Typechecking in the $\lambda\Pi$-Calculus Modulo: Theory and Practice

PhD thesis defense

Ronan SAILLARD

MINES ParisTech, PSL Research University
INRIA, Deducteam

September 25, 2015

Formal Proving

Computers can help mathematicians and engineers prove theorems.

- Theorem provers.
- Proof checkers.

Examples:

The Kepler Conjecture

Operating System of Driverless Subway
(Paris, Line 14)

Logical Frameworks

There exist many tools for proving/checking

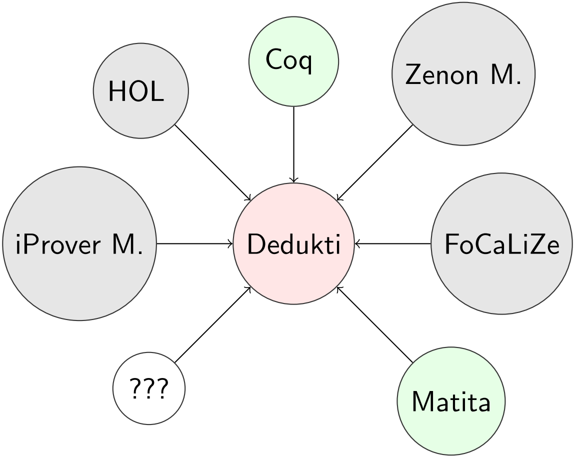
Agda, Beluga, Coq, . . . and DEDUKTI.

DEDUKTI: a Logical Framework

A tool to implement logical systems.

- Prototyping of proof systems.
- Independent proof checking.

A Universal Proof Checker



Long-Term Goal: allowing these programs to cooperate thanks to a unique proof format.

The Curry-Howard Correspondence [Curry 1958 and Howard 1969]

Observation

$$\frac{\Gamma \vdash A \implies B \qquad \Gamma \vdash A}{\Gamma \vdash B} \quad \approx \quad \frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash f\ u : B}$$

(Modus Ponens) **(Typing Rule for Application)**

Consequence

Proof checking can be reduced to Type checking

DEDUKTI is at the same time a proof checker and a type checker.

The Curry-Howard Correspondence [Curry 1958 and Howard 1969]

Observation

$$\frac{\Gamma \vdash A \implies B \qquad \Gamma \vdash A}{\Gamma \vdash B} \quad \approx \quad \frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash f\ u : B}$$

(Modus Ponens) **(Typing Rule for Application)**

Consequence

Proof checking can be reduced to Type checking

DEDUKTI is at the same time a proof checker and a type checker.

The $\lambda\Pi$-Calculus Modulo [Cousineau and Dowek, 2007]

The $\lambda\Pi$-**Calculus Modulo** is a typed calculus based on two features:

- Dependent Types.
- Rewrite Rules.

Dependent Types

The λ-Calculus with Dependent Types is called $\lambda\Pi$-Calculus or LF.

Idea

Types can be parameterized by terms.
Functions can return values whose types depend on their input.

Lists Parameterized by their Size

$\mathtt{nil} : \mathtt{Vector}\ 0$

$\mathtt{cons} : \Pi n : \mathtt{Nat.Elt} \longrightarrow \mathtt{Vector}\ n \longrightarrow \mathtt{Vector}\ (\mathtt{S}\ n)$

Typing Rules

$$\frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[x \leftarrow u]}\ (\text{Application})$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \textbf{Type} \qquad A \equiv_\beta B}{\Gamma \vdash t : B}\ (\text{Conversion})$$

Rewrite Rules

$\lambda\Pi$-Calculus Modulo

- β-reduction,
- A set $\mathcal{R}$ of rewrite rules $(f\,\vec{l} \hookrightarrow r)$.

Example

$\texttt{plus}\ n\ 0 \hookrightarrow n$
$\texttt{plus}\ n\ (\texttt{S}\ m) \hookrightarrow \texttt{S}\ (\texttt{plus}\ n\ m)$

Extended Conversion Rule

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \textbf{Type} \qquad A \equiv_{\beta\mathcal{R}} B}{\Gamma \vdash t : B}$$

Benefits

- Allows the design of small encodings of proof systems.
- Allows encoding more systems.

Encoding Propositional Logic

In the $\lambda\Pi$-**Calculus**:

$\texttt{prop} : \textbf{Type}.$
$\texttt{prf} : \texttt{prop} \longrightarrow \textbf{Type}.$

$\dot{\Longrightarrow} : \texttt{prop} \longrightarrow \texttt{prop} \longrightarrow \texttt{prop}.$

$\texttt{elim} : \Pi A : \texttt{prop}.\Pi B : \texttt{prop}.\ \texttt{prf}\ (A \dot{\Longrightarrow} B) \longrightarrow \texttt{prf}\ A \longrightarrow \texttt{prf}\ B.$
$\texttt{intro} : \Pi A : \texttt{prop}.\Pi B : \texttt{prop}.\ (\texttt{prf}\ A \longrightarrow \texttt{prf}\ B) \longrightarrow \texttt{prf}\ (A \dot{\Longrightarrow} B).$

In the $\lambda\Pi$-**Calculus Modulo**:
$\texttt{prf}\ (A \dot{\Longrightarrow} B) \hookrightarrow (\texttt{prf}\ A \longrightarrow \texttt{prf}\ B).$

Meta-Theorem (in both cases)

$\Sigma \vdash P$ iff $\exists t\ (\Gamma; \dot{\Sigma} \vdash t : \texttt{prf}\ \dot{P}).$

In the $\lambda\Pi$-Calculus Modulo, proof terms are usually smaller and can be checked faster.

