DEDUKTI: A Universal Proof Checker

Quentin Carbonneaux[2]
Supervised by Mathieu Boespflug[1] Olivier Hermant[3]

[1]McGill University

[2]INRIA and ENPC

[3]INRIA and ISEP

MPRI defense 2012

CONTENTS

LOGICAL FRAMEWORKS AND THE $\lambda\Pi$-CALCULUS

A calculus with dependent types: $array : nat \to \mathrm{Type}$.

In a Curry-de Bruijn-Howard fashion, the $\lambda\Pi$-calculus is a language representing proofs of minimal predicate logic.

At least two possibilities to increase expressiveness:

1. enrich the $\lambda\Pi$-calculus by adding more deduction rules (*e.g.* CIC);
2. liberalize the conversion rule ($\lambda\Pi$-calculus modulo).

THE λΠ-CALCULUS MODULO

$$\begin{aligned} \mathrm{Var} &\ni x, y, z \\ \mathrm{Term} &\ni t, A, B ::= x \mid \lambda x{:}A.\ M \mid \Pi x{:}A.\ B \mid M\ N \mid \mathsf{Type} \mid \mathsf{Kind} \end{aligned}$$

FIGURE : Grammar of the λΠ-calculus modulo

TYPING RULES: ABSTRACTIONS

$$(\mathit{prod})\ \frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B : s}{\Gamma \vdash \Pi x{:}A.\ B : s}$$

$$(\mathit{abs})\ \frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B : s \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\ M : \Pi x{:}A.\ B}$$

$$s \in \{\mathsf{Type}, \mathsf{Kind}\}$$

TYPING RULES: DEPENDENT APPLICATION

$$(app)\ \frac{\Gamma \vdash M : \Pi x{:}A.\ B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : \{N/x\}B}$$

Typing rules: Conversion modulo

$$(conv)\;\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A : s \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B}\; A \equiv_{\beta\mathcal{R}} B$$

A DEDUKTI SIGNATURE

$$\forall y, 0 + y = y$$
$$\forall x, \forall y, \mathsf{S}\ x + y = \mathsf{S}\ (x + y).$$

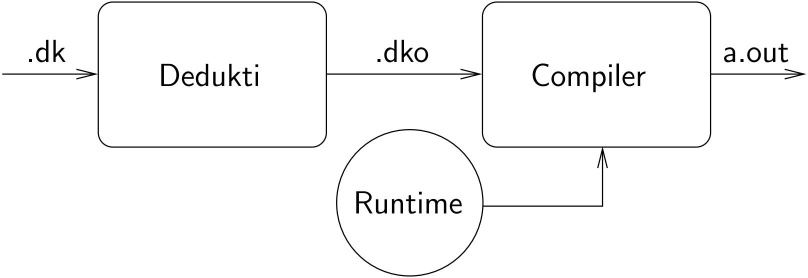
```
nat : Type.
Z : nat.
S : nat → nat.
plus : nat → nat → nat.

[y:nat]        plus Z     y ↪ y
[x:nat, y:nat] plus (S x) y ↪ S (plus x y).
```

DEDUKTI'S GOALS & TOOLS

- Versatility (HOL *v.s.* proofs by reflexion).
- Simplest compilation scheme.
- Small proof terms.
- Use compilation techniques.
  - Plenty of efficient compilers available;
  - reuse them off the shelf (separate concerns).

DEDUKTI'S ARCHITECTURE



DEDUKTI'S CALCULUS

$$\mathit{App}\ \frac{\vdash M \Rightarrow C \qquad C \longrightarrow_w^* \Pi x : A.\ B \qquad \vdash N \Leftarrow A}{\vdash M\ N \Rightarrow \{N/x\}B}$$

$$\mathit{Lam}\ \frac{C \longrightarrow_w^* \Pi x : A.\ B \qquad \vdash \{[y : A]/x\}M \Leftarrow \{y/x\}B}{\vdash \lambda x.\ M \Leftarrow C}$$

DEDUKTI'S CALCULUS

$$\mathit{App}\ \frac{\vdash M \Rightarrow C \qquad C \longrightarrow_w^* \Pi x : A.\ B \qquad \vdash N \Leftarrow A}{\vdash M\ N \Rightarrow \{N/x\}B}$$

$$\mathit{Lam}\ \frac{C \longrightarrow_w^* \Pi x : A.\ B \qquad \vdash \{[y : A]/x\}M \Leftarrow \{y/x\}B}{\vdash \lambda x.\ M \Leftarrow C}$$

- Compute on terms (find whnf).
- Inspect terms.
- Substitute variables.

