Towards Extraction of Continuity Moduli in Coq

Yannick Forster, Dominik Kirst, Florian Steinberg

TYPES'20, Torino, Italy
March 2-4, 2020

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Inria

Background

- Florian works on a Coq library of computable analysis[1]
  - Typical goal: prove some particular $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ continuous
  - Doing this by hand is repetitive and in fact unnecessary, since...
  - Metatheorem: in vanilla Coq, every definable function is continuous

- Yannick works on the MetaCoq project[2]
  - Internal representation of Coq terms with quote and unquote
  - Support for monadic programming to implement metatheorems

- I learned about the continuity theorem for System T at PC'18
  - Translated Escardó's Agda implementation[3] to Coq

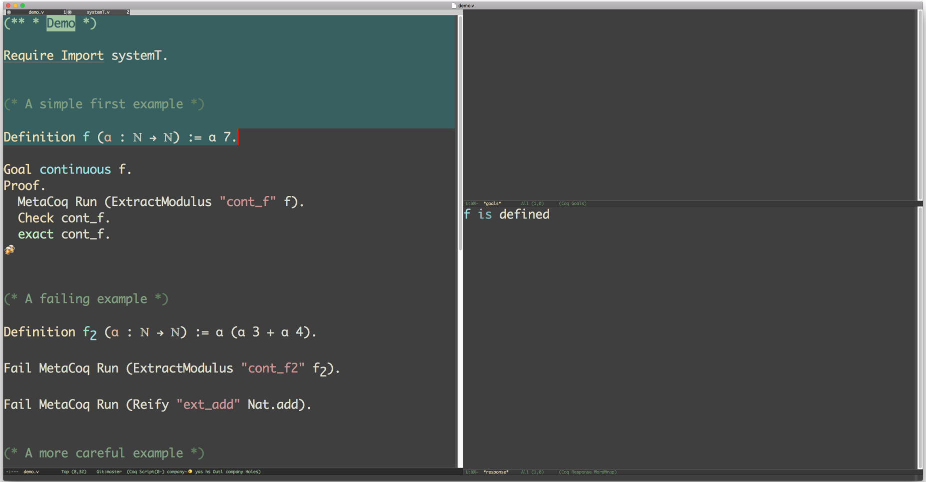
Project idea when we met at PC'19:
Implement a PoC continuity plugin

[1][Steinberg et al. '19] [2][Sozeau et al. '19] [3][Escardó '13]

Demo

```
(** * Demo *)

Require Import systemT.


(* A simple first example *)

Definition f (α : N → N) := α 7.

Goal continuous f.
Proof.
  MetaCoq Run (ExtractModulus "cont_f" f).
  Check cont_f.
  exact cont_f.


(* A failing example *)

Definition f₂ (α : N → N) := α (α 3 + α 4).

Fail MetaCoq Run (ExtractModulus "cont_f2" f₂).

Fail MetaCoq Run (Reify "ext_add" Nat.add).


(* A more careful example *)
```

