Autosubst: Automation for de Bruijn Substitutions in Lean

Second Bachelor Seminar Talk

Sarah Mameche
Advisor: Kathrin Stark
Supervisor: Prof. Dr. Gert Smolka

Okt 12, 2018

Previously: Autosubst ...

Goal: support formal reasoning about languages with binders in a proof assistant

$$s, t \in tm \;:=\; x \mid s\; t \mid \lambda\, s \quad (x \in \mathbb{N})$$

$\downarrow$

de Bruijn

The σ−calculus + **Autosubst**

Previously: Autosubst and Lean

THEOREM PROVER

Microsoft Research

interactive theorem proving + automated theorem proving

Overview

- Recap
- Theory
  - Extension of the σ–calculus: vector substitutions
- Realisation in Lean
  - Implementation
  - Proof term generation
  - Reflection
- Outlook

Extension of the σ−calculus: vector substitutions[1]

call-by-value System F

$$
\begin{aligned}
A, B &\in \text{ty} := X \mid A\ B \mid \forall.\ A \\
s, t &\in \text{tm} := s\ t \mid s\ A \mid v \\
u, v &\in \text{vl} := x \mid \lambda A.\ s \mid \Lambda.\ s
\end{aligned}
$$

substitutions $\sigma\colon \mathbb{N} \to ty, \tau\colon \mathbb{N} \to vl$

instantiation	$x[\sigma, \tau]$
stream cons	$s \cdot (\sigma, \tau)$
up	$\Uparrow^{vl}_{tm} \sigma,\ \Uparrow^{ty}_{tm} \tau$

Example:

$$
\begin{aligned}
x[\sigma, \tau] &= \tau\ x \qquad (x \in \text{vl}) \\
(\lambda A.\ s)[\sigma, \tau] &= \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}(\sigma, \tau)] \\
\Uparrow^{vl}_{tm} (\sigma, \tau) &= (\sigma, 0_{vl} \cdot \tau \circ (\text{id}_{ty}, \uparrow))
\end{aligned}
$$

- project to correct substitution
- select subvector
- pass binder of a sort

[1] [Kaiser, Schäfer, Stark 17]

Implementation

- As in Autosubst 2: inductive types from HOAS specification
  - dependencies between sorts
  - identify sorts which need variables
  - derive substitution operations and prove lemmas

```
ty, tm, vl : Type

arr : ty -> ty -> ty
all : (ty -> ty) -> ty

app  : tm -> tm -> tm
tapp : tm -> ty -> tm
vt   : vl -> tm

lam  : ty -> (vl -> tm) -> vl
tlam : (ty -> tm) -> vl
```

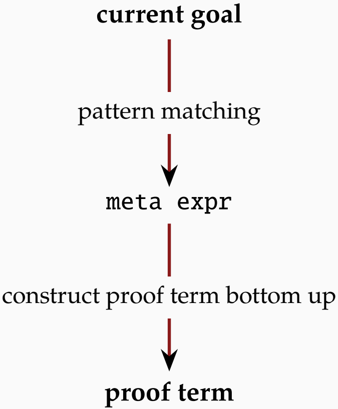
```
inductive ty :
  | var_ty : nat → ty
  | arr : ty → ty → ty
  | arr : ty → ty
inductive tm :
  | app : tm → tm → tm
  | tapp : tm → ty → tm
  | vt : vl → tm
with vl:
  | var_vl : nat → vl
  | lam : ty → tm → vl
  | tlam : tm → vl
```

Implementation issues

General problems

- equation compiler does not yet support structural recursion for mutually inductive types
  $\rightarrow$ well founded recursion + custom tactic
- support for mutual meta definitions not implemented yet
  $\rightarrow$ can be bypassed by providing functions with extra argument

Generating proof terms



Generating proof terms

current goal

pattern matching

`meta expr`

construct proof term bottom up

proof term

$$\dfrac{\dfrac{}{s = s'}\,\text{refl} \qquad \dfrac{}{t = t'}\,\text{refl}}{s\ t = s'\ t'}\,\text{congrApp}$$

Generating proof terms

current goal

pattern matching

`meta expr`

construct proof term bottom up

proof term

```
normalize_term :
  | (s t) = (normalize_term s)
     (normalize_term t)
  | (λ s) = λ(normalize_term s)
  | s[σ] = normalize_inst
     (normalize_term s)
     (normalize_subst σ)
  | n = refl n
```

Generating proof terms

current goal

pattern matching

`meta expr`

construct proof term bottom up

proof term

```
normalize_subst :
  | s · σ = normalize_cons
     (normalize_term s)
     (normalize_subst σ)
  | ⇑ σ = normalize_up
     (normalize_subst σ)
  | σ ∘ τ = normalize_cmp
     (normalize_subst σ)
     (normalize_subst τ)
  | σ = refl σ
```

