

Experience report: Formalizing F^{\forall} using the locally nameless approach

Enjoying and suffering with locally nameless

Benoît Montagu

<http://gallium.inria.fr/~montagu/>
INRIA Paris-Rocquencourt

5th Workshop on Mechanizing Metatheory
Baltimore, Maryland — September 25th, 2010

Yet another formal proof on metatheory

Core F^\forall (F-zip)

- ▶ A language featuring *open* existential types
- ▶ Proofs: *subject reduction* and *progress*
- ▶ Tools: Coq v8.2pl1, Ott, LNgem
- ▶ Sources available online
<http://gallium.inria.fr/~montagu/proofs/FzipCore/>



Benoît Montagu and Didier Rémy.

Modeling abstract types in modules with open existential types.

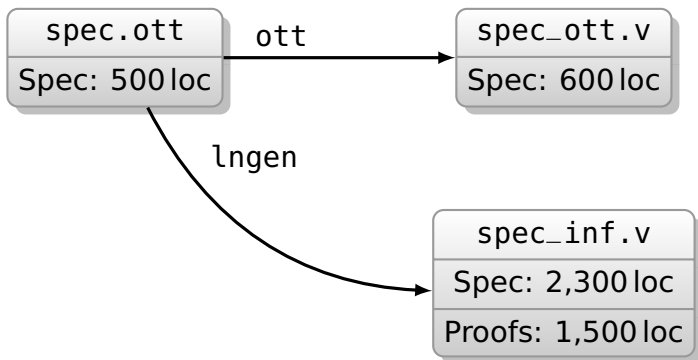
In Proceedings of ACM SIGPLAN Symposium on Principles of Programming Languages, January 2009.

