Generic Approach to Certified Static Checking of Module-like Language Constructs

Julia Belyakova
julbinb@gmail.com
Southern Federal University

June 20th 2017
FTfJP 2017: Formal Techniques for Java-like Programs
ECOOP 2017 Series
Contents

1 Introduction

2 Generic Coq Library for Certified Checking of Modules

3 STLC with Concept Parameters

4 Generic Coq Library for Certified Checking of Modules
Interactive Theorem Provers for PL

Interactive Theorem Provers have been used for both:

1. mechanizing **formal models** of programming languages;
   - Featherweight Java with mutability [Mackay et al. 2012];
   - Dependent Object Types [Rompf and Amin 2016];
   - JSCert [Bodin et al. 2014].

2. building **certified**\(^1\) compilers and interpreters.
   - C Compiler CompCert [Blazy and Leroy 2009];
   - SML Compiler CakeML [Tan et al. 2016];
   - JavaScript Interpreter JSRef [Bodin et al. 2014].

---

1“Certified” means “corresponds to the formal model”. For example, \(\text{type\_check}(\Gamma, t) = \text{Some } \tau \Rightarrow \Gamma \vdash t : \tau.\)
Many programming languages have some notion of **module**:

- package;
- ML module;
- class;
- trait;
- Haskell type class.

“Modules” are often used as a means of abstraction and come in pairs of an **interface** and an **implementation**:

- interfaces/traits/protocols and classes (OO);
- signatures and modules (ML);
- type classes and instances (Haskell).
Let’s build an efficient certified static checker for a simple language with modules.
Concept describes an interface — list of name-type pairs.
Model defines an implementation of the concept — list of name-term pairs.

Example

```coq
// Concepts (interfaces)
concept CMonoid { ident : Nat; binop : Nat -> Nat -> Nat }
concept CFoo { boo : Nat; bar : Nat -> Bool }

// Models (implementations)
model MSum of CMonoid {
  ident = 0
  binop = \x:Nat.\y:Nat. x + y
}
model MProd of CMonoid {
  ident = 1
  binop = \x:Nat.\y:Nat. x * y
}
model MFoo of CFoo {
  boo = 5;
  bar = \x:Nat. if x > boo then true else false
}
```
Static Checking of Concepts and Models

1 Concept is well-defined in CT (concepts table)
   ⇔ all names are distinct ∧ all types are well-defined in CT.

2 Concept Section is well-defined
   ⇔ all names are distinct ∧ all concepts are well-defined².

3 Model M is a well-defined model of C in (CT, MT)
   ⇔ all names are distinct ∧ all concept members are defined² ∧ all terms have expected types in (CT, MT).

4 Model Section is well-defined
   ⇔ all names are distinct ∧ all models are well-defined².

Example: Model M of C \{ f_1 = e_{f_1}; f_2 = e_{f_2} \}

\[ \vdash e_{f_1} : \tau_{f_1} \quad \land \quad f_1 : \tau_{f_1} \vdash e_{f_2} : \tau_{f_2} \]

² Later defined elements can refer to the previous ones.
Definition of well-definedness of a module [formal model]:
\[ DOk : M \rightarrow Prop. \]

Definition \( f_{\text{mem}} (\text{okCl} : \text{Prop} \times \text{ctxloc}) (\text{dt} : \text{data}) : \text{Prop} \times \text{ctxloc} \)
:= \text{match} \ \text{okAndCl} \ \text{with} \ (\text{ok}, \ \text{cl}) \Rightarrow
  \text{let} \ \text{ok}' := \text{update\_prop} \ \text{ok} \ \text{cl} \ \text{dt} \ \text{in}
  \text{let} \ \text{cl}' := \text{update\_ctxloc} \ \text{cl} \ \text{dt} \ \text{in} \ (\text{ok}', \ \text{cl}').

Definition \( \text{module\_ok} (\text{dts} : \text{list data}) : \text{Prop} \times \text{ctxloc} \)
:= \text{let} (\text{ok}, \ \text{m}) := \text{List.fold\_left} \ \text{f\_mem} \ \text{dts} (\text{True}, \ \text{ctxloc\_init}) \ \text{in}
  (\text{List.NoDup} \ (\text{get\_names} \ \text{dts}) \ \backslash \ \text{ok}, \ \text{m}).

Algorithm checking that a module is well-defined [compiler/interpreter]: \( AOk : M \rightarrow \text{bool} \).

