Chapter 4: Types

Why Have Types?

Detect simple kinds of errors:

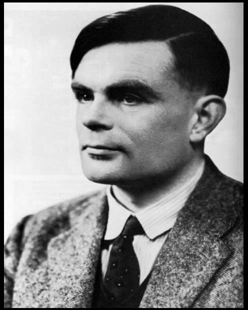
```
 int square(int x) { return x*x; }

 ...

 int bogus = square(“hello”);


String bogus2 = square(3);
```

Why Have Types?

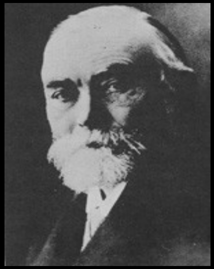
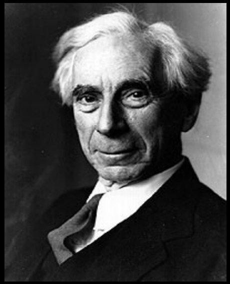
Detect simple kinds of errors:

```
double diff(double x, double y) {
   return x – y;
}
...
double d = diff(3.2, 4.0, -5.1);
```

Why Have Types?



Recall Russell’s Paradox: The barber who shaves only those who don’t shave themselves (naïve set theory)

Static vs. Dynamic (Latent) Type-Checking

- *Static*: checked by compiler, or part of grammar (PS4 argument-count example)

- *Dynamic*: Every value is *tagged* with a *latent* type; types are checked at run-time (Scheme)

```
(+                  3            'abc)
[int*int->int]      [int]        [string]
```

A Little Statically-Typed Language

<type-exp> ::= **`int`**

```
int-type-exp ()
```

<type-exp> ::= **`bool`**

```
bool-type-exp ()
```

e.g., **`(int*int) -> bool`**

<type-exp> ::= **`(`**{type-exp>}*(*) **`->`** <type-exp>**`)`**

```
proc-type-exp (arg-texps result-texp)
```

Strong Static Type Checking

- No program that passes the type checker (compiler) will contain a type error.

- Want a procedure **`type-of-expression`** to compute the value of a given type in a given type-environment.

- Will have to contain a separate rule (case) for every kind of construct we support; e.g., **`if`**:

Static Type Checking

(type-of-expression <<*test-exp*>> *tenv* **)** = **bool**

(type-of-expression <<*true-exp*>> *tenv* **)** = *x*

(type-of-expression <<*false-exp*>> *tenv* **)** = *x*

(**type-of-expression**

if <<*test-exp* **then** *true-exp* **else** *false-exp*>>

tenv **)** = *x*

(Book uses *t* instead of *x* - confusing!)

Mathematical Background

