A Verified Compiler for an Impure Functional Language

水野 雅之

2015 年 12 月 10 日

Motivation

Planning to verify MinCaml for graduate thesis
There are several previous reserches (such as CompCert) but...

MinCaml features

- First class functions
- Impure operations
- Foreign function interface

Thus tried to survey closely related paper
"A Verified Compiler for an Impure Functional Language"

A Verified Compiler for an Impure Functional Language?

Adam Chlipala, POPL10

verifies impure functional language with Coq

$$
\begin{aligned}
e \ ::= \ & c \mid e = e \mid x \mid e\ e \mid \mathtt{fix}\ f(x).\ e \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \\
& \mid () \mid \langle e, e \rangle \mid \mathtt{fst}(e) \mid \mathtt{snd}(e) \mid \mathtt{inl}(e) \mid \mathtt{inr}(e) \\
& \mid \mathtt{case}\ e\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow e \mid \mathtt{inr}(x) \Rightarrow e \\
& \mid \mathtt{ref}(e) \mid !e \mid e := e \mid \mathtt{raise}(e) \mid e\ \mathtt{handle}\ x \Rightarrow e
\end{aligned}
$$

$\rightarrow$ close to MinCaml's source language

Outline

1. Introduction

Outline

A lot of sucks around mechanized proof

- wrong abstractions (e.g. nested variable binders)
- copious lemma and case analysis
- Once add a new constructor, Modify proof anywhere

Proposing "engineering" approach to reduce development costs

- *Parametric Higher-Order Abstract Syntax*
- new semantic approach
- sophisticated proof automation

Source Language

untyped, subset of ML

$$
\begin{aligned}
e \ ::= \ & c \mid e = e \mid x \mid e\ e \mid \mathtt{fix}\ f(x).\ e \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \\
& \mid () \mid \langle e, e \rangle \mid \mathtt{fst}(e) \mid \mathtt{snd}(e) \mid \mathtt{inl}(e) \mid \mathtt{inr}(e) \\
& \mid \mathtt{case}\ e\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow e \mid \mathtt{inr}(x) \Rightarrow e \\
& \mid \mathtt{ref}(e) \mid !e \mid e := e \mid \mathtt{raise}(e) \mid e\ \mathtt{handle}\ x \Rightarrow e
\end{aligned}
$$

no variable-arity features (e.g. sum, product)

no compound pattern matching

Target Assembly Language Syntax

idealized assembly language

$$
\begin{array}{lcl}
r & ::= & r_0 \mid \ldots \mid r_{N-1} \\
n & \in & \mathbb{N} \\
L & ::= & r \mid [r+n] \mid [n] \\
R & ::= & n \mid r \mid [r+n] \mid [n] \\
I & ::= & L ::= R \mid r \stackrel{+}{=} n \mid L := R \stackrel{?}{=} R \mid \texttt{jnz}\ R, n \\
J & ::= & \texttt{halt}\ R \mid \texttt{fail}\ R \mid \texttt{jmp}\ R \\
B & ::= & (I^*, J) \\
P & ::= & (B^*, B)
\end{array}
$$

finite registers and no interface for memory managements

Source Language Semantics

big step operational semantics $(h_1, e) \Downarrow (h_2, r)$

$$\frac{}{(h, \texttt{fix}\ f(x).\ e) \Downarrow (h, \texttt{Ans}(\texttt{fix}\ f(x).\ e))}$$

$$\frac{(h_1, e_1) \Downarrow (h_2, \texttt{Ans}(\texttt{fix}\ f(x).\ e)) \qquad (h_2, e_2) \Downarrow (h_3, \texttt{Ans}(e')) \qquad (h_3, e[f \mapsto \texttt{fix}\ f(x).\ e][x \mapsto e']) \Downarrow (h_4, r)}{(h_1, e_1\ e_2) \Downarrow (h_4, r)}$$

$$\frac{(h_1, e) \Downarrow (h_2, \texttt{Ex}(v))}{(h_1, e_1\ e_2) \Downarrow (h_3, \texttt{Ex}(v))}$$

$$\frac{(h_1, e_1) \Downarrow (h_2, \texttt{Ans}(\texttt{fix}\ f(x).\ e)) \qquad (h_2, e_2) \Downarrow (h_3, \texttt{Ex}(v))}{(h_1, e_1\ e_2) \Downarrow (h_3, \texttt{Ex}(v))}$$

$$\frac{(h_1, e) \Downarrow (h_2, \texttt{Ans}(v))}{(h_1, \texttt{ref}(e)) \Downarrow (v :: h_2, \texttt{Ans}(\texttt{ref}(|h_2|)))}$$

$$\frac{(h_1, e) \Downarrow (h_2, \texttt{Ans}(\texttt{ref}(n))) \qquad h_2.n = v}{(h_1, !e) \Downarrow (h_2, \texttt{Ans}(v))}$$

Source Language Semantics

$$\frac{(h_1, e) \Downarrow (h_2, \texttt{Ans}(v))}{(h_1, \texttt{raise}(e)) \Downarrow (h_2, \texttt{Ex}(v))}$$

$$\frac{(h_1, e_1) \Downarrow (h_2, \texttt{Ex}(v)) \qquad (h_2, e_2[x \mapsto v]) \Downarrow (h_3, r)}{(h_1, e_1 \ \texttt{handle} \ x \Rightarrow e_2) \Downarrow (h_3, r)}$$

Theorems ignore non-termination.