

On the use of confluence
in
type theory modulo rewriting

Frédéric Blanqui

Deduc-eam

The logo for Inria, featuring the word "Inria" in a red, cursive script font.The logo for École normale supérieure paris-saclay, consisting of the text "école", "normale", "supérieure", and "paris-saclay" stacked vertically, with horizontal lines extending to the right from the end of each line.The logo for LSU, featuring the letters "LSU" in a blue, stylized, rounded font.

30 June 2020

Outline

Rewriting in proof assistants

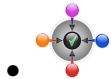
Dedukti and the $\lambda\Pi$ -calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)

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

Conclusion

Rewriting in proof assistants


increasing interest in using rewriting in proof assistants



• Dedukti

Rewriting in proof assistants

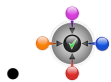
increasing interest in using rewriting in proof assistants

-  Dedukti

-  Agda

Rewriting in proof assistants

increasing interest in using rewriting in proof assistants



Dedukti



Agda

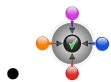


Coq ?

- *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/>

The screenshot shows a web browser window with the URL `https://agda.readthedocs.io/en/v2.6.1/language/rewriting.html`. The left sidebar contains a navigation menu with the following items: Postulates, Pragmas, Prop, Record Types, Reflection, **Rewriting** (expanded), Rewrite rules by example, General shape of rewrite rules, Confluence checking, Advanced usage, Run-time Irrelevance, Safe Agda, Sized Types, Syntactic Sugar, Syntax Declarations, Telescopes, Termination Checking, Universe Levels, With-Abstraction, and Without K. The main content area shows the breadcrumb `Docs » Language Reference » Rewriting`, followed by the title

Rewriting

. Below the title is a paragraph: "Rewrite rules allow you to extend Agda's evaluation relation with new computation rules." A blue note box contains the text: "This page is about the `--rewriting` option and the associated `REWRITE` builtin. You might be looking for the documentation on the `rewrite` construct instead." Below the note is the section title

Rewrite rules by example

, followed by a paragraph: "To enable rewrite rules, you should run Agda with the flag `--rewriting` and import the modules `Agda.Builtin.Equality` and `Agda.Builtin.Equality.Rewrite`:" Below this is a code block with a light green background containing the following Agda code:

```
{-# OPTIONS --rewriting #-}  
  
module language.rewriting where  
  
open import Agda.Builtin.Equality  
open import Agda.Builtin.Equality.Rewrite
```

Confluence checking in Agda

The screenshot shows a web browser window with the URL `https://agda.readthedocs.io/en/v2.6.1/language/rewriting.html#confluence-checking`. The left sidebar contains a navigation menu with the following items: Postulates, Pragmas, Prop, Record Types, Reflection, **Rewriting** (expanded), Rewrite rules by example, General shape of rewrite rules, **Confluence checking** (selected), Advanced usage, Run-time Irrelevance, Safe Agda, Sized Types, Syntactic Sugar, Syntax Declarations, Telescopes, Termination Checking, Universe Levels, With-Abstraction, and Without K.

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 p₁σ ... pₙσ` is rewritten to `vσ`, where `σ` is any substitution on the pattern variables `x₁, ... xₙ`.

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 `--confluence-check` flag.

Advanced usage

Instead of importing `Agda.Builtin.Equality.Rewrite`, a different type may be chosen as the rewrite relation by registering it as the `REWRITE` builtin. For example, using the pragma `{-# BUILTIN REWRITE _~ #-}` registers the type `_~_` 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.

