On the use of confluence in type theory modulo rewriting

Frédéric Blanqui

DeducL-eam

30 June 2020
Outline

Rewriting in proof assistants

Dedukti and the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/R$)

Properties of the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/R$)

Conclusion
Rewriting in proof assistants

increasing interest in using rewriting in proof assistants

- Dedukti
Rewriting in proof assistants

increasing interest in using rewriting in proof assistants

- Dedukti
- Agda
Rewriting in proof assistants

increasing interest in using rewriting in proof assistants

- Dedukti
- Agda
- Coq?
  - *How to Tame your Rewrite Rules* (TYPES 2019)
  - *Modular Confluence for Rewrite Rules in MetaCoq* (TYPES 2020)
Rewriting in proof assistants

increasing interest in using rewriting in proof assistants

- Dedukti
- Agda
- Coq?
  - *How to Tame your Rewrite Rules* (TYPES 2019)
  - *Modular Confluence for Rewrite Rules in MetaCoq* (TYPES 2020)

**common point:** all these systems use dependent types
Rewriting in Agda

https://agda.readthedocs.io/

Rewriting

Rewrite rules allow you to extend Agda’s evaluation relation with new computation rules.

Note

This page is about the `--rewriting` option and the associated `REWRITE` builtin. You might be looking for the documentation on the `rewrite` construct instead.

Rewrite rules by example

To enable rewrite rules, you should run Agda with the flag `--rewriting` and import the modules

```agda
{-# OPTIONS --rewriting #-}

module language.rewriting where

open import Agda.Builtin.Equality
```
Confluence checking in Agda

Once a rewrite rule has been added, Agda automatically rewrites all instances of the left-hand side to the corresponding instance of the right-hand side during reduction. More precisely, a term (definitionally equal to) \( f p_1 \sigma \ldots p_n \sigma \) is rewritten to \( v \sigma \), where \( \sigma \) is any substitution on the pattern variables \( x_1, \ldots, x_k \).

Since rewriting happens after normal reduction, rewrite rules are only applied to terms that would otherwise be neutral.

Confluence checking

Agda can optionally check (local) confluence of rewrite rules by enabling the flag.

Advanced usage

Instead of importing \texttt{Agda.Builtin.Equality.Rewrite}, a different type may be chosen as the rewrite relation by registering it as the \texttt{REWRITE} builtin. For example, using the pragma \texttt{{-# BUILTIN REWRITE _ _ #-}} registers the type \texttt{_ _} as the rewrite relation. To qualify as the rewrite relation, the type must take at least two arguments, and the final two arguments should be visible.
Dedukti:
- purely functional “programming” language (λ-calculus)
Dedukti:

- purely functional “programming” language ($\lambda$-calculus)
- with dependent types (types can take values as arguments)
Dedukti:
- purely functional “programming” language (\(\lambda\)-calculus)
- with dependent types (types can take values as arguments)
- functions and types \(!\) can be defined (by rewrite rules \(\mathcal{R}\))
Dedukti:
- purely functional “programming” language (\(\lambda\)-calculus)
- with dependent types (types can take values as arguments)
- functions and types can be defined (by rewrite rules \(\mathcal{R}\))
- implements the \(\lambda\Pi\)-calculus modulo rewriting (\(\lambda\Pi/\mathcal{R}\))
Dedukti

Dedukti:
- purely functional “programming” language (λ-calculus)
- with dependent types (types can take values as arguments)
- functions and types \( \vartriangle \) can be defined (by rewrite rules \( R \))
- implements the \( \lambda \Pi \)-calculus modulo rewriting (\( \lambda \Pi / R \))

Example:

```plaintext
symbol N:TYPE symbol 0:N symbol s:N → N
symbol F:N → TYPE
rule F 0 ↦ N
with F (s $x) ↦ N → F $x
assert F 2 ≡ N → N → N // convertible expressions
```
Applications of Dedukti

- logical framework for representing the theories and the proofs of many logical systems (HOL-Light, Coq, Agda, PVS, etc.)
  see Guillaume Genestier’s FSCD talk on July 3rd at 15:00 😊

- independant proof checker

- proof transformations
Dedukti v2

https://deducteam.github.io/

A Logical Framework

What Is Dedukti?
Dedukti is a logical framework based on the λΠ-calculus modulo in which many theories and logics can be expressed.

Where does the name "Dedukti" comes from?
"Dedukti" means "to deduce" in Esperanto.

How to get/install/use Dedukti?
See the GitHub repository.
See the manual for the current version (v2.5.1) of Dedukti.
There is also a tutorial.

Dedukti libraries

Manual developments
- Dklib is a library defining basic data structures.
- Sigmoid (SIGMA-calculus In Dedukti) is an encoding of the simply-typed c-calculus in Dedukti.
- Libraries A github repository that aims to contain every hand-written Dedukti files. In the short run, the two links above should be outdated.

Generated libraries
Each tarball contains a Makefile in order to check the library. You may want to modify the variables DKDEP and DKCHECK that are the paths of the
- The Holide library is the library produced by Holide on the standard library of the common format for HOL proof assistant: OpenTheory
- The Matita arithmetic library is a library of Dedukti files generated by Krajono.
- The Focalide library is a library of Dedukti files generated by Focalide.
- The Zenon Modulo Set Theory library is a library of Dedukti files generated by Zenon Modulo and dealing with the B Method set theory. (Rem
Dedukti v3 aka Lambdapi

https://github.com/Deducteam/lambdapi/

User manual for Lambdapi

Lambdapi is a proof assistant based on the λΠ-calculus modulo rewriting, mostly compatible with the proof checker Dedukti. This document provides a good starting point for anyone wishing to use or to contribute to the project.