Demo

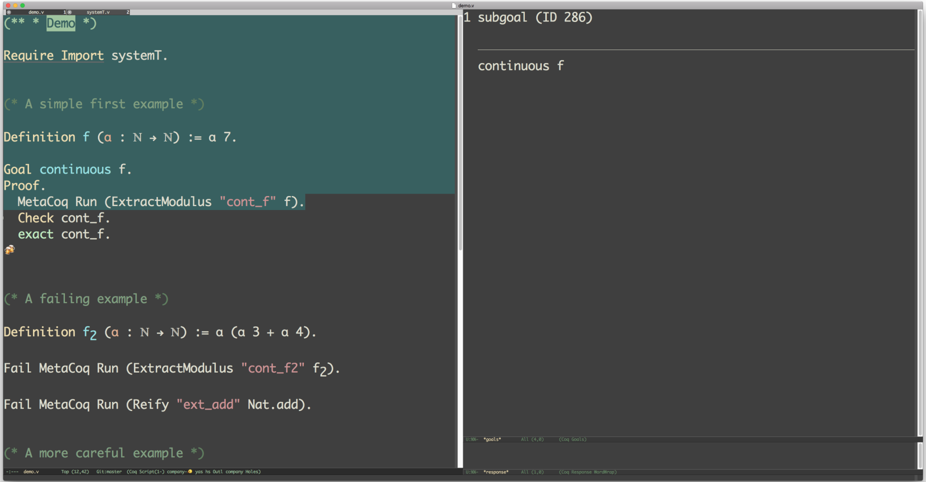

(** * Demo *)
Require Import systemT.
(* A simple first example *)
Definition f (a : N → N) := a 7.
Goal continuous f.
Proof.
MetaCoq Run (ExtractModulus "cont_f" f).
Check cont_f.
exact cont_f.
(* A failing example *)
Definition f2 (a : N → N) := a (a 3 + a 4).
Fail MetaCoq Run (ExtractModulus "cont_f2" f2).
Fail MetaCoq Run (Reify "ext_add" Nat.add).
(* A more careful example *)
f is defined

Demo

```
(** * Demo *)

Require Import systemT.


(* A simple first example *)

Definition f (a : N → N) := a 7.

Goal continuous f.
Proof.
  MetaCoq Run (ExtractModulus "cont_f" f).
  Check cont_f.
  exact cont_f.


(* A failing example *)

Definition f₂ (a : N → N) := a (a 3 + a 4).

Fail MetaCoq Run (ExtractModulus "cont_f2" f₂).

Fail MetaCoq Run (Reify "ext_add" Nat.add).


(* A more careful example *)
```

```
1 subgoal (ID 286)

────────────────────────────
continuous f
```

Demo


(** * Demo *)
Require Import systemT.
(* A simple first example *)
Definition f (a : N → N) := a 7.
Goal continuous f.
Proof.
MetaCoq Run (ExtractModulus "cont_f" f).
Check cont_f.
exact cont_f.
(* A failing example *)
Definition f2 (a : N → N) := a (a 3 + a 4).
Fail MetaCoq Run (ExtractModulus "cont_f2" f2).
Fail MetaCoq Run (Reify "ext_add" Nat.add).
(* A more careful example *)
1 subgoal (ID 286)
continuous f

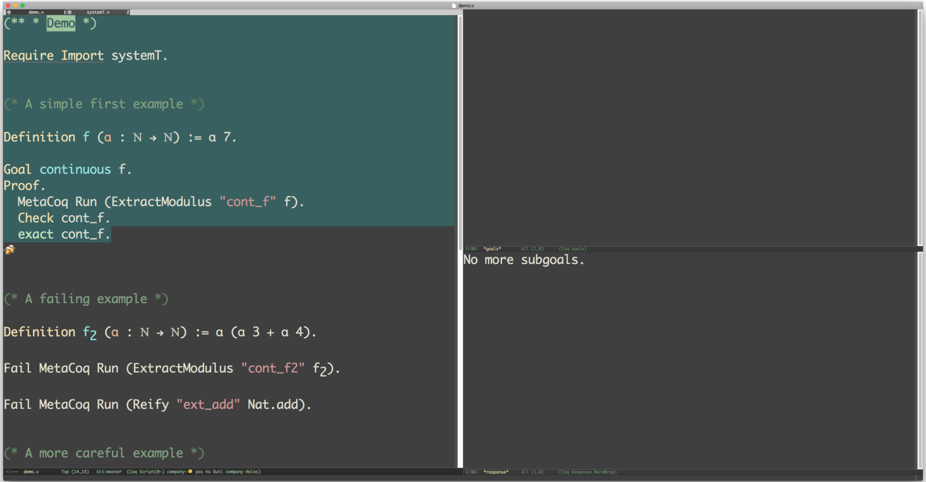
Demo

```
(** * Demo *)

Require Import systemT.


(* A simple first example *)

Definition f (a : N → N) := a 7.

Goal continuous f.
Proof.
  MetaCoq Run (ExtractModulus "cont_f" f).
  Check cont_f.
  exact cont_f.


(* A failing example *)

Definition f₂ (a : N → N) := a (a 3 + a 4).

Fail MetaCoq Run (ExtractModulus "cont_f2" f₂).

Fail MetaCoq Run (Reify "ext_add" Nat.add).


(* A more careful example *)
```

```
1 subgoal (ID 286)

  ______________________________________
  continuous f
```

```
cont_f
     : continuous f
```

