

4471028: 프로그래밍언어

Lecture 6 — 표현식
Expressions

임현승
2020 봄학기

강의 주제

● 1부 기본기

- ▶ 귀납법(induction)
- ▶ OCaml을 이용한 함수형 프로그래밍

● 2부 개념

- ▶ 추상 문법 구조(abstract syntax), 실행 의미구조(operational semantics)
- ▶ 표현식(expression), 함수(procedure, function)
- ▶ 변수의 범위(scoping), 변수 바인딩(binding), 변경 가능한 상태(mutable state)
- ▶ 타입 검사(type-checking), 타입 추론(type inference)

● 3부 고급 주제 (optional)

- ▶ 다형 타입 시스템(polymorphic type system)
- ▶ 모듈 시스템(module system)
- ▶ 객체 지향 프로그래밍(object-oriented programming)

개요

간단하고 작은 언어부터 시작해서 단계적으로 새로운 기능들을 추가하며 더 큰 언어들을 정의하고 구현해봄으로써 프로그래밍 언어에서 사용되는 다양한 핵심 개념들을 배운다.

\mathcal{L}^{let} : 간단한 표현식 언어



$\mathcal{L}^{fun} = \mathcal{L}^{let} + \text{고차 함수}$



$\mathcal{L}^{rec} = \mathcal{L}^{fun} + \text{재귀 함수}$



$\mathcal{L}^{ref} = \mathcal{L}^{fun} + \text{변경 가능한 상태}$



$\mathcal{L}^{typed} = \mathcal{L}^{fun} + \text{타입 검사}$



⋮

프로그래밍 언어 정의하기

언어의 문법과 의미구조를 정의해야 한다:

- 문법(Syntax): 프로그램을 작성하는 방법
- 의미구조(Semantics): 프로그램의 의미

본 수업에서는 귀납법을 이용하여 문법과 의미구조를 기술하고 OCaml을 이용하여 구현한다.

\mathcal{L}^{let} : 간단한 표현식 언어

문법구조

Program	$P ::= E$	
Expression	$E ::= n$	정수
	x	변수
	$E + E$	덧셈식
	$E - E$	뺄셈식
	iszero E	0값 테스트
	if E then E else E	조건문
	let $x = E$ in E	지역변수 선언

의미구조

반복문 **while**의 의미를 어떻게 기술할 수 있을까?

while E **do** C

- Informal semantics: *“The command C is executed repeatedly so long as the value of the expression E remains true. The test takes place before each execution of the command”.*
- 장점: 직관적이며 상대적으로 사람이 이해하기가 용이함
- 단점: 애매모호하며 프로그램의 성질을 엄밀하게 분석하고 추론하는데는 부적합함

의미구조

반복문 **while**의 의미를 어떻게 기술할 수 있을까?

while E **do** C

- Formal semantics: 프로그램의 의미가 추론규칙을 이용하여 수학적으로 정의됨

$$\frac{M \vdash E \Rightarrow false}{M \vdash \mathbf{while} E \mathbf{do} C \Rightarrow M}$$

$$\frac{M \vdash E \Rightarrow true \quad M \vdash C \Rightarrow M_1 \quad M_1 \vdash \mathbf{while} E \mathbf{do} C \Rightarrow M_2}{M \vdash \mathbf{while} E \mathbf{do} C \Rightarrow M_2}$$

- 오해의 소지가 적고, 프로그램의 성질을 분석하고 추론하기 위한 기초로 사용될 수 있음

값(Value)

- 실행 의미구조를 정의하기 위해서 먼저 값을 정의해야 된다.
- 프로그램의 의미는 실행 결과로 얻어지는 최종 결과 값
- \mathcal{L}^{let} 에서 사용되는 값들의 집합 Val 은 정수 집합 \mathbb{Z} 와 불리언 집합 $Bool$ 을 포함

$$Val = \mathbb{Z} + Bool$$

실행 환경(Environment)

- 다음으로 변수의 값을 저장하는 실행 환경을 정의해야 함
- 환경은 변수로부터 값으로의 부분 함수(partial function)

$$Env = Var \rightarrow Val$$

표기법:

- 임의의 환경을 지칭하기 위해 메타 변수 ρ 를 사용, i.e., $\rho \in Env$
- $[]$: 아무 정보가 없는 빈 환경(empty environment)
- 환경 예: $[x \mapsto 1]$, $[x \mapsto 1, y \mapsto 3]$, $[x \mapsto 1, y \mapsto 3, z \mapsto true]$, ...
- $\rho(x)$: 환경 ρ 에서 변수 x 의 값을 찾는 연산

$$\rho(x) = \begin{cases} v & \text{if } x \mapsto v \in \rho \\ \text{undefined} & \text{otherwise} \end{cases}$$

실행 환경 업데이트

- $[x \mapsto v]\rho$: 환경 ρ 에서 변수 x 가 값 v 를 가리키도록 업데이트하는 연산

$$([x \mapsto v]\rho)(y) = \begin{cases} v & \text{if } x = y \\ \rho(y) & \text{otherwise} \end{cases}$$

- $[x_1 \mapsto v_1, x_2 \mapsto v_2]\rho$: 환경 ρ 에서 변수 x_1 은 값 v_1 을, x_2 는 v_2 를 가리키도록 업데이트

$$[x_1 \mapsto v_1, x_2 \mapsto v_2]\rho = [x_1 \mapsto v_1]([x_2 \mapsto v_2]\rho)$$

\mathcal{L}^{let} 의 실행 의미구조, 추론 규칙을 이용하여

판단문 $\rho \vdash E \Rightarrow v$ 는 “환경 ρ 에서 표현식 E 는 값 v 로 계산된다”는 의미

$$\frac{}{\rho \vdash n \Rightarrow n} \quad \frac{}{\rho \vdash x \Rightarrow \rho(x)}$$

$$\frac{\rho \vdash E_1 \Rightarrow n_1 \quad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 + E_2 \Rightarrow n_1 + n_2} \quad \frac{\rho \vdash E_1 \Rightarrow n_1 \quad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 - E_2 \Rightarrow n_1 - n_2}$$

$$\frac{\rho \vdash E \Rightarrow 0}{\rho \vdash \mathbf{iszero} E \Rightarrow \mathbf{true}} \quad \frac{\rho \vdash E \Rightarrow n}{\rho \vdash \mathbf{iszero} E \Rightarrow \mathbf{false}} \quad (n \neq 0)$$

$$\frac{\rho \vdash E_1 \Rightarrow \mathbf{true} \quad \rho \vdash E_2 \Rightarrow v}{\rho \vdash \mathbf{if} E_1 \mathbf{then} E_2 \mathbf{else} E_3 \Rightarrow v} \quad \frac{\rho \vdash E_1 \Rightarrow \mathbf{false} \quad \rho \vdash E_3 \Rightarrow v}{\rho \vdash \mathbf{if} E_1 \mathbf{then} E_2 \mathbf{else} E_3 \Rightarrow v}$$

$$\frac{\rho \vdash E_1 \Rightarrow v_1 \quad [x \mapsto v_1]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \mathbf{let} x = E_1 \mathbf{in} E_2 \Rightarrow v}$$

“표현식 E 가 환경 ρ 하에서 의미를 갖는다”는 말은 “어떤 값 v 에 대해서 판단문 $\rho \vdash E \Rightarrow v$ 의 증명 나무를 (공리에서부터 시작해서 추론 규칙을 유한 번만 적용해서) 유도할 수 있다”는 말과 동치

예제: 정수식

- 환경 $\rho = [i \mapsto 1, v \mapsto 5, x \mapsto 10]$ 하에서 프로그램 $(x - 3) - (v - i)$ 는 의미를 가지며 그 값은 3
- 증명

$$\frac{\frac{\overline{\rho \vdash x \Rightarrow 10} \quad \overline{\rho \vdash 3 \Rightarrow 3}}{\rho \vdash x - 3 \Rightarrow 7} \quad \frac{\overline{\rho \vdash v \Rightarrow 5} \quad \overline{\rho \vdash i \Rightarrow 1}}{\rho \vdash v - i \Rightarrow 4}}{\rho \vdash (x - 3) - (v - i) \Rightarrow 3}$$