Workflow and statistics

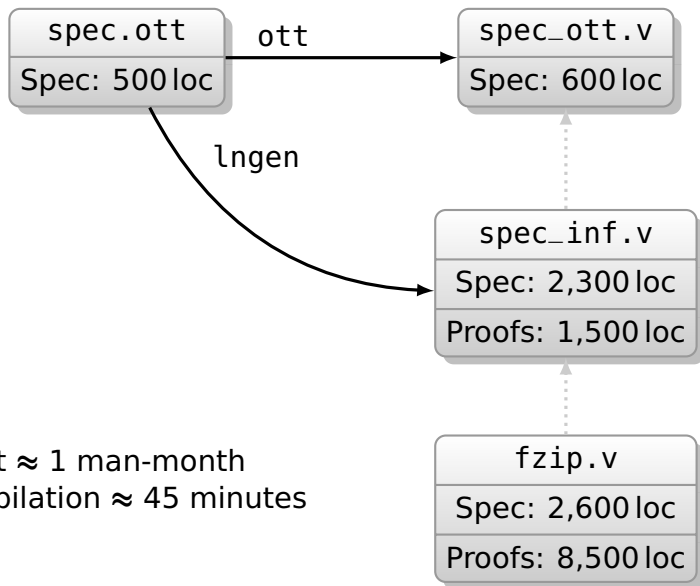
spec.ott

Spec: 500loc

Workflow and statistics

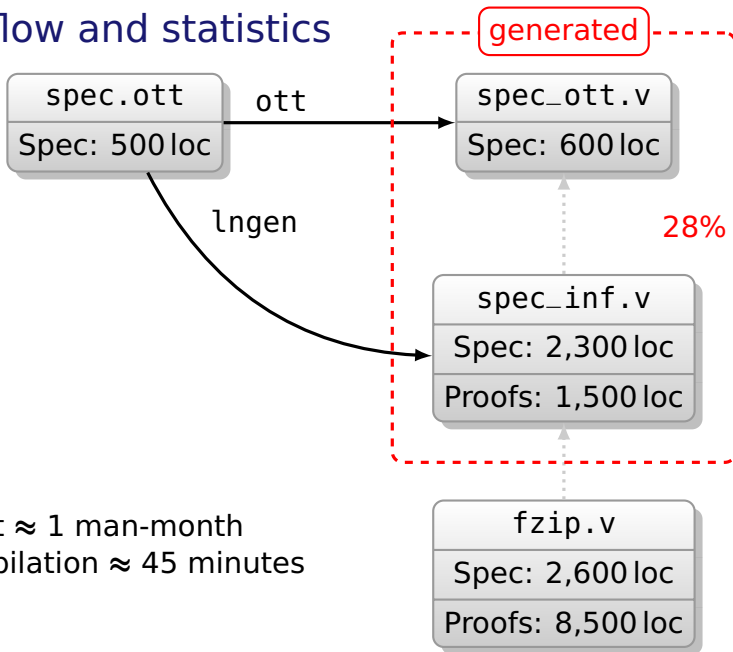


Workflow and statistics



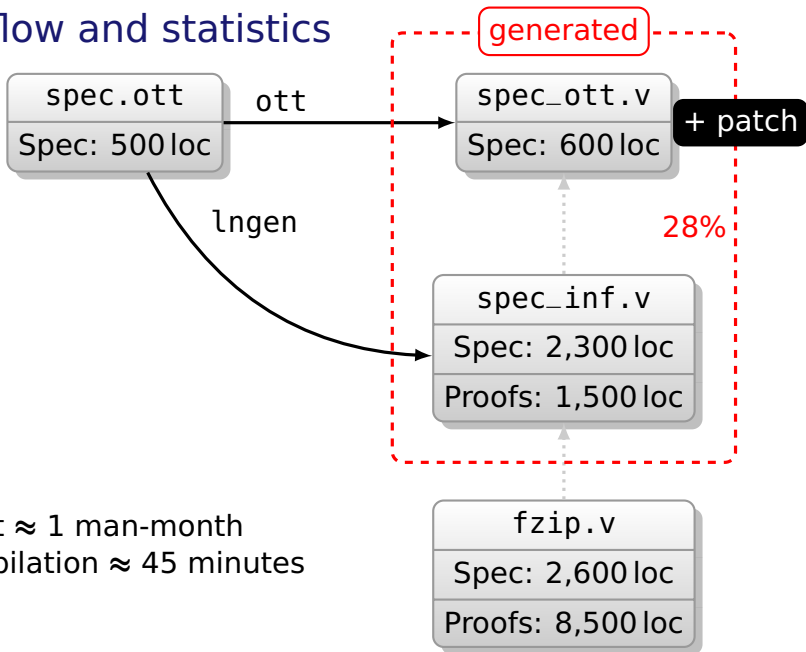
Effort \approx 1 man-month
Compilation \approx 45 minutes

Workflow and statistics



Effort \approx 1 man-month
Compilation \approx 45 minutes

Workflow and statistics



Effort \approx 1 man-month
Compilation \approx 45 minutes

Coq + LNgen + Ott = the perfect toolbox?

Yes!!

- ▶ Ott: automatic encoding with cofinite quantification
- ▶ LNgen: automatic generation of the infrastructure
- ▶ UPenn library: high level of automatization

Coq + LNgen + Ott = the perfect toolbox?

Yes!!

- ▶ Ott: automatic encoding with cofinite quantification
- ▶ LNgen: automatic generation of the infrastructure
- ▶ UPenn library: high level of automatization

But...

- ▶ As long as you stay in the *standard* cases
- ▶ And Core F^{\forall} has many non-standard features!

The peculiarities of F^{\forall}

With respect to binding:

- ▶ Syntax: not closed under substitution of types
- ▶ Reduction under binders: need for renaming lemmas
- ▶ Nested binders in reduction rules
- ▶ Extrusion of binders
- ▶ Exchange of binders

Other particularities:

- ▶ Swapping of bindings in contexts
- ▶ Linearity in contexts

The peculiarities of F^{\downarrow}

currently not supported by LNgem

With respect to binding:

- ▶ Syntax: not closed under substitution of types
- ▶ Reduction under binders: need for renaming lemmas
- ▶ Nested binders in reduction rules
- ▶ Extrusion of binders
- ▶ Exchange of binders

Other particularities:

- ▶ Swapping of bindings in contexts
- ▶ Linearity in contexts

The peculiarities of F^{\downarrow}

With respect to binding:

- ▶ Syntax: not closed under substitution of types
- ▶ Reduction under binders: need for renaming lemmas
- ▶ Nested binders in reduction rules
- ▶ Extrusion of binders
- ▶ Exchange of binders

currently not supported by LNgen

no support in Coq for easier proofs

Other particularities:

- ▶ Swapping of bindings in contexts
- ▶ Linearity in contexts

The peculiarities of F^{\downarrow}

With respect to binding:

- ▶ Syntax: not closed under substitution of types
- ▶ Reduction under binders: need for renaming lemmas
- ▶ Nested binders in reduction rules
- ▶ Extrusion of binders
- ▶ Exchange of binders

currently not supported by LNgem

no support in Coq for easier proofs

Other particularities:

- ▶ Swapping of bindings in contexts
- ▶ Linearity in contexts

currently not supported by Ott

The peculiarities of F^λ

With respect to binding:

- ▶ Syntax: not closed under substitution of types
- ▶ Reduction under binders: need for renaming lemmas
- ▶ Nested binders in reduction rules
- ▶ Extrusion of binders
- ▶ Exchange of binders

currently not supported by LNgén

no support in Coq for easier proofs

Other particularities:

- ▶ Swapping of bindings in contexts
- ▶ Linearity in contexts

No link with binding issues

currently not supported by Ott

The peculiarities of F^λ

With respect to binding:

- ▶ Syntax: not closed under substitution of types
- ▶ Reduction under binders: need for renaming lemmas
- ▶ Nested binders in reduction rules
- ▶ Extrusion of binders
- ▶ Exchange of binders

Other particularities:

- ▶ Swapping of bindings in contexts
- ▶ Linearity in contexts

Working with cofinite quantification

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x. M : \tau_1 \rightarrow \tau_2}$$

On paper

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda(\text{close } x \ M) : \tau_1 \rightarrow \tau_2}$$

Closed form

$$\frac{\Gamma, x : \tau_1 \vdash \text{open } M \ x : \tau_2 \quad x \# M}{\Gamma \vdash \lambda M : \tau_1 \rightarrow \tau_2}$$

Existentially
opened form

$$\frac{\forall x \quad \Gamma, x : \tau_1 \vdash \text{open } M \ x : \tau_2}{\Gamma \vdash \lambda M : \tau_1 \rightarrow \tau_2}$$

Universally
opened form

Working with cofinite quantification

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x. M : \tau_1 \rightarrow \tau_2}$$

On paper

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda(\text{close } x \ M) : \tau_1 \rightarrow \tau_2}$$

Closed form

$$\frac{\Gamma, x : \tau_1 \vdash \text{open } M \ x : \tau_2 \quad x \# M}{\Gamma \vdash \lambda M : \tau_1 \rightarrow \tau_2}$$

Existentially opened form

$$\frac{\forall x \notin \mathcal{L} \quad \Gamma, x : \tau_1 \vdash \text{open } M \ x : \tau_2}{\Gamma \vdash \lambda M : \tau_1 \rightarrow \tau_2}$$

Cofinitely
opened form

Working with cofinite quantification

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x. M : \tau_1 \rightarrow \tau_2}$$

On paper

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda(\text{close } x \ M) : \tau_1 \rightarrow \tau_2}$$

Closed form

$$\frac{\Gamma, x : \tau_1 \vdash \text{open } M \ x : \tau_2 \quad x \# M}{\Gamma \vdash \lambda M : \tau_1 \rightarrow \tau_2}$$

Existentially opened form

$$\frac{\forall x \notin \mathcal{L} \quad \Gamma, x : \tau_1 \vdash \text{open } M \ x : \tau_2}{\Gamma \vdash \lambda M : \tau_1 \rightarrow \tau_2}$$

Cofinitely opened form

Stronger induction principle

Working with cofinite quantification

Example: β -reduction

$$(\lambda x. M) v \rightsquigarrow M[x \leftarrow v]$$

On paper

$$\frac{\text{lc } (\lambda M) \quad \text{lc } v}{(\lambda M) v \rightsquigarrow \text{open } M v}$$

Recommended
(produced by Ott)

$$\frac{\text{lc } (\lambda M) \quad \text{lc } v \quad \forall x \notin \mathcal{L}, \quad M' = (\text{open } M x)[x \leftarrow v]}{(\lambda M) v \rightsquigarrow M'}$$