Table of contents

- What is Lambdapi?
- Installation
- Getting started
- Command line options
- User interfaces
  - Emacs
  - VSCode
  - Vim
  - Atom
- Module system
- Syntax of terms
- Commands
- Tactics
- Compatibility with Dedukti
- Bibliographic references
- Including Lambdapi code into a LaTeX document
Rewrite rules and matching in Dedukti

- LHS can be overlapping:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0 + \ y) → (y)</td>
<td></td>
</tr>
<tr>
<td>with (s \ (\ x + \ y) → s ((x + \ y))</td>
<td></td>
</tr>
<tr>
<td>with (x + 0) → (x)</td>
<td></td>
</tr>
<tr>
<td>with (x + s \ y) → s ((x + \ y))</td>
<td></td>
</tr>
</tbody>
</table>

See Gabriel Hondet's FSCD talk on July 3rd at 15:30
Rewrite rules and matching in Dedukti

- matching on defined symbols:

\[
\text{rule } (x + y) + z \mapsto x + (y + z)
\]
Rewrite rules and matching in Dedukti

- LHS can be non-linear:

```latex
\text{rule} \quad \begin{array}{l}
x + (-x) \quad \rightarrow \quad 0
\end{array}
```
Rewrite rules and matching in Dedukti

- higher-order pattern-matching:

```plaintext
rule diff(\lambda x.\sin f[x]) \leftrightarrow diff(\lambda x.f[x])*\cos
rule lam(\lambda x.app f[] x) \leftrightarrow f[] // \eta\text{-rule}
```
Rewrite rules and matching in Dedukti

- LHS can be overlapping:

\[
\begin{align*}
\text{rule} & \quad 0 + $y \leftrightarrow $y \\
\text{with} & \quad s \; $x + $y \leftrightarrow s \; ($x + $y) \\
\text{with} & \quad $x + 0 \leftrightarrow $x \\
\text{with} & \quad $x + s \; $y \leftrightarrow s \; ($x + $y)
\end{align*}
\]

- matching on defined symbols:

\[
\begin{align*}
\text{rule} & \quad ($x + $y) + $z \leftrightarrow $x + ($y + $z)
\end{align*}
\]

- LHS can be non-linear:

\[
\begin{align*}
\text{rule} & \quad $x + (- \; $x) \leftrightarrow 0
\end{align*}
\]

- higher-order pattern-matching:

\[
\begin{align*}
\text{rule} & \quad \text{diff}(\lambda x.\sin \; $f[x]) \leftrightarrow \text{diff}(\lambda x.$f[x]) * \cos \\
\text{rule} & \quad \text{lam}(\lambda x.\text{app} \; $f[] \; x) \leftrightarrow $f[] \quad \text{// } \eta - \text{rule}
\end{align*}
\]

See Gabriel Hondet’s FSCD talk on July 3rd at 15:30 😊
Confluence checking in Dedukti

Dedukti rewrite systems can be exported to the HRS format of the confluence competition (CoCo) but:

- the HRS format does not accept dependent types
- the TRS format does not accept $\lambda$-abstractions

---

Confluence checking

Lambdapi provides an option `--confluence CMD` to check the confluence of the rewriting system by calling an external prover with the command `CMD`. The given command receives HRS formatted text on its standard input, and it is expected to output on the first line of its standard output either YES, NO or MAYBE.

As an example, `echo MAYBE` is the simplest possible (valid) confluence-check that one may use.

For now, only the `CSI^h` confluence checker has been tested with Lambdapi. It can be called using the flag `--confluence "path/to/csiho.sh --ext trs --stdin"`.

To inspect the `.trs` file generated by Lambdapi, one may use the following dummy command: `--confluence "cat > output.trs; echo MAYBE"`.

Termination checking

Lambdapi provides an option `--termination CMD` to check the termination of the rewriting system by calling an external prover with the command `CMD`. The given command receives XTC formatted text on its standard input, and it is expected to output on the first line of its standard output either YES, NO or MAYBE.

As for confluence, `echo MAYBE` is the simplest possible (valid) command for checking termination.

To the best of our knowledge, the only termination checker that is compatible with all the features of Lambdapi is `SizeChangeTool`. It can be called using the flag `--termination "path/to/sct.native --no-color --stdin=xml"

If no type-level rewriting is used `Wanda` can also be used. However, it does not directly accept input on its standard input, so it is tricky to have Lambdapi call it directly. Alternatively, one can first generate a `.wda` file as described below.
Confluence checking in Dedukti

Dedukti rewrite systems can be exported to the HRS format of the confluence competition (CoCo) but:
– the HRS format does not accept dependent types

---

**Confluence checking**

Lambdapi provides an option `--confluence CMD` to check the confluence of the rewriting system by calling an external prover with the command `CMD`. The given command receives HRS formatted text on its standard input, and it is expected to output on the first line of its standard output either YES, NO or MAYBE.

As an example, `echo MAYBE` is the simplest possible (valid) confluence-check that one may use.

For now, only the `CSI:no` confluence checker has been tested with Lambdapi. It can be called using the flag `--confluence "path/to/csi:no.sh --ext trs --stdin"`.

To inspect the `.trs` file generated by Lambdapi, one may use the following dummy command: `--confluence "cat > output.trs; echo MAYBE"`.

---

**Termination checking**

Lambdapi provides an option `--termination CMD` to check the termination of the rewriting system by calling an external prover with the command `CMD`. The given command receives XTC formatted text on its standard input, and it is expected to output on the first line of its standard output either YES, NO or MAYBE.

As for confluence, `echo MAYBE` is the simplest possible (valid) command for checking termination.

To the best of our knowledge, the only termination checker that is compatible with all the features of Lambdapi is `SizeChangeTool`. It can be called using the flag `--termination "path/to/scnt.native --no-color --stdin=xml"`.

If no type-level rewriting is used `wanda` can also be used. However, it does not directly accept input on its standard input, so it is tricky to have Lambdapi call it directly. Alternatively, one can first generate a `.trn` file as described below.
Confluence checking in Dedukti

Dedukti rewrite systems can be exported to the HRS format of the confluence competition (CoCo) but:

– the HRS format does not accept dependent types
– the TRS format does not accept λ-abstractions
Outline

Rewriting in proof assistants

Dedukti and the λΠ-calculus modulo rewriting (λΠ/Δ)

Properties of the λΠ-calculus modulo rewriting (λΠ/Δ)

Conclusion
\[\lambda\Pi\text{-calculus modulo a set } \mathcal{R} \text{ of rewrite rules} \ (\lambda\Pi/\mathcal{R})\]

terms/types \( t, u, A, B = \)

| \( f \) | (function/type symbol)  
| \( x \) | (variable)  
| \( \lambda x : A. t \) | (abstraction)  
| \( tu \) | (application)  

\( \lambda \Pi \)-calculus modulo a set \( \mathcal{R} \) of rewrite rules (\( \lambda \Pi / \mathcal{R} \))

terms/types \( t, u, A, B = \)

