs ?

Logique des prédicats constructive

$$\begin{aligned}
 t &::= x \mid f(t, \dots, t) \\
 A &::= P(t_1, \dots, t_n) \mid A \Rightarrow A \mid \forall_s x A \mid \top \mid \perp \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists_s x A
 \end{aligned}$$

Étendre le $\lambda\Pi$ -calcul modulo théorie

- ▶ avec de nouveaux connecteurs
- ▶ avec des inductifs ?

Non, on peut rester avec le même base

Idée :

- ▶ Plongement profond dans un type \circ
- ▶ Remontée superficielle avec opérateur eps
en utilisant encodage imprédicatif des connecteurs

Encodage imprédictatif

```
o : Type.  
bot : o.  
imp : o -> o -> o.  
or : o -> o -> o.  
  
def eps : o -> Type.  
  
[a,b] eps (imp a b) --> eps a -> eps b.
```


Encodage imprédictif

```
o : Type.  
bot : o.  
imp : o -> o -> o.  
or : o -> o -> o.  
  
def eps : o -> Type.  
  
[a,b] eps (imp a b) --> eps a -> eps b.  
[] eps bot --> z : o -> eps z.
```

Encodage imprédictatif

```
o : Type.
```

```
bot : o.
```

```
imp : o -> o -> o.
```

```
or : o -> o -> o.
```

```
def eps : o -> Type.
```

```
[a,b] eps (imp a b) --> eps a -> eps b.
```

```
[] eps bot --> z : o -> eps z.
```

```
[a,b] eps (or a b) -->
```

```
  z : o -> (eps a -> eps z) -> (eps b -> eps z) -> eps z.
```

iProverModulo

iProver [Korovin 2008]

- ▶ prouveur automatique pour la logique des prédicats classique
- ▶ basé (entre autres) sur la résolution
- ▶ écrit en OCaml

patché pour intégrer la déduction modulo théorie [Burel 2011]

capable de sortir des preuves Dedukti pour la partie résolution [Burel 2013]
(qui est constructive!)

Logique des prédicats classique

Idée : introduire des connecteurs différents

$$\begin{aligned}
 t &::= x \mid f(t, \dots, t) \\
 a &::= P(t, \dots, t) \\
 A &::= a \mid A \Rightarrow A \mid \forall x A \mid \top \mid \perp \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \\
 &\quad a \mid A \Rightarrow_c A \mid \forall_c x A \mid \top_c \mid \perp_c \mid \neg_c A \mid A \wedge_c A \mid A \vee_c A \mid \exists_c x A
 \end{aligned}$$

Logique des prédicats classique

Idée : introduire des connecteurs différents

$$\begin{aligned}
 t &::= x \mid f(t, \dots, t) \\
 a &::= P(t, \dots, t) \\
 A &::= \triangleright a \mid A \Rightarrow A \mid \forall x A \mid \top \mid \perp \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \\
 &\quad \triangleright_c a \mid A \Rightarrow_c A \mid \forall_c x A \mid \top_c \mid \perp_c \mid \neg_c A \mid A \wedge_c A \mid A \vee_c A \mid \exists_c x A
 \end{aligned}$$

Logique des prédicats classique

Idée : introduire des connecteurs différents

$$t ::= x \mid f(t, \dots, t)$$

$$a ::= P(t, \dots, t)$$

$$A ::= \triangleright a \mid A \Rightarrow A \mid \forall x A \mid \top \mid \perp \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \\ \triangleright_c a \mid A \Rightarrow_c A \mid \forall_c x A \mid \top_c \mid \perp_c \mid \neg_c A \mid A \wedge_c A \mid A \vee_c A \mid \exists_c x A$$

Définir les connecteurs classiques en fonction des intuitionnistes en utilisant une $\neg\neg$ -traduction.

$$\triangleright_c a := \neg(\neg \triangleright a)$$

$$a \vee_c b := \neg(\neg(a \vee b))$$

Zenon Modulo

Extension de Zenon pour traiter la déduction modulo théorie [Halmagrand 2013]

Capable de produire des preuves Dedukti

600MB de preuves gzippées (à partir d'obligation de preuves B)

Outline

- Introduction
- Le $\lambda\Pi$ -calcul modulo théorie
- Logiques des prédicats
- Langages de programmation
- Logique d'ordre supérieur
- Calcul des constructions inductives

λ -calcul non typé

$$app\ (lam\ f)\ t \longrightarrow^{f,t} f\ t$$

λ -calcul simplement typé

Plongement profond des types simples
Remontée superficielle avec `term`

ML

Utilisation de destructeurs à la place du filtrage de motifs

$$\begin{aligned} \text{destr}_C R \ (C \ t) \ f \ d &\longrightarrow f \ t \\ \text{destr}_C R \ (C' \ t') \ f \ d &\longrightarrow d \quad (\text{for all other constructors } C' \text{ for type } \tau') \end{aligned}$$

Blocage de la récursion :

$$\begin{aligned} @ : A : \text{type} \rightarrow B : \text{type} \rightarrow ((\text{eps } A \rightarrow \text{eps } B) \rightarrow \text{eps } A \rightarrow \text{eps } B) \\ [R, f, t] @ \tau' R \ f \ (C \ t) &\longrightarrow f \ (C \ t) \end{aligned}$$

Outline

- Introduction
- Le $\lambda\Pi$ -calcul modulo théorie
- Logiques des prédicats
- Langages de programmation
- Logique d'ordre supérieur
- Calcul des constructions inductives

Outline

- Introduction
- Le $\lambda\Pi$ -calcul modulo théorie
- Logiques des prédicats
- Langages de programmation
- Logique d'ordre supérieur
- Calcul des constructions inductives