Alternate

$$\frac{\text{lc } M \quad \text{lc } v}{(\lambda (\text{close } x M)) v \rightsquigarrow M[x \leftarrow v]}$$

Closed form

Working with cofinite quantification

Extrusions

$$(M_1, \nu\alpha. M_2) \rightsquigarrow \nu\alpha. (M_1, M_2)$$

On paper

$$\frac{\forall\alpha \notin \mathcal{L}, \text{ open } M'_1 \ \alpha = M_1}{(M_1, \nu M_2) \rightsquigarrow \nu (M'_1, M_2)}$$

Using cofinite
quantification

(conditions for local closure are omitted)

Working with cofinite quantification

Extrusions

$$(M_1, \nu \alpha. M_2) \rightsquigarrow \nu \alpha. (M_1, M_2)$$

On paper

proof

$$\frac{\forall \alpha \notin \mathcal{L}, \text{ open } M'_1 \ \alpha = M_1}{(M_1, \nu M_2) \rightsquigarrow \nu (M'_1, M_2)}$$

Using cofinite
quantification

$$\frac{\alpha \# M_1}{(M_1, \nu(\text{close } \alpha \ M_2)) \rightsquigarrow \nu(\text{close } \alpha \ (M_1, M_2))}$$

Closed form

(conditions for local closure are omitted)

Working with cofinite quantification

Exchanging binders

$$\nu\beta. \nu\alpha. M \rightsquigarrow \nu\alpha. \nu\beta. M$$

On paper

$$\frac{\forall\beta \notin \mathcal{L}, \forall\alpha \notin \mathcal{L} \cup \{\beta\}, \text{open}^0(\text{open}^1 M \beta) \alpha = \text{open}^0(\text{open}^1 M' \alpha) \beta}{\nu(\nu M) \rightsquigarrow \nu(\nu M')}$$

Using cofinite
quantification

(conditions for local closure are omitted)

Working with cofinite quantification

Exchanging binders

$$\nu\beta. \nu\alpha. M \rightsquigarrow \nu\alpha. \nu\beta. M$$

On paper

$$\frac{\forall\beta \notin \mathcal{L}, \forall\alpha \notin \mathcal{L} \cup \{\beta\}, \text{open}^0(\text{open}^1 M \beta) \alpha = \text{open}^0(\text{open}^1 M' \alpha) \beta}{\nu(\nu M) \rightsquigarrow \nu(\nu M')}$$

Using cofinite quantification

proof



$$\begin{aligned} &\nu(\text{close } \beta (\nu (\text{close } \alpha M))) \\ &\rightsquigarrow \nu(\text{close } \alpha (\nu (\text{close } \beta M))) \end{aligned}$$

Closed form

(conditions for local closure are omitted)

Working with cofinite quantification

Exchanging binders

Lemma: For any α, β, γ, M and n ,
$$\text{open}^n \beta \left(\text{open}^{n+1} \gamma \left(\text{close}^{n+1} \alpha M \right) \right) =$$
$$\text{open}^n \gamma \left(\text{open}^{n+1} \beta \left(\text{close}^n \alpha M \right) \right)$$

On paper

$$\frac{\forall \beta \notin \mathcal{L}, \forall \alpha \notin \mathcal{L} \cup \{\beta\}, \text{open}^0 \left(\text{open}^1 M \beta \right) \alpha = \text{open}^0 \left(\text{open}^1 M' \alpha \right) \beta}{\nu(\nu M) \rightsquigarrow \nu(\nu M')}$$

Using cofinite quantification

proof

$$\nu(\text{close } \beta (\nu (\text{close } \alpha M)))$$
$$\rightsquigarrow \nu(\text{close } \alpha (\nu (\text{close } \beta M)))$$

Closed form

(conditions for local closure are omitted)

Lessons learned

Sometimes your syntax is *not* closed under substitution!

- ▶ I had to consider a *larger* syntax

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Lessons learned

Sometimes your syntax is *not* closed under

Tools should support that!

- ▶ I had to consider a *larger* syntax

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for your rules

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for your rules

We need tool support!

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for your rules

We need tool support!