<table>
<thead>
<tr>
<th>( f )</th>
<th>(function/type symbol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>(variable)</td>
</tr>
<tr>
<td>( \lambda x : A . t )</td>
<td>(abstraction)</td>
</tr>
<tr>
<td>( tu )</td>
<td>(application)</td>
</tr>
<tr>
<td>( s \in { \text{TYPE}, \text{KIND} } )</td>
<td>(sort)</td>
</tr>
<tr>
<td>( \Pi x : A . B )</td>
<td>(dependent product, written ( A \rightarrow B ) if ( x \notin B ))</td>
</tr>
</tbody>
</table>
\[\lambda\Pi\text{-calculus modulo a set } R \text{ of rewrite rules } (\lambda\Pi/R)\]

**Terms/Types** \( t, u, A, B = \)

- \( f \) (function/type symbol)
- \( x \) (variable)
- \( \lambda x:A.t \) (abstraction)
- \( tu \) (application)
- \( s \in \{\text{TYPE, KIND}\} \) (sort)
- \( \Pi x:A.B \) (dependent product, written \( A \to B \) if \( x \notin B \))

**Typing Environments** \( \Gamma, \Delta = \)

- \( \emptyset \) (empty environment)
- \( \Gamma, x:A \) (variable declaration)
\(\lambda\Pi\)-calculus modulo a set \(\mathcal{R}\) of rewrite rules \((\lambda\Pi/\mathcal{R})\)

**Terms/Types**

\[ t, u, A, B = \]
- \(f\) (function/type symbol)
- \(x\) (variable)
- \(\lambda x:A.t\) (abstraction)
- \(tu\) (application)
- \(s \in \{\text{TYPE, KIND}\}\) (sort)
- \(\Pi x:A.B\) (dependent product, written \(A \rightarrow B\) if \(x \notin B\))

**Typing Environments**

\[ \Gamma, \Delta = \]
- \(\emptyset\) (empty environment)
- \(\Gamma, x:A\) (variable declaration)

**Rewrite Rules**

\[ \rho = \]
- \(\Delta \vdash f t_1 \ldots t_n \leftrightarrow u\) (rewrite rule)
Typing rules of $\lambda\Pi/\mathcal{R}$

\[
\begin{array}{ll}
\text{(empty)} & \frac{\emptyset \text{ valid}}{} \\
\text{(decl)} & \frac{\Gamma \text{ valid} \quad \Gamma \vdash A : s}{\Gamma, x:A \text{ valid}}
\end{array}
\]
Typing rules of $\lambda\Pi/R$

- **(fun)** $\Gamma$ valid $\frac{}{\Gamma \vdash f : A_f}$
- **(var)** $\Gamma, x : A, \Gamma' \text{ valid}$ $\frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$
- **(abs)** $\Gamma, x : A \vdash t : B$ $\Gamma \vdash \Pi x : A. B : s$ $\frac{}{\Gamma \vdash \lambda x : A. t : \Pi x : A. B}$
- **(app)** $\Gamma \vdash t : \Pi x : A. B$ $\Gamma \vdash u : A$ $\frac{}{\Gamma \vdash tu : B\{x \mapsto u\}}$

Remark: the type of a term is unique up to $\downarrow^{\beta, R}$.
Typing rules of $\lambda\Pi/R$

\[
\begin{align*}
\text{(sort)} & \quad \Gamma \text{ valid} \quad \frac{}{\Gamma \vdash \text{TYPE} : \text{KIND}} \\
\text{(prod)} & \quad \Gamma \vdash A : \text{TYPE} \quad \Gamma, x:A \vdash B : s \quad \frac{}{\Gamma \vdash \prod x:A. B : s} \\
\text{(conv)} & \quad \Gamma \vdash t : A \quad A \downarrow_{\beta R} B \quad \Gamma \vdash B : s \quad \frac{}{\Gamma \vdash t : B} \\
& \text{if } A \text{ and } B \text{ have a common reduct wrt the } \beta \text{-rule of } \lambda\text{-calculus and the user-defined rules } R
\end{align*}
\]
Typing rules of $\lambda\Pi/\mathcal{R}$

\[
\begin{align*}
\text{(empty)} & \quad \frac{}{\emptyset \text{ valid}} & \quad \text{(decl)} & \quad \frac{\Gamma \text{ valid} \quad \Gamma \vdash A : s}{\Gamma, x:A \text{ valid}} \\
\text{(fun)} & \quad \frac{\Gamma \text{ valid}}{\Gamma \vdash f : A_f} & \quad \text{(var)} & \quad \frac{\Gamma, x:A, \Gamma' \text{ valid}}{\Gamma, x:A, \Gamma' \vdash x : A} \\
\text{(abs)} & \quad \frac{\Gamma, x:A \vdash t : B \quad \Gamma \vdash \Pi x:A.B : s}{\Gamma \vdash \lambda x:A.t : \Pi x:A.B} \\
\text{(sort)} & \quad \frac{\Gamma \text{ valid}}{\Gamma \vdash \text{TYPE} : \text{KIND}} & \quad \text{(app)} & \quad \frac{\Gamma \vdash t : \Pi x:A.B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B\{x \mapsto u\}} \\
\text{(conv)} & \quad \frac{\Gamma \vdash t : A \quad A \downarrow_{\beta\mathcal{R}} B \quad \Gamma \vdash B : s}{\Gamma \vdash t : B}
\end{align*}
\]

$A \downarrow_{\beta\mathcal{R}} B$ if $A$ and $B$ have a common reduct wrt the $\beta$-rule of $\lambda$-calculus and the user-defined rules $\mathcal{R}$

Remark: the type of a term is unique up to $\downarrow^*_\beta\mathcal{R}$
Outline

Rewriting in proof assistants

Dedukti and the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/$R$)$

Properties of the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/$R$)$

Conclusion
Some important properties

<table>
<thead>
<tr>
<th>TC</th>
<th>decidability of the typing relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>termination of $\hookrightarrow_{\beta R}$ from typable terms</td>
</tr>
<tr>
<td>$\text{SR}_{\beta}$</td>
<td>preservation of typing by $\hookrightarrow_{\beta}$</td>
</tr>
<tr>
<td>$\text{SR}_{R}$</td>
<td>preservation of typing by $\hookrightarrow_{R}$</td>
</tr>
<tr>
<td>LCR</td>
<td>local confluence of $\hookrightarrow_{\beta R}$ on arbitrary terms</td>
</tr>
<tr>
<td>CR</td>
<td>confluence of $\hookrightarrow_{\beta R}$ from typable terms</td>
</tr>
</tbody>
</table>