Navigation buttons: [Previous](#) [Next](#)

Dedukti

Dedukti:

- purely functional “programming” language (λ -calculus)


Dedukti

Dedukti:

- purely functional “programming” language (λ -calculus)
- with dependent types (types can take values as arguments)


Dedukti

Dedukti:

- purely functional “programming” language (λ -calculus)
- with dependent types (types can take values as arguments)
- functions and types  can be defined (by rewrite rules \mathcal{R})

Dedukti

Dedukti:

- purely functional “programming” language (λ -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 $\triangle!$ can be defined (by rewrite rules \mathcal{R})
- implements the $\lambda\Pi$ -calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)

Example:

```
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

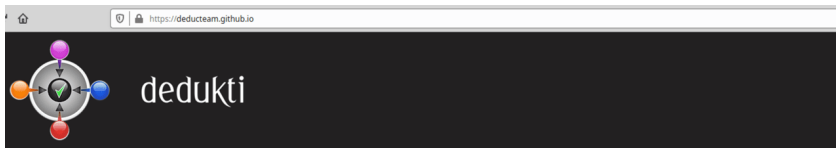
assert F 2 ≡ N → N → N //convertible expressions
```

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 $\lambda\Pi$ -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.
- [SigmaId](#) (SIGMA-calculus In Dedukti) is an encoding of the simply-typed ζ -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/>

  <https://github.com/Deducteam/lambdapi/blob/master/doc/DOCUMENTATION.md>

User manual for Lambdapi

Lambdapi is a proof assistant based on the $\lambda\Pi$ -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:

```
rule      0 + $y  ↦ $y
with s $x + $y  ↦ s ($x + $y)
with     $x + 0  ↦ $x
with     $x + s $y ↦ s ($x + $y)
```

Rewrite rules and matching in Dedukti

- matching on defined symbols:

```
rule ($x + $y) + $z ↦ $x + ($y + $z)
```

Rewrite rules and matching in Dedukti

- LHS can be non-linear:

```
rule $x + (- $x) ↦ 0
```

Rewrite rules and matching in Dedukti

- higher-order pattern-matching:

```
rule diff( $\lambda x.$ sin $f[x])  $\hookrightarrow$  diff( $\lambda x.$ $f[x])*cos  
rule lam( $\lambda x.$ app $f[] x)  $\hookrightarrow$  $f[] //  $\eta$ -rule
```

Rewrite rules and matching in Dedukti

- LHS can be overlapping:

```
rule    0 + $y  ↪ $y
with s  $x + $y  ↪ s ($x + $y)
with    $x + 0   ↪ $x
with    $x + s $y ↪ s ($x + $y)
```

- matching on defined symbols:

```
rule ($x + $y) + $z ↪ $x + ($y + $z)
```

- LHS can be non-linear:

```
rule $x + (- $x) ↪ 0
```

- higher-order pattern-matching:

```
rule diff(λx.sin $f[x]) ↪ diff(λx.$f[x])*cos
rule lam(λx.app $f[] x) ↪ $f[] // η-rule
```

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:

<https://github.com/Deducteam/lambdapi/blob/master/doc/options.md>

Confluence checking

Lambdapl 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^ho` confluence checker has been tested with Lambdapl. It can be called using the flag `--confluence "path/to/csiho.sh --ext trs --stdin"`.

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

Termination checking

Lambdapl 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 Lambdapl 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 Lambdapl call it directly. Alternatively, one can first generate a `.trs` 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

<https://github.com/Deducteam/lambdapl/blob/master/doc/options.md>

Confluence checking

Lambdapl 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^ho` confluence checker has been tested with Lambdapl. It can be called using the flag `--confluence "path/to/csiho.sh --ext trs --stdin"`.

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

Termination checking

Lambdapl 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 Lambdapl 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 Lambdapl call it directly. Alternatively, one can first generate a `.trs` 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

<https://github.com/Deducteam/lambdapl/blob/master/doc/options.md>

Confluence checking

Lambdapl 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^ho` confluence checker has been tested with Lambdapl. It can be called using the flag `--confluence "path/to/csiho.sh --ext trs --stdin"`.

