Yarmin

October 5, 2017

Contents

Chapter 1

Introduction

Yarmin is an imperative (also a functional) interpreter written in OCaml.
The interpreter is capale of handling expressions, commands and declarations all within a program.
A program can be parsed to into valid code from file/string thanks to reflection.

Chapter 2

Desing Choice

As a starting point, I used the denotation interpreter. I chose to develop a denotational interpreter since adding new constructs are simpler than the operative interpreter.

The interpreter is characterized by a dynamic local environment and a static non-local environment.

To simplify writing code from terminal to ocaml, I used rlwrap.
The interpreter therefore consists of a series of functions divided into several files.
To start the program:
1. Get in the Path of the project directory;
2. Run the script **compileAll.sh** to compile.
3. Run the command **rlwrap ocaml**
4. Insert **#use "loadAll.ml";;**

In the development of the reflection operation, I decided to write a parser via **Menhir**, which is the evolution of the previous Ocamlyacc parser for OCaml. Using it together with the use of ocamllex, or lexer, I have written a grammar for the language so that I can properly interpret correctly any string passed as an argument of the reflect function and so on assigning the correct semantic evaluation.
All various test, can be executed by the command **#use "YarminTest.ml";;**

2.0.1 Files

- **loadAll.ml**: Used for run the program.
- **Funstore.mli**: Store Interface .
- **Funstore.ml**: Store function implementation.
- **Funenv.mli**: Environment Interface.
- **Funenv.ml**: Environment function implementation.
- **YarminParser.mly**: Parser definition for Menhir.
- **YarminLexer.mll**: Lexer definition for OCamllex.
- **Yarmin.mli**: General Interface with types.
- **Yarmin.ml**: General Implementation of the language.

2.0.2 Functions

Semantic Valutation

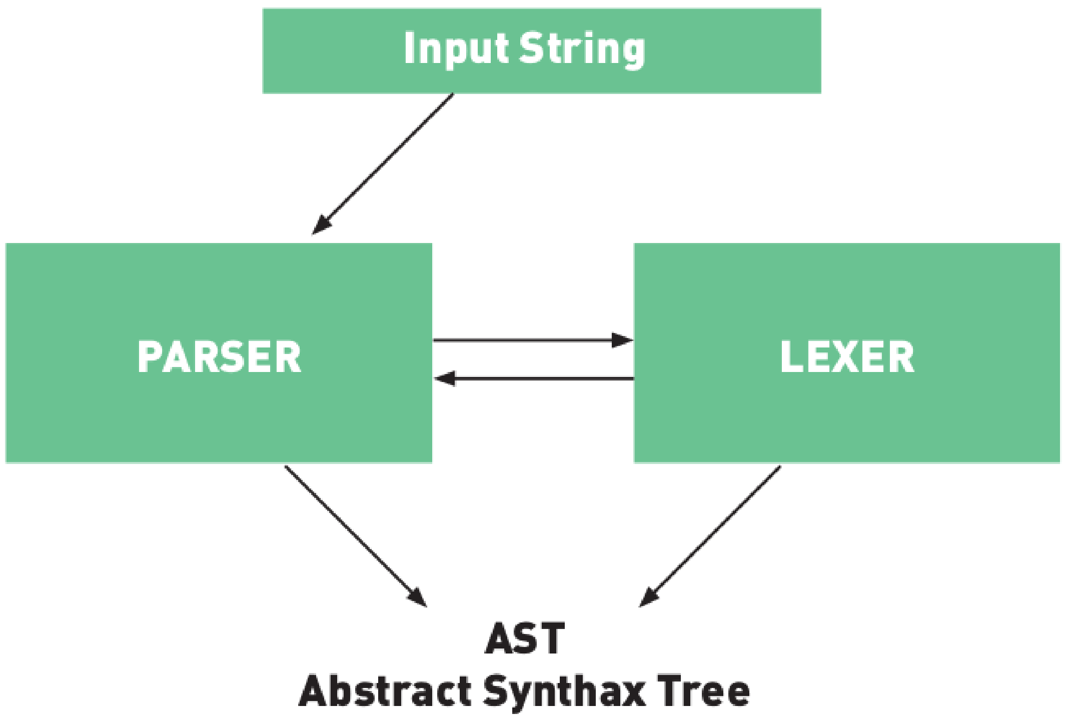
- SEM
- SEMDEN
- SEMC
- SEMDV
- SEMDR
- SEMB