Generating proof terms

current goal

pattern matching

`meta expr`

construct proof term bottom up

proof term

Example:

```
lemma someGoal (s t σ) :
   (app s t).[σ] =
    app (s.[σ]) (t.[σ])
```

```
nm_term :
 |‘((%%s).[%%σ])
    = nm_inst (nm_term s) (nm_subst σ)
 | ‘(λ %%s)
    = λ (nm_tm s)
 | (...)
```

```
λ s t σ, rwlemma1
 (congr_app (refl s) (refl σ))
 (congr_app (refl t) (refl σ))
```

Generating proof terms

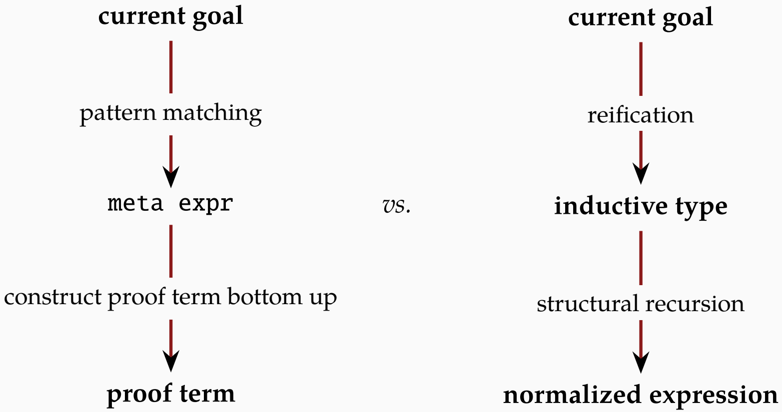
Advantages

- avoid some of the overhead of rewriting
- not tied to functional extensionality

Problems

- pattern matching on meta type `expr` inefficient,
  causes problems with equation compiler
  $\rightarrow$ too inefficient for e.g. system F

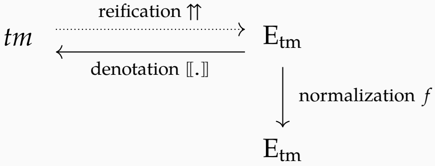
Reflection


current goal
pattern matching
meta expr
construct proof term bottom up
proof term
vs.
current goal
reification
inductive type
structural recursion
normalized expression

Reflection

- synthetic type captures the class of substitution expressions

$$
\begin{aligned}
s, t \in E_{tm} &:= \mu \mid s\ t \mid \lambda\ s \mid s[\sigma] \mid s \\
\sigma, \tau \in E_{subst} &:= \mu \mid \uparrow \mid \mathrm{id} \mid \Uparrow \sigma \\
&\quad \mid s \cdot \sigma \mid \sigma \circ \tau \mid \sigma
\end{aligned}
$$



- **soundness**: $\forall\, e,\ e'.\, f\ e = e' \rightarrow [\![e]\!] = [\![e']\!]$

Reflection

Advantages

- normalization as a function on inductive type
- verified decision procedure

Problems

- termination of normalization function
  $\rightarrow$ first for renamings

Roadmap

So far:

- Lemmas and automation

Up next:

- finish approach 2
- normalization in larger context
- case study

Outlook

Future versions of Lean:

- Lean 4: implemented in Lean itself (elaborator, expressions ...) [1]
- hopefully improved support for mutual recursion at some point

[1] `http://leanprover.github.io/talks/LeanAtGalois.pdf`

References i

Abadi, M., Cardelli, L., Curien, P.-L., and Lévy, J.-J. (1991).
Explicit substitutions.
In *Journal of Functional Programming*, volume 1(4), 375-416.

de Moura, L. M., Kong, S., Avigad, J., van Doorn, F., and von Raumer, J. (2015).
The lean theorem prover (system description).
In *CADE*, volume 9195, pages 378–388.

Ebner, G., Ullrich, S., Roesch, J., Avigad, J., and de Moura, L. (2017).
A metaprogramming framework for formal verification.
Proc. ACM Program. Lang., 1(ICFP).

References ii

Kaiser, J., Schäfer, S., and Stark, K. (2017).
Autosubst 2: Towards reasoning with multi-sorted de bruijn terms and vector substitutions.
In *LFMTP*, LFMTP.

Schäfer, S., Smolka, G., and Tebbi, T. (2015a).
Completeness and decidability of de bruijn substitution algebra in coq.
In *Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India*, pages 67–73. ACM.

Schäfer, S., Tebbi, T., and Smolka, G. (2015b).
Autosubst: Reasoning with de bruijn terms and parallel substitutions.
In Zhang, X. and Urban, C., editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015*, LNAI. Springer-Verlag.