Dedukti In a Nutshell

Ronan Saillard

INRIA

MINES ParisTech, PSL Research University

January 9, 2015

What is Dedukti?

Dedukti is a type-checker for the $\lambda\Pi$-calculus modulo, a functional programming language featuring dependent types and user-defined rewrite rules.

Plan

- The $\lambda\Pi$-calculus modulo.
- Example 1: Programming with Dependent Types.
- Example 2: Efficient Encodings.

What is λΠ-calculus modulo?

λΠ-calculus (aka LF)

- λ-calculus with dependent types.
- Types are equal modulo β-conversion.

λΠ-calculus modulo

Types are equal modulo $\beta\mathcal{R}$-conversion where $\mathcal{R}$ is a set of rewrite rules.

Summary

λΠ-calculus modulo = λΠ-calculus + conversion extended with rewrite rules.

Example 1: Programming with Dependent Types

Programming with Dependent Types (1)

```
(; Peano Naturals ;)
Nat : Type.
0 : Nat.
S : Nat -> Nat.

(; Addition with Peano Naturals ;)
plus : Nat -> Nat -> Nat.
(; 0 + n = n ;)
[ n: Nat ] plus 0 n --> n.
(; (n1+1) + n2 = (n1+n2) + 1 ;)
[ n1: Nat, n2: Nat ]
      plus (S n1) n2 --> S (plus n1 n2).
```

Programming with Dependent Types (2)

```
A : Type.
Vector : Nat -> Type.
Nil : Vector 0.
Cons : n:Nat -> A -> Vector n -> Vector (S n).

(; Concatenation on Vectors ;)
append : n1:Nat -> n2:Nat -> Vector n1 ->
           Vector n2 -> Vector (plus n1 n2).
(; Nil @ v = v ;)
[ n: Nat, v: Vector n ] append 0 n Nil v --> v.
(; (a::v1) @ v2 = a::(v1 @ v2) ;)
[ n1:Nat, n2:Nat, v1:Vector n1, v2:Vector n2,a:A]
  append (S n1) n2 (Cons n1 a v1) v2 -->
    Cons (plus n1 n2) a (append n1 n2 v1 v2).
```

Programming with Dependent Types (3)

```
eq_vec : n:Nat -> Vector n -> Vector n -> Type.
refl : n:Nat -> v:Vector n -> eq_vec n v v.

n: Nat.
v: Vector n.

theorem1 : eq_vec n v (append n 0 Nil v)
    := refl n v v.
(; Typing Error ;)
conjecture2 : eq_vec n v (append n 0 v Nil).
```

Problem

Vector n is not convertible to **Vector (plus n 0)**.

Solution

Add a rewrite rule to extend the conversion.

Programming with Dependent Types (4)

```
plus : Nat -> Nat -> Nat.
[ n: Nat] plus 0 n --> n.
[ n1: Nat, n2: Nat ]
    plus (S n1) n2 --> S (plus n1 n2).
(...)
[ n: Nat ] plus n 0 --> n.

conjecture2 : eq_vec n l (append n 0 l Nil).

[ n1:Nat, n2:Nat ]
  plus n1 (S n2) --> S (plus n1 n2).
[ n1:Nat, n2:Nat, n3:Nat ]
  plus n1 (plus n2 n3) --> plus (plus n1 n2) n3.
```

- Confluence and termination must be preserved.
- Nothing to be proved: the rules are part of the definition of **plus**.
- **Conclusion:** Rewriting as a way to have a precise control over the conversion relation.

Example 2: Efficient Encodings

Encodings in the $\lambda\Pi$-calculus (modulo)

Dedukti is a logical framework

- One defines (encodes) his logic in Dedukti.
- Then one uses Dedukti to write theorems and check proofs in this logic.

Why rewrite rules?

Allows to design shallower encodings.

- *Embedding Pure Type Systems in the $\lambda\Pi$-calculus modulo, D. Cousineau and G. Dowek, 2007.*
- *The $\lambda\Pi$-calculus Modulo as a Universal Proof Language, M. Boespflug and Q. Carbonneaux and O. Hermant, 2012.*

Benchmarks

Encoding	**Standard**	**With Rewriting**	**Factor**
HOL/OpenTheory			
Size (Mo)	1024	53	19
Checking Time (s)	250	11	23
Zenon/TPTP			
Size (Mo)	192	9	21
Checking Time (s)	278	5	56

Conclusion

The $\lambda\Pi$-calculus modulo makes a good proof certificate format.

- Simple definition of your logic.
- Small proof terms.
- Efficient proof checking.

Conclusion

Rewriting as implemented in Dedukti:

- simplifies programing with dependent types through a precise control of the conversion relation.
- allows the design of simple and efficient encodings of logics.

Thank you

Dedukti

You can get **Dedukti** at http://dedukti.gforge.inria.fr.

Programs that use Dedukti as backend

- **Holide** (encoding of HOL proofs),
- **Coqine** (encoding of Coq proof),
- **Zenonide** (Zenon),
- **Focalide** (FoCaLiZe),
- **iProver**

at https://www.rocq.inria.fr/deducteam/software.html

Programs developed by Ali Assaf, Mathieu Boespflug, Guillaume Burel, Raphaël Cauderlier, Quentin Carbonneaux and Frédéric Gilbert.