Proof of correctness of the algorithm with respect to the formal definition (soundness and maybe completeness).
\[ AOk(M) = \text{true} \iff DOk(M) \]

Efficient representation of a well-defined module (ctxloc).
STLC with Concept Parameters: Syntax

Types

\[ \tau :\!\!::= \text{Nat} | \text{Bool} | \tau \rightarrow \tau | C \# \tau \]  
\[ \phi :\!\!::= \{f_i : \tau_i\} \]  
\[ \psi :\!\!::= (C, \{f_i = e_i\}) \]  

Terms

\[ e :\!\!::= x | \lambda x : \tau. e | e e \]  
\[ | n | e + e | \ldots \]  
\[ | \lambda c \# C. e \]  
\[ | e \# M \]  
\[ | c::f \]  
\[ p :\!\!::= \text{CSec MSec e} \]
STLC with Concept Parameters: Typing

Typing Judgement

\[ CT \ast MT ; \Gamma \vdash e : \tau, \]

where CT and MT are finite maps built from CSec and MSec.

\[
\frac{C \in dom(CT)}{CT \ast MT ; \Gamma \vdash \lambda c \# C. e : C \# \tau} \quad (T-CAbs)
\]

\[
\frac{c \# C \in \Gamma \quad C \in dom(CT) \quad f : \tau_f \in CT(C)}{CT \ast MT ; \Gamma \vdash c :: f : \tau_f} \quad (T-CInvc)
\]

\[
\frac{M \circ C \{ \ldots \} \in MT \quad f : \tau_f \in CT(C) \quad M \# C' \notin \Gamma}{CT \ast MT ; \Gamma \vdash M :: f : \tau_f} \quad (T-MInvc)
\]
Example

\[ f = \lambda c \# \text{CMonoid}. \lambda x : \text{Nat}. \ c :: \text{binop} \ x \ 5 : \text{CMonoid} \# \text{Nat} \to \text{Nat} \]

\[ f \# \text{MSum} : \text{Nat} \to \text{Nat} \]

\[ f \# \text{MSum} \text{ evaluates to} \]
\[ \lambda x : \text{Nat}. \ \text{MSum} :: \text{binop} \ x \ 5 \]

\[ f \# \text{MSum} 3 \text{ evaluates to} \]
\[ \text{MSum} :: \text{binop} \ 3 \ 5 \to (\lambda x : \text{Nat}. \ \lambda y : \text{Nat}. \ x + y) \ 3 \ 5 \to^* 8 \]
Small-Step Operational Semantics

\[ \text{CT} \ast \text{MT} ; t \rightarrow t' \]

\[
\frac{M \text{ of } \text{C}\{\ldots\} \in \text{MT} \quad f = t_f \in \text{MT}(M)}{\text{CT} \ast \text{MT} ; M::f \rightarrow QMM(\text{MT}, M, t_f)} \quad \text{(E-CInvc)}
\]

Where \( QMM(\text{MT}, M, t) \) qualifies with model name \( M \) all free variables of \( t_f \) that appear in \( \text{MT}(M) \).

Example

\[
\text{MFoo::bar} = (\lambda x: \text{Nat}. \text{if } x > \text{boo} \text{ then } \ldots)
\]

\[
\text{MFoo::bar 42} \rightarrow (\lambda x: \text{Nat}. \text{if } x > \text{MFoo::boo} \text{ then } \ldots) \ 42
\]
STLC with Concept Parameters: Soundness Problem

To prove type soundness, we need to prove that evaluation of the member invocation preserves typing:

\[ \text{CT} \ast \text{MT} \vdash M :: f : \tau_f \land M :: f \rightarrow QMM(t_f) \implies \text{CT} \ast \text{MT} \vdash QMM(t_f) : \tau_f \]

This has something to do with the definition of well-definedness for models and a model section.

Have to unfold generic definitions of well-definedness; copy-paste driven reasoning about fold-left based definitions.

We need a generic principle of reasoning about the definitions.
Current Progress

- Low-level library for certified transformation of lists into sets (MSet), and lists of pairs into finite maps (FMap): ~2000 LOC.
- Generic library for certified checking of simple modules, single-pass modules, and single-pass modules-implementations: ~1500 LOC.
- STLC with Concept Parameters: soundness proof up to 2 lemmas about well-definedness of models: ~6000 LOC.

Source code: concept-params at github/julbinb.
Future Work

1. Reasoning principles for generic definitions of well-definedness.
2. More strategies of checking modules (e.g. all members can refer to each other; nested modules; first-class modules).