Question: what are the dependencies between those properties?
Decidability of type-checking (TC)
Decidability of type-checking (TC)

mix type-inference \( \uparrow \) and type-checking \( \downarrow \)

\[
\frac{\frac{\Gamma \vdash t \uparrow A \quad A \downarrow^{\ast} B}{\Gamma \vdash t \downarrow B}}{(\text{conv})}
\]
Decidability of type-checking (TC)
mix type-inference ↑ and type-checking ↓

(conv) \[
\Gamma \vdash t \uparrow A \quad A \downarrow^{*}_\beta R B \\
\frac{}{\Gamma \vdash t \downarrow B}
\]

(sort) \[
\Gamma \text{ valid} \\
\frac{}{\Gamma \vdash \text{TYPE} \uparrow \text{KIND}}
\]

(prod) \[
\Gamma \vdash A \downarrow \text{TYPE} \quad \Gamma, x:A \vdash B \uparrow s \\
\frac{}{\Gamma \vdash \Pi x:A.B \uparrow s}
\]

(fun) \[
\Gamma \text{ valid} \\
\frac{}{\Gamma \vdash f \uparrow A_f}
\]

(var) \[
\Gamma, x:A, \Gamma' \text{ valid} \\
\frac{}{\Gamma, x:A, \Gamma' \vdash x \uparrow A}
\]

(abs) \[
\Gamma \vdash A \downarrow \text{TYPE} \quad \Gamma, x:A \vdash t \uparrow B \\
B \neq \text{KIND} \\
\frac{}{\Gamma \vdash \lambda x:A.t \uparrow \Pi x:A.B}
\]

(app) \[
\Gamma \vdash t \uparrow C \\
C \rightarrow^{*}_\beta R \Pi x:A.B \\
\Gamma \vdash u \downarrow A \\
\frac{}{\Gamma \vdash tu \uparrow B\{x \mapsto u\}}
\]
Decidability of type-checking (TC)

mix type-inference ↑ and type-checking ↓

\[
\Gamma \vdash t \uparrow C 
\quad C \rightarrow^*_{\beta \eta} \Pi x : A.B 
\quad \Gamma \vdash u \downarrow A \\
\Gamma \vdash tu \uparrow B \{x \mapsto u\}
\]

Conclusion: for TC we use SN, SR, LCR
Decidability of type-checking (TC)

mix type-inference $\uparrow$ and type-checking $\downarrow$

(1) $\Gamma \vdash t \uparrow A \quad A \downarrow^{*}_R B$

(2) $\Gamma \vdash t \downarrow B$

(s) $\Gamma$ valid

(3) $\Gamma \vdash \textsc{type} \uparrow \textsc{knd}$

(p) $\Gamma \vdash A \downarrow \textsc{type} \quad \Gamma, x:A \vdash B \uparrow s$

(f) $\Gamma$ valid

(4) $\Gamma \vdash f \uparrow A_f$

(v) $\Gamma, x:A, \Gamma' \vdash x \uparrow A$

(a) $\Gamma \vdash A \downarrow \textsc{type} \quad \Gamma, x:A \vdash t \uparrow B \quad B \neq \textsc{knd}$

(6) $\Gamma \vdash \lambda x:A.t \uparrow \Pi x:A.B$

(a) $\Gamma \vdash t \uparrow C \quad C \xrightarrow{\neq \beta_R} \Pi x:A.B \quad \Gamma \vdash u \downarrow A$

(7) $\Gamma \vdash tu \uparrow B \{x \mapsto u\}$

Conclusion: for TC we use SN, SR, LCR
Termination (SN)
Step 1 define a function

\[
\begin{align*}
\left\llbracket \cdot \right\rrbracket : \text{SN types} & \rightarrow \text{subsets of SN} \\
A & \mapsto \left[ A \right]
\end{align*}
\]

invariant by reduction: \( A \xrightarrow{\beta R} A' \Rightarrow \left[ A \right] = \left[ A' \right] \)

+ other conditions
Termination (SN) [B. Genestier Hermant, FSCD 2019]

**Step 1** define a function

\[
\begin{align*}
\llbracket \rrbracket : \text{SN types} & \rightarrow \text{subsets of SN} \\
A & \mapsto \llbracket A \rrbracket
\end{align*}
\]

invariant by reduction: \( A \xrightarrow{\beta R} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket \)

+ other conditions

**Step 2** prove \( \Gamma \vdash t : A \Rightarrow t \in \llbracket A \rrbracket \)
Goal: define a function

\[
\begin{align*}
\left[ \right] : & \text{ SN types } \rightarrow \text{ subsets of SN} \\
A & \mapsto [A]
\end{align*}
\]

invariant by reduction: \( A \xrightarrow{\beta R} A' \Rightarrow [A] = [A'] \)

+ other conditions
Goal: define a function

\[
\begin{array}{ccl}
\llbracket \cdot \rrbracket : & \text{SN types} & \rightarrow \text{subsets of SN} \\
A & \mapsto & \llbracket A \rrbracket
\end{array}
\]

Invariant by reduction: \( A \xrightarrow{\beta R} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket \)

+ other conditions

Solution: \( \llbracket A \rrbracket = \begin{cases} 
\ldots & \text{if } A \text{ is in normal form (nf)} \\
\llbracket_{nf}(A) \rrbracket & \text{otherwise}
\end{cases} \)

using SR and LCR
Termination (SN) – Step 1

Goal: define a function

\[
\begin{align*}
[ \ ] : \text{SN types} & \rightarrow \text{subsets of SN} \\
A & \mapsto [A]
\end{align*}
\]

invariant by reduction: \( A \leftrightarrow_{\beta R} A' \Rightarrow [A] = [A'] \)

+ other conditions

Solution: \( [A] = \begin{cases} 
\ldots & \text{if } A \text{ is in normal form (nf)} \\
[nf(A)] & \text{otherwise}
\end{cases} \)

using SR and LCR

⚠️ previous works assumed no critical pairs on types
Goal: prove $\Gamma \vdash t : A \implies t \in [A]$
Goal: prove $\Gamma \vdash t : A \Rightarrow t \in \llbracket A \rrbracket$

