4471028: 프로그래밍언어

Lecture 8 — 변수의 유효 범위와 바인딩
Scoping and Binding

임현승
2020 봄학기

참조와 선언

프로그래밍 언어에서 변수는 두 가지 형태로 나타남:

- 변수 참조(variable reference): 변수를 사용할 때(use of the variable)
- 변수 선언(variable declaration): 어떤 값에 대한 이름으로써 변수를 도입할 때
- 문법적으로 올바른 프로그램(well-formed program)에서 모든 변수 참조는 반드시 변수 선언문에 묶여 있다(where the variable is *bound to* its value).
- 예제:

```
fun x -> x + 3
```

```
let x = y + 7 in x + 3
```

바인딩

- 바인딩(binding): 변수와 값의 결합, 연결(association); *i.e.,* 실행 환경은 변수 바인딩들을 모아 놓은 것
- $\mathcal{L}^{rec}$에서 바인딩은 아래 표현식들에서만 만들어짐
  - `let` 표현식:

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

  - `let rec` 표현식:

$$\frac{[f \mapsto (f, x, E_1, \rho)]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{let rec}\ f\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v}$$

  - 함수 적용식:

$$\frac{\rho \vdash E_1 \Rightarrow (x, E, \rho') \qquad \rho \vdash E_2 \Rightarrow v \qquad [x \mapsto v]\rho' \vdash E \Rightarrow v'}{\rho \vdash E_1\ E_2 \Rightarrow v'}$$

유효 범위 규칙

- 변수 참조가 가리키는 변수 선언문을 결정하는 방법은? 유효 범위 규칙(scoping rules)을 이용
- 대부분의 프로그래밍 언어에서는 *lexical scoping* 규칙을 이용—where the declaration of a variable is found by searching outward from the use until we find a declaration of the variable:

```
let x = 3 in          // call this x1
let y = 4 in
(let x = y + 5 in     // call this x2
 x * y)               // Here x refers to x2
 + x                  // Here x refers to x1
```

