Software Verification

Lecture 02. OCaml Programming II

Ji, Yong-Hyeon

24. 07. 25 (Thu)

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

Table of Contents

Solutions for Homework

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	v	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	v	Visual Basic	2.08%	-0.82%
9	11	^	Fortran	2.05%	+0.80%
10	8	v	SQL	2.04%	+0.57%
11	15	^^	Delphi/Object Pascal	1.89%	+0.91%
12	10	v	MATLAB	1.34%	+0.08%
13	17	^^	Rust	1.18%	+0.29%
14	16	^	Ruby	1.16%	+0.25%
15	12	v	Scratch	1.15%	+0.08%

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%

Basic OCaml Programming

OCaml 프로그램의 기본 단위 공식

- 프로그램을 구성하는 두 가지 기본 단위

Statement:

```
x = x + 1
```

Expression:

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

```
let hello = "Hello"
let world = "World"
let helloworld = hello ^ " " ^ world
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;;
```

▷ **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

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

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

▷ **Function Call Expression (함수 호출식)**

$$e_1 \quad e_2$$

```
# (fun x -> x * x) 3;;
- : int = 9
# (fun x -> if x > 0 then x + 1 else x * x) 1;;
- : int = 2
# (fun x -> fun y -> fun z -> x + y + z) 1 2 3;;
- : int = 6
```

```
# (fun f -> f * 1) (fun x -> x * x);;
- : int = 1
# (fun x -> x * x) ((fun x -> if x > 0 then 1 else
2) 3);;
- : int = 2
```

▷ **Let Expressions**

값에 이름 붙이기!

$$\texttt{let}\ x = e_1\ \texttt{in}\ e_2$$

- e_1의 값을 x라고 하고 e_2를 계산
  * x: variable (변수, 값의 이름)
  * e_1: binding expression (정의식)
  * e_2: body expression (몸통식)
- e_2: scope of x (유효범위)

```
# let x = 1 in x + x;;
- : int = 2
# (let x = 1 in x) + x;;
Error: Unbound value x
# (let x = 1 in x) + (let x = 2 in x);;
- : int = 3
```

```
# let x = (let y = 1 in y + 1) in x + 1;;
- : int = 3
# let x = 1 in
        let y = 2 in
                x + y;;
- : int = 3
```

```
# let square = fun x -> x * x in square 2;;
- : int = 4
# let add x y = x + y in add 1 2;;
- : int = 3
```

```
# let rec factorial n =
        if n = 0 then 1
        else n * factorial (n - 1);;
val factorial : int -> int = <fun>
# factorial 5;;
- : int = 120
```

▷ **Pattern Matching (패턴 매칭)**

- 패턴 매칭을 이용한 값의 구조 분석

```
# let rec factorial n =
if n = 0 then 1 else n * factorial (n - 1);;
val factorial : int -> int = <fun>
```

```
# let factorial a =
match a with
0 -> 1
|_ -> a * factorial (a-1);;
val factorial : int -> int = <fun>
```

▷ **Polymorphic Type (다형 타입)**

- 패턴 매칭을 이용한 값의 구조 분석

```
# let rec factorial n =
if n = 0 then 1 else n * factorial (n - 1);;
val factorial : int -> int = <fun>
```

```
# let factorial a =
match a with
0 -> 1
|_ -> a * factorial (a-1);;
val factorial : int -> int = <fun>
```

▷ **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
```

▷ **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 = ()
```

▷ **Conditional Expression (조건식)**

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

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

▷ **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
```

▷ **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
```

▷ **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 ...