Type safety of rewrite rules in dependent types

Frédéric Blanqui

Deduc⊢eam

Inria — école normale supérieure paris–saclay — LSV

3 July 2020

Outline

Type safety, aka subject-reduction (SR) in typed programming languages

assume a typed prog. language with operational semantics $\hookrightarrow$

subject-reduction property (SR):

$$\boxed{\text{if } t : A \text{ and } t \hookrightarrow u, \text{ then } u : A}$$

meaning: an expression checked of type A at compile time can only evaluate to a value of type A

– fondamental property of *statically-typed* prog. languages

– ensure memory safety

SR in type-based logical systems

assume a type system with cut-elimination relation $\hookrightarrow$

subject-reduction property (SR):

$$\boxed{\text{if } t : A \text{ and } t \hookrightarrow u, \text{ then } u : A}$$

meaning: a proof of proposition A can only reduce to a proof of A

- correctness of cut-elimination
- correctness of type inference in dependent type theories

Outline

Dedukti v3 aka Lambdapi

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

- 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 v3 aka Lambdapi

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

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

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
```

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 talk just after at 15:30 🙂

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 just after at 15:00 🙂

- independant proof checker

- proof transformations

`https://deducteam.github.io/`

`http://logipedia.science/`

Outline

$\lambda\Pi$-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 $t, u, A, B =$

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

typing environments $\Gamma =$

$\emptyset$	(empty environment)
$\Gamma, x{:}A$	(variable declaration)

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

terms/types $t, u, A, B =$

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

typing environments $\Gamma =$

$\mid \emptyset$	(empty environment)
$\mid \Gamma, x{:}A$	(variable declaration)

rewrite rules $\rho =$

$\mid f\ t_1 \ldots t_n \hookrightarrow u$	(rewrite rule)

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

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

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

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

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

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

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

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

$A \downarrow_{\beta\mathcal{R}} B$ 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 $\lambda\Pi/\mathcal{R}$

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

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

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

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

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

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

<u>remark:</u> the type of a term is unique up to $\downarrow^*_{\beta\mathcal{R}}$

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

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

undecidable in $\lambda\Pi/\mathcal{R}$ [Saillard, 2015]

A first (not so good) idea

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

there exists B such that $l : B$ and $r : B$

$\Rightarrow$ enforces many rules to be non-linear

$\Rightarrow$ rewriting is less efficient and confluence more difficult to prove

Example: tail function on vectors

```
symbol A:TYPE

symbol V:N → TYPE
symbol nil:V0
symbol cons:A → Πn:N,Vn → V(sn)

symbol tail:Πn:N,V(sn) → Vn

rule tail $n (cons $x $p $v) ↪ $v
```

the LHS is not typable:

	`cons x p v`	has type	`V(sp)`
but	`tail n`	expects an argument of type	`V(sn)`

replacing `p` by `n` makes it typable but non-linear

$\Rightarrow$ rewriting is less efficient and confluence more difficult to prove

Example: tail function on vectors (continued)

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

symbol tail:Πn:N,V(sn) → Vn
```

yet the rule preserves typing (SR property):

– let `tail n (cons x p v)` be a typable instance of the LHS

– by inversion of typing rules, we get:

$$\underbrace{\texttt{tail}\ \underbrace{\texttt{n}}_{:\texttt{N}}\ \underbrace{(\texttt{cons}\ \underbrace{\texttt{x}}_{:\texttt{A}}\ \underbrace{\texttt{p}}_{:\texttt{N}}\ \underbrace{\texttt{v}}_{:\texttt{Vp}})}_{:\texttt{V(sp)}\downarrow^*_{\beta\mathcal{R}}\texttt{V(sn)}}}_{:\texttt{Vn}} \hookrightarrow \underbrace{\texttt{v}}_{:\texttt{Vp}}$$

– since `V` and `s` are undefined, $\texttt{V(sp)} \downarrow^*_{\beta\mathcal{R}} \texttt{V(sn)}$ implies $\texttt{p} \downarrow^*_{\beta\mathcal{R}} \texttt{n}$

Outline

Contribution

A procedure to prove that a rewrite rule preserves typing in $\lambda\Pi/\mathcal{R}$:

Contribution

A procedure to prove that a rewrite rule preserves typing in $\lambda\Pi/\mathcal{R}$:

Step 1: compute the equations $\mathcal{E}$ that must be satisfied
for the LHS to be of type C (fresh constant)

goal: prove that the RHS has type C modulo $\mathcal{E}$

Contribution

A procedure to prove that a rewrite rule preserves typing in $\lambda\Pi/\mathcal{R}$:

Step 1: compute the equations $\mathcal{E}$ that must be satisfied
for the LHS to be of type C (fresh constant)

goal: prove that the RHS has type C modulo $\mathcal{E}$

problem: how to type-check modulo equations?