General Contribution: More Safety

Previous versions of DEDUKTI could give incorrect results if the input problem did not verify the subject reduction property (preservation of types by reduction). And DEDUKTI did not check subject reduction compromising its soundness.

More Safety

- I studied the subject reduction property and showed how it can be checked.
- I implemented the verification in DEDUKTI.

General Contribution: More Expressiveness

From Algebraic Rewrite Rules

- Left-hand sides are algebraic terms (built from constant applications and variables only).
- **Example:**
  $\texttt{plus}\ n\ 0 \hookrightarrow n$
  $\texttt{plus}\ n\ (\texttt{S}\ m) \hookrightarrow \texttt{S}\ (\texttt{plus}\ n\ m)$

To Higher-Order Rewrite Rules

- Left-hand sides may contain abstractions.
- **Example:**
  - $\texttt{D}\ (\lambda x : R.\texttt{Exp}\ (f\ x)) \hookrightarrow (\texttt{D}\ (\lambda x : R.f\ x)) \times (\lambda x : R.\texttt{Exp}\ (f\ x))$.
  - Encoding of Coq's universes [Assaf, 2014].

$\lambda\Pi$-Calculus Modulo vs Dedukti

$\lambda\Pi$-Calculus Modulo

- The set of rewrite rules $\mathcal{R}$ is fixed.
- Rewrite rules are typed outside the system.

Dedukti

- Rewrite rules can be added at any time.
- Rewrite rules are typed iteratively.
- More rules can be checked.

$\lambda\Pi$-Calculus Modulo with Global Contexts

Global Contexts and Local Contexts

$$\begin{array}{lcl} \Gamma & ::= & () \mid \Gamma(c : A) \mid \Gamma(f\vec{l} \hookrightarrow r) \\ \Delta & ::= & () \mid \Delta(x : A) \end{array}$$

Conversion Rule

$$\frac{\Gamma; \Delta \vdash t : A \qquad \Gamma; \Delta \vdash B : s \qquad A \equiv_{\beta\Gamma} B}{\Gamma; \Delta \vdash t : B}$$

Improvements

- Allows typing more rewrite rules.
- Eases the reasoning about DEDUKTI (soundness/completeness proofs).

Publication: Towards Explicit Rewrite Rules in the $\lambda\Pi$-Calculus Modulo, R. Saillard in IWIL, 2013.

A Fundamental Property

Subject Reduction

$\Gamma; \Delta \vdash t_1 : T \wedge t_1 \rightarrow_{\beta\Gamma} t_2 \implies \Gamma; \Delta \vdash t_2 : T$

Subject Reduction is necessary

for proving any non-trivial property about the type system and in particular

- the soundness/completeness of proof embeddings.
- the soundness/completeness of typechecking algorithms.
- termination.

Subject reduction may not hold!

Product Compatibility and Well-Typedness of Rewrite Rules

The proof of subject reduction can be reduced to the proof of two simpler properties.

Product Compatibility [Geuvers, 1992]

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

Well-Typed Rewrite Rules [Blanqui, 2005]

For all $(l \hookrightarrow r) \in \Gamma$ and substitution σ, if $\Gamma; \Delta \vdash \sigma(l) : T$, then $\Gamma; \Delta \vdash \sigma(r) : T$.

Remark

These properties are undecidable. To check them in DEDUKTI, we need to find decidable criteria.

1. A $\lambda\Pi$-Calculus Modulo with Global Contexts
2. Product Compatibility and Higher-Order Rewrite Rules
3. Typing Rewrite Rules
4. Conclusion

Product Compatibility

Product Compatibility (PC)

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

Theorem: PC for Object-Level Systems [Barbanera et al,1994]

Product Compatibility holds when there are no type-level rewrite rules.

Theorem: PC by Confluence

Product Compatibility follows from the confluence of $\rightarrow_{\beta\Gamma}$.

Higher-Order Rewrite Rules

Derivation Operation $(e^f)' = f' \times e^f$

$\mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ (f\ x)) \hookrightarrow (\mathtt{D}\ (\lambda x : R.f\ x)) \times (\lambda x : R.\mathtt{Exp}\ (f\ x)).$

Critical Pair

$$\begin{tikzcd} & \mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ ((\lambda y : R.y)\ x)) \arrow[dl, "\mathtt{D}"'] \arrow[dr, "\beta"] & \\ (\mathtt{D}\ (\lambda x : R.(\lambda y : R.y)\ x)) \times (\lambda x : R.(\mathtt{Exp}\ ((\lambda y : R.y)\ x))) & & \mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ x) \end{tikzcd}$$

The critical peak cannot be joined; **confluence is lost**.

Remark

In the λΠ-Calculus Modulo, matching is syntactic.

Two Problems

- How to prove product compatibility?
- How to decide the congruence $\equiv_{\beta\Gamma}$?

Rewriting Modulo beta

$$
\begin{tikzcd}
& \mathtt{D}\;(\lambda x : R.\mathtt{Exp}\;((\lambda y : R.y)\;x)) \arrow[dl, "\mathtt{D}"'] \arrow[dr, "\beta"] & \\
\ldots \;\times\; (\lambda x : R.(\mathtt{Exp}\;((\lambda y : R.y)\;x))) \arrow[dr, "\beta"'] & & \mathtt{D}\;(\lambda x : R.\mathtt{Exp}\;x) \arrow[dl, dashed, red, "?"'] \\
& (\mathtt{D}\;(\lambda x : R.x)) \;\times\; (\lambda x : R.(\mathtt{Exp}\;x)) &
\end{tikzcd}
$$

Technical Choice

Use Higher-Order Rewrite Systems [Nipkow, 1991] to define rewriting modulo beta in the λΠ-Calculus Modulo.

