Software Verification

Lecture 02. OCaml Programming I

Ji, Yong-Hyeon

24. 07. 18 (Thu)

Coding & Optimization Together (CO2)
Crypto & Security Engineering Lab (CSE)
Department of Information Security, Cryptology, and Mathematics

Table of Contents

Motivation

1. Motivation

https://www.tiobe.com/tiobe-index/

Jul 2024	Jul 2023	Change	Programming Language	Ratings	Change
1	1		Python	16.12%	+2.70%
2	3	↑	C++	10.34%	-0.46%
3	2	↓	C	9.48%	-2.08%
4	4		Java	8.59%	-1.91%
5	5		C#	6.72%	-0.15%
6	6		JavaScript	3.79%	+0.68%
7	13	⇈	Go	2.19%	+1.12%
8	7	↓	Visual Basic	2.08%	-0.82%
9	11	↑	Fortran	2.05%	+0.80%
10	8	↓	SQL	2.04%	+0.57%
11	15	⇈	Delphi/Object Pascal	1.89%	+0.91%
12	10	↓	MATLAB	1.34%	+0.08%
13	17	⇈	Rust	1.18%	+0.29%
14	16	↑	Ruby	1.16%	+0.25%
15	12	↓	Scratch	1.15%	+0.08%
16	9	⇊	PHP	1.15%	-0.27%
17	18	↑	Swift	1.13%	+0.25%
18	14	⇊	Assembly language	1.11%	+0.10%
19	20	↑	COBOL	1.08%	+0.21%
20	26	⇈	Kotlin	1.05%	+0.35%

1. Motivation

Position	Programming Language	Ratings
21	Classic Visual Basic	0.91%
22	R	0.83%
23	SAS	0.79%
24	Ada	0.78%
25	Dart	0.74%
26	D	0.72%
27	Lisp	0.67%
28	Prolog	0.67%
29	(Visual) FoxPro	0.66%
30	Perl	0.66%
31	Haskell	0.65%
32	Lua	0.60%
33	Scala	0.59%
34	Julia	0.56%
35	Objective-C	0.40%
36	VBScript	0.40%
37	GAMS	0.33%
38	ML	0.31%
39	Solidity	0.30%
40	Logo	0.28%
41	PL/SQL	0.28%
42	Transact-SQL	0.27%
43	PowerShell	0.27%
44	TypeScript	0.26%
45	Bash	0.26%
46	Scheme	0.24%
47	ABAP	0.24%
48	Awk	0.22%
49	Ladder Logic	0.22%
50	RPG	0.21%

Basic OCaml Programming

2.1 OCaml 기본 구성

OCaml 프로그램의 기본 단위 공식

- 프로그램을 구성하는 두 가지 기본 단위

Statement:

```
x = x + 1
```

Expression:

```
(x + y) * 2
```

2.1 OCaml 기본 구성

▷ OCaml 프로그램의 기본 단위 공식

- 프로그램을 구성하는 두 가지 기본 단위

  * Statement

    ```
    x = x + 1
    ```

  * Expression

    ```
    (x + y) * 2
    ```

2.1 OCaml 기본 구성

▷ OCaml 프로그램의 기본 단위 공식

- 프로그램을 구성하는 두 가지 기본 단위

  * Statement (명령문): 기계 상태를 변경

    ```
    x = x + 1
    ```

  * Expression (식): 상태 변경 없이 값을 계산

    ```
    (x + y) * 2
    ```

2.1 OCaml 기본 구성

▷ OCaml 프로그램의 기본 단위 공식

- 프로그램을 구성하는 두 가지 기본 단위
  * Statement (명령문): 기계 상태를 변경

```
x = x + 1
```

  * Expression (식): 상태 변경 없이 값을 계산

```
(x + y) * 2
```

- 프로그래밍 언어를 구분하는 한가지 기준:
  * 명령문을 중심으로 프로그램을 작성
    - C, C++, Java, Python, JavaScript, etc.
    - "Imperative Languages"
  * 식을 중심으로 프로그램을 작성
    - ML, Haskell, Scala, Lisp, etc.
    - "Functional Programming"

2.1 OCaml 기본 구성

▷ OCaml 프로그램의 기본 구조

- 값 정의들의 나열

$$\begin{aligned} &\texttt{let } x_1 = e_1 \\ &\texttt{let } x_2 = e_2 \\ &\qquad \vdots \\ &\texttt{let } x_n = e_n \end{aligned}$$

