PA2 – OCaml F1VAE Interpreter

Programming Languages (SWE3006-41)
Fall, 2023

Instructor:
Sungjae Hwang
- Jason.sungjae.hwang@gmail.com
- https://softsec-lab.github.io

Software Security Lab
SUNGKYUNKWAN UNIV

Introduction

- Deadline : 2023.11.23. (Delay Submission 2023.11.25., 25% deduction per day)
- Write interpreter using OCaml.
- Submit source codes (*ml) for each exercise.
  - ✓ You will not get any points if your source code does not compile well.
  - ✓ Submit "PA2_StudentID.zip" through icampus.
  - ✓ Zip file should contains:

  ast.ml, dune, dune-project, fStore.ml, interpreter.ml, main.ml, store.ml, value.ml, test.ml
- Do not modify files except interpreter.ml.
- Please leave the questions in the google sheet

  https://docs.google.com/spreadsheets/d/19T2K5LmaounOnL-LGaF-nzdu1G_jJXY5nRjSMQr3hVc/edit#gid=259444581

Prequisites

- Install following packages with correct version
  - ✓ Opam >= 2.0
    - ➢ Ocaml package manager
    - ➢ https://opam.ocaml.org/doc/install.html
  - ✓ OCaml >= 4.12.0
    - ➢ Ocaml interpreter & Compiler
    - ➢ https://ocaml.org/docs/install.html
  - ✓ Dune >= 2.0
    - ➢ Ocaml build system
    - ➢ https://github.com/ocaml/dune

Required Libraries

- Install following libraries
  - ✓ Core
    - ➢ Ocaml standard library
    - ➢ $ opam install core
  - ✓ ppx_compare
    - ➢ Preprocessor of Ocaml (included in Core, if not enter command)
    - ➢ $ opam install ppx_compare
  - ✓ Menhir
    - ➢ Ocaml parser generator
    - ➢ $ opam install menhir

Goal

- Your task is to implement interpreter for F1VAE programming language that we have designed in the lectures.

  ✓ Should support multiple functions and parameters
  ✓ Write below functions in "interpreter.ml"

```
let rec interp_expr (e: Ast.expr) (g: FStore.t) (s: Store.t) : Value.t =  (* ... *)
let interp_fundef (d: Ast.fundef) (g: FStore.t) : FStore.t =  (* ... *)
let interp (p: Ast.prog) : Value.t =  (* ... *)
```

  ✓ You only need to modify "interpreter.ml“
    ➢ Do not modify other files
    ➢ You only need to write above 3 functions.

Syntax of F1VAE

$$\boldsymbol{p} ::= \overline{d}\ e$$

$$\boldsymbol{d} ::= def\ x\ \overline{x}\ =\ \boldsymbol{e}$$

$$\boldsymbol{e} ::= n \mid x \mid \boldsymbol{e} + \boldsymbol{e} \mid \boldsymbol{e} - \boldsymbol{e} \mid let\ x\ =\ \boldsymbol{e}\ in\ \boldsymbol{e} \mid x(\overline{\boldsymbol{e}})$$

$$n \in \mathbb{Z}$$

$$x \in \mathbb{V}ar$$

Semantics of F1VAE

Write interpreter which evaluate program as below

[rule Num] : n

$$\frac{}{\Lambda, \sigma \vdash n \Downarrow_E n}$$

[rule Id (1)] : x

$$\frac{x \in \text{Domain}(\sigma)}{\Lambda, \sigma \vdash x \Downarrow_E \sigma(x)}$$

[rule Add] : e1 + e2

$$\frac{\Lambda, \sigma \vdash e_1 \Downarrow_E n_1 \quad \Lambda, \sigma \vdash e_2 \Downarrow_E n_2}{\Lambda, \sigma \vdash e_1 + e_2 \Downarrow_E n_1 +_z n_2}$$

[rule Sub] : e1 - e2

$$\frac{\Lambda, \sigma \vdash e_1 \Downarrow_E n_1 \quad \Lambda, \sigma \vdash e_2 \Downarrow_E n_2}{\Lambda, \sigma \vdash e_1 + e_2 \Downarrow_E n_1 -_z n_2}$$