Advantages

- Confluence results of HRSs [van Oostrom, 1995].
- Automatic confluence checkers (CSIˆho [Nagele, 2015], ACPH [Onowawa et al, 2015])

Higher-Order Rewrite Systems [Nipkow, 1991]

Terms

Simply typed λ-terms in $\beta\overline{\eta}$-normal form over some signature.

Higher-Order Patterns and Rewrite Rules

- A **rewrite rule** is a pair of terms $(l \hookrightarrow r)$ where l is a **higher-order pattern** [Miller, 1991].
- A term is a **higher-order pattern** if the arguments of its free variables are lists of terms η-equivalent to distinct bound variables.
- Rewriting is performed modulo $\beta\eta$.

Example: λ-calculus

$\Sigma = \{\ \texttt{Lam}, \texttt{App}\ \}\quad (\beta)\quad \texttt{App}(\texttt{Lam}(\underline{\lambda}x.F(x)), A) \hookrightarrow F(A)$

Defining Rewriting Modulo beta

Encoding Terms

- We encode **untyped $\lambda\Pi$-terms** as **typed HRS-terms**.
- **Example:**
  $\|\mathtt{D}\ (\lambda x : \mathtt{Nat}.f\ x)\| = \mathtt{App}(\mathtt{D}, \mathtt{Lam}(\mathtt{Nat}, \underline{\lambda} x.\mathtt{App}(f, x)))$

Encoding Rewrite Rules

- $(l \hookrightarrow r) \mapsto \|(l \hookrightarrow r)\| \neq (\|l\| \hookrightarrow \|r\|)$.
- **Example:**
  $\|\mathtt{D}\ (\lambda x : \mathtt{Nat}.f\ x) \hookrightarrow f\ 0\| = \mathtt{App}(\mathtt{D}, \mathtt{Lam}(\mathtt{Nat}, \underline{\lambda} x.f(x))) \hookrightarrow f(0)$
- Restricted to be higher-order patterns.

Rewriting Modulo beta

$t_1 \rightarrow_{\Gamma^{beta}} t_2 =_{def} \|t_1\| \rightarrow_{\|\Gamma\|_{\beta\eta}} \|t_2\|$

Defining Rewriting Modulo beta

Encoding Terms

- We encode **untyped $\lambda\Pi$-terms** as **typed HRS-terms**.
- **Example:**
  $\|\mathtt{D}\ (\lambda x : \mathtt{Nat}.f\ \ x)\| = \mathtt{App}(\mathtt{D}, \mathtt{Lam}(\mathtt{Nat}, \underline{\lambda} x.\mathtt{App}(f, x)))$

Encoding Rewrite Rules

- $(l \hookrightarrow r) \mapsto \|(l \hookrightarrow r)\| \neq (\|l\| \hookrightarrow \|r\|)$.
- **Example:**
  $\|\mathtt{D}\ (\lambda x : \mathtt{Nat}.f\ \ x) \hookrightarrow f\ \ \mathtt{0}\| = \mathtt{App}(\mathtt{D}, \mathtt{Lam}(\mathtt{Nat}, \underline{\lambda} x.f(x))) \hookrightarrow f(0)$
- Restricted to be higher-order patterns.

Rewriting Modulo beta

$t_1 \to_{\Gamma^{beta}} t_2 =_{def} \|t_1\| \to_{\|\Gamma\|_{\beta\eta}} \|t_2\|$

Confluence Modulo beta

Theorem

Product Compatibility follows from the confluence of $\rightarrow_{\beta\Gamma^{beta}}$.

Theorem

If $\rightarrow_{\beta\Gamma^{beta}}$ terminates, then $\equiv_{\beta\Gamma}$ is decidable.

Key Lemma

The congruences $\equiv_{\beta\Gamma}$ and $\equiv_{\beta\Gamma^{beta}}$ are equal.

Proof:

- $\rightarrow_{\beta\Gamma} \quad \subset \quad \rightarrow_{\beta\Gamma^{beta}}$.
- $\rightarrow_{\Gamma^{beta}} \quad \subset \quad \leftarrow_{\beta}^{*} . \rightarrow_{\Gamma}$.

Remark

Adding rewriting modulo beta to the λΠ-Calculus Modulo does not modify the type system.

Summary

Product Compatibility

A **new criterion** for proving product compatibility that

- is strictly more general than the previous one based on (usual) confluence.
- can be used in presence of higher-order rewrite rules.

Implementation

- DEDUKTI now implements higher-order rewrite rules.
  It has been used to encode Coq's Universes [Assaf, 2014].
- DEDUKTI now checks confluence (modulo beta) using an external checker.
  This allowed us to find bugs in existing DEDUKTI developments.

Publication: Rewriting Modulo β in the $\lambda\Pi$-Calculus Modulo, R. Saillard in LFMTP, 2015.

Colored $\lambda\Pi$-Calculus Modulo

Problem

- Proving product compatibility in presence of type-level rules and a non-confluent rewrite system (for instance due to non-left-linear rules).
- **Difficulty:** conversions may contain ill-typed terms; we cannot assume subject reduction.

Colored $\lambda\Pi$-Calculus Modulo

- Approximate typing by a weak notion of typing for which subject reduction is easy to prove.
- Constrain the conversion to be weakly well-typed.
- Use weak typing to show that more rewrite systems verify product compatibility.

Typing Rewrite Rules

Well-Typed Rewrite Rules

A rule $(l \hookrightarrow r)$ is **well-typed** for Γ if, $\forall \sigma \forall \Delta \forall T$
$\Gamma; \Delta \vdash \sigma(l) : T \implies \Gamma; \Delta \vdash \sigma(r) : T$.

