데이터타입 제너릭 프로그래밍기법을 Coq 증명에 활용한 라이브러리 (Part 2: How to Use GMeta)

이계식

(joint work with Bruno C.d.S. Oliveira, 조성근 , 이광근)

서울대 , ROPAS

ROSAEC Center 4th Workshop

Outline

Outline

Coq, a Theorem Prover

- A formal language providing
  - mathematical definitions,
  - executable algorithms and theorems,
- A formal proof management system with
  - environments for interactive development of machine-checked proofs.
- Based on Calculus of Inductive Constructions
- Used to formalize proofs in a variety of fields
  - programming languages
  - mathematics
  - ...

Coq, a Theorem Prover

- A formal language providing
  - mathematical definitions,
  - executable algorithms and theorems,
- A formal proof management system with
  - environments for interactive development of machine-checked proofs.
- Based on Calculus of Inductive Constructions
- Used to formalize proofs in a variety of fields
  - programming languages
  - mathematics
  - ...

Outline

What You Usually Start with

- Choice of a theorem prover
  - Coq, Isabelle\HOL, Agda, ACL2, Nuprl, PVS, Mizar, ...
- Choice of a representation style
  - de Bruijn indices
  - Locally nameless approach
  - Locally-named approach
  - Nominal approah
  - Higher-Order Abstract Syntax
  - ...
- Specification of the target language
- There are many other choices to be made.

What You Usually Start with

- Choice of a theorem prover
  - Coq, Isabelle\HOL, Agda, ACL2, Nuprl, PVS, Mizar, ...
- Choice of a representation style
  - de Bruijn indices
  - Locally nameless approach
  - Locally-named approach
  - Nominal approah
  - Higher-Order Abstract Syntax
  - ...
- Specification of the target language
- There are many other choices to be made.

What You Usually Start with

- Choice of a theorem prover
  - Coq, Isabelle\HOL, Agda, ACL2, Nuprl, PVS, Mizar, ...
- Choice of a representation style
  - de Bruijn indices
  - Locally nameless approach
  - Locally-named approach
  - Nominal approah
  - Higher-Order Abstract Syntax
  - ...
- Specification of the target language
- There are many other choices to be made.

What You Usually Start with

- Choice of a theorem prover
  - Coq, Isabelle\HOL, Agda, ACL2, Nuprl, PVS, Mizar, ...
- Choice of a representation style
  - de Bruijn indices
  - Locally nameless approach
  - Locally-named approach
  - Nominal approah
  - Higher-Order Abstract Syntax
  - ...
- Specification of the target language
- There are many other choices to be made.

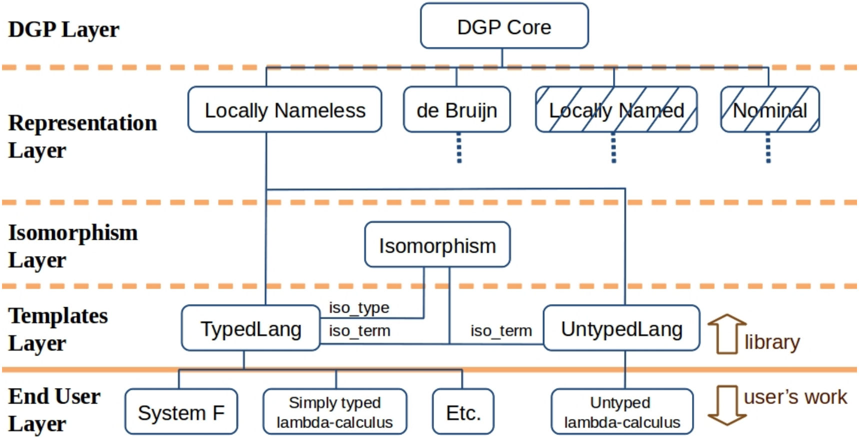
Outline

An Example

Outline

1. About Coq
   - What is a Theorem Prover
2. Approach to a Theorem Prover
   - What People Usually Do
   - POPLmark Challenge
3. Stepwise Use of GMeta
   - Overview of GMeta Library
   - Four Steps in Using GMeta

GMeta Library



DGP Core

Universe of Representations

```
Inductive Rep : Type :=
| UNIT  : Rep
| CONST : Rep -> Rep
| REPR  : Rep -> Rep
| PLUS  : Rep -> Rep -> Rep
| PROD  : Rep -> Rep -> Rep
| BIND  : Rep -> Rep -> Rep
| REC   : Rep.
```