[Rule Prog] : d e

$$\frac{\vdash d \Downarrow_F \Lambda \qquad \Lambda, \varnothing \vdash e \Downarrow_E n}{\vdash d\ e \Downarrow_p n}$$

Semantics of F1VAE

Write interpreter which evaluate program as below

[Rule LetIn] : let x = e1 in e2

$$\frac{\Lambda, \sigma \vdash e_1 \Downarrow_E n_1 \quad \Lambda, \sigma[x_1 \mapsto n_1] \vdash e2 \Downarrow_E n_2}{\Lambda, \sigma \vdash \text{let } x = e1 \text{ in } e1 \Downarrow_E n_2}$$

[Rule FDef] : def x1 x2 = e

$$\frac{x2 = x21 \ldots x2n}{\Lambda \vdash \text{def } x_1\ x_2 = e \Downarrow_F \Lambda[x_1 \rightarrow ([x21; \ldots ; x2n], e)]}$$

[Rule FCall] : x(e)

$$\frac{\mathbf{e} = e1 \ldots e1k \quad \Lambda, \sigma \vdash e11 \Downarrow_E n11 \ldots \Lambda, \sigma \vdash e1k \Downarrow_E n1k \qquad \Lambda(x) = ([x1;\ldots;xk], e2) \quad \Lambda, [x1 \mapsto n11;\ldots;xk \mapsto n1k] \vdash e2 \Downarrow_E n_2.}{\Lambda, \sigma \vdash x(e) \Downarrow_E n_2}$$

Abstract Memory for Functions

- Define abstract memory for storing function definition
  - ✓ Fstore : set of functions that receives function name and returns pair of a parameter and function body (x, e)
    - ➢ $Fstore : Var \rightarrow Var \times E$
  - ✓ Λ is one element of Fstore
    - ➢ $\Lambda \in Fstore$

Fstore is implemented in fStore.ml

Use functions in fStore.ml to store and find functions from interpreter

Abstract Memory for Identifier

- Define abstract memory for storing variable
  - ✓ Store is set of functions which receive variable × return integer n
    - ➢ $Store : Var \rightarrow Z$
  - ✓ σ is one element of Store
    - ➢ $\sigma \in Store$

Store is implemented in store.ml

Use functions in store.ml to store and find identifiers from interpreter

Files

- ast.ml : ast of F1VAE programming languages
- dune, dune-project, main.ml : build files
- fStore.ml : abstract memory for functions
- store.ml : abstract memory for identifier
- test.ml : test cases
- value.ml : value ($\sigma \in Z$)

To Run

- $ dune runtest
  - All test cases are passed

```
stell@stell-virtual-machine:~/pa2$ dune runtest
File "test.ml", line 6, characters 0-142: <<try let ast = Prog ([], (Add ((Id "x"), (Num [...]>> (0.000 sec)
File "test.ml", line 12, characters 0-178: <<try   let ast =     Prog ([FunDef ("x", [], ([...]>> (0.000 sec)
File "test.ml", line 18, characters 0-232: <<try   let ast =     Prog       ([FunDef ("x",[...]>> (0.000 sec)
File "test.ml", line 24, characters 0-237: <<try   let ast =     Prog       ([FunDef ("x",[...]>> (0.000 sec)
File "test.ml", line 30, characters 0-122: <<(interp ast) = (NumV 14)>> (0.000 sec)
File "test.ml", line 34, characters 0-130: <<(interp ast) = (NumV 4)>> (0.000 sec)
File "test.ml", line 38, characters 0-143: <<(interp ast) = (NumV 7)>> (0.000 sec)
File "test.ml", line 42, characters 0-144: <<(interp ast) = (NumV 5)>> (0.000 sec)
File "test.ml", line 46, characters 0-195: <<(interp ast) = (NumV 7)>> (0.000 sec)
File "test.ml", line 50, characters 0-102: <<(interp ast) = (NumV 53)>> (0.000 sec)
File "test.ml", line 54, characters 0-102: <<(interp ast) = (NumV 65)>> (0.000 sec)
File "test.ml", line 58, characters 0-102: <<(interp ast) = (NumV 0)>> (0.000 sec)
File "test.ml", line 62, characters 0-113: <<(interp ast) = (NumV 8)>> (0.000 sec)
File "test.ml", line 66, characters 0-169: <<try   let ast = Prog ([], (LetIn ("x", (Num 5[...]>> (0.000 sec)
File "test.ml", line 72, characters 0-186: <<try   let ast =     Prog       ([FunDef ("x",[...]>> (0.000 sec)
stell@stell-virtual-machine:~/pa2$
```