Criterion used by Dedukti < 2.3

- l is algebraic
- $\Gamma; \Delta \vdash l : T$
- $\Gamma; \Delta \vdash r : T$
- $\Gamma \vdash^{ctx} \Delta$ and $dom(\Delta) = FV(l)$.

Theorem

Let Γ be a global context such that $\to_{\beta\Gamma}$ is confluent. If the hypotheses above are verified, then the rewrite rule is well-typed.

What about Higher-order Rewrite Rules?

First Limitation

Cannot type higher-order rewrite rules.

Analysis

The hypothesis that l is algebraic is used to show that, if $\Gamma; \Delta \vdash l : T$ and $\Gamma; \Delta_2 \vdash \sigma(l) : T_2$, then σ is well-typed, that is

$$\forall x. \Gamma; \Delta_2 \vdash \sigma(x) : \sigma(\Delta(x)).$$

What is really needed

- the types of the variables should be uniquely determined by the shape of l.
- the types of the variables should be inferable.

Typing Left-Hand Sides

Bidirectional Typing [Pierce, 1997] of Patterns

- (Synthesis) $\Gamma; \Delta_1; \Sigma \Vdash_s t \Rightarrow A, \Delta_2$
- (Checking) $\Gamma; \Delta_1; \Sigma \Vdash_c t \Leftarrow A \mid \Delta_2$

Rules

$$\frac{\Gamma; \Delta_1; \Sigma \Vdash_s u \Rightarrow T, \Delta_2 \qquad T \rightarrow^*_{\beta\Gamma} \Pi x : A.B \qquad \Gamma; \Delta_2; \Sigma \Vdash_c v \Leftarrow A \mid \Delta_3}{\Gamma; \Delta_1; \Sigma \Vdash_s uv \Rightarrow B[x/u], \Delta_3} \textbf{ (Application)}$$

$$\frac{\Gamma; \Delta_1; \Sigma \Vdash_s u \Rightarrow A_2, \Delta_2 \qquad A_1 \equiv_{\beta\Gamma} A_2}{\Gamma; \Delta_1; \Sigma \Vdash_c u \Leftarrow A_1 \mid \Delta_2} \textbf{ (Inversion)}$$

$$\frac{x \notin dom(\Sigma \cup \Delta_1)}{\Gamma; \Delta_1; \Sigma \Vdash_c x \Leftarrow B \mid \Delta_1(x : B)} \textbf{ (Free Variable)}$$

A First Generalization

Theorem

If

- $\Gamma; \emptyset; \emptyset \Vdash_s l \Rightarrow T, \Delta$
- and $\Gamma; \Delta \vdash r : T$,

then $(l \hookrightarrow r)$ is well-typed in Γ.

Remark

If t is a well-typed higher-order pattern, then
$\Gamma; \emptyset, \emptyset \Vdash_s t \Rightarrow T, \Delta$ for some T and Δ.

Rewrite Rules and Dependent Typing

Example

$\texttt{head} \;:\; \Pi n : \texttt{Nat}.\texttt{Vector}\ (\texttt{S}\ n) \longrightarrow \texttt{Elt}.$

$\texttt{head}\ n\ (\texttt{cons}\ n\ e\ v) \;\hookrightarrow\; e.$

$\texttt{tail} \;:\; \Pi n : \texttt{Nat}.\texttt{Vector}\ (\texttt{S}\ n) \longrightarrow \texttt{Vector}\ n.$

$\texttt{tail}\ n\ (\texttt{cons}\ n\ e\ v) \;\hookrightarrow\; v.$

Problem

These rewrite rules are not left-linear.

Implementing head and tail differently:

$\texttt{head}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \;\hookrightarrow\; e.$

$\texttt{tail}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \;\hookrightarrow\; v.$

These rewrite rules are well-typed [Blanqui, 2005].

We know that, if the redex $\sigma(\texttt{head}\ \ldots)$ is well-typed, then $\sigma(n_1) \equiv_{\beta\Gamma} \sigma(n_2)$

Weakening Bidirectional Checking (1)

Inversion

$$\frac{\Gamma; \Delta_1; \Sigma \Vdash_s u \Rightarrow A_2, \Delta_2 \qquad A_1 \equiv_{\beta\Gamma} A_2}{\Gamma; \Delta_1; \Sigma \Vdash_c u \Leftarrow A_1 \mid \Delta_2} \quad \longrightarrow \quad \frac{\Gamma; \Delta_1; \Sigma \Vdash_s u \Rightarrow A_2, \Delta_2}{\Gamma; \Delta_1; \Sigma \Vdash_c u \Leftarrow A_1 \mid \Delta_2}$$

Head

- $\Gamma; \emptyset; \emptyset \Vdash_s \texttt{head}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \Rightarrow \texttt{Elt}, \Delta$
  for $\Delta = (n_1 : \texttt{Nat})(n_2 : \texttt{Nat})(e : \texttt{Elt})(v : \texttt{Vector}\ n_2)$
- $\Gamma; \Delta \vdash e : \texttt{Elt}$

Thus, $(\texttt{head}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \quad \hookrightarrow \quad e)$ is well-typed.

Weakening Bidirectional Checking (2)

Tail

- $\Gamma;\emptyset;\emptyset \Vdash_s \texttt{tail}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \Rightarrow \texttt{Vector}\ n_1, \Delta$
  for $\Delta = (n_1 : \texttt{Nat})(n_2 : \texttt{Nat})(e : \texttt{Elt})(v : \texttt{Vector}\ n_2)$
- $\Gamma;\Delta \vdash v : \texttt{Vector}\ n_2$

The criterion still does not apply.

However
We know, by typing, that if $\sigma(\texttt{tail}\ n_1\ (\texttt{cons}\ n_2\ e\ v))$ is well-typed, then $\sigma(n_1) \equiv_{\beta\Gamma} \sigma(n_2)$.