Implementations

- LEN
- CAT
- SUBSTR
- EREFLECT

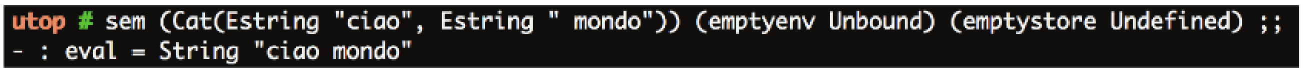
Chapter 3

Parsing and Lexing



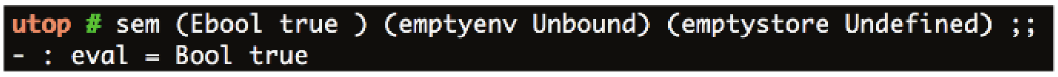
Chapter 4

Yarmin Test



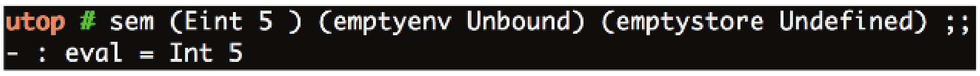
```
utop # sem (Cat(Estring "ciao", Estring " mondo")) (emptyenv Unbound) (emptystore Undefined) ;;
- : eval = String "ciao mondo"
```

sem (Cat(Estring “ciao”, Estring “ mondo”)) (emptyenv Unbound)
(emptystore Undefined) ;;



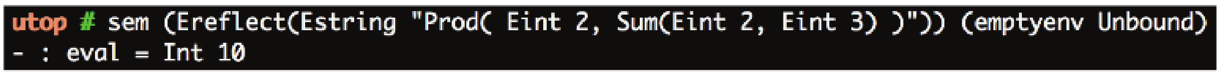
```
utop # sem (Ebool true ) (emptyenv Unbound) (emptystore Undefined) ;;
- : eval = Bool true
```

sem (Ebool true) (emptyenv Unbound) (emptystore Undefined) ;;



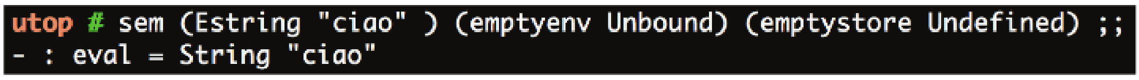
```
utop # sem (Eint 5 ) (emptyenv Unbound) (emptystore Undefined) ;;
- : eval = Int 5
```

sem (Eint 5) (emptyenv Unbound) (emptystore Undefined) ;;



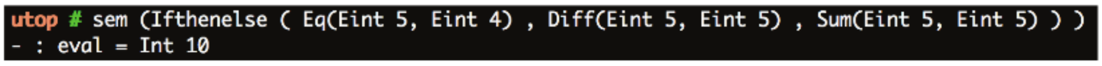
```
utop # sem (Ereflect(Estring "Prod( Eint 2, Sum(Eint 2, Eint 3) )")) (emptyenv Unbound)
- : eval = Int 10
```

sem (Ereflect(Estring “Prod(Eint 2, Sum(Eint 2, Eint 3))”)) (emptyenv Unbound)
(emptystore Undefined) ;;



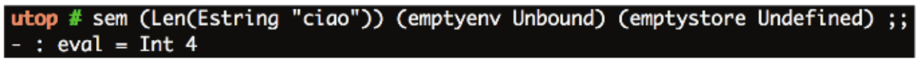
```
utop # sem (Estring "ciao" ) (emptyenv Unbound) (emptystore Undefined) ;;
- : eval = String "ciao"
```

sem (Estring “ciao”) (emptyenv Unbound) (emptystore Undefined) ;;



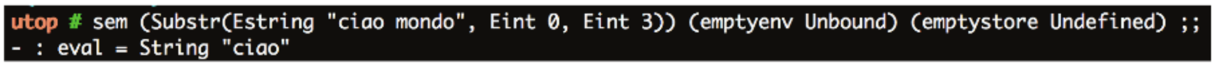
```
utop # sem (Ifthenelse ( Eq(Eint 5, Eint 4) , Diff(Eint 5, Eint 5) , Sum(Eint 5, Eint 5) ) )
- : eval = Int 10
```

sem (Ifthenelse(Eq(Eint 5, Eint 4) , Diff(Eint 5, Eint 5) , Sum(Eint 5, Eint 5)))
(emptyenv Unbound) (emptystore Undefined) ;;



```
utop # sem (Len(Estring "ciao")) (emptyenv Unbound) (emptystore Undefined) ;;
- : eval = Int 4
```

sem (Len(Estring “ciao”)) (emptyenv Unbound) (emptystore Undefined) ;;



```
utop # sem (Substr(Estring "ciao mondo", Eint 0, Eint 3)) (emptyenv Unbound) (emptystore Undefined) ;;
- : eval = String "ciao"
```

sem (Substr(Estring “ciao mondo”, Eint 0, Eint 3)) (emptyenv Unbound)
(emptystore Undefined) ;;