Meta-Library for Locally Nameless Style

- bfsubst_var_intro

$$a \notin FV(T) \rightarrow \forall k, \{U \backslash k\}T = [U \backslash a](\{a \backslash k\}T)$$

- wfT_wf

$$\mathsf{wfT}_{r_0}(T) \rightarrow \forall k\,(U : Interpret\, r_0), T = \{U \backslash k\}T$$

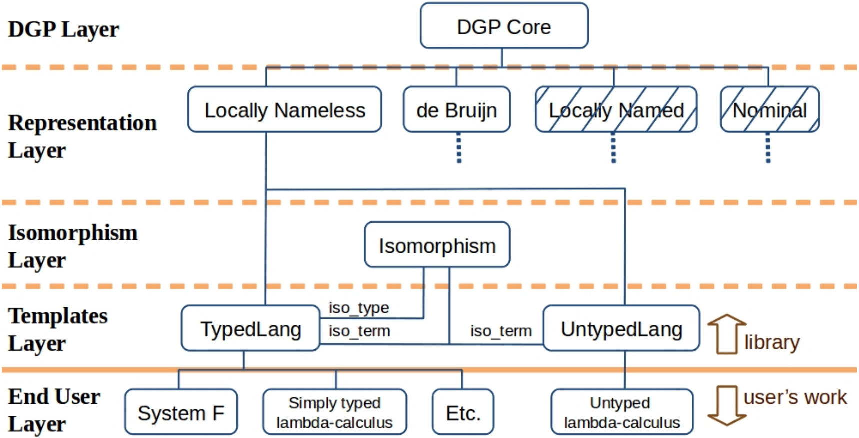
- bfsubst_permutation_core

$$\mathsf{wfT}_{r_1} U \rightarrow \forall k, \{[U \backslash a]V \backslash k\}([U \backslash a]T) = [U \backslash a](\{V \backslash k\}T)$$

- wfT_bsubst_hetero

$$\mathsf{wfT}_{r_0}(\{a \backslash k\}_{r_1} T) \rightarrow r_0 \neq r_1 \rightarrow \mathsf{wfT}_{r_0}(T)$$

GMeta Library



Outline

First Step: Defining Syntax

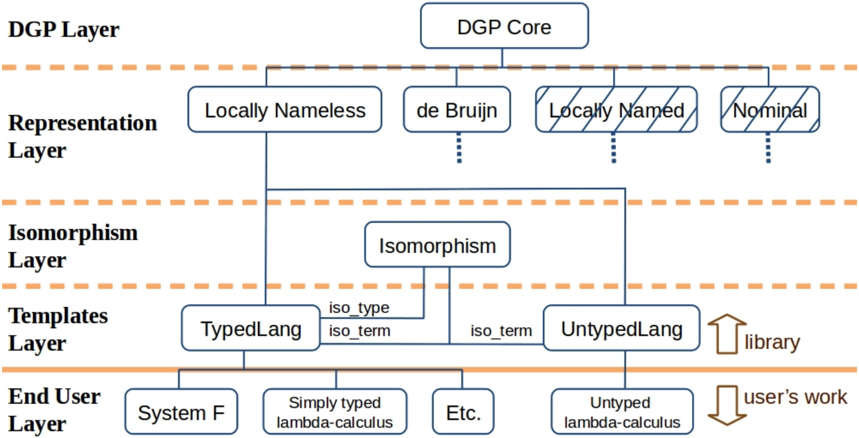
type

```
Inductive typ : Type :=
| typ_var : atm -> typ
| typ_arrow : typ -> typ -> typ.
```

term

```
Inductive trm : Set :=
| trm_bvar : nat -> trm
| trm_fvar : var -> trm
| trm_abs : trm -> trm
| trm_app : trm -> trm -> trm.
```

GMeta Library



Second Step: Defining Isomorphisms

iso_trm

```
T := trm
R := PLUS (BIND REC REC) (PROD REC REC)
From : T -> R
To : R -> T
To_From : ∀ (t:T), To (From t) = t
From_To : ∀ (t:Interpret R), From (To t) = t
```

There are 2 ways of creating isomorphism modules

- Using the generation tool
- Manual creation

Automatic Generation of Isomorphisms

Isomorphism for type

```
(*@Iso STLC_typ_iso *)

Inductive typ :=
| typ_var     : nat -> typ
| typ_arrow   : typ -> typ -> typ.
```