DEDUKTI'S CALCULUS

$$App\ \frac{\vdash M \Rightarrow C \qquad C \longrightarrow_w^* \Pi x : A.\ B \qquad \vdash N \Leftarrow A}{\vdash M\ N \Rightarrow \{N/x\}B}$$

$$Lam\ \frac{C \longrightarrow_w^* \Pi x : A.\ B \qquad \vdash \{[y : A]/x\}M \Leftarrow \{y/x\}B}{\vdash \lambda x.\ M \Leftarrow C}$$

- Compute on terms (find whnf).
- Inspect terms.
- Substitute variables.

Hence the need of *two* translations.

TWO INTERPRETATIONS

The static version of terms in HOAS ($\ulcorner . \urcorner$).

```
data Term =
     Lam (Term → Term)
   | App Term Term
   | V Term
```

$$\begin{aligned} \ulcorner x \urcorner &= \mathsf{V}\ x \\ \ulcorner \lambda x.\ t \urcorner &= \mathsf{Lam}\ (\lambda x. \ulcorner t \urcorner) \\ \ulcorner a\ b \urcorner &= \mathsf{App}\ \ulcorner a \urcorner\ \ulcorner b \urcorner \end{aligned}$$

TWO INTERPRETATIONS

The static version of terms in HOAS ($\ulcorner . \urcorner$).

```
data Term =
      Lam (Term → Term)
    | App Term Term
    | V Term
```

With this interpreter:

$$\ulcorner x \urcorner = \mathsf{V}\ x \qquad\qquad \mathsf{eval}\ (\mathsf{V}\ x) = x$$

$$\ulcorner \lambda x.\ t \urcorner = \mathsf{Lam}\ (\lambda x. \ulcorner t \urcorner) \qquad\qquad \mathsf{eval}\ (\mathsf{Lam}\ f) = \lambda x. \mathsf{eval}\ (f\ x)$$

$$\ulcorner a\ b \urcorner = \mathsf{App}\ \ulcorner a \urcorner\ \ulcorner b \urcorner \qquad\qquad \mathsf{eval}\ (\mathsf{App}\ a\ b) = (\mathsf{eval}\ a)(\mathsf{eval}\ b)$$

How to peel the result of the evaluation?

TWO INTERPRETATIONS

$$\begin{aligned} \text{eval'}\ (\text{V}\ x) &= x \\ \text{eval'}\ (\text{Lam}\ f) &= \text{L}\ (\lambda x.\text{eval'}\ (f\ x)) \\ \text{eval'}\ (\text{App}\ a\ b) &= \text{app}\ (\text{eval'}\ a)\ (\text{eval'}\ b) \end{aligned}$$

$$\begin{aligned} \text{app}\ (\text{L}\ f)\ x &= f\ x \\ \text{app}\ a\ b &= \text{A}\ a\ b \end{aligned}$$

Two interpretations

$$\begin{aligned}\text{eval' } (\text{V } x) &= x\\ \text{eval' } (\text{Lam } f) &= \text{L } (\lambda x.\text{eval' } (f\ x))\\ \text{eval' } (\text{App } a\ b) &= \text{app } (\text{eval' } a)\ (\text{eval' } b)\end{aligned}$$

$$\begin{aligned}\text{app } (\text{L } f)\ x &= f\ x\\ \text{app } a\ b &= \text{A } a\ b\end{aligned}$$

The dynamic version of terms ($[\![.]\!]$).