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

Termination checking

Lambdapl 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 Lambdapl 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 Lambdapl call it directly. Alternatively, one can first generate a `.wanda` file as described below:

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

$\lambda\Pi$ -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)

$\lambda\Pi$ -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)
- | $s \in \{\text{TYPE}, \text{KIND}\}$ (sort)
- | $\Pi x:A.B$ (dependent product, written $A \rightarrow B$ if $x \notin B$)

$\lambda\Pi$ -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)
- | $s \in \{\text{TYPE}, \text{KIND}\}$ (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)

$\lambda\Pi$ -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)
- | $s \in \{\text{TYPE}, \text{KIND}\}$ (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 $\rho =$

- | $\Delta \vdash f t_1 \dots t_n \hookrightarrow u$ (rewrite rule)

Typing rules of $\lambda\Pi/\mathcal{R}$

$$\text{(empty)} \frac{}{\emptyset \text{ valid}} \quad \text{(decl)} \frac{\Gamma \text{ valid} \quad \Gamma \vdash A : s}{\Gamma, x:A \text{ valid}}$$

Typing rules of $\lambda\Pi/\mathcal{R}$

$$\text{(fun)} \frac{\Gamma \text{ valid}}{\Gamma \vdash f : A_f} \quad \text{(var)} \frac{\Gamma, x:A, \Gamma' \text{ valid}}{\Gamma, x:A, \Gamma' \vdash x : A}$$

$$\text{(abs)} \frac{\Gamma, x:A \vdash t : B \quad \Gamma \vdash \Pi x:A. B : s}{\Gamma \vdash \lambda x:A. t : \Pi x:A. B} \quad \text{(app)} \frac{\Gamma \vdash t : \Pi x:A. B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B\{x \mapsto u\}}$$

Typing rules of $\lambda\Pi/\mathcal{R}$

$$\text{(sort)} \quad \frac{\Gamma \text{ valid}}{\Gamma \vdash \text{TYPE} : \text{KIND}}$$

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

$$\text{(conv)} \quad \frac{\Gamma \vdash t : A \quad A \downarrow_{\beta\mathcal{R}} B \quad \Gamma \vdash B : s}{\Gamma \vdash t : B}$$

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

Typing rules of $\lambda\Pi/\mathcal{R}$

$$\text{(empty)} \frac{}{\emptyset \text{ valid}} \quad \text{(decl)} \frac{\Gamma \text{ valid} \quad \Gamma \vdash A : s}{\Gamma, x:A \text{ valid}}$$

$$\text{(fun)} \frac{\Gamma \text{ valid}}{\Gamma \vdash f : A_f} \quad \text{(var)} \frac{\Gamma, x:A, \Gamma' \text{ valid}}{\Gamma, x:A, \Gamma' \vdash x : A}$$

$$\text{(abs)} \frac{\Gamma, x:A \vdash t : B \quad \Gamma \vdash \Pi x:A. B : s}{\Gamma \vdash \lambda x:A. t : \Pi x:A. B} \quad \text{(app)} \frac{\Gamma \vdash t : \Pi x:A. B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B\{x \mapsto u\}}$$

$$\text{(sort)} \frac{\Gamma \text{ valid}}{\Gamma \vdash \text{TYPE} : \text{KIND}} \quad \text{(prod)} \frac{\Gamma \vdash A : \text{TYPE} \quad \Gamma, x:A \vdash B : s}{\Gamma \vdash \Pi x:A. B : s}$$

$$\text{(conv)} \frac{\Gamma \vdash t : A \quad A \downarrow_{\beta\mathcal{R}} B \quad \Gamma \vdash B : s}{\Gamma \vdash t : B} \quad A \downarrow_{\beta\mathcal{R}} B \text{ if } A \text{ and } B \text{ have a common reduct wrt the } \beta\text{-rule of } \lambda\text{-calculus and the user-defined rules } \mathcal{R}$$

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 ($\lambda\Pi/\mathcal{R}$)

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

Conclusion

Some important properties

TC	decidability of the typing relation
SN	termination of $\hookrightarrow_{\beta\mathcal{R}}$ from typable terms
SR_{β}	preservation of typing by \hookrightarrow_{β}
$SR_{\mathcal{R}}$	preservation of typing by $\hookrightarrow_{\mathcal{R}}$
LCR	local confluence of $\hookrightarrow_{\beta\mathcal{R}}$ on arbitrary terms
CR	confluence of $\hookrightarrow_{\beta\mathcal{R}}$ from typable terms