Solution: for a rule $\Delta \vdash f \mid l \leftrightarrow r$ with $f : \Pi x : A. B$, we assume

$$\Delta \vdash r : B\{x \mapsto l\}$$

in any sub-system of $\lambda\Pi/\mathcal{R} +$ other conditions
Termination (SN) – Step 2

Goal: prove $\Gamma \vdash t : A \Rightarrow t \in \llbracket A \rrbracket$

Solution: for a rule $\Delta \vdash f \leftarrow l \mapsto r$ with $f : \Pi x : A. B$, we assume

$$\Delta \vdash r : B\{x \mapsto l\}$$

in any sub-system of $\lambda\Pi/\mathcal{R} +$ other conditions

Conclusion: for SN we use SR, LCR, TC
Subject-reduction (SR)
Subject-reduction (SR)
aka preservation of typing by reduction

\[
\text{for all } \Gamma, \ t, \ u, \ A, \ \text{if } \Gamma \vdash t : A \ \text{and} \ t \xrightarrow{\beta R} u, \ \text{then} \ \Gamma \vdash u : A
\]

applications:

– in programming languages: ensures memory safety
– in logical systems: correctness of cut elimination
Subject-reduction for $\beta$-reduction ($\text{SR}_\beta$)
Subject-reduction (SR) - Case of $\rightarrow_{\beta}$

Goal: $\Gamma \vdash (\lambda x : A.t)u : C \Rightarrow \Gamma \vdash t\{x \mapsto u\} : C$ ?
Subject-reduction (SR) - Case of $\rightarrow^\beta$

**Goal:** $\Gamma \vdash (\lambda x : A.t)u : C \Rightarrow \Gamma \vdash t\{x \mapsto u\} : C$ ?

$$\Gamma \vdash (\lambda x : A.t)u : C$$
Subject-reduction (SR) - Case of $\rightarrow_\beta$

Goal: \[ \Gamma \vdash (\lambda x : A. t)u : C \quad \Rightarrow \quad \Gamma \vdash t\{x \mapsto u\} : C \quad ? \]

\[
\begin{align*}
\Gamma \vdash (\lambda x : A.t)u : B'\{x \mapsto u\} & \quad \quad B'\{x \mapsto u\} \downarrow^*_\beta C \quad \Gamma \vdash C : s'' \\
\hline
\Gamma \vdash (\lambda x : A.t)u : C
\end{align*}
\]
**Subject-reduction (SR) - Case of $\rightarrow_\beta$**

**Goal:**
$$\Gamma \vdash (\lambda x : A.t)u : C \quad \Rightarrow \quad \Gamma \vdash t\{x \mapsto u\} : C$$

---

$$\frac{\Gamma \vdash \lambda x : A.t : \Pi x : A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x : A.t)u : B'\{x \mapsto u\}}$$ (\text{app})

$$\frac{B'\{x \mapsto u\} \downarrow^*_{\beta \mathcal{R}} C \quad \Gamma \vdash C : s''}{\Gamma \vdash (\lambda x : A.t)u : C}$$ (\text{conv})
Subject-reduction (SR) - Case of $\rightarrow^\beta$

Goal: \[ \Gamma \vdash (\lambda x : A. t) u : C \quad \Rightarrow \quad \Gamma \vdash t\{x \mapsto u\} : C \quad ? \]

\[ \Gamma \vdash \lambda x : A. t : \Pi x : A. B \quad \Pi x : A. B \downarrow^*_R \Pi x : A'. B' \quad \Gamma \vdash \Pi x : A'. B' : s' \quad \text{(conv)} \]

\[ \Gamma \vdash \lambda x : A. t : \Pi x : A'. B' \]

\[ \Gamma \vdash \lambda x : A. t : \Pi x : A'. B' \quad \Gamma \vdash u : A' \quad \text{(app)} \]

\[ \Gamma \vdash (\lambda x : A. t) u : B'\{x \mapsto u\} \quad B'\{x \mapsto u\} \downarrow^*_R C \quad \Gamma \vdash C : s'' \quad \text{(conv)} \]

\[ \Gamma \vdash (\lambda x : A. t) u : C \]
Subject-reduction (SR) - Case of $\rightarrow^\beta$

Goal: $\Gamma \vdash (\lambda x : A.t)u : C \quad \Rightarrow \quad \Gamma \vdash t\{x \mapsto u\} : C$  

$\Gamma, x : A \vdash t : B \quad \Pi x : A. B : s$

$\Gamma \vdash \lambda x : A.t : \Pi x : A. B$

$\Pi x : A. B \downarrow_{\beta_R}^* \Pi x : A'. B' \quad \Gamma \vdash \Pi x : A'. B' : s'$

$\Gamma \vdash \lambda x : A.t : \Pi x : A'. B'$

$\Gamma \vdash \lambda x : A.t : \Pi x : A'. B'$  

$\Gamma \vdash u : A'$

$\Gamma \vdash (\lambda x : A.t)u : B'\{x \mapsto u\}$

$B'\{x \mapsto u\} \downarrow_{\beta_R}^* C \quad \Gamma \vdash C : s''$

$\Gamma \vdash (\lambda x : A.t)u : C$

Problem: $A' \downarrow_{\beta_R}^* \Gamma \vdash A$ and $B \downarrow_{\beta_R}^* C$

Solution: $\Pi x : A. B \downarrow_{\beta_R}^* \Pi x : A'. B'$

Conclusion: for SR $\beta$ we use CR
Subject-reduction (SR) - Case of $\rightarrow^\beta$

**Goal:** \( \Gamma \vdash (\lambda x : A. t) u : C \quad \Rightarrow \quad \Gamma \vdash t \{ x \mapsto u \} : C \) ?

\[
\begin{align*}
\Gamma, x : A &\vdash t : B & \Pi x : A.B : s \\
\hline
\Gamma &\vdash \lambda x : A. t : \Pi x : A.B & (\text{abs}) \\
\Pi x : A.B &\downarrow^*_\mathcal{R} \Pi x : A'.B' & \Gamma \vdash \Pi x : A'.B' : s' \\
\hline
\Gamma &\vdash \lambda x : A. t : \Pi x : A'.B' & (\text{conv}) \\
\Gamma &\vdash \lambda x : A. t : \Pi x : A'.B' & \Gamma \vdash u : A' \\
\hline
\Gamma &\vdash (\lambda x : A. t) u : B' \{ x \mapsto u \} & (\text{app}) \\
B' \{ x \mapsto u \} &\downarrow^*_\mathcal{R} C & \Gamma \vdash C : s'' & (\text{conv}) \\
\hline
\Gamma &\vdash (\lambda x : A. t) u : C \\
\end{align*}
\]