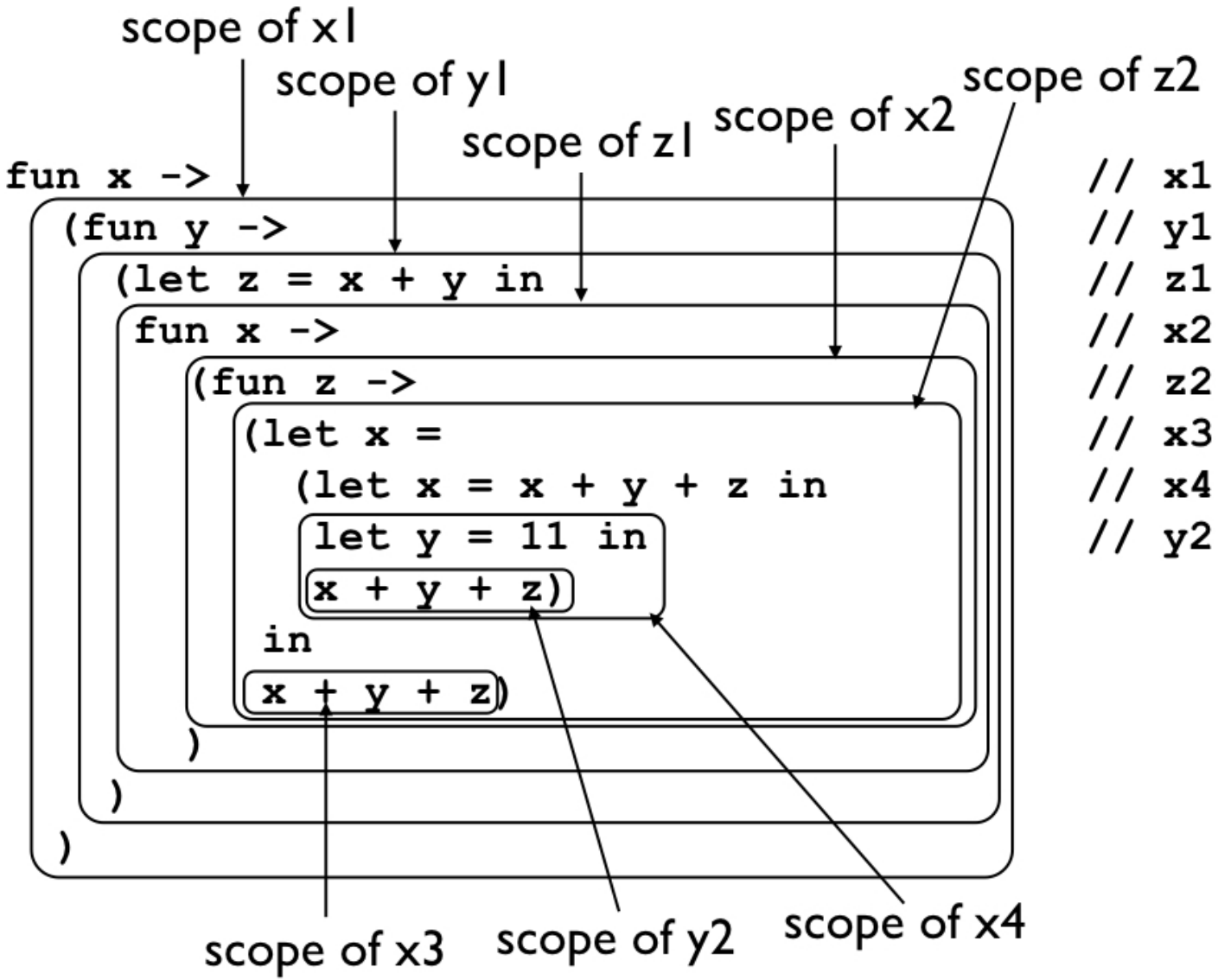
변수의 유효 범위

Lexical scope은 중첩될 수 있음: each scope lies entirely within another.

```
fun x ->                                          // x1
  (fun y ->                                       // y1
    (let z = x + y in                             // z1
     fun x ->                                     // x2
       (fun z ->                                  // z2
         (let x =                                 // x3
           (let x = x + y + z in                  // x4
            let y = 11 in                         // y2
            x + y + z)
          in
          x + y + z)
       )
    )
  )
```

변수의 유효 범위(Contour Diagram)

Lexical scope은 중첩될 수 있음: each scope lies entirely within another.



프로그램의 정적 특성과 동적 특성

- 정적 특성(static properties)은 컴파일 시간에 결정됨
  - 예: 변수의 선언, 유효 범위 등
- 동적 특성(dynamic properties)은 실행 시간에 결정됨
  - 예: 변수의 값, 수명, 오류의 부재 등

Lexical Address

- 변수 참조의 *lexical depth*는 연관된 선언문을 찾기 위해 가로지르는 변수 선언문의 수(the number of declarations crossed to find the associated declaration)

```
let x = 1 in
let y = 2 in
x + y
```

- 변수 참조의 lexical depth를 이용하여 변수가 가리키는 선언문을 식별할 수 있음
- 따라서 변수 참조를 해당 lexical address로 치환함으로써 프로그램에서 변수 이름을 모두 제거할 수 있음.

```
let 1 in
let 2 in
#1 + #0
```

이러한 표기법을 "*nameless*" 또는 "*De Bruijn*" 표기법이라고 함.

연습: 이름 없는 표현법(Nameless Representation)