Contribution

A procedure to prove that a rewrite rule preserves typing in $\lambda\Pi/\mathcal{R}$:

Step 1: compute the equations $\mathcal{E}$ that must be satisfied
for the LHS to be of type C (fresh constant)

Step 2: turn the equations into a convergent rewrite system $\mathcal{S}$
using Knuth-Bendix completion

Step 3: check that the RHS has type C in $\lambda\Pi/\mathcal{R} + \mathcal{S}$

Step 1: compute typability constraints $\mathcal{E}$ of the LHS

$$\underbrace{t}_{\text{term}} \uparrow \underbrace{A}_{\text{type}}\ \underbrace{[\mathcal{E}]}_{\text{equations}} \qquad \text{(input: } t\text{; output: } A[\mathcal{E}]\text{)}$$

$$\text{(var)}\ \frac{}{y \uparrow \widehat{y}\,[\emptyset]} \qquad (\widehat{y} \text{ new constant for the unknown type of } y)$$

$$\text{(fun)}\ \frac{f : \Pi x_1{:}T_1, \ldots, \Pi x_n{:}T_n, U \quad t_1 \uparrow A_1[\mathcal{E}_1] \quad t_n \uparrow A_n[\mathcal{E}_n]}{f t_1 \ldots t_n \uparrow U\sigma[\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n \cup \{A_1 = T_1\sigma, \ldots, A_n = T_n\sigma\}]}$$
$$\text{where } \sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

$$\underbrace{\texttt{tail}\ \underbrace{\texttt{n}}_{\uparrow\widehat{\texttt{n}}=\texttt{N}}\ \underbrace{(\texttt{cons}\ \underbrace{\texttt{x}}_{\uparrow\widehat{\texttt{x}}=\texttt{A}}\ \underbrace{\texttt{p}}_{\uparrow\widehat{\texttt{p}}=\texttt{N}}\ \underbrace{\texttt{v}}_{\uparrow\widehat{\texttt{v}}=\texttt{Vp}})}_{\uparrow\texttt{V(sp)}=\texttt{V(sn)}}}_{\uparrow\texttt{Vn}}$$

Step 2: turn $\mathcal{E}$ into a convergent rewrite system $\mathcal{S}$

using Knuth-Bendix completion procedure (KB)
with any well-founded order total on ground terms (e.g. LPO)

remark: KB always terminates on ground equations in this case

example: $\widehat{\mathtt{x}} > \widehat{\mathtt{v}} > \widehat{\mathtt{p}} > \widehat{\mathtt{n}} > \mathtt{V} > \mathtt{T} > \mathtt{N} > \mathtt{s} > \mathtt{p} > \mathtt{n}$

$$\begin{array}{lccccc} \mathcal{E}: & \widehat{\mathtt{x}} = \mathtt{A} & \widehat{\mathtt{p}} = \mathtt{N} & \widehat{\mathtt{v}} = \mathtt{Vp} & \widehat{\mathtt{n}} = \mathtt{N} & \mathtt{V(sp)} = \mathtt{V(sn)} \\ \mathcal{S}: & \widehat{\mathtt{x}} \hookrightarrow \mathtt{A} & \widehat{\mathtt{p}} \hookrightarrow \mathtt{N} & \widehat{\mathtt{v}} \hookrightarrow \mathtt{Vp} & \widehat{\mathtt{n}} \hookrightarrow \mathtt{N} & \mathtt{V(sp)} \hookrightarrow \mathtt{V(sn)} \end{array}$$

Step 3: check that RHS has same type as LHS modulo $\mathcal{S}$

$$\underbrace{\texttt{tail}\ \underbrace{\texttt{n}}_{\uparrow\widehat{\texttt{n}}=\texttt{N}}\ \underbrace{(\texttt{cons}\ \underbrace{\texttt{x}}_{\uparrow\widehat{\texttt{x}}=\texttt{A}}\ \underbrace{\texttt{p}}_{\uparrow\widehat{\texttt{p}}=\texttt{N}}\ \underbrace{\texttt{v}}_{\uparrow\widehat{\texttt{v}}=\texttt{Vp}})}_{\uparrow\texttt{V(sp)}=\texttt{V(sn)}}}_{\uparrow\texttt{Vn}} \quad \hookrightarrow \quad \texttt{v}$$

$$\mathcal{S}: \quad \widehat{\texttt{x}} \hookrightarrow \texttt{A} \quad \widehat{\texttt{p}} \hookrightarrow \texttt{N} \quad \widehat{\texttt{v}} \hookrightarrow \texttt{Vp} \quad \widehat{\texttt{n}} \hookrightarrow \texttt{N} \quad \texttt{V(sp)} \hookrightarrow \texttt{V(sn)}$$

we now want to check if

$$\boxed{\texttt{v} : \texttt{Vn} \text{ modulo } \mathcal{S} \text{ ?}}$$

but it doesn't work since $\texttt{v} : \widehat{\texttt{v}}$ and $\widehat{\texttt{v}} \not\to^*_{\beta\mathcal{RS}} \texttt{Vn}$ 😕