- 어떤 값 v 에 대해서도 다음 판단문을 증명할 수 없기 때문에 표현식 $y - 3$ 은 의미가 없음

$$\rho \vdash y - 3 \Rightarrow v$$

- 환경 $\rho = [x \mapsto \text{true}]$ 하에서 어떤 값 v 에 대해서도 다음 판단문을 증명할 수 없기 때문에 $x + 1$ 의 의미는 정의되지 못 함

$$\rho \vdash x + 1 \Rightarrow v$$

예제: 조건식

$\rho = [x \mapsto 33, y \mapsto 22]$ 일 때,

if iszero ($x - 11$) **then** $y - 2$ **else** $y - 4$

는 잘 정의되어 있으며(well-defined), 그 값은 18:

$$\frac{\frac{\overline{\rho \vdash x \Rightarrow 33} \quad \overline{\rho \vdash 11 \Rightarrow 11}}{\rho \vdash x - 11 \Rightarrow 22}}{\rho \vdash \text{iszero}(x - 11) \Rightarrow \text{false}} \quad \frac{\overline{\rho \vdash y \Rightarrow 22} \quad \overline{\rho \vdash 4 \Rightarrow 4}}{\rho \vdash y - 4 \Rightarrow 18}$$
$$\rho \vdash \text{if iszero}(x - 11) \text{ then } y - 2 \text{ else } y - 4 \Rightarrow 18$$

예제: 지역 변수 선언식

let-표현식은 새로운 변수 바인딩(variable binding)을 환경에 추가함



$$\frac{[] \vdash 5 \Rightarrow 5 \quad \frac{\frac{[x \mapsto 5] \vdash x \Rightarrow 5}{[x \mapsto 5] \vdash 3 \Rightarrow 3}}{[x \mapsto 5] \vdash x - 3 \Rightarrow 2}}{[] \vdash \mathbf{let\ } x = 5 \mathbf{ in\ } x - 3 \Rightarrow 2}$$

- 환경이 $[x \mapsto 7, y \mapsto 2]$ 일 때, 프로그램

let $y = (\mathbf{let\ } x = x - 1 \mathbf{ in\ } x - y) \mathbf{ in\ } x - 8 - y$

은 -5 로 계산됨:

$$\frac{\frac{\frac{\dots}{[x \mapsto 7, y \mapsto 2] \vdash x - 1 \Rightarrow 6} \quad \frac{\frac{\dots}{[x \mapsto 6, y \mapsto 2] \vdash x - y \Rightarrow 4}}{[x \mapsto 7, y \mapsto 2] \vdash \mathbf{let\ } x = x - 1 \mathbf{ in\ } x - y \Rightarrow 4} \quad \frac{\dots}{[x \mapsto 7, y \mapsto 4] \vdash x - 8 - y \Rightarrow -5}}{[x \mapsto 7, y \mapsto 2] \vdash \mathbf{let\ } y = (\mathbf{let\ } x = x - 1 \mathbf{ in\ } x - y) \mathbf{ in\ } x - 8 - y \Rightarrow -5}$$

구현: 문법

OCaml 타입 정의를 이용하여 작성한 추상 문법 구조:

```
type program = exp          (* P ::= E          *)
and exp =                   (* E ::=          *)
  CONST of int              (*      n          *)
  | VAR of var              (*      | x          *)
  | ADD of exp * exp        (*      | E + E      *)
  | SUB of exp * exp        (*      | E - E      *)
  | ISZERO of exp           (*      | iszero E    *)
  | IF of exp * exp * exp   (*      | if E then E else E *)
  | LET of var * exp * exp  (*      | let x = E in E      *)
and var = string
```

예제:

```
# ADD (CONST 1, VAR "x");;
- : exp = ADD (CONST 1, VAR "x")
# IF (ISZERO (CONST 1), ADD (CONST 1, VAR "x"), CONST 3);;
- : exp = IF (ISZERO (CONST 1), ADD (CONST 1, VAR "x"), CONST 3)
```