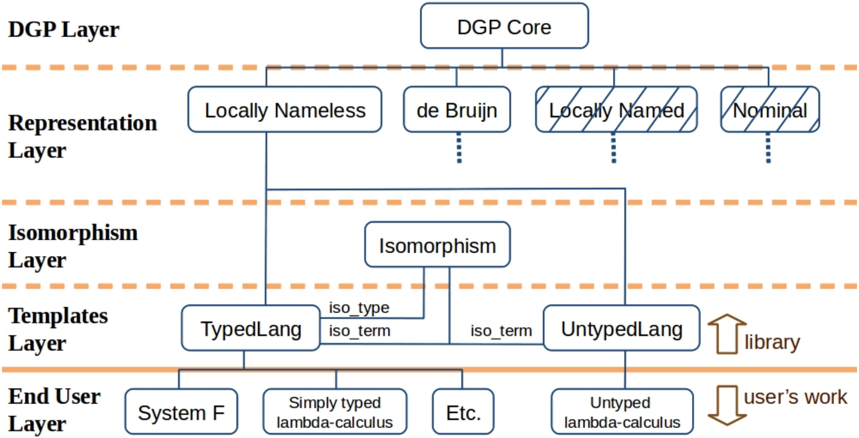
Automatic Generation of Isomorphisms

Isomorphism for term

```
(*@Iso STLC_trm_iso {
   Parameter  trm_fvar,
   Variable   trm_bvar,
   Binder     trm_abs
}*)

Inductive trm :=
| trm_bvar : nat -> trm
| trm_fvar : nat -> trm
| trm_abs  : trm -> trm
| trm_app  : trm -> trm -> trm.
```

GMeta Library



Third Step: Importing the Infrastructure

```
Module Import M := Iso_Infra Iso_trm Iso_typ.
```

Saved Lemmas

Fixpoint open_rec

Fixpoint subst

Fixpoint fv

Lemma subst_fresh

Lemma subst_intro

...

Definition Tbsubst

Definition Tfsubst

Definition Tfv

Lemma Tfsubst_fresh

Lemma Tbfsubst_var_intro

Fourth Step: Starting with Formalization

Typing rules

```
Inductive typing : env -> trm -> typ -> Prop :=
| typing_var : forall E x T,
  uniq E ->
  binds x T E ->
  E |- (trm_fvar x)  : T

| typing_abs : forall L E U T t1,
  (∀ x, x ∉ L ->(x , U) :: E |- [0 / x] t1  :  T) ->
  E |- (trm_abs t1)  :  (typ_arrow U T)

| typing_app : forall S T E t1 t2,
  E  |-  t1  :  (typ_arrow S T)  ->
  E  |-  t2  :  S  ->
  E  |-  (trm_app t1 t2)  :  T
```

Fourth Step: Starting with Formalization

Typing rules with substitution

```
Lemma typing_subst : forall E U F t T z u,
  (E ++ (z , U) ++ F)  |-  t : T ->
  F  |-  u : U  ->
  (E ++ F) |-  [u/z] t : T.
```

Fourth Step: Starting with Formalization

Preservation and progress

```
Definition preservation := forall E t t' T,
  E |- t : T ->
  t ~> t' ->
  E |- t' : T.

Definition progress := forall t T,
  empty_env |- t : T ->
  (value t  exists t', t ~> t').
```

Final lemmas

```
Lemma preservation_result : preservation.

Lemma progress_result : progress.
```

Fourth Step: Starting with Formalization

Preservation and progress

```
Definition preservation := forall E t t' T,
  E |- t : T ->
  t ~> t' ->
  E |- t' : T.

Definition progress := forall t T,
  empty_env |- t : T ->
  (value t  exists t', t ~> t').
```

Final lemmas

```
Lemma preservation_result : preservation.

Lemma progress_result : progress.
```

Case Studies based on Locally Nameless Approach

STLC

	boilerplate	total	ratio
Aydemir et al.	17	31	55%
GMeta basic	7	21	33%
GMeta full	1	15	7%

System $F_{<:}$

	boilerplate	total	ratio
Aydemir et al.	60	93	65%
GMeta basic	25	58	43%
GMeta full	11	45	24%

Thank you!

Questions and Comments?