Recording Conversion Tests

Solution

- Record conversion tests.

$$\frac{\Gamma; \Delta_1; \Sigma \Vdash_s u \Rightarrow A_2, \Delta_2, \mathcal{C}}{\Gamma; \Delta_1; \Sigma \Vdash_c u \Leftarrow A_1 \mid \Delta_2, \mathcal{C} \cup \{(A_1, A_2)\}}$$

- Use this information when typing the right-hand side of the rule.

Example

- $\Gamma; \emptyset; \emptyset \Vdash_s \texttt{tail}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \Rightarrow \texttt{Vector}\ n_1, \Delta, \mathcal{C}$ and $(\texttt{Vector}\ (\texttt{S}\ n_1), \texttt{Vector}\ (\texttt{S}\ n_2)) \in \mathcal{C}$
- Any solution σ of $\mathcal{C}$ verifies $\sigma(n_1) \equiv_{\beta\Gamma} \sigma(n_2)$
- $\Gamma; \Delta[n_2 \leftarrow n_1] \vdash v : \texttt{Vector}\ n_1$

Thus, $(\texttt{tail}\ n_1\ (\texttt{cons}\ n_2\ e\ v) \ \hookrightarrow\ \ v)$ is well-typed.

Summary

A general criterion for typing rewrite rules

- Compatible with higher-order rewrite rules.
- Allows typing linearized versions of rewrite rules when non-left linearity is due to typing constraints.

Implementation in DEDUKTI

- It replaces the unsafe way of linearizing rewrite rules implemented in the previous versions of DEDUKTI.
- Users do not need to give the typing context Δ anymore.

Perspectives

Termination:

Termination is necessary for deciding typing.

- Design termination criteria for $\rightarrow_{\beta\Gamma}$ and $\rightarrow_{\beta\Gamma^{beta}}$
- Implement them in DEDUKTI.

DEDUKTI as a Proof Assistant

- Refiner (Already done by G. Gilbert).
- Tactics.
- Standard Library.
- Theorem Prover.

Contributions

Making DEDUKTI Safer

DEDUKTI now checks the **subject reduction property**.

- **Product compatibility** is ensured by confluence of **rewriting modulo beta**.
- **Well-typedness of rewrite rules**.

Making DEDUKTI More Expressive

Adding **higher-order rewrite rules** to DEDUKTI.
Used to import proofs from Coq and Matita.

New Concepts

- The $\lambda\Pi$-Calculus Modulo with **Global Contexts**.
- A notion of **rewriting modulo beta**.
- The **Colored $\lambda\Pi$-Calculus Modulo**.

And a **new implementation** of DEDUKTI in OCaml.

Typechecking in the $\lambda\Pi$-Calculus Modulo: Theory and Practice

PhD thesis defense

Ronan SAILLARD

MINES ParisTech, PSL Research University
INRIA, Deducteam

September 25, 2015

Permanently Well-Typed Rewrite Rules

Example (continued)

$\texttt{Vector}\ (\texttt{S}\ n) \hookrightarrow \texttt{NonEmptyVector}$.

For $\Delta = (n_1 : \texttt{Nat})(n_2 : \texttt{Nat})(e : \texttt{Elt})(l : \texttt{Vector}\ n_2)$, we have $\Gamma'; \Delta \vdash \texttt{tail}\ n_1\ (\texttt{cons}\ n_2\ e\ l) : \texttt{Vector}\ n_1$.
(because
$\texttt{Vector}\ (\texttt{S}\ n_1) \equiv_{\beta\Gamma'} \texttt{NonEmptyVector} \equiv_{\beta\Gamma'} \texttt{Vector}\ (\texttt{S}\ n_2)$)

But we have
$\texttt{tail}\ k_1\ (\texttt{cons}\ k_2\ e\ l) \rightarrow_{\Gamma'} l$.
$\Gamma_2; \Delta \vdash l : \texttt{Vector}\ n_2$ and $\Gamma_2; \Delta \nvdash l : \texttt{Vector}\ n_1$.

The rewrite rule is no more well-typed.

Explanation

$\tau = \{n_2 \mapsto n_1\}$ is a prefix for $\mathcal{C}$ in Γ but not in Γ_2.

Morality

Rewrite Rules may become ill-typed a posteriori.

Permanently Well-Typed Rewrite Rules

Definition

A rewrite rule is permanently well-typed in Γ if it is well-typed in any (well-formed) extension of Γ.

Except for the last one, all the previous criteria provide permanent well-typedness.

Static Symbols

Static symbols are constants for which we (implicitly) assume that they will never be associated to rewrite rules.

Tail

If `Vector` and `S` are declared as static symbols, then
$\tau = \{n_2 \mapsto n_1\}$ will remain a prefix in any extension of the context.
In this case, the rule on `tail` is permanently well-typed.

Typing Rules for Global Contexts

$$\text{(Empty)}\ \frac{}{\emptyset\ \mathbf{wf}}$$

$$\text{(Declaration)}\ \frac{\Gamma\ \mathbf{wf} \qquad \Gamma;\emptyset \vdash T : \mathbf{Type} \qquad c \notin dom(\Gamma)}{\Gamma(c:T)\ \mathbf{wf}}$$

$$\frac{\Gamma\ \mathbf{wf} \qquad \rightarrow_{\beta(\Gamma\Xi)^{beta}} \text{ is confluent} \qquad \begin{array}{c}(\forall i)\Gamma \vdash u_i \hookrightarrow v_i\\ \Xi = (u_1 \hookrightarrow v_1)\dots(u_n \hookrightarrow v_n)\end{array}}{\Gamma\Xi\ \mathbf{wf}}$$

Theorem

If Γ **wf**, then Γ verifies subject reduction.