Demo



(** * Demo *)
Require Import systemT.
(* A simple first example *)
Definition f (a : N → N) := a 7.
Goal continuous f.
Proof.
MetaCoq Run (ExtractModulus "cont_f" f).
Check cont_f.
exact cont_f.
(* A failing example *)
Definition f2 (a : N → N) := a (a 3 + a 4).
Fail MetaCoq Run (ExtractModulus "cont_f2" f2).
Fail MetaCoq Run (Reify "ext_add" Nat.add).
(* A more careful example *)
No more subgoals.

Demo

```
(** * Demo *)

Require Import systemT.


(* A simple first example *)

Definition f (α : ℕ → ℕ) := α 7.

Goal continuous f.
Proof.
  MetaCoq Run (ExtractModulus "cont_f" f).
  Check cont_f.
  exact cont_f.


(* A failing example *)

Definition f₂ (α : ℕ → ℕ) := α (α 3 + α 4).

Fail MetaCoq Run (ExtractModulus "cont_f2" f₂).

Fail MetaCoq Run (Reify "ext_add" Nat.add).


(* A more careful example *)
```

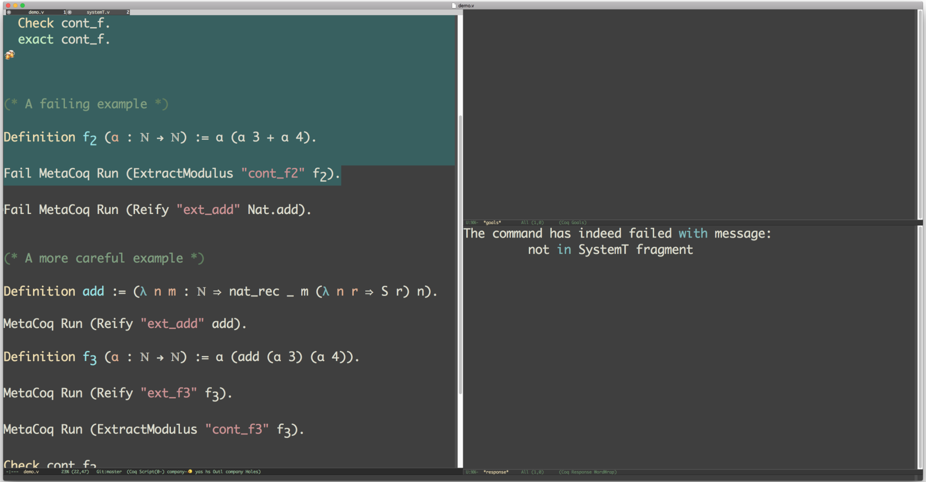
```
Unnamed_thm is defined
```

Demo

```
  Check cont_f.
  exact cont_f.


(* A failing example *)

Definition f₂ (α : ℕ → ℕ) := α (α 3 + α 4).

Fail MetaCoq Run (ExtractModulus "cont_f2" f₂).

Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : ℕ ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f₃ (α : ℕ → ℕ) := α (add (α 3) (α 4)).

MetaCoq Run (Reify "ext_f3" f₃).

MetaCoq Run (ExtractModulus "cont_f3" f₃).

Check cont_f₃
```

```
f₂ is defined
```

Demo



Check cont_f.
exact cont_f.
(* A failing example *)
Definition f2 (a : N → N) := a (a 3 + a 4).
Fail MetaCoq Run (ExtractModulus "cont_f2" f2).
Fail MetaCoq Run (Reify "ext_add" Nat.add).
(* A more careful example *)
Definition add := (λ n m : N ⇒ nat_rec _ m (λ n r ⇒ S r) n).
MetaCoq Run (Reify "ext_add" add).
Definition f3 (a : N → N) := a (add (a 3) (a 4)).
MetaCoq Run (Reify "ext_f3" f3).
MetaCoq Run (ExtractModulus "cont_f3" f3).
The command has indeed failed with message:
not in SystemT fragment