* 식 $e_1, e_2, \ldots, e_n$을 순차적으로 계산

* 변수 x_i는 식 e_i의 값을 지칭 (assignment vs binding)

```
1 let hello = "Hello"
2 let world = "World"
3 let helloworld = hello ^ " " ^ world
4 let _ = print_endline helloworld
```

Compile

```
@:~$>ocaml helloworld.ml
```

OCaml REPL(Real-Eval-Print Loop)

```
@:~$>ocaml

# #use "helloworld.ml";;
val hello : string = "Hello"
val world : string = "World"
val helloworld : string = "Hello World"
Hello World
- : unit = ()
# exit 1;;
```

2.1 OCaml 기본 구성

▷ Arithmetic Expression (산술식)

- 정수와 실수

```
# 1 + 2 * 3;;
- : int = 7
# 1.1 +. 2.2 *. 3.3;;
- : float = 8.36
```

정수값을 위한 산술 연산자

$$+, \quad -, \quad *, \quad /, \quad \texttt{mod}$$

실수값을 위한 산술 연산자

$$+., \quad -., \quad *., \quad /., \quad \texttt{mod\_float(.,.)}$$

2.1 OCaml 기본 구성

▷ Arithmetic Expressions (산술식)

- 정수 타입과 실수 타입을 명확히 구분하자

```
# 3 + 2.0;;
Error: This expression has type float but an
expression was expected of type int
# 3 + int_of_float 2.0;;
- : int = 5
```

2.1 OCaml 기본 구성

▷ Boolean Expressions (논리식)

- 논리값

```
# true;;
- : bool = true
# false;;
- : bool = false
```

- 비교 연산자 (산술식 → 논리식)

```
# 1 = 2;;
- : bool = false
# 1 <> 2;;
- : bool = true
# 2 <= 2;;
- : bool = true
```

2.1 OCaml 기본 구성

▷ Boolean Expressions (논리식)

- 논리 연산자 (논리식 → 논리식)

```
# true && (false || not false);;
- : bool = true
# (2 > 1) && (3 > 2);;
- : bool = false
```

2.1 OCaml 기본 구성

▷ Primitive Values (기본값)

- OCmal은
  - integer (정수)
  - float (실수)
  - boolean (논리)
  - character (문자)
  - string (문자열)
  - unit (유닛)

  을 제공

```
# 'c';;
- : char = 'c'
# "Objecive " ^ "Caml";;
- : string = "Objective Caml"
# ();;
- : unit = ()
```

2.1 OCaml 기본 구성

▷ Conditional Expression (조건식)

$$\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3$$

```
# if 1 then 2 else 3;;
```

2.1 OCaml 기본 구성

▷ Conditional Expression (조건식)

$$\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3$$

- e_1은 반드시 논리식이어야 함. 즉 e_1의 값은 true or false

```
# if 1 then 2 else 3;;
Error: This expression has type int but an
expression was expected of type bool because it is
in the condition of an if-statement
```

2.1 OCaml 기본 구성

▷ Conditional Expression (조건식)

- 조건식의 값은 e_1 값에 따라서 결정

```
# if 2 > 1 then 0 else 1;;
- : int = 0
# if 2 < 1 then 0 else 1;;
- : int = 1
```

- e_2와 e_3는 타입이 같아야 함

```
# if true then 1 else true;;
Error: This expression has type bool but an
expression was expected of type int
```

2.1 OCaml 기본 구성

▷ Function Expression (함수식)

$$\texttt{fun}\ x\ \texttt{->}\ e$$

- 함수의 예:
  * $\texttt{fun}\ x\ \texttt{->}\ x + 1$
  * $\texttt{fun}\ y\ \texttt{->}\ y * y$
  * $\texttt{fun}\ x\ \texttt{->}\ \texttt{if}\ x > 0\ \texttt{then}\ x + 1\ \texttt{else}\ x * x$
  * $\texttt{fun}\ x\ \texttt{->}\ \texttt{fun}\ y\ \texttt{->}\ x + y$
  * $\texttt{fun}\ x\ \texttt{->}\ \texttt{fun}\ y\ \texttt{->}\ \texttt{fun}\ z\ \texttt{->}\ x + y + z$

- Syntactic Sugar

$$\texttt{fun}\ x_1\ \ldots\ x_n\ \texttt{->}\ e$$

  * $\texttt{fun}\ x\ y\ \texttt{->}\ x + y$
  * $\texttt{fun}\ x\ y\ z\ \texttt{->}\ x + y + z$

```
@:~$>ocaml

# let f = fun x y -> x + y;;
val f : int -> int -> int = <fun>
# f 1 2;
- : int = 3
# let g = f 1;
# g 2;;
- : int = 3
```

Advanced OCaml Programming

To be continue ...