- `(let a = 5 in fun x -> x - a) 7`
- ```
  (let x = 37 in
   fun y ->
     let z = y - x in
     x - y
  ) 10
  ```
  ```

Lexical Address

- 실행 환경을 association list로 표현할 때, 변수의 lexical address는 실행 환경에서 변수의 위치
- ```
  let x = 1 in
  let y = 2 in
  x + y
  ```
- ```
  (let a = 5 in fun x -> x - a) 7
  ```

이름 없는 $\mathcal{L}^{fun}$

문법 구조

Program	P	$::=$	N	
Expression	N	$::=$	n	
		$\mid$	$\#n$	(lexical address)
		$\mid$	$N + N$	
		$\mid$	$N - N$	
		$\mid$	`iszero` N	
		$\mid$	`if` N `then` N `else` N	
		$\mid$	`let` N `in` N	
		$\mid$	`fun` N	
		$\mid$	$N\ N$	

이름 없는 $\mathcal{L}^{fun}$

의미 구조

$$\begin{aligned} NVal &= \mathbb{Z} + Bool + NFun \\ NFun &= NExp \times NEnv \\ \varphi \in NEnv &= NVal\ list \end{aligned}$$

$$\frac{}{\varphi \vdash n \Rightarrow n} \qquad \frac{}{\varphi \vdash \#n \Rightarrow nth(\varphi, n)} \qquad \frac{\varphi \vdash N_1 \Rightarrow n_1 \qquad \varphi \vdash N_2 \Rightarrow n_2}{\varphi \vdash N_1 + N_2 \Rightarrow n_1 + n_2}$$

$$\frac{\varphi \vdash N \Rightarrow 0}{\varphi \vdash \texttt{iszero}\ N \Rightarrow true} \qquad \frac{\varphi \vdash N \Rightarrow n}{\varphi \vdash \texttt{iszero}\ N \Rightarrow false}\ (n \neq 0)$$

$$\frac{\varphi \vdash N_1 \Rightarrow true \qquad \varphi \vdash N_2 \Rightarrow v}{\varphi \vdash \texttt{if}\ N_1\ \texttt{then}\ N_2\ \texttt{else}\ N_3 \Rightarrow v} \qquad \frac{\varphi \vdash N_1 \Rightarrow false \qquad \varphi \vdash N_3 \Rightarrow v}{\varphi \vdash \texttt{if}\ N_1\ \texttt{then}\ N_2\ \texttt{else}\ N_3 \Rightarrow v}$$

$$\frac{\varphi \vdash N_1 \Rightarrow v_1 \qquad v_1 :: \varphi \vdash N_2 \Rightarrow v}{\varphi \vdash \texttt{let}\ N_1\ \texttt{in}\ N_2 \Rightarrow v} \qquad \frac{}{\varphi \vdash \texttt{fun}\ N \Rightarrow (N, \varphi)}$$

$$\frac{\varphi \vdash N_1 \Rightarrow (N, \varphi') \qquad \varphi \vdash N_2 \Rightarrow v \qquad v :: \varphi' \vdash N \Rightarrow v'}{\varphi \vdash N_1\ N_2 \Rightarrow v'}$$

예제

$$[] \vdash \texttt{(let 37 in fun (let (\#0 - \#1) in (\#2 - \#1))) 10} \Rightarrow \mathbf{27}$$

코드 변환

아래 함수 **T**는 프로그램 P를 이름 없는 프로그램으로 변환

$$\mathbf{T} : Exp \to Var\ list \to NExp$$

$$\begin{aligned}
\mathbf{T}(n)(\varphi) &= n \\
\mathbf{T}(x)(\varphi) &= \#n \qquad (n \text{ is the first position of } x \text{ in } \varphi) \\
\mathbf{T}(E_1 + E_2)(\varphi) &= \mathbf{T}(E_1)(\varphi) + \mathbf{T}(E_2)(\varphi) \\
\mathbf{T}(\texttt{iszero}\ E)(\varphi) &= \texttt{iszero}\ (\mathbf{T}(E)(\varphi)) \\
\mathbf{T}(\texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3)(\varphi) &= \texttt{if}\ \mathbf{T}(E_1)(\varphi)\ \texttt{then}\ \mathbf{T}(E_2)(\varphi)\ \texttt{else}\ \mathbf{T}(E_3)(\varphi) \\
\mathbf{T}(\texttt{let}\ x = E_1\ \texttt{in}\ E_2)(\varphi) &= \texttt{let}\ \mathbf{T}(E_1)(\varphi)\ \texttt{in}\ \mathbf{T}(E_2)(x :: \varphi) \\
\mathbf{T}(\texttt{fun}\ x \texttt{ -> } E)(\varphi) &= \texttt{fun}\ \mathbf{T}(E)(x :: \varphi) \\
\mathbf{T}(E_1\ E_2)(\varphi) &= \mathbf{T}(E_1)(\varphi)\ \mathbf{T}(E_2)(\varphi)
\end{aligned}$$

예제

$$\mathbf{T}\left(\begin{array}{l}\texttt{(let x = 37 in}\\ \texttt{ fun y ->}\\ \texttt{ let z = y - x in}\\ \texttt{ x - y}\\ \texttt{) 10}\end{array}\right)([]) =$$

요약

- Lexical scoping에서 유효 범위 규칙은 정적 특성: 표현식을 lexical address를 이용하여 이름 없는 표현식으로 변환 가능
- Lexical address는 실행 환경에서 해당 변수의 위치
- 컴파일러에서 이름 없는 표현식을 이용: 주어진 프로그램 P에 대해
  1. 먼저 $\mathbf{T}(P)([])$로 변환하고
  2. 이름 없는 프로그램을 실행