To Run

- $ dune runtest
  - Failed case

```
stell@stell-virtual-machine:~/pa2$ dune runtest
File "test.ml", line 6, characters 0-142: <<try let ast = Prog ([], (Add ((Id "x"), (Num [...]>> (0.000 sec)
File "test.ml", line 12, characters 0-178: <<try   let ast =     Prog ([FunDef ("x", [], ([...]>> (0.000 sec)
File "test.ml", line 18, characters 0-232: <<try   let ast =     Prog       ([FunDef ("x",[...]>> (0.000 sec)
File "test.ml", line 24, characters 0-237: <<try   let ast =     Prog       ([FunDef ("x",[...]>> (0.000 sec)
File "test.ml", line 30, characters 0-122: <<(interp ast) = (NumV 14)>> (0.000 sec)
File "test.ml", line 34, characters 0-130: <<(interp ast) = (NumV 4)>> (0.000 sec)
File "test.ml", line 38, characters 0-143: <<(interp ast) = (NumV 7)>> (0.000 sec)
File "test.ml", line 42, characters 0-144: <<(interp ast) = (NumV 5)>> (0.000 sec)
File "test.ml", line 46, characters 0-195: <<(interp ast) = (NumV 7)>> (0.000 sec)
File "test.ml", line 50, characters 0-102: <<(interp ast) = (NumV 53)>> (0.000 sec)
File "test.ml", line 54, characters 0-102: <<(interp ast) = (NumV 65)>> (0.000 sec)
File "test.ml", line 58, characters 0-102: <<(interp ast) = (NumV 0)>> (0.000 sec)
File "test.ml", line 62, characters 0-113: <<(interp ast) = (NumV 8)>> (0.000 sec)
File "test.ml", line 66, characters 0-169: <<try   let ast = Prog ([], (LetIn ("x", (Num 5[...]>> (0.000 sec)
File "test.ml", line 72, characters 0-186: <<try   let ast =     Prog       ([FunDef ("x",[...]>> (0.000 sec)

======================================================================
File "test.ml", line 18, characters 0-232: <<try   let ast =     Prog       ([FunDef ("x",[...]>> is false.

File "test.ml", line 24, characters 0-237: <<try   let ast =     Prog       ([FunDef ("x",[...]>> is false.

FAILED 2 / 15 tests
stell@stell-virtual-machine:~/pa2$
```

← Failed case

← Number of failed case

Error Handling

- If we call undefined functions
  - failwith "Undefined function: f_name“
- If we functions with wrong number of arguments
  - failwith "The number of arguments of x mismatched: Required:x, Actual: x"
- If we access free identifier
  - failwith "Free identifier: id_name"
- Please see testcases in test.ml file

Points

- Implementation of Prog Semantics (10 pts)
- Implementation of Num Semantics (10 pts)
- Implementation of Add & Sub Semantics (5 pts each, total 10 pts)
- Implementation of Id Semantics (10 pts)
  - Catch Free identifier (10 pts)
- Implementation of LetIn Semantics (10 pts)
- Implementation of Call Semantics (10 pts)
  - Catch Arguments mismatch (10 pts)
  - Catch Undefined function (10 pts)
- Implementation of FunDef Semantics (10 pts)

- Total 100 pts

- See ast.ml file how the program traverse AST and implement semantics.