Demo

```
  Check cont_f.
  exact cont_f.

(* A failing example *)

Definition f2 (a : N → N) := a (a 3 + a 4).

Fail MetaCoq Run (ExtractModulus "cont_f2" f2).

Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : N ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f3 (a : N → N) := a (add (a 3) (a 4)).

MetaCoq Run (Reify "ext_f3" f3).

MetaCoq Run (ExtractModulus "cont_f3" f3).

Check cont_f3
```

```
The command has indeed failed with message:
        not in SystemT fragment
```

Demo

```
Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)
Definition add := (λ n m : ℕ ⇒ nat_rec _ m (λ n r ⇒ S r) n).
MetaCoq Run (Reify "ext_add" add).
Definition f₃ (α : ℕ → ℕ) := α (add (α 3) (α 4)).
MetaCoq Run (Reify "ext_f3" f₃).
MetaCoq Run (ExtractModulus "cont_f3" f₃).
Check cont_f₃.

(* Modulus extraction *)
Print continuous.
Compute (get_modulus cont_f₃ (λ n ⇒ 3)).
Compute (get_modulus cont_f₃ (λ n ⇒ n)).
```

```
add is defined
```

Demo

```
Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : ℕ ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f3 (α : ℕ → ℕ) := α (add (α 3) (α 4)).

MetaCoq Run (Reify "ext_f3" f3).

MetaCoq Run (ExtractModulus "cont_f3" f3).

Check cont_f3.


(* Modulus extraction *)

Print continuous.

Compute (get_modulus cont_f3 (λ n ⇒ 3)).

Compute (get_modulus cont_f3 (λ n ⇒ n)).
```

```
(ilambda (ilambda (iapp (iapp (iapp (irec N) (ivar F1))
(ilambda (ilambda (iapp isucc (ivar F1))))) (ivar (FS F1)))))
```

Demo

```
Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : ℕ ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f3 (α : ℕ → ℕ) := α (add (α 3) (α 4)).

MetaCoq Run (Reify "ext_f3" f3).

MetaCoq Run (ExtractModulus "cont_f3" f3).

Check cont_f3.

(* Modulus extraction *)

Print continuous.

Compute (get_modulus cont_f3 (λ n ⇒ 3)).

Compute (get_modulus cont_f3 (λ n ⇒ n)).
```

```
f3 is defined
```

Demo

```
Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : ℕ ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f₃ (a : ℕ → ℕ) := a (add (a 3) (a 4)).

MetaCoq Run (Reify "ext_f3" f₃).

MetaCoq Run (ExtractModulus "cont_f3" f₃).

Check cont_f₃.


(* Modulus extraction *)

Print continuous.

Compute (get_modulus cont_f₃ (λ n ⇒ 3)).

Compute (get_modulus cont_f₃ (λ n ⇒ n)).
```

```
(ilambda
   (iapp (ivar F₁)
      (iapp
         (iapp (ilambda (ilambda (iapp (iapp (iapp (irec N)
(ivar F₁)) (ilambda (ilambda (iapp isucc (ivar F₁))))) (ivar
(FS F₁)))))
            (iapp (ivar F₁) (iapp isucc (iapp isucc (iapp
isucc izero))))) (iapp (ivar F₁) (iapp isucc (iapp isucc
(iapp isucc (iapp isucc izero))))))))
```