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

$$\text{(conv)} \frac{\Gamma \vdash t \uparrow A \quad A \downarrow_{\beta\mathcal{R}}^* B}{\Gamma \vdash t \downarrow B}$$

Decidability of type-checking (TC)

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

$$\text{(conv)} \frac{\Gamma \vdash t \uparrow A \quad A \downarrow_{\beta\mathcal{R}}^* B}{\Gamma \vdash t \downarrow B}$$

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

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

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

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

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

$$\text{(app)} \frac{\Gamma \vdash t \uparrow C \quad C \hookrightarrow_{\beta\mathcal{R}}^* \Pi x:A. B \quad \Gamma \vdash u \downarrow A}{\Gamma \vdash tu \uparrow B\{x \mapsto u\}}$$

Decidability of type-checking (TC)

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

$$\text{(app)} \frac{\Gamma \vdash t \uparrow C \quad C \xrightarrow{\beta\mathcal{R}}^* \Pi x:A.B \quad \Gamma \vdash u \downarrow A}{\Gamma \vdash tu \uparrow B\{x \mapsto u\}}$$

Decidability of type-checking (TC)

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

$$\text{(conv)} \frac{\Gamma \vdash t \uparrow A \quad A \downarrow_{\beta\mathcal{R}}^* B}{\Gamma \vdash t \downarrow B}$$

$$\text{(sort)} \frac{\Gamma \text{ valid}}{\Gamma \vdash \text{TYPE} \uparrow \text{KIND}} \quad \text{(prod)} \frac{\Gamma \vdash A \downarrow \text{TYPE} \quad \Gamma, x:A \vdash B \uparrow s}{\Gamma \vdash \Pi x:A. B \uparrow s}$$

$$\text{(fun)} \frac{\Gamma \text{ valid}}{\Gamma \vdash f \uparrow A_f} \quad \text{(var)} \frac{\Gamma, x:A, \Gamma' \text{ valid}}{\Gamma, x:A, \Gamma' \vdash x \uparrow A}$$

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

$$\text{(app)} \frac{\Gamma \vdash t \uparrow C \quad C \hookrightarrow_{\beta\mathcal{R}}^* \Pi x:A. B \quad \Gamma \vdash u \downarrow A}{\Gamma \vdash tu \uparrow B\{x \mapsto u\}}$$

Conclusion: for TC we use **SN**, **SR**, **LCR**

Termination (SN)

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

Step 1 define a function

$$\begin{array}{lcl} \llbracket \cdot \rrbracket : & \text{types} & \rightarrow \text{subsets of SN} \\ & A & \mapsto \llbracket A \rrbracket \end{array}$$

invariant by reduction: $A \hookrightarrow_{\beta\mathcal{R}} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket$

+ other conditions

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

Step 1 define a function

$$\begin{array}{ccc} \llbracket \cdot \rrbracket : & \text{types} & \rightarrow \text{subsets of SN} \\ & A & \mapsto \llbracket A \rrbracket \end{array}$$

invariant by reduction: $A \hookrightarrow_{\beta\mathcal{R}} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket$

+ other conditions

Step 2 prove $\Gamma \vdash t : A \Rightarrow t \in \llbracket A \rrbracket$

Termination (SN) – Step 1

Goal: define a function

$$\begin{array}{lcl} \llbracket \cdot \rrbracket : & \text{types} & \rightarrow \text{ subsets of SN} \\ & A & \mapsto \llbracket A \rrbracket \end{array}$$

invariant by reduction: $A \hookrightarrow_{\beta\mathcal{R}} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket$

+ other conditions

Termination (SN) – Step 1

Goal: define a function

$$\begin{array}{lcl} \llbracket \cdot \rrbracket : & \text{types} & \rightarrow \text{subsets of SN} \\ & A & \mapsto \llbracket A \rrbracket \end{array}$$

invariant by reduction: $A \hookrightarrow_{\beta\mathcal{R}} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket$

