Safe, Fast, Concurrent Proof Checking
for the lambda-Pi Calculus Modulo Rewriting

Michael Färber

2022-01-18
Section 1

Dedukti
Dedukti is a proof checker based on the $\lambda\Pi$-calculus modulo rewriting. It checks proofs from systems such as Coq, HOL Light, Isabelle, ... Proofs can become quite large and take long to check.

Question

How can we check Dedukti proofs faster, while keeping a small kernel?
Dedukti: Theories

Concepts

- Theory: a sequence of commands
- Command: introduces a constant or adds a rewrite rule

A Theory About Implication

\[
\begin{align*}
\text{prop} &: \text{Type} & (1) \\
\text{imp} &: \text{prop} \to \text{prop} \to \text{prop} & (2) \\
\text{prf} &: \text{prop} \to \text{Type} & (3) \\
\text{prf} (\text{imp} x y) &: \leftrightarrow \text{prf} x \to \text{prf} y & (4)
\end{align*}
\]
Dedukti: Checking

Parsing:
\[ [x, y] \text{ prf (imp } x \text{ y) } \rightarrow \text{ prf } x \rightarrow \text{ prf } y \text{ becomes } \]
\[ \text{prf (imp } x y) \leftrightarrow \text{ prf } x \rightarrow \text{ prf } y. \]

Type Inference: \[ \text{prf (imp } x y) : A \]

Type Checking: \[ \text{prf } x \rightarrow \text{ prf } y : A? \]
Dedukti: Checking

1. Parsing:
   \[ [x, y] \text{ prf (imp x y)} \rightarrow \text{prf x} \rightarrow \text{prf y} \] becomes
   \[ \text{prf (imp x y)} \leftrightarrow \text{prf x} \rightarrow \text{prf y}. \]

2. Type Inference: \[ \text{prf (imp x y)} : A \]

3. Type Checking: \[ \text{prf x} \rightarrow \text{prf y} : A? \]
Dedukti: Checking

1. Parsing:
   \[(x, y) \text{ prf (imp } x \ y) \rightarrow \text{ prf } x \rightarrow \text{ prf } y\] becomes
   \[\text{prf (imp } x \ y) \hookrightarrow \text{ prf } x \rightarrow \text{ prf } y.\]

2. Type Inference: \(\text{prf (imp } x \ y) : \text{ Type}\)

3. Type Checking: \(\text{prf } x \rightarrow \text{ prf } y : \text{ Type?}\)
Dedukti: Checking

1. Parsing:
   \[ [x, y] \text{ prf } (\text{imp } x \ y) \rightarrow \text{ prf } x \rightarrow \text{ prf } y \text{ becomes } \text{ prf } (\text{imp } x \ y) \leftrightarrow \text{ prf } x \rightarrow \text{ prf } y. \]

2. Type Inference: \text{ prf } (\text{imp } x \ y) : \text{ Type} \\

3. Type Checking: \text{ prf } x \rightarrow \text{ prf } y : \text{ Type? ✓}
Dedukti: Concurrency

- Dedukti checks multiple theories concurrently (one process per theory).
- For each theory, it processes only one command at a time.
- Can we somehow process multiple commands concurrently?
**Programming Languages**

**OCaml**
- Dedukti is implemented in OCaml
- Multicore support not (yet) available

**Rust**
- Functional systems programming language
- Memory- and thread-safe (unlike C)
- Focus on performance and concurrency

**Goal**
- Reimplement core of Dedukti in Rust
- Process multiple *commands* concurrently, using threads
Section 2

Concurrent Proof Checking
Sequential Proof Checking

Most time is spent in parsing and type checking
(69% for HOL Light and 85% for Isabelle/HOL corpora)

Concurrency
- Delegate parsing to an own thread
- Delegate type checking to multiple threads
Concurrent Parsing

Parse commands in a thread and send them via a channel to main thread:

Parse thread
parse → parse → ···

Main thread
infer → check → infer → check → ···

Best-case improvement: Reduce proof checking time by parsing time

In practice: channel overhead too large to make it pay off
Concurrent Type Checking

Launch a thread for every type checking task:

Main thread

Check thread

Check thread

Best-case improvement: Reduce proof checking time by type checking time
Section 3

Terms
Terms are *the* central data structure in Dedukti:

\[
  t := c \mid x \mid tu \mid \lambda x : t. u \mid \Pi x : t. u,
\]

where \( t \) and \( u \) are terms, \( c \) is a constant, \( x \) is a variable.
Pointer Types

Rust requires use of pointers to obtain inductive types (such as terms).