Demo

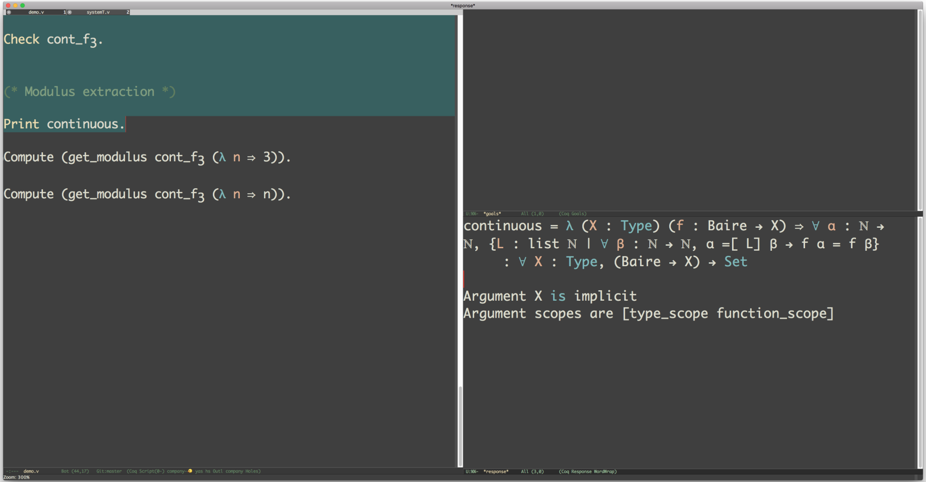
```
Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : ℕ ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f₃ (α : ℕ → ℕ) := α (add (α 3) (α 4)).

MetaCoq Run (Reify "ext_f3" f₃).

MetaCoq Run (ExtractModulus "cont_f3" f₃).

Check cont_f₃.

(* Modulus extraction *)

Print continuous.

Compute (get_modulus cont_f₃ (λ n ⇒ 3)).

Compute (get_modulus cont_f₃ (λ n ⇒ n)).
```

Demo

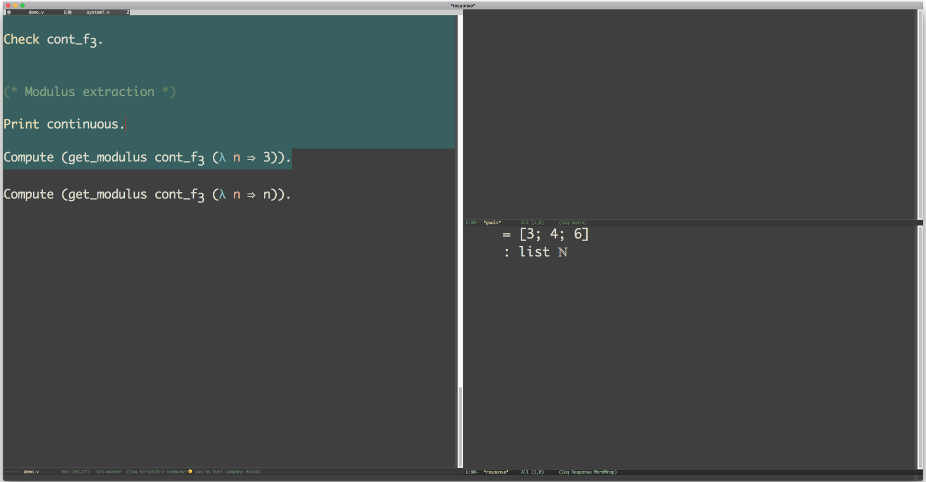
```
Fail MetaCoq Run (Reify "ext_add" Nat.add).

(* A more careful example *)

Definition add := (λ n m : N ⇒ nat_rec _ m (λ n r ⇒ S r) n).

MetaCoq Run (Reify "ext_add" add).

Definition f3 (α : N → N) := α (add (α 3) (α 4)).

MetaCoq Run (Reify "ext_f3" f3).

MetaCoq Run (ExtractModulus "cont_f3" f3).

Check cont_f3.


(* Modulus extraction *)

Print continuous.

Compute (get_modulus cont_f3 (λ n ⇒ 3)).

Compute (get_modulus cont_f3 (λ n ⇒ n)).
```

```
cont_f3
     : continuous f3
```