(Sort) $$\frac{}{\Gamma; \Delta \vdash \textbf{Type} : \textbf{Kind}}$$

(Variable) $$\frac{(x : A) \in \Delta}{\Gamma; \Delta \vdash x : A}$$

(Constant) $$\frac{(c : A) \in \Gamma}{\Gamma; \Delta \vdash c : A}$$

(Application) $$\frac{\Gamma; \Delta \vdash t : \Pi x : A.B \qquad \Gamma; \Delta \vdash u : A}{\Gamma; \Delta \vdash tu : B[x/u]}$$

(Abstraction) $$\frac{\Gamma; \Delta(x : A) \vdash t : B \qquad \Gamma; \Delta \vdash \Pi x : A.B : s}{\Gamma; \Delta \vdash \lambda x : A.t : \Pi x : A.B}$$

(Product) $$\frac{\Gamma; \Delta \vdash A : \textbf{Type} \qquad \Gamma; \Delta(x : A) \vdash B : s}{\Gamma; \Delta \vdash \Pi x : A.B : s}$$

(Conversion) $$\frac{\Gamma; \Delta \vdash t : A \qquad \Gamma; \Delta \vdash B : s \qquad A \equiv_{\beta\Gamma} B}{\Gamma; \Delta \vdash t : B}$$

Typing Rules for Local Contexts

(Empty)
$$\frac{}{\Gamma \vdash^{ctx} \emptyset}$$

(Var)
$$\frac{\Gamma \vdash^{ctx} \Delta \qquad \Gamma; \Delta \vdash \underline{U} : \textbf{Type} \qquad x \notin dom(\Delta)}{\Gamma \vdash^{ctx} \Delta(x : \underline{U})}$$

Typing Rules for Global Contexts

$$\text{(Empty)}\ \frac{}{\emptyset\ \mathbf{wf}}$$

$$\text{(Declaration)}\ \frac{\Gamma\ \mathbf{wf} \qquad \Gamma;\emptyset \vdash T : \mathbf{Type} \qquad c \notin dom(\Gamma)}{\Gamma(c : T)\ \mathbf{wf}}$$

$$\frac{\Gamma\ \mathbf{wf} \qquad \rightarrow_{\beta(\Gamma\Xi)^{beta}} \text{ is confluent} \qquad \Xi = (u_1 \hookrightarrow v_1)\ldots(u_n \hookrightarrow v_n) \quad (\forall i)\Gamma \vdash u_i \hookrightarrow v_i}{\Gamma\Xi\ \mathbf{wf}}$$

Theorem

If Γ **wf**, then Γ verifies subject reduction.

Nipkow's Higher-Order Rewrite Systems

Terms of HRS

Simply typed λ-terms in $\beta\overline{\eta}$-normal form over some signature Σ.
($u, v ::= c \in \Sigma \mid x \mid \underline{\lambda} x.t \mid u(v)$, variables have a type)

Patterns and Rewrite Rules

- A **rewrite rule** is a pair of terms $(l \hookrightarrow r)$ where l is a **pattern**.
- A term is a (Miller) **pattern** if the arguments of its free variables are lists of terms η-equivalent to distinct bound variables.
- Higher-order unification (and matching) of patterns is decidable and most general substitutions exist.

Higher-Order Rewriting

Let R be a set of rewrite rules.

- if $(l \hookrightarrow r) \in R$, then $\updownarrow_{\beta}^{\eta} (\theta(l)) \to_R \updownarrow_{\beta}^{\eta} (\theta(r))$.
- if $\vec{t_1} \to_R \vec{t_2}$, then $t_0(t_1) \to_R t_0(t_2)$, $t_1(t_0) \to_R t_2(t_0)$ and $\underline{\lambda} x.t_1 \to_R \underline{\lambda} x.t_2$;

Example of HRS: the λ-calculus

Signature

$$\begin{aligned}\Sigma = \{\ & \mathtt{Lam} : (\mathtt{Term} \longrightarrow \mathtt{Term}) \longrightarrow \mathtt{Term},\\ & \mathtt{App} : \mathtt{Term} \longrightarrow \mathtt{Term} \longrightarrow \mathtt{Term}\ \}\end{aligned}$$

Rewrite Rule

$$(\beta) \quad \mathtt{App}(\mathtt{Lam}(\underline{\lambda} x.F(x)), A) \hookrightarrow F(A)$$

Example

$$\begin{aligned}\mathtt{App}(\mathtt{Lam}(\underline{\lambda} x.x), c) &= \updownarrow_{\beta}^{\eta} (\mathtt{App}(\mathtt{Lam}(\underline{\lambda} x.(\underline{\lambda} y.y)(x)), c))\\ &\hookrightarrow \updownarrow_{\beta}^{\eta} ((\underline{\lambda} y.y)(c))\\ &= c\end{aligned}$$

Encoding $\lambda\Pi$-Terms

$$\begin{aligned}\Sigma = \{\ c\ : \mathtt{Term} \mid c \in \mathcal{C}\ \} \cup \{\ &\mathtt{Type}\ :\ \mathtt{Term},\ \mathtt{Kind}\ :\ \mathtt{Term},\\ &\mathtt{Lam} : \mathtt{Term} \longrightarrow (\mathtt{Term} \longrightarrow \mathtt{Term}) \longrightarrow \mathtt{Term},\\ &\mathtt{App} : \mathtt{Term} \longrightarrow \mathtt{Term} \longrightarrow \mathtt{Term},\\ &\mathtt{Pi} : \mathtt{Term} \longrightarrow (\mathtt{Term} \longrightarrow \mathtt{Term}) \longrightarrow \mathtt{Term}\ \}\end{aligned}$$