**Problem:** \( A' \downarrow^*_\mathcal{R} A \) and \( B \{ x \mapsto u \} \downarrow^*_\mathcal{R} C \) ?
Subject-reduction (SR) - Case of $\rightarrow_\beta$

**Goal:**

$$\Gamma \vdash (\lambda x : A. t)u : C \quad \Rightarrow \quad \Gamma \vdash t\{x \mapsto u\} : C$$

$$\Gamma, x : A \vdash t : B \quad \Pi x : A. B : s$$

$$\frac{}{\Gamma \vdash \lambda x : A. t : \Pi x : A. B} \quad \text{(abs)}$$

$$\Pi x : A. B \downarrow_{\beta_R}^* \Pi x : A'. B' \quad \Gamma \vdash \Pi x : A'. B' : s'$$

$$\frac{}{\Gamma \vdash \lambda x : A. t : \Pi x : A'. B'} \quad \text{(conv)}$$

$$\Gamma \vdash \lambda x : A. t : \Pi x : A'. B'$$

$$\vdash$$

$$\Gamma \vdash \lambda x : A. t : \Pi x : A'. B' \quad \Gamma \vdash u : A'$$

$$\frac{}{\Gamma \vdash (\lambda x : A. t)u : B'\{x \mapsto u\}} \quad \text{(app)}$$

$$B'\{x \mapsto u\} \downarrow_{\beta_R}^* C \quad \Gamma \vdash C : s''$$

$$\frac{}{\Gamma \vdash (\lambda x : A. t)u : C} \quad \text{(conv)}$$

**Problem:**

$$A' \downarrow_{\beta_R}^* A \text{ and } B\{x \mapsto u\} \downarrow_{\beta_R}^* C$$

**Solution:**

$$\Pi x : A. B \downarrow_{\beta_R}^* \Pi x : A'. B' \quad \text{CR} \quad A \downarrow_{\beta_R}^* A' \land B \downarrow_{\beta_R}^* B'$$
Subject-reduction (SR) - Case of $\rightarrow_\beta$

**Goal:** \( \Gamma \vdash (\lambda x : A.t)u : C \quad \Rightarrow \quad \Gamma \vdash t\{x \mapsto u\} : C \quad ? \)

\[ \Gamma, x:A \vdash t : B \quad \Pi x:A.B : s \]

\[ \frac{}{\Gamma \vdash \lambda x:A.t : \Pi x:A.B} \quad (\text{abs}) \]

\[ \Pi x:A.B \downarrow^*_\mathcal{R} \quad \Pi x:A'.B' \quad \Gamma \vdash \Pi x:A'.B' : s' \]

\[ \frac{}{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B'} \quad (\text{conv}) \]

\[ \Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A' \]

\[ \frac{}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \quad (\text{app}) \]

\[ B'\{x \mapsto u\} \downarrow^*_\mathcal{R} \quad C \quad \Gamma \vdash C : s'' \quad \text{(conv)} \]

\[ \frac{}{\Gamma \vdash (\lambda x:A.t)u : C} \]

**Problem:** \( A' \downarrow^*_\mathcal{R} \quad A \quad \text{and} \quad B\{x \mapsto u\} \downarrow^*_\mathcal{R} \quad C \quad ? \)

**Solution:** \( \Pi x:A.B \downarrow^*_\mathcal{R} \quad \Pi x:A'.B' \quad \text{CR} \quad A \downarrow^*_\mathcal{R} \quad A' \quad \land \quad B \downarrow^*_\mathcal{R} \quad B' \)

**Conclusion:** for SR$_\beta$ we use CR
Subject-reduction for rewriting ($\text{SR}_R$)
Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

**Goal:** $\forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C$
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

**Goal:** \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \)

see my FSCD talk on July 3rd at 14:30 ! 😊
Example: tail function on vectors

\[\text{symbol } V : \mathbb{N} \rightarrow \text{TYPE}\]
\[\text{symbol } \text{nil} : V_0\]
\[\text{symbol } \text{cons} : A \rightarrow \Pi n : \mathbb{N}, V_n \rightarrow V(sn)\]
\[\text{symbol } \text{tail} : \Pi n : \mathbb{N}, V(sn) \rightarrow V_n\]

\[\text{tail } n \left(\text{cons } x \ p \ v\right) \mapsto v\]
Example: tail function on vectors

symbol \( V : \mathbb{N} \rightarrow \text{TYPE} \)

symbol \( \text{nil} : V_0 \)

symbol \( \text{cons} : A \rightarrow \prod n : \mathbb{N}, V_n \rightarrow V(sn) \)

symbol \( \text{tail} : \prod n : \mathbb{N}, V(sn) \rightarrow V_n \)

\[
\text{tail } n \ (\text{cons} \ x \ p \ v) \quad \rightarrow \quad v
\]

\[
\text{tail } n \ (\text{cons} \ x \ p \ v) \quad \rightarrow \quad V_n
\]
Example: tail function on vectors

symbol $V : N \to \text{TYPE}$
symbol $\text{nil} : V_0$
symbol $\text{cons} : A \to \prod n : N, V_n \to V(sn)$

symbol $\text{tail} : \prod n : N, V(sn) \to V_n$
Subject-reduction (SR) - Case of a rule \( l \leftrightarrow r \)

**Goal:** \( \forall \Gamma, \sigma, C, \Gamma \vdash l\sigma : C \Rightarrow \Gamma \vdash r\sigma : C \)
Subject-reduction (SR) - Case of a rule \( l \rightarrow r \)

Goal: \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \)

Solution:

Step 1: let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)
Subject-reduction (SR) - Case of a rule \( l \leftrightarrow r \)