구현: 값과 실행 환경

값:

```
type value = Int of int | Bool of bool
```

환경:

```
type env = var -> value
exception Not_found
(* val empty : env *)
let empty = fun x -> raise Not_found
(* val lookup : var -> env -> value *)
let lookup x e = e x
(* val extend : var -> value -> env -> env *)
let extend x v e = fun y -> if x = y then v else lookup e y
```

- $[] = \text{empty}$
- $\rho(x) = \text{lookup } x \rho$
- $[x \mapsto v]\rho = \text{extend } x v \rho$

구현: 실행 의미구조

```
let rec eval : exp -> env -> value
= fun exp env ->
  match exp with
  | CONST n -> Int n
  | VAR x -> lookup x env
  | ADD (e1,e2) ->
    let v1 = eval e1 env in
    let v2 = eval e2 env in
    (match v1,v2 with
     | Int n1, Int n2 -> Int (n1 + n2)
     | _ -> raise (Failure "Type Error: non-numeric values"))
  | SUB (e1,e2) ->
    let v1 = eval e1 env in
    let v2 = eval e2 env in
    (match v1,v2 with
     | Int n1, Int n2 -> Int (n1 - n2)
     | _ -> raise (Failure "Type Error: non-numeric values"))
  ...
```

고차 함수를 이용한 코드 재사용

ADD와 SUB을 처리하는데 사용되는 공통 패턴을 다음과 같은 고차 함수로 추출할 수 있음

```
let rec eval_bop :  
  (int -> int -> int) -> exp -> exp -> env -> value  
= fun op e1 e2 env ->  
  let v1 = eval e1 env in  
  let v2 = eval e2 env in  
  (match v1,v2 with  
  | Int n1, Int n2 -> Int (op n1 n2)  
  | _ -> raise (Failure "Type Error: non-numeric values"))
```

eval 정의에서 다음과 같이 eval_bop을 호출

```
| ADD (e1,e2) -> eval_bop (+) e1 e2 env  
| SUB (e1,e2) -> eval_bop (-) e1 e2 env
```

구현: 실행 의미구조

```
let rec eval : exp -> env -> value
= fun exp env ->
  ...
  | ISZERO e ->
    (match eval e env with
     | Int n -> if n = 0 then Bool true else Bool false
     | _ -> raise (Failure "Type Error"))
  | IF (e1,e2,e3) ->
    (match eval e1 env with
     | Bool true -> eval e2 env
     | Bool false -> eval e3 env
     | _ -> raise (Failure "Type Error"))
  | LET (x,e1,e2) ->
    let v1 = eval e1 env in
    eval e2 (extend x v1 env)
```

예제

프로그램 실행:

```
let run : program -> value =  
  fun pgm -> eval pgm empty
```

예제:

```
# let e1 = LET ("x", CONST 1, ADD (VAR "x", CONST 2));;  
val e1 : exp = LET ("x", CONST 1, ADD (VAR "x", CONST 2))  
# run e1;;  
- : value = Int 3
```

요약

간단한 표현식 언어인 \mathcal{L}^{let} 을 정의하고 구현해봤음

$$\begin{array}{lcl} P & ::= & E \\ E & ::= & n \\ & | & x \\ & | & E + E \\ & | & E - E \\ & | & \text{iszero } E \\ & | & \text{if } E \text{ then } E \text{ else } E \\ & | & \text{let } x = E \text{ in } E \end{array}$$

배운 것들:

- 프로그래밍 언어의 문법과 의미구조를 엄밀하게 정의하는 방법
- 핵심 언어 요소: 값과 실행 환경, 변수 바인딩
- 주어진 언어 정의를 보고 이를 구현하는 법

간단한 숙제

- \mathcal{L}^{let} 을 구현한 `let.ml` 파일을 다운 받아
- 아래 프로그램을 추상 문법 구조로 작성하고
- `run` 함수를 이용하여 계산해볼 것

```
▶ let x = 7 in
  let y = 2 in
  let y =
    let x = x - 1 in
    x - y
  in
  (x - 8) - y

▶ let z = 5 in
  let x = 3 in
  let y = x - 1 in
  let x = 4 in
  z - (x - y)
```