The Encoding

$\Vert\mathbf{Kind}\Vert$	$:=$	$\mathtt{Kind}$	$\Vert\mathbf{Type}\Vert$	$:=$	$\mathtt{Type}$
$\Vert x\Vert$	$:=$	x (of type $\mathtt{Term}$)	$\Vert c\Vert$	$:=$	$\mathtt{c}$
$\Vert\lambda x : A.t\Vert$	$:=$	$\mathtt{Lam}(\Vert A\Vert, \underline{\lambda} x.\Vert t\Vert)$	$\Vert uv\Vert$	$:=$	$\mathtt{App}(\Vert u\Vert, \Vert v\Vert)$
$\Vert\Pi x : A.B\Vert$	$:=$	$\mathtt{Pi}(\Vert A\Vert, \underline{\lambda} x.\Vert B\Vert)$			

Isomorphism

This encoding is an **isomorphism** between (untyped) $\lambda\Pi$-terms and HRS-terms (which are $\beta\overline{\eta}$-normal) of type $\mathtt{Term}$.

$$t_1 \rightarrow_\beta t_2 \iff \Vert t_1\Vert \rightarrow_\beta \Vert t_2\Vert.$$

If we take $\Vert(l \hookrightarrow r)\Vert = \Vert l\Vert \hookrightarrow \Vert r\Vert$, then

$$t_1 \rightarrow_\Gamma t_2 \iff \Vert t_1\Vert \rightarrow_{\Vert\Gamma\Vert} \Vert t_2\Vert.$$

Encoding Rewrite Rules

Second Encoding of Terms

$$\begin{array}{lll} \|\textbf{Kind}\|^2 & := & \mathtt{Kind} \\ \dots & := & \dots \\ \|uv\|^2 & := & \mathtt{App}(\|u\|^2, \|v\|^2) \textit{ if } uv \neq x\ \vec{w} \textit{ for } x \textit{ free} \\ \|x\vec{v}\|^2 & := & x(\|\vec{v}\|^2) \textit{ if } x \textit{ free } (x \text{ of type } \mathtt{Term} \longrightarrow \dots \longrightarrow \mathtt{Term}). \end{array}$$

Second Encoding of Rewrite Rules

$$\|(l \hookrightarrow r)\|^2 = \|l\|^2 \hookrightarrow \|r\|^2$$

Conditions

- $\|l\|^2$ must be a pattern;
- all occurrences of a free variable in l and r must be applied to the same number of arguments.

Example

Rewrite Rule

$$\|\mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ (f\ x)) \hookrightarrow (\mathtt{D}\ (\lambda x : R.f\ x))\ \times\ (\lambda x : R.\mathtt{Exp}\ (f\ x))\|^2$$
$$= \mathtt{App}(\mathtt{D}, \mathtt{Lam}(R, \underline{\lambda} x.\mathtt{App}(\mathtt{Exp}, f(x)))) \hookrightarrow \mathtt{App}(\mathtt{App}(\times, \dots f(x) \dots f(x))))$$

Reduction

$$\begin{aligned}
\|\mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ x)\| &= \mathtt{App}(\mathtt{D}, \mathtt{Lam}(R, \underline{\lambda} x.\mathtt{App}(\mathtt{Exp}, x))) \\
&= \updownarrow^{\eta}_{\beta} \mathtt{App}(\mathtt{D}, \mathtt{Lam}(R, \underline{\lambda} x.\mathtt{App}(\mathtt{Exp}, (\underline{\lambda} y.y)(x)))) \\
&\to \updownarrow^{\eta}_{\beta} \mathtt{App}(\mathtt{App}(\times, \dots (\underline{\lambda} y.y)(x) \dots (\underline{\lambda} y.y)(x)))) \\
&= \mathtt{App}(\mathtt{App}(\times, \dots x \dots x))) \\
&= \|(\mathtt{D}\ (\lambda x : R.x))\ \times\ (\lambda x : R.\mathtt{Exp}\ x)\|
\end{aligned}$$

$$\begin{tikzcd}
 & \mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ ((\lambda y : R.y)\ x)) \arrow[dl, "\mathtt{D}"'] \arrow[dr, "\beta"] & \\
\dots\ \times\ (\lambda x : R.(\mathtt{Exp}\ ((\lambda y : R.y)\ x))) \arrow[dr, two heads, "\beta"] & & \mathtt{D}\ (\lambda x : R.\mathtt{Exp}\ x) \arrow[dl, red] \\
 & (\mathtt{D}\ (\lambda x : R.x))\ \times\ (\lambda x : R.(\mathtt{Exp}\ x)) &
\end{tikzcd}$$

(Sort)

$$\frac{}{\Gamma; \Delta; \Sigma; \mathcal{C} \Vdash_s \textbf{Type} \Rightarrow (\Delta, \textbf{Kind}, \mathcal{C})}$$

(Constant)

$$\frac{(f : A) \in \Gamma}{\Gamma; \Delta; \Sigma; \mathcal{C} \Vdash_s f \Rightarrow (\Delta, A, \mathcal{C})}$$

(ΣΔ-Variable)

$$\frac{(x : A) \in \Sigma \cup \Delta}{\Gamma; \Delta; \Sigma; \mathcal{C} \Vdash_s x \Rightarrow (\Delta, A, \mathcal{C})}$$

(S-Application)

$$\frac{\begin{array}{c}\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_s u \Rightarrow (\Delta_2, T_2, \mathcal{C}_2) \\ \Gamma; \Delta_2; \Sigma; \mathcal{C}_2 \Vdash_c v \Leftarrow A \mid (\Delta_3, \mathcal{C}_3) \qquad T_2 \rightarrow^*_{\beta\Gamma} \Pi x : A.B\end{array}}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_s u\ v \Rightarrow (\Delta_3, B[x/v], \mathcal{C}_3)}$$