+ other conditions

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

using **SR** and **LCR**

Termination (SN) – Step 1

Goal: define a function

$$\begin{array}{lcl} \llbracket \cdot \rrbracket : & \text{types} & \rightarrow \text{subsets of SN} \\ & A & \mapsto \llbracket A \rrbracket \end{array}$$

invariant by reduction: $A \hookrightarrow_{\beta\mathcal{R}} A' \Rightarrow \llbracket A \rrbracket = \llbracket A' \rrbracket$

+ other conditions

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

using **SR** and **LCR**

 previous works assumed no critical pairs on types

Termination (SN) – Step 2

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

Termination (SN) – Step 2

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

Solution: for a rule $\Delta \vdash f l \hookrightarrow r$ 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 $\Delta \vdash f l \hookrightarrow r$ 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

for all Γ, t, u, A , if $\Gamma \vdash t : A$ and $t \hookrightarrow_{\beta\mathcal{R}} u$, then $\Gamma \vdash u : A$

applications:

- in programming languages: ensures memory safety
- in logical systems: correctness of cut elimination

Subject-reduction for β -reduction (SR_β)

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

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

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\frac{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\} \qquad B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s''}{\Gamma \vdash (\lambda x:A.t)u : C} \text{ (conv)}$$

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\frac{\frac{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \text{ (app)} \quad B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s''}{\Gamma \vdash (\lambda x:A.t)u : C} \text{ (conv)}$$

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\begin{array}{c}
 \frac{\Gamma \vdash \lambda x:A.t : \Pi x:A.B \quad \Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \quad \Gamma \vdash \Pi x:A'.B' : s'}{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B'} \text{ (conv)} \\
 \nearrow \\
 \frac{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \text{ (app)} \\
 \frac{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\} \quad B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s''}{\Gamma \vdash (\lambda x:A.t)u : C} \text{ (conv)}
 \end{array}$$

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\begin{array}{c}
 \frac{\Gamma, x:A \vdash t : B \quad \Pi x:A.B : s}{\Gamma \vdash \lambda x:A.t : \Pi x:A.B} \text{ (abs)} \quad \Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \quad \Gamma \vdash \Pi x:A'.B' : s' \\
 \hline
 \Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \text{(conv)} \\
 \nearrow \\
 \frac{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \text{ (app)} \quad B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s'' \\
 \hline
 \Gamma \vdash (\lambda x:A.t)u : C \quad \text{(conv)}
 \end{array}$$

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\frac{\Gamma, x:A \vdash t : B \quad \Pi x:A.B : s}{\Gamma \vdash \lambda x:A.t : \Pi x:A.B} \text{ (abs)} \quad \frac{\Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \quad \Gamma \vdash \Pi x:A'.B' : s'}{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B'} \text{ (conv)}$$

\nearrow

$$\frac{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \text{ (app)} \quad \frac{B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s''}{\Gamma \vdash (\lambda x:A.t)u : C} \text{ (conv)}$$

Problem: $A' \downarrow_{\beta\mathcal{R}}^* A$ and $B\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad ?$

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\frac{\Gamma, x:A \vdash t : B \quad \Pi x:A.B : s}{\Gamma \vdash \lambda x:A.t : \Pi x:A.B} \text{ (abs)} \quad \frac{\Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \quad \Gamma \vdash \Pi x:A'.B' : s'}{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B'} \text{ (conv)}$$

\nearrow

$$\frac{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \text{ (app)} \quad \frac{B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s''}{\Gamma \vdash (\lambda x:A.t)u : C} \text{ (conv)}$$

Problem: $A' \downarrow_{\beta\mathcal{R}}^* A$ and $B\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad ?$

Solution: $\Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \xRightarrow{\text{CR}} A \downarrow_{\beta\mathcal{R}}^* A' \wedge B \downarrow_{\beta\mathcal{R}}^* B'$