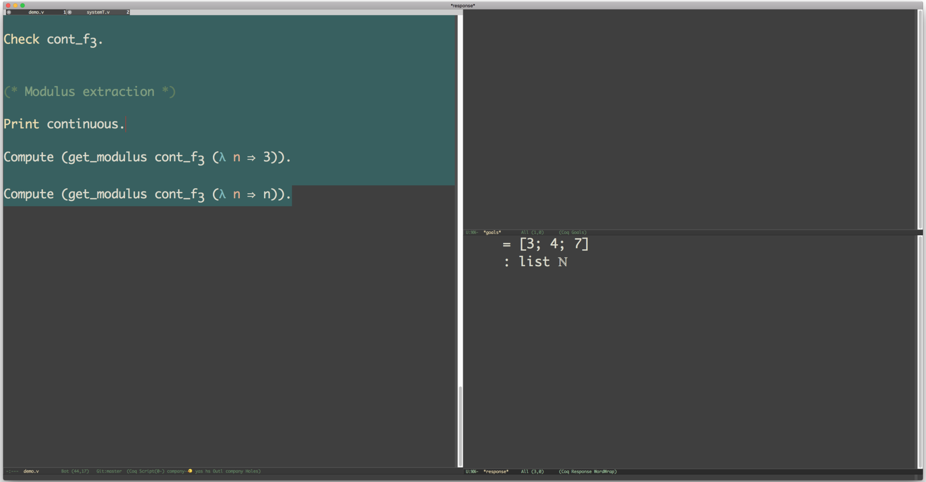
Demo


Check cont_f3.
(* Modulus extraction *)
Print continuous.
Compute (get_modulus cont_f3 (λ n ⇒ 3)).
Compute (get_modulus cont_f3 (λ n ⇒ n)).
continuous = λ (X : Type) (f : Baire → X) ⇒ ∀ α : ℕ → ℕ, {L : list ℕ | ∀ β : ℕ → ℕ, α =[L] β → f α = f β}
: ∀ X : Type, (Baire → X) → Set
Argument X is implicit
Argument scopes are [type_scope function_scope]

Demo


Check cont_f3.
(* Modulus extraction *)
Print continuous.
Compute (get_modulus cont_f3 (λ n ⇒ 3)).
Compute (get_modulus cont_f3 (λ n ⇒ n)).
= [3; 4; 6]
: list N

Demo


Check cont_f3.
(* Modulus extraction *)
Print continuous.
Compute (get_modulus cont_f3 (λ n ⇒ 3)).
Compute (get_modulus cont_f3 (λ n ⇒ n)).
= [3; 4; 7]
: list ℕ

Plugin Pipeline

Expected input: T-definable functional $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

1. Reify f to the internal MetaCoq representation f_C
2. Try to translate f_C to an untyped T-term f_U
3. Infer a type for f_U to obtain an intrinsically typed T-term f_T
4. Compute the continuity information for f_T
5. Verify that f_T corresponds to f

Gödel's System T

Simply typed lambda calculus with natural numbers and recursors:

$$A, B ::= \mathbb{N} \mid A \to B$$

$$s, t ::= x \mid \lambda x.\, s \mid s\, t \mid 0 \mid S \mid R_A$$

Usual typing rules $\Gamma \vdash s : A$ of simply typed lambda calculus plus:

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}} \qquad \frac{}{\Gamma \vdash S : \mathbb{N} \to \mathbb{N}} \qquad \frac{}{\Gamma \vdash R_A : A \to (\mathbb{N} \to A \to A) \to \mathbb{N} \to A}$$

Natural denotational semantics in type theory:
Judgements $\vdash s : A$ translate to terms $[\![s]\!] : [\![A]\!]$

Continuity of T-Definable Functionals (cf. Escardó '13)

A functional $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is continuous if

- "it accesses only finitely many positions of every input sequence"
- $\forall(\alpha : \mathbb{N} \to \mathbb{N}).\, \Sigma(L : \mathbb{N}^*).\, \forall(\beta : \mathbb{N} \to \mathbb{N}).\, \alpha \approx_L \beta \to f\,\alpha = f\,\beta$ [1]

[1] Projecting out $\mu_f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}^*$ yields the modulus of continuity.

Theorem

If $\vdash s : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is typable in System T, then $[\![s]\!]$ is continuous.

Proof sketch.