(S-Abstraction)

$$\frac{\begin{array}{c}\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c A \Leftarrow \textbf{Type} \mid (\Delta_2, \mathcal{C}_2) \\ \Gamma; \Delta_2; \Sigma(x : A); \mathcal{C}_2 \Vdash_s u \Rightarrow (\Delta_3, B, \mathcal{C}_3)\end{array}}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_s \lambda x : A.u \Rightarrow (\Delta_3, \Pi x : A.B, \mathcal{C}_3)}$$

(Product)

$$\frac{\begin{array}{c}\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c A \Leftarrow \textbf{Type} \mid (\Delta_2, \mathcal{C}_2) \\ \Gamma; \Delta_2; \Sigma(x : A); \mathcal{C}_2 \Vdash_c B \Leftarrow s \mid (\Delta_3, \mathcal{C}_3)\end{array}}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_s \Pi x : A.B \Rightarrow (\Delta_3, s, \mathcal{C}_3)}$$

(Free Variable)

$$\frac{FV(A) \cap dom(\Sigma) = \emptyset \qquad x \notin dom(\Delta_1) \cup dom(\Sigma)}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c x \Leftarrow A \mid (\Delta_1(x : A), \mathcal{C}_1)}$$

(C-Abstraction)

$$\frac{\begin{array}{c} T \rightarrow^*_{\beta\Gamma} \Pi x : A_2.B \\ \Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c A_1 \Leftarrow \textbf{Type} \mid (\Delta_2, \mathcal{C}_2) \\ \Gamma; \Delta_2; \Sigma(x : A_1); \mathcal{C}_2 \Vdash_c u \Leftarrow B \mid (\Delta_3, \mathcal{C}_3) \end{array}}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c \lambda x : A_1.u \Leftarrow T \mid (\Delta_3, \mathcal{C}_3 \cup \{(A_1, A_2)\})}$$

(C-Application)

$$\frac{(x : A) \in \Sigma \qquad \Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c u \Leftarrow \Pi x : A.B \mid (\Delta_2, \mathcal{C}_2)}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c u\ x \Leftarrow B \mid (\Delta_2, \mathcal{C}_2)}$$

(Inversion)

$$\frac{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_s u \Rightarrow (\Delta_2, A_2, \mathcal{C}_2)}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c u \Leftarrow A_1 \mid (\Delta_2, \mathcal{C}_2 \cup \{(A_1, A_2)\})}$$

(App-No-Check)

$$\frac{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_s v \Rightarrow (\Delta_2, A, \mathcal{C}_2)}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c u\ v \Leftarrow B \mid (\Delta_2, \mathcal{C}_2)}$$

(No-Check)

$$\frac{}{\Gamma; \Delta_1; \Sigma; \mathcal{C}_1 \Vdash_c u \Leftarrow T \mid (\Delta_1, \mathcal{C}_1)}$$

Colors

The stripping function $\|.\|$, from types and kinds to weak types, is defined as follows:

$$
\begin{array}{lcl}
\|\mathbf{Kind}\| & = & \mathbf{Kind} \\
\|\mathbf{Type}\| & = & \mathbf{Type} \\
\|C\| & = & \mathbf{Color}(C) \\
\|At\| & = & \|A\| \\
\|\lambda x : A.B\| & = & \|B\| \\
\|\Pi x : A.B\| & = & \|A\| \to \|B\|
\end{array}
$$

(Sort)

$$\frac{}{\Gamma; \Delta \vdash_w \textbf{Type} : \textbf{Kind}}$$

(Variable)

$$\frac{(x : A) \in \Delta}{\Gamma; \Delta \vdash_w x : \|A\|}$$

(Constant)

$$\frac{(c : A) \in \Gamma}{\Gamma; \Delta \vdash_w c : \|A\|}$$

(Application)

$$\frac{\Gamma; \Delta \vdash_w t : A \to B \qquad \Gamma; \Delta \vdash_w u : A}{\Gamma; \Delta \vdash_w tu : B}$$

(Abstraction)

$$\frac{\Gamma; \Delta \vdash_w A : \textbf{Type} \qquad \Gamma; \Delta(x : A) \vdash_w t : B \qquad B \neq \textbf{Kind}}{\Gamma; \Delta \vdash_w \lambda x : A.t : \|A\| \to B}$$

(Product)

$$\frac{\Gamma; \Delta \vdash_w A : \textbf{Type} \qquad \Gamma; \Delta(x : A) \vdash_w B : s}{\Gamma; \Delta \vdash_w \Pi x : A.B : s}$$

(Conversion)

$$\frac{\Gamma; \Delta \vdash_w t : A \qquad \Gamma; \Delta \vdash_w B : s \qquad A \equiv^w_\Gamma \|B\|}{\Gamma; \Delta \vdash_w t : \|B\|}$$

Non-left-linear Rewrite Rules

$\mathtt{minus}\ n\ n \hookrightarrow 0.$
$\mathtt{minus}\ (\mathtt{S}\ n)\ n \hookrightarrow \mathtt{S}\ 0.$

Let Y be Turing's fixpoint combinator.
$\mathtt{minus}\ (Y\mathtt{S})\ (Y\mathtt{S}) \rightarrow_{\Gamma} 0.$
$\mathtt{minus}\ (Y\mathtt{S})\ (Y\mathtt{S}) \rightarrow_{\beta}^{*} \mathtt{minus}\ (\mathtt{S}\ (Y\mathtt{S}))\ (Y\mathtt{S}) \rightarrow_{\Gamma} \mathtt{S}\ 0.$