Subject-reduction (SR) - Case of \hookrightarrow_{β}

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

$$\frac{\Gamma, x:A \vdash t : B \quad \Pi x:A.B : s}{\Gamma \vdash \lambda x:A.t : \Pi x:A.B} \text{ (abs)} \quad \Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \quad \Gamma \vdash \Pi x:A'.B' : s'$$

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

\nearrow

$$\frac{\Gamma \vdash \lambda x:A.t : \Pi x:A'.B' \quad \Gamma \vdash u : A'}{\Gamma \vdash (\lambda x:A.t)u : B'\{x \mapsto u\}} \text{ (app)} \quad B'\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C \quad \Gamma \vdash C : s''$$

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

Problem: $A' \downarrow_{\beta\mathcal{R}}^* A$ and $B\{x \mapsto u\} \downarrow_{\beta\mathcal{R}}^* C$?

Solution: $\Pi x:A.B \downarrow_{\beta\mathcal{R}}^* \Pi x:A'.B' \xRightarrow{\text{CR}} A \downarrow_{\beta\mathcal{R}}^* A' \wedge B \downarrow_{\beta\mathcal{R}}^* B'$

Conclusion: for SR_{β} we use **CR**

Subject-reduction for rewriting ($\text{SR}_{\mathcal{R}}$)

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

see my FSCD talk on July 3rd at 14:30 ! 😊

Example: tail function on vectors

```
symbol V : N → TYPE
symbol nil : V 0
symbol cons : A → Π n : N, V n → V (sn)

symbol tail : Π n : N, V (sn) → V n
```

$$\text{tail } n \text{ (cons } x \text{ p } v) \hookrightarrow v$$

Example: tail function on vectors

```
symbol V : N → TYPE
symbol nil : V0
symbol cons : A → Πn : N, Vn → V(s n)

symbol tail : Πn : N, V(s n) → Vn
```

$$\underbrace{\text{tail } \underbrace{n}_{:N} \left(\underbrace{(\text{cons } x \text{ p } v)}_{:V(s n)} \right)}_{:Vn} \quad \hookrightarrow \quad \underbrace{v}_{:Vn ?}$$

Example: tail function on vectors

```
symbol V : N → TYPE
symbol nil : V 0
symbol cons : A → Π n : N, V n → V (sn)

symbol tail : Π n : N, V (sn) → V n
```

$$\text{tail } \underbrace{n}_{:N} \left(\text{cons } \underbrace{x}_{:A} \underbrace{p}_{:N} \underbrace{v}_{:Vp} \right) \hookrightarrow \underbrace{v}_{:Vp} \downarrow_{\beta^* \mathcal{R}}^* V n ?$$

$\underbrace{\hspace{15em}}_{:Vn}$

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

$$\Delta = \hat{n} : \text{TYPE}, n : \hat{n}, \hat{x} : \text{TYPE}, x : \hat{x}, \dots$$

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

$$\Delta = \widehat{n} : \text{TYPE}, n : \widehat{n}, \widehat{x} : \text{TYPE}, x : \widehat{x}, \dots$$

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

$$\Delta = \hat{n} : \text{TYPE}, n : \hat{n}, \hat{x} : \text{TYPE}, x : \hat{x}, \dots$$

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

$$\underbrace{\text{tail} \underbrace{n}_{:\hat{n}=\text{N}} \left(\text{cons} \underbrace{x}_{:\hat{x}=\text{A}} \underbrace{p}_{:\hat{p}=\text{N}} \underbrace{v}_{:\hat{v}=\text{Vp}} \right)}_{:\text{V}(\text{sp})=\text{V}(\text{sn})}}_{:\text{Vn}=\text{X}}$$

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

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

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

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

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

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

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

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

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

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

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

Step 1: let $\Delta = \dots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \dots$ 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

$$\hat{n} = N \quad \hat{x} = A \quad \hat{p} = N \quad \hat{v} = V_p \quad p = n \quad V_n = X$$

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

Step 1: let $\Delta = \dots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \dots$ 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