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for

We need tool support!

Automation to derive them?

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for

We need tool support!

Automation to derive them?

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Use closed forms!

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for

We need tool support!

Automation to derive them?

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Use closed forms!

Other problems

- ▶ How to define functions over terms with binders?
- ▶ How much do you trust your translator (Ott)?

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for

We need tool support!

Automation to derive them?

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Use closed forms!

Other problems

- ▶ How to define functions over terms with binders
- ▶ How much do you trust your translator (O)

Should be as simple as possible!

Lessons learned

Sometimes your syntax is *not* closed under

- ▶ I had to consider a *larger* syntax

Tools should support that!

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for

We need tool support!

Automation to derive them?

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Use closed forms!

Other problems

- ▶ How to define functions over terms with binders
- ▶ How much do you trust your translator (O)

Should be as simple as possible!

Lessons learned

Sometimes your syntax is *not* closed under substitution!

- ▶ I had to consider a *larger* syntax

Eventually, renaming lemmas are needed

- ▶ To prove existence lemmas
- ▶ Renaming lemmas are a sanity check for your rules

Cofinite quantification

- ▶ Provides strong induction principles
- ▶ Can decrease the readability of your specs!
- ▶ Can expose the user to the implementation of binders!

Other problems

- ▶ How to define functions over terms with binders?
- ▶ How much do you trust your translator (Ott)?

Mechanized metatheory for the masses?

(if you want to use Coq as your proof assistant)

Much easier than a few years ago

- ▶ Keep up the good work!

But...

- ▶ The tools still need improvements
- ▶ The techniques still need exploration

Mechanized metatheory for the masses?

(if you want to use Coq as your proof assistant)

Much easier than a few years ago

- ▶ Keep up the good work!

But...

- ▶ The tools still need improvements
- ▶ The techniques still need exploration

Claim

- ▶ Since your proofs are verified, only one requirement remains: **trust your specifications**

Mechanized metatheory for the masses?

(if you want to use Coq as your proof assistant)

Much easier than a few years ago

- ▶ Keep up the good work!

But...

- ▶ The tools still need improvements
- ▶ The techniques still need exploration

Claim

- ▶ Since your proofs are verified, only one requirement remains: **trust your specifications**
- ▶ **Formal specifications should be as close as possible to their paper versions**

Thanks!

<http://gallium.inria.fr/~montagu/proofs/FzipCore/>

Extra material



Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, , and Stephanie Weirich.

Engineering formal metatheory.

In Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 3–15. ACM, 2008.



Brian Aydemir and Stephanie Weirich.

LNgen: Tool support for locally nameless representations.
Draft.



Benoît Montagu and Didier Rémy.

Modeling abstract types in modules with open existential types.

In Proceedings of ACM SIGPLAN Symposium on Principles of Programming Languages. ACM, January 2009.



Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strnivsa.

Ott: Effective tool support for the working semanticist.

JFP, 20(1):71–122, 2010.

Working with cofinite quantification

Nested binders

$$\frac{\beta \# \tau}{\nu \beta. \Sigma \langle \beta \rangle (\alpha = \tau) M \rightsquigarrow M[\alpha \leftarrow \beta][\beta \leftarrow \tau]}$$

On paper

$$\frac{\begin{array}{l} \forall \beta \notin \mathcal{L}, \forall \alpha \notin \mathcal{L} \cup \{\beta\}, \\ \text{open}(\Sigma 0 \tau M) \beta = \Sigma \beta (\text{open } \tau \beta) M_1 \rightarrow \\ \text{open } M_1 \alpha = M_2 \rightarrow \\ \beta \notin \text{open } \tau \beta \\ \wedge M' = M_2 [\beta \leftarrow \alpha][\alpha \leftarrow \text{open_type } \tau \beta] \end{array}}{\nu(\Sigma 0 \tau M) \rightsquigarrow M'}$$

Using cofinite quantification

$$\frac{\beta \# \tau}{\nu(\text{close } \beta (\Sigma \beta \tau (\text{close } \alpha M))) \rightsquigarrow M[\alpha \leftarrow \beta][\beta \leftarrow \tau]}$$

Closed form

(conditions for local closure are omitted)