Figure 1: Three commonly used pointer types.

Thread-safe
Box
Arc
Fast
&
Rc
Shared
Three Types of Terms

Terms using different pointer types have different downsides:

- **Box-terms** take linear time to duplicate.
- **Rc-terms** cannot be used across threads.
- **Arc-terms** are slow.

<table>
<thead>
<tr>
<th>Task</th>
<th>Mode</th>
<th>Term pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>Any</td>
<td>Box</td>
</tr>
<tr>
<td>Type checking</td>
<td>Sequential</td>
<td>Rc</td>
</tr>
<tr>
<td>Type checking</td>
<td>Concurrent</td>
<td>Arc</td>
</tr>
</tbody>
</table>
Increasing Term Performance: Unboxing

- Omit pointers around constants and variables (do not have subterms)
- Reduces runtime by 20% when using Rc-terms and 29% when using Arc-terms.
Section 4

Implementation
Kontroli is a minimal concurrent proof checker for the $\lambda\Pi$-calculus modulo. 

https://github.com/01mf02/kontroli-rs

<table>
<thead>
<tr>
<th></th>
<th>Program</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedukti</td>
<td>3470 LOC</td>
<td></td>
</tr>
<tr>
<td>Kontroli</td>
<td>663 LOC</td>
<td></td>
</tr>
</tbody>
</table>

- Kontroli supports only a subset of Dedukti’s features
- Large enough to verify HOL-based theories
Evaluation

HOL Light

<table>
<thead>
<tr>
<th>Description</th>
<th>Runtime [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DK</td>
<td>344</td>
</tr>
<tr>
<td>KO</td>
<td>220</td>
</tr>
<tr>
<td>$DK \cap p$</td>
<td>77</td>
</tr>
<tr>
<td>$KO \cap p$</td>
<td>21</td>
</tr>
<tr>
<td>$DK_{t=\infty}$</td>
<td>307</td>
</tr>
<tr>
<td>$KO_{p=1}$</td>
<td>247</td>
</tr>
<tr>
<td>$KO_{c=1}$</td>
<td>282</td>
</tr>
<tr>
<td>$KO_{c=2}$</td>
<td>182</td>
</tr>
<tr>
<td>$KO_{c=4}$</td>
<td>147</td>
</tr>
<tr>
<td>$KO_{c=8}$</td>
<td>146</td>
</tr>
<tr>
<td>$KO \setminus c$</td>
<td>89</td>
</tr>
</tbody>
</table>

Isabelle/HOL

<table>
<thead>
<tr>
<th>Description</th>
<th>Runtime [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DK</td>
<td>415</td>
</tr>
<tr>
<td>KO</td>
<td>306</td>
</tr>
<tr>
<td>$DK \cap p$</td>
<td>195</td>
</tr>
<tr>
<td>$KO \cap p$</td>
<td>43</td>
</tr>
<tr>
<td>$DK_{t=\infty}$</td>
<td>414</td>
</tr>
<tr>
<td>$KO_{p=1}$</td>
<td>336</td>
</tr>
<tr>
<td>$KO_{c=1}$</td>
<td>355</td>
</tr>
<tr>
<td>$KO_{c=2}$</td>
<td>223</td>
</tr>
<tr>
<td>$KO_{c=4}$</td>
<td>153</td>
</tr>
<tr>
<td>$KO_{c=8}$</td>
<td>120</td>
</tr>
<tr>
<td>$KO \setminus c$</td>
<td>87</td>
</tr>
</tbody>
</table>
Section 5

Conclusion
Terms using Box, Rc, and Arc nicely fit parsing, sequential type checking, and parallel type checking.
Fewer pointers in the term type greatly benefit performance.
Parsing is one of the largest bottlenecks in Dedukti.
Concurrent parsing increases runtime, due to channel overhead.
Concurrent type checking significantly reduces runtime (up to 6.6x for 8 threads).

A small & safe proof checker with fast concurrency is possible!
Conclusion

- Terms using Box, Rc, and Arc nicely fit parsing, sequential type checking, and parallel type checking.
- Fewer pointers in the term type greatly benefit performance.
- Parsing is one of the largest bottlenecks in Dedukti.
- Concurrent parsing increases runtime, due to channel overhead.
- Concurrent type checking significantly reduces runtime (up to 6.6x for 8 threads).

A small & safe proof checker with fast concurrency is possible!

Thank you for your attention!