Lecture 17
Scoping & Binding

T. METIN SEZGIN

What is the value of this expression?

```
let a = 3
in let p = proc (x) -(x,a)
       a = 5
   in -(a,(p 2))
```

Denoted values

- Variables
  - References

```
(f x y)
```

  - Declarations

```
(lambda (x) (+ x 3))

(let ((x (+ y 7))) (+ x 3))
```

- Semantics
  - Binding
  - Scope

scope

Did you mean: -scope (suffix), scope (philosophy), radarscope, cathode-ray oscilloscope (electronics), Scope (drink), Viewing instrument, Scope (charity), Scope (project management) **More...**

Dictionary: scope (skōp)

n.

1. The range of one's perceptions, thoughts, or actions.
2. Breadth or opportunity to function. See synonyms at **room**.
3. The area covered by a given activity or subject. See synonyms at **range**.
4. The length or sweep of a mooring cable.
5. *Informal.* A viewing instrument such as a periscope, microscope, or telescope.

tr.v. Slang, **scoped**, **scop·ing**, **scopes**.
To examine or investigate. Often used with *out*: *"Their World Wide Web site is, for now, the best place to scope out the future of the media business in cyberspace."* (Marc Gunther).

[Italian *scopo*, aim, purpose, from Greek *skopos*, target, aim.]

Denoted values

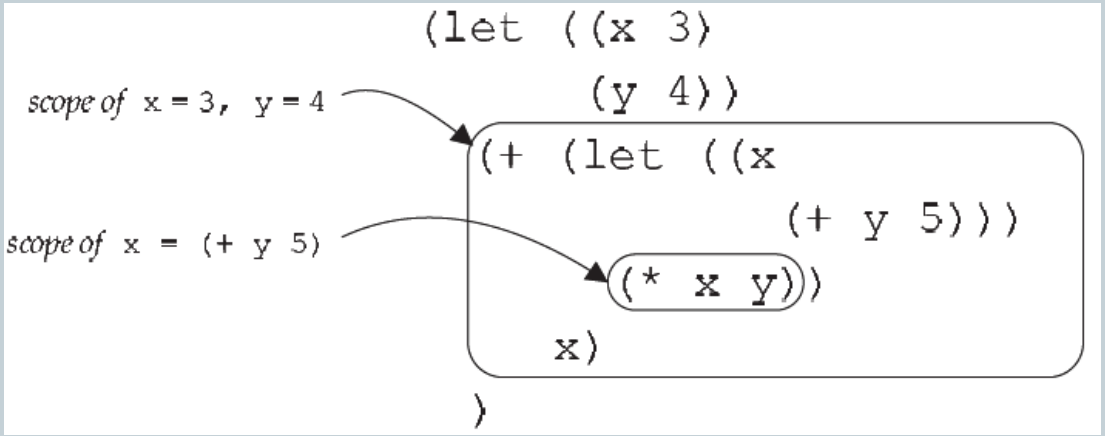
- Variables
  - References

```
(f x y)
```

  - Declarations

```
(lambda (x) (+ x 3))
(let ((x (+ y 7))) (+ x 3))
```

- Semantics
  - Binding
  - Scope



we need rules to define scoping

Scoping

- Static scoping
  - Declarations and references can be matched without code execution
  - Search "outward"

```
(let ((x 3)                          Call this x1
      (y 4))
  (+ (let ((x                        Call this x2
           (+ y 5)))
       (* x y))                      Here x refers to x2
     x))                             Here x refers to x1
```

- Dynamic scoping
  - Declarations and references are matched during code execution
  - a in the proc bound to 5

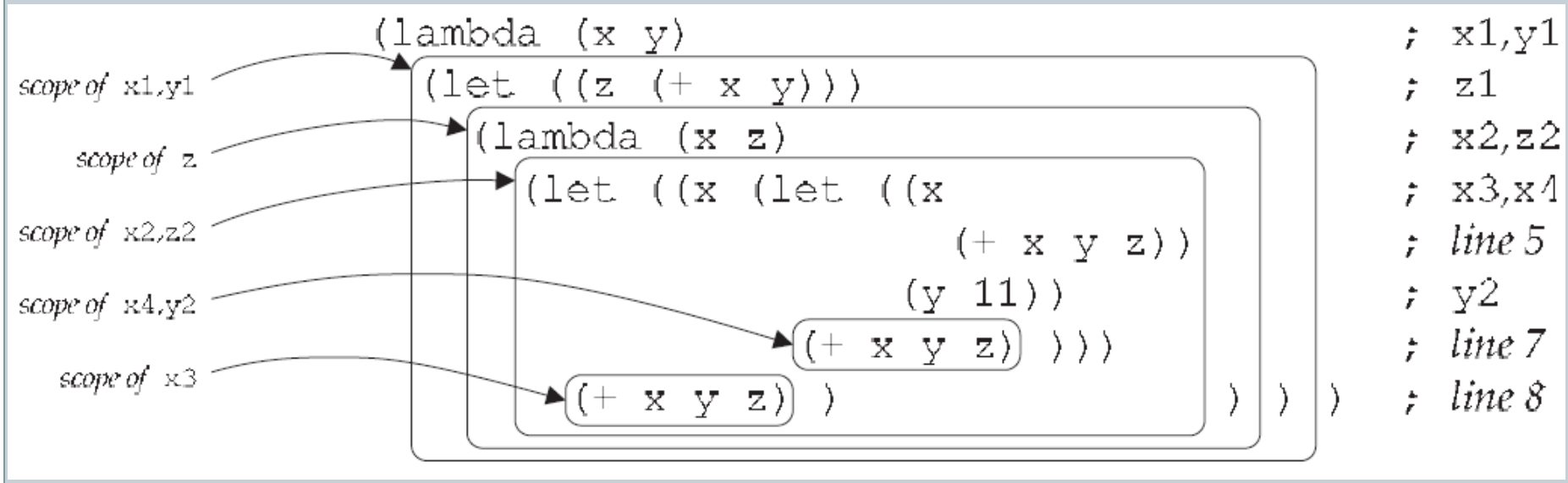
```
let a = 3
in let p = proc (x) -(x,a)
       a = 5
   in -(a,(p 2))
```

Concepts

- Shadowing
- Holes
- Extent
  - Duration of the binding
- Contour diagram
  - Helps resolving bindings
- Lexical depth

```
(let ((x 3)                          Call this x1
      (y 4))
  (+ (let ((x                        Call this x2
           (+ y 5)))
       (* x y))                      Here x refers to x2
     x))                             Here x refers to x1
```

Another example



How are the binding rules defined?

How are the binding rules defined?

```
(apply-procedure (procedure var body ρ) val)
= (value-of body (extend-env var val ρ))


(value-of (let-exp var val body) ρ)
= (value-of body (extend-env var val ρ))


(value-of
   (letrec-exp proc-name bound-var proc-body letrec-body)
   ρ)
= (value-of
     letrec-body
      (extend-env-rec proc-name bound-var proc-body ρ))
```