**Goal:** \( \forall \Gamma, \sigma, C, \Gamma \vdash l\sigma : C \Rightarrow \Gamma \vdash r\sigma : C \) ?

**Solution:**

**Step 1:** let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

\[
\Delta = \hat{n} : \text{TYPE}, n : \hat{n}, \hat{x} : \text{TYPE}, x : \hat{x}, \ldots
\]
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

**Goal:** \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \implies \Gamma \vdash r\sigma : C \) ?

**Solution:**

**Step 1:** let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

\[ \Delta = \hat{n} : \text{TYPE}, n : \hat{n}, \hat{x} : \text{TYPE}, x : \hat{x}, \ldots \]

**Step 2:** compute the equations on \( \hat{x}_i, x_i, X \) for having \( \Delta \vdash l : X \)
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

**Goal:** \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \)

**Solution:**

**Step 1:** let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

\[
\Delta = \hat{n} : \text{TYPE}, n : \hat{n}, \hat{x} : \text{TYPE}, x : \hat{x}, \ldots
\]

**Step 2:** compute the equations on \( \hat{x}_i, x_i, X \) for having \( \Delta \vdash l : X \)

\[
\begin{align*}
tail & (\text{cons}\ x\ p\ v) \\
: \hat{n} & = N \\
: \hat{x} & = A \\
: \hat{p} & = N \\
: \hat{v} & = Vp \\
: V(s p) & = V(s n) \\
: Vn & = X
\end{align*}
\]
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

**Goal:** \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \)

**Solution:**

**Step 1:** let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

**Step 2:** compute equations on \( \hat{x}_i, x_i, X \) for having \( \Delta \vdash l : X \)

\[
\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = Vp \quad V(sp) = V(sn) \quad Vn = X
\]
Subject-reduction (SR) - Case of a rule \( l \rightarrow r \)

**Goal:** \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \)

**Solution:**

**Step 1:** let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

**Step 2:** compute equations on \( \hat{x}_i, x_i, X \) for having \( \Delta \vdash l : X \)

\[
\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = Vp \quad V(sp) = V(sn) \quad Vn = X
\]

**Step 3:** replace \( gt = gu \) by \( t = u \) if \( g \) is undefined
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

Goal: \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \implies \Gamma \vdash r\sigma : C \) ?

Solution:

Step 1: let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

Step 2: compute equations on \( \hat{x}_i, x_i, X \) for having \( \Delta \vdash l : X \)

\[
\begin{align*}
\hat{n} &= N \\
\hat{x} &= A \\
\hat{p} &= N \\
\hat{v} &= Vp \\
V(sp) &= V(sn) \\
Vn &= X
\end{align*}
\]

Step 3: replace \( gt = gu \) by \( t = u \) if \( g \) is undefined

\[
\begin{align*}
\hat{n} &= N \\
\hat{x} &= A \\
\hat{p} &= N \\
\hat{v} &= Vp \\
p &= n \\
Vn &= X
\end{align*}
\]
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

Goal: $\forall \Gamma, \sigma, C, \Gamma \vdash l\sigma : C \implies \Gamma \vdash r\sigma : C$ ?

Solution:

Step 1: let $\Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots$ be the variables of $l$

Step 2: compute equations on $\hat{x}_i, x_i, X$ for having $\Delta \vdash l : X$

Step 3: replace $gt = gu$ by $t = u$ if $g$ is undefined

\[
\begin{align*}
\hat{n} &= N & \hat{x} &= A & \hat{p} &= N & \hat{v} &= Vp & p &= n & Vn &= X
\end{align*}
\]
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

Goal: $\forall \Gamma, \sigma, C, \Gamma \vdash l\sigma : C \Rightarrow \Gamma \vdash r\sigma : C$ ?

Solution:

Step 1: let $\Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots$ be the variables of $l$

Step 2: compute equations on $\hat{x}_i, x_i, X$ for having $\Delta \vdash l : X$

Step 3: replace $gt = gu$ by $t = u$ if $g$ is undefined

$$\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = Vp \quad p = n \quad Vn = X$$

Step 4: apply Knuth-Bendix completion to transform the equations into a convergent rewrite system $S$
Subject-reduction (SR) - Case of a rule $l \leftrightarrow r$

Goal: $\forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \implies \Gamma \vdash r\sigma : C$ ?

Solution:

Step 1: let $\Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots$ be the variables of $l$

Step 2: compute equations on $\hat{x}_i, x_i, X$ for having $\Delta \vdash l : X$

Step 3: replace $gt = gu$ by $t = u$ if $g$ is undefined

\[
\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = Vp \quad p = n \quad Vn = X
\]

Step 4: apply Knuth-Bendix completion to transform the equations into a convergent rewrite system $S$

\[
\hat{n} \leftrightarrow N \quad \hat{x} \leftrightarrow A \quad \hat{p} \leftrightarrow N \quad \hat{v} \leftrightarrow Vn \quad p \leftrightarrow n \quad X \leftrightarrow Vn
\]
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

**Goal:** \( \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \)

**Solution:**

**Step 1:** let \( \Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots \) be the variables of \( l \)

**Step 2:** compute equations on \( \hat{x}_i, x_i, X \) for having \( \Delta \vdash l : X \)

**Step 3:** replace \( gt = gu \) by \( t = u \) if \( g \) is undefined

\[
\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = Vp \quad p = n \quad Vn = X
\]

**Step 4:** apply Knuth-Bendix completion to transform the equations into a convergent rewrite system \( S \)

\[
\hat{n} \leftrightarrow N \quad \hat{x} \leftrightarrow A \quad \hat{p} \leftrightarrow N \quad \hat{v} \leftrightarrow Vn \quad p \leftrightarrow n \quad X \leftrightarrow Vn
\]

**Step 5:** check \( \Delta \vdash r : X \) in any sub-system of \( \lambda \Pi / R + S \)
Subject-reduction (SR) - Case of a rule $l \rightarrow r$

Goal: $\forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C$ ?

