Certified Meta-Programming With Template Coq

Abhishek Anand - Cornell
Simon Boulier & Nicolas Tabareau - Inria Nantes
Matthieu Sozeau - Inria Paris

CSEC Kick-off
Santiago, Chile
March 7th 2018

TemplateCoq

Initially developed by G. Malecha

Quoting and unquoting of terms and declarations

- `Quote Definition quoted_t : Ast.t := t.`
- `Make Definition denoted_t := quoted_t.`

Ideally "faithful" representation of Coq terms

Differences: Strings for global_reference and lists instead of arrays. But see native integers and arrays…

Demo

Ast.v (term) & template-demo.v

The TemplateCoq Monad

- Similar to MetaCoq's monad (shallow vs deep terms)
- Allows crawling the environment and modifying it, calling the type checker etc…
- WIP OCaml version on the extracted version for building plugins.

- Could be used to **justify** MetaCoq programs and run them **without oracles**, on bare metal.
- But **first** need to formalize the unification algorithm (Ziliani & Sozeau) to actually build interesting tactics (part of CSEC program)

Demo

Ast.v (TemplateMonad) & template-demo.v

Application: CertiCoq

Gallina → Clight

```
compile : Ast.term -> Compcert.Csyntax
```

Theorem (forward simulation)

```
∀ t v : Ast.term, closed t →
 t ~>_wcbv v →
 ∃ v', compile t ~>_C v' ∧ v ~ v'
```

Erases proofs, type labels, types, parameters of constructors, and lambdas of match branches, then CPS, closure conversion, shrink reduction… binding to a GC.

Application: CertiCoq Extraction-Based Path

1. Extract `compile` and bind it to CompCert
2. Reifier in ML from Coq's `constr` to TemplateCoq's **extracted** `Coq_term`
3. Voilà! "`CertiCoq Compile foobar`" (Extraction in the TCB).

Bootstraping à la CakeML in the future.

CIC's Typing Judgments

To improve the theorem, need a spec of reduction in CIC

Current focus on the specification of CIC **as implemented in** COQ:

- Inductive specifications of typing, conversion and reduction on `Ast.term`
- Strict positivity and guard condition (C. Mangin).
- No modules yet: PMP, Derek Dreyer, Joshua Yanovski and I have a "plan" (involving ω-universes…)

Demo: Typing.v

A Certified Typechecker?

- Requires to formally specify the actual implementation of Coq's type inference and its correspondence with the formal semantics defined as a typing judgment.
- **WIP:** A (partial) typechecker and conversion test for `Ast.term` (based on fuel, totality needs SN).
  Comments / contributions welcome!
- **Disclaimer: no** positivity condition, **no** guardedness checking yet.
- Extract it or `CertiCoq Compile` it to get a verified type checker for Coq in ML or as a certified binary.
- `Template Check foo.`

Certified Translations

Definitional translations from TT to TT, e.g. forcing, weaning, parametricity, syntactical models (Boulier et al, CPP'16), exceptional type theory (Pédrot & Tabareau, ESOP'18)

Two parametricity translations:

1. Standard binary parametricy by S. Boulier, using de Bruijn and calling type inference

2. Uniform Propositions by A. Anand's and G. Morrisett (talk this afternoon), switching to a named representation.

Such translations can also be plugged on top of CertiCoq, e.g. to optimize before compilation

Conclusion

Write your plugins in Coq!

Certify them in Coq!

Run them natively using a certified compiler!

http://template-coq.github.io/template-coq