We translate Escardó's Agda development to Coq

- Logical statements placed in impredicative universe of propositions
- Computationally relevant notions placed in predicative type universes
- Based on intrinsically typed de Bruijn representation of System T

Reification (cf. Forster/Kunze '19)

- MetaCoq type `Ast.tm` represents untyped Coq terms
- MetaCoq program `tmQuote` reifies Coq terms to `Ast.tm`

```
Module Ast.
Inductive tm : Set :=
| tRel : nat -> tm
| tConstruct : inductive -> nat ->
           universe_instance -> tm
| tFix : mfixpoint tm -> nat -> tm
| tLambda : name -> tm -> tm -> tm
| tApp : tm -> tm -> tm
| (* ... *).
End Ast.
```

```
Module SystemT.
Inductive tm : Type :=
| var : nat -> tm
| zero : tm
| succ : tm
| rec : type -> tm
| lambda : type -> tm -> tm
| app : tm -> tm -> tm.

End SystemT.
```

- Implemented translation to `SystemT.tm` as monadic program `reify`

Type Inference

- Explicit annotations in `SystemT.tm` allow for unique type inference
- Composes to reification from Coq functions f to typed T-terms f_T

```
Definition Reify (def : ident) {A} (f : A) :=
  f <- tmEval hnf f;;
  s <- tmQuote f;;
  s' <- reify 42 (trans s);;
  s' <- tmEval cbv (infer s' empty_env);;
  match s' with
  | Some (_; s') => tmDefinitionRed def (Some Common.hnf) s'
  | None => tmFail "could not infer type"
  end.
```

Extraction Plugin

- Feeds the extracted typed T-term f_T into the continuity theorem
- Tries to show that $f = [\![f_T]\!]$ by reflexivity
- Concludes the continuity of f

```
Definition ExtractModulus def f :=
  syn <- tmQuote f;;
  m <- ExtractModulus' f;;
  let (A, s) := m in
  match type_eq A ((N -> N) -> N) with
  | left H => s <- tmQuote s;;
              H <- tmUnquoteTyped (continuous f) (cnst syn s);;
              H <- tmEval cbv H;;
              tmDefinition def H
  | right _ => @tmFail (continuous f) "wrong type"
  end.
```

Debatable Design Decisions

- Syntax translations as monadic programs instead of Coq functions
  - `Ast.tm` to `SystemT.tm` is morally the identity function
  - Normalisation needed to eliminate unexpressible features
  - Supported by MetaCoq's `tmEval` program
  - Not internally verifiable!
- Ad-hoc type inference for reified terms
  - Forget typing information of `Ast.tm`, can be easily reconstructed
  - Allows simple syntax transformations to `SystemT.tm` in empty context
  - Only works for fully annotated representation of System T!
- Intermediate language to express the supported fragment
  - Natural starting point given previous developments
  - Works for well-defined fragments of Coq
  - Might be difficult to adjust to more language features!

Future Directions

Extend the supported syntax fragment to

- More data types like $\mathbb{B}$, sums, pairs, lists, rational numbers etc.
- Definitions using Coq's fix/match instead of explicit recursors
- More expressiveness with dependent and informative types

  $\Rightarrow$ Clarify how notion and status of continuity scale!

Verify the current pipeline:

- Requires verified normalisation and type inference (Sozeau et al. '20)
- Soundness: $f = [\![f_T]\!]$ holds whenever reification succeeds
- Completeness: reification succeeds for all T-definable f

```
http://www.ps.uni-saarland.de/extras/modulus-extraction/
```

Bibliography

Yannick Forster and Fabian Kunze.
A Certifying Extraction with Time Bounds from Coq to CBV Lambda Calculus.
In *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter.
Coq Coq correct! verification of type checking and erasure for Coq, in Coq.
Proceedings of the ACM on Programming Languages, 4(POPL):8, 2020.

Florian Steinberg, Laurent Théry, and Holger Thies.
Quantitative Continuity and Computable Analysis in Coq.
In *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

Martín Escardó.
Continuity of Gödel's System T definable functionals via effectful forcing.
Electronic Notes in Theoretical Computer Science, 298:119–141, 2013.

Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter.
The MetaCoq Project.
`https://github.com/MetaCoq/metacoq`, June 2019.

Continuity of T-Definable Functionals (cf. Escardó '13)

Theorem

If $\vdash s : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ *is typable in System T, then* $[\![s]\!]$ *is continuous.*

Proof sketch.

□

Metacoq

content