Solution:

Step 1: let $\Delta = \ldots, \widehat{x}_i : \text{TYPE}, x_i : \widehat{x}_i, \ldots$ be the variables of $l$

Step 2: compute equations on $\widehat{x}_i, x_i, X$ for having $\Delta \vdash l : X$

Step 3: replace $gt = gu$ by $t = u$ if $g$ is undefined

$$
\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = Vp \quad p = n \quad Vn = X
$$

Step 4: apply Knuth-Bendix completion to transform the equations into a convergent rewrite system $S$

$$
\hat{n} \hookrightarrow N \quad \hat{x} \hookrightarrow A \quad \hat{p} \hookrightarrow N \quad \hat{v} \hookrightarrow Vn \quad p \hookrightarrow n \quad X \hookrightarrow Vn
$$

Step 5: check $\Delta \vdash r : X$ in any sub-system of $\lambda\Pi/R + S$

$$
\Delta \vdash v : X
$$
Subject-reduction (SR) - Case of a rule $l \leftrightarrow r$

Goal: \[ \forall \Gamma, \sigma, C, \quad \Gamma \vdash l\sigma : C \quad \Rightarrow \quad \Gamma \vdash r\sigma : C \quad ? \]

Solution:

Step 1: let $\Delta = \ldots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \ldots$ be the variables of $l$

Step 2: compute equations on $\hat{x}_i, x_i, X$ for having $\Delta \vdash l : X$

Step 3: replace $gt = gu$ by $t = u$ if $g$ is undefined

Step 4: apply Knuth-Bendix completion to transform the equations into a convergent rewrite system $S$

Step 5: check $\Delta \vdash r : X$ in any sub-system of $\lambda \Pi / R + S$

Conclusion: for $\text{SR}_R$ we use TC, CR
Summary
Dependencies between properties

- - $\rightarrow$ for dependency on a sub-system
Dependencies between properties

- - → for dependency on a sub-system

We need a finer analysis . . .
do we really need

– LCR?
do we really need

- LCR ? ok
do we really need

- LCR ? ok

- TC ?
do we really need

- LCR ? ok

- TC ? yes but $\text{SN}(\mathcal{R} + l \leftrightarrow r)$ requires $\text{TC}(\mathcal{R})$ only
do we really need

- LCR ? ok
- TC ? yes but $\text{SN}(\mathcal{R} + l \leftrightarrow r)$ requires $\text{TC}(\mathcal{R})$ only
- SR ?

Termination (SN) revised
do we really need

- LCR ? ok
- TC ? yes but $\text{SN}(\mathcal{R} + l \leftrightarrow r)$ requires $\text{TC}(\mathcal{R})$ only
- SR ? no (conjecture, ongoing work)
  a simple syntactic condition seems sufficient: that every rule maps an object to an object, and a type to a type
Outline

Rewriting in proof assistants

Dedukti and the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)

Properties of the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)

Conclusion
Dependencies between properties

- - → for dependency on a sub-system
Preservation of properties by rule addition

assume we have a calculus $\lambda\Pi/R$ with LCR, SN, $SR_\beta$, $SR_R$

how to prove LCR', SN' and SR' for $\lambda\Pi/R'$ with $R \subset R'$?
Preservation of properties by rule addition

assume we have a calculus $\lambda\Pi/\mathcal{R}$ with LCR, SN, SR$_\beta$, SR$_\mathcal{R}$

how to prove LCR', SN' and SR' for $\lambda\Pi/\mathcal{R}'$ with $\mathcal{R} \subseteq \mathcal{R}'$?

Step 1: try to prove LCR’
Preservation of properties by rule addition

assume we have a calculus $\lambda\Pi/\mathcal{R}$ with LCR, SN, $SR_\beta$, $SR_\mathcal{R}$

how to prove LCR', SN' and SR' for $\lambda\Pi/\mathcal{R}'$ with $\mathcal{R} \subset \mathcal{R}'$?

Step 1: try to prove LCR'
Step 2: try to prove SN' using LCR' and TC
Preservation of properties by rule addition

assume we have a calculus $\lambda\Pi/\mathcal{R}$ with LCR, SN, SR$_{\beta}$, SR$_{\mathcal{R}}$

how to prove LCR', SN' and SR' for $\lambda\Pi/\mathcal{R}'$ with $\mathcal{R} \subset \mathcal{R}'$?

Step 1: try to prove LCR'
Step 2: try to prove SN' using LCR' and TC
Step 3: then CR' by Newman's Lemma
Preservation of properties by rule addition

assume we have a calculus $\lambda\Pi/\mathcal{R}$ with LCR, SN, $SR_\beta$, $SR_\mathcal{R}$

how to prove LCR’, SN’ and SR’ for $\lambda\Pi/\mathcal{R}'$ with $\mathcal{R} \subset \mathcal{R}'$?

Step 1: try to prove LCR’
Step 2: try to prove SN’ using LCR’ and TC
Step 3: then CR’ by Newman’s Lemma
Step 4: then SR’$_\beta$
Preservation of properties by rule addition

assume we have a calculus $\lambda\Pi/\mathcal{R}$ with LCR, SN, SR$_\beta$, SR$_\mathcal{R}$

how to prove LCR', SN' and SR' for $\lambda\Pi/\mathcal{R}'$ with $\mathcal{R} \subset \mathcal{R}'$ ?

Step 1: try to prove LCR'
Step 2: try to prove SN' using LCR' and TC
Step 3: then CR' by Newman’s Lemma
Step 4: then SR'$_\beta$
Step 5: try to prove SR'$_\mathcal{R}'$ using Knuth-Bendix completion and TC$_{\mathcal{R}+S}$
Preservation of properties by rule addition

assume we have a calculus $\lambda \Pi / \mathcal{R}$ with LCR, SN, $\text{SR}_\beta$, $\text{SR}_\mathcal{R}$

how to prove $\text{LCR}'$, $\text{SN}'$ and $\text{SR}'$ for $\lambda \Pi / \mathcal{R}'$ with $\mathcal{R} \subset \mathcal{R}'$?

Step 1: try to prove $\text{LCR}'$
Step 2: try to prove $\text{SN}'$ using $\text{LCR}'$ and $\text{TC}$
Step 3: then $\text{CR}'$ by Newman’s Lemma
Step 4: then $\text{SR}'_\beta$
Step 5: try to prove $\text{SR}'_{\mathcal{R}'}$ using Knuth-Bendix completion and $\text{TC}_{\mathcal{R} + S}$

$\Rightarrow$ we need termination and confluence criteria for $\beta + \mathcal{R} + S$
when $S$ is closed and there are shared symbols
Another approach: prove CR without assuming SN

possible methods:
– (weakly) orthogonal systems
– development-closed critical pairs
– locally decreasing diagrams
Conclusion

- for finding out new criteria
- for providing tools

Thank you!