Step 1': simplify equations using confluence of $\hookrightarrow_{\beta\mathcal{R}}$

$$\mathcal{E}: \quad \widehat{\mathtt{x}} = \mathtt{A} \quad \widehat{\mathtt{p}} = \mathtt{N} \quad \widehat{\mathtt{v}} = \mathtt{Vp} \quad \widehat{\mathtt{n}} = \mathtt{N} \quad \mathtt{V(sp)} = \mathtt{V(sn)}$$

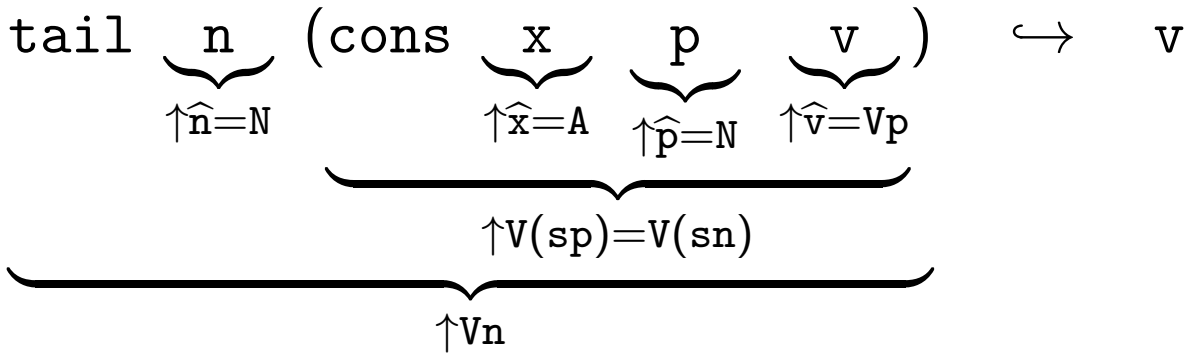
since V and s are undefined, $\mathcal{E}$ is equivalent to:

$$\mathcal{E}': \quad \widehat{\mathtt{x}} = \mathtt{A} \quad \widehat{\mathtt{p}} = \mathtt{N} \quad \widehat{\mathtt{v}} = \mathtt{Vp} \quad \widehat{\mathtt{n}} = \mathtt{N} \quad \mathtt{p} = \mathtt{n}$$

step 3 (KB) with $\widehat{\mathtt{x}} > \widehat{\mathtt{v}} > \widehat{\mathtt{p}} > \widehat{\mathtt{n}} > \mathtt{V} > \mathtt{T} > \mathtt{N} > \mathtt{s} > \mathtt{p} > \mathtt{n}$:

$$\mathcal{S}': \quad \widehat{\mathtt{x}} \hookrightarrow \mathtt{A} \quad \widehat{\mathtt{p}} \hookrightarrow \mathtt{N} \quad \widehat{\mathtt{v}} \hookrightarrow \mathtt{Vn} \quad \widehat{\mathtt{n}} \hookrightarrow \mathtt{N} \quad \mathtt{p} \hookrightarrow \mathtt{n}$$

Step 3: check that RHS has same type as LHS modulo $\mathcal{S}$



$$\mathcal{S}' : \quad \widehat{\mathtt{x}} \hookrightarrow \mathtt{A} \quad \widehat{\mathtt{p}} \hookrightarrow \mathtt{N} \quad \widehat{\mathtt{v}} \hookrightarrow \mathtt{Vn} \quad \widehat{\mathtt{n}} \hookrightarrow \mathtt{N} \quad \mathtt{p} \hookrightarrow \mathtt{n}$$

we want to check if

$$\boxed{\mathtt{v} : \mathtt{Vn} \text{ modulo } \mathcal{S}' \text{ ?}}$$

it now works since $\mathtt{v} : \widehat{\mathtt{v}}$ and $\widehat{\mathtt{v}} \hookrightarrow \mathtt{Vn}$ 🙂

Conclusion

A procedure to prove that a rewrite rule preserves typing in $\lambda\Pi/\mathcal{R}$:

Step 1: compute the equations $\mathcal{E}$ that must be satisfied for the LHS to be of type C

Step 2: simplify equations assuming confluence of $\hookrightarrow_{\beta\mathcal{R}}$

Step 3: turn the equations into a convergent rewrite system $\mathcal{S}$ using Knuth-Bendix completion

Step 4: check that the RHS has type C in $\lambda\Pi/\mathcal{R}+\mathcal{S}$

Also in the paper:

- for decidability of Step 4: termination criterion for $\hookrightarrow_{\beta\mathcal{R}} \cup \hookrightarrow_{\mathcal{S}}$ when $\hookrightarrow_{\beta\mathcal{R}}$ and $\hookrightarrow_{\mathcal{S}}$ terminate and $\mathcal{S}$ is a ground rewrite system