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

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

Step 1: let $\Delta = \dots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \dots$ 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 \mathcal{S}

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

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

Step 1: let $\Delta = \dots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \dots$ 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 \mathcal{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 $\lambda\Pi/\mathcal{R} + \mathcal{S}$

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

Step 1: let $\Delta = \dots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \dots$ 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 \mathcal{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 $\lambda\Pi/\mathcal{R} + \mathcal{S}$

$$\Delta \vdash v : X$$

Subject-reduction (SR) - Case of a rule $l \hookrightarrow r$

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

Solution:

Step 1: let $\Delta = \dots, \hat{x}_i : \text{TYPE}, x_i : \hat{x}_i, \dots$ 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

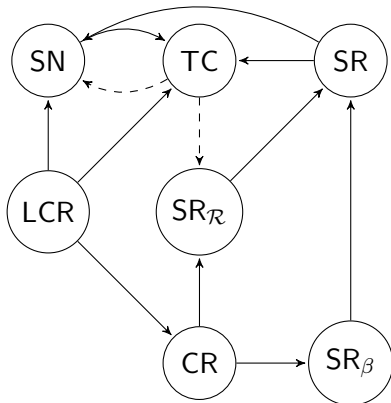
Step 4: apply **Knuth-Bendix** completion to transform the equations into a convergent rewrite system \mathcal{S}

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

Conclusion: for $\text{SR}_{\mathcal{R}}$ we use **TC**, **CR**

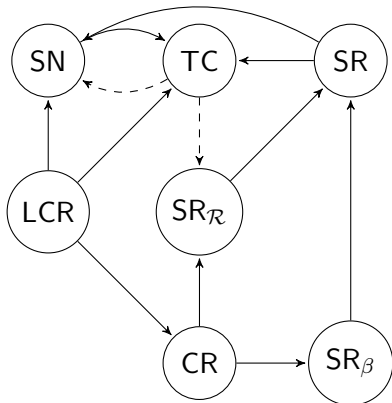
Summary

Dependencies between properties



- - \rightarrow for dependency on a sub-system

Dependencies between properties



- - \rightarrow for dependency on a sub-system

We need a finer analysis ...

Termination (SN) revised

do we really need

– LCR ?

Termination (SN) revised

do we really need

– LCR ? ok

Termination (SN) revised

do we really need

- LCR ? ok
- TC ?

Termination (SN) revised

do we really need

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

Termination (SN) revised

do we really need

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

Termination (SN) revised

do we really need

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

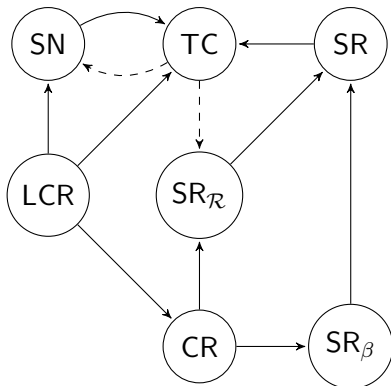
Rewriting in proof assistants

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



- - \rightarrow for dependency on a sub-system

Preservation of properties by rule addition

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

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

Preservation of properties by rule addition

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

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

Step 1: try to prove LCR'

Preservation of properties by rule addition

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

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_β , $SR_{\mathcal{R}}$

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_β , $SR_{\mathcal{R}}$

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'_β

Preservation of properties by rule addition

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

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'_β

Step 5: try to prove $SR'_{\mathcal{R}'}$ using Knuth-Bendix completion and $TC_{\mathcal{R}+\mathcal{S}}$

Preservation of properties by rule addition

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

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'_β

Step 5: try to prove $SR'_{\mathcal{R}'}$ using Knuth-Bendix completion and $TC_{\mathcal{R}+\mathcal{S}}$

\Rightarrow we need termination and confluence criteria for $\beta + \mathcal{R} + \mathcal{S}$
when \mathcal{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



**WE NEED
YOU**

- for finding out new criteria
- for providing tools

Thank you!