Fun with Template-Coq and CertiCoq

M. Sozeau
INRIA

Coq WG
October 4th 2017

Template-Coq

- A quoter for Coq terms and declarations.
  - `Quote Definition quoted_t : Ast.t := t.`
  - `Denote denoted_t := quoted_t.`
    - Ideal: “Faithful” representation of Coq terms Differences: String.t for global_reference and lists instead of arrays, no univ poly yet, type levels as positives.
- Initially developed by G. Malecha, with contribs from A. Anand and myself. Now maintaining it and using it in the CertiCoq project.

Template-Coq

- One option “Set Template Cast Propositions”, reifies `(t : T : Prop)` into `tCast (reify t : reify T : reify Prop)`

CertiCoq

A certified compiler for Gallina terms:

- `compile : Ast.t -> Compcert.Csyntax`
- ```
  forall t : Ast.t,
   t ~>_cbv_Coq u ->
   compile t ~> C compile u /\
   obs_equivalence Coq C t (compile u).
  ```

- Compilation first erases proofs (as in Extraction), type labels, types, parameters of constructors, and lambdas of match branches (after eta expanding them if necessary).
- Then CPS, closure conversion, shrink reduction and beta-reductions, optimisation of constructor representations and binding to a certified gc.
  ```

CertiCoq

- We can now extract `compile` and bind it to compcert or gcc on the backend.
- Implement a reifier in ML from Coq `constr` to Template-Coq's *extracted* `Ast.t` and use it as the frontend:
- Voilà: "CertiCoq Compile foobar" gives the value of foobar, adding only Extraction to the TCB.
- We could try bootstrapping à la CakeML, using `vm_compute (compile quoted_compile)`, trusting little more than the printer of Coq in addition to the kernel.

A Certified Typechecker?

- To prove CertiCoq's semantics preservation theorem, we need to start from a spec of Coq's reduction.
- WIP: extend Template-Coq with definitions of typing, conversion and reduction and a (partial) typechecker for Ast.t (based on fuel, totality needs SN).
- Extract it or CertiCoq Compile it to get a verified type checker for Coq in ML or as a certified binary.

Byproducts

- Requires to formally specify the actual implementation of Coq's type inference and its correspondence with a formal semantics defined as a typing judgment.
- Disclaimer: Currently none of the hard parts done (positivity condition, guardedness checking).
- Typing function can be used to help writing definitional translation, such as parametricity (S. Boulier).
- Such translations can also be compiled by CertiCoq, and could form the basis for certified definitional extensions of the theory. I.e. to add internal "computational axioms":

```
param_type : ∀ {A : Type}, A -> Type
```
and
```
param_proof : ∀ {A) (a : A) : param_type a
```