$$[\![.]\!] = \text{eval'} \circ \ulcorner . \urcorner$$

$$\begin{aligned}[\![x]\!] &= x\\ [\![\lambda x.\ t]\!] &= \text{L } (\lambda x.[\![t]\!])\\ [\![a\ b]\!] &= \text{app } [\![a]\!]\ [\![b]\!]\end{aligned}$$

TWO INTERPRETATIONS

$$\begin{aligned}
\mathsf{eval'}\ (\mathsf{V}\ x) &= x \\
\mathsf{eval'}\ (\mathsf{Lam}\ f) &= \mathsf{L}\ (\lambda x.\mathsf{eval'}\ (f\ x)) \\
\mathsf{eval'}\ (\mathsf{App}\ a\ b) &= \mathsf{app}\ (\mathsf{eval'}\ a)\ (\mathsf{eval'}\ b)
\end{aligned}$$

$$\begin{aligned}
\mathsf{app}\ (\mathsf{L}\ f)\ x &= f\ x \\
\mathsf{app}\ a\ b &= \mathsf{A}\ a\ b
\end{aligned}$$

$$\begin{aligned}
\ulcorner x \urcorner &= \mathsf{V}\ x \\
\ulcorner \lambda x.\ t \urcorner &= \mathsf{Lam}\ (\lambda x.\ulcorner t \urcorner) \\
\ulcorner a\ b \urcorner &= \mathsf{App}\ \ulcorner a \urcorner\ \ulcorner b \urcorner
\end{aligned}$$

$$\begin{aligned}
[\![x]\!] &= x \\
[\![\lambda x.\ t]\!] &= \mathsf{L}\ (\lambda x.[\![t]\!]) \\
[\![a\ b]\!] &= \mathsf{app}\ [\![a]\!]\ [\![b]\!]
\end{aligned}$$

COMPILATION TO LUA

DEDUKTI generates one time usage Lua type checkers.

- Lua is a minimal programming language.
- Lua enjoys a very fast cutting edge JIT (luajit).
- Lua is not statically typed, not statically scoped.

THE JIT COMPROMISE

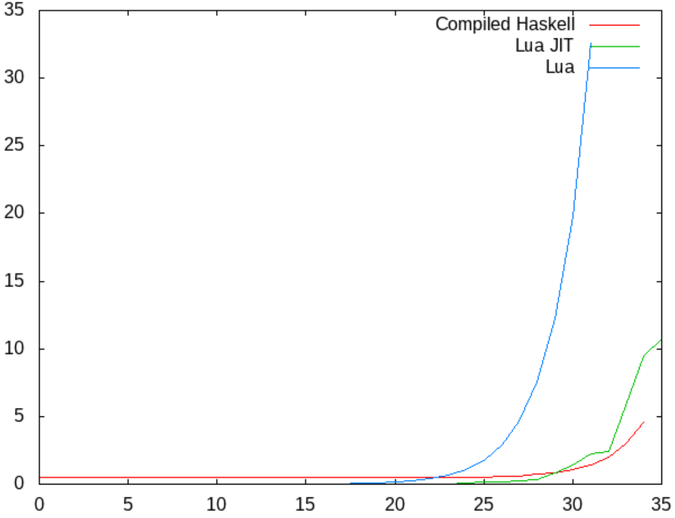


FIGURE : Compilation vs JIT vs Interpreter ($y = \text{Time}(1000\ \mathit{fib}(x))$)

CONTRIBUTIONS OF THIS WORK

- Brand new type checker using JIT compilation;
- some new optimizations w.r.t Boespflug's previous implementation;
- smaller proof terms using bidirectional type checking;
- combination of bidirectional and context-free systems in a "modulo" setting proven sound;
- hacking on COQINE to match the new implementation (available in the current release).

CONCLUSION

- DEDUKTI is
    - 1285 lines of C (+ 451 lines of comments);
    - blazingly fast on resonably sized examples;
    - not worse than an interpreter for computation free examples;
    - generating Lua code.
- Using a JIT allows a smoother behavior of type checking times.
- Accepted system description in the PxTP workshop.
- Next steps: improve our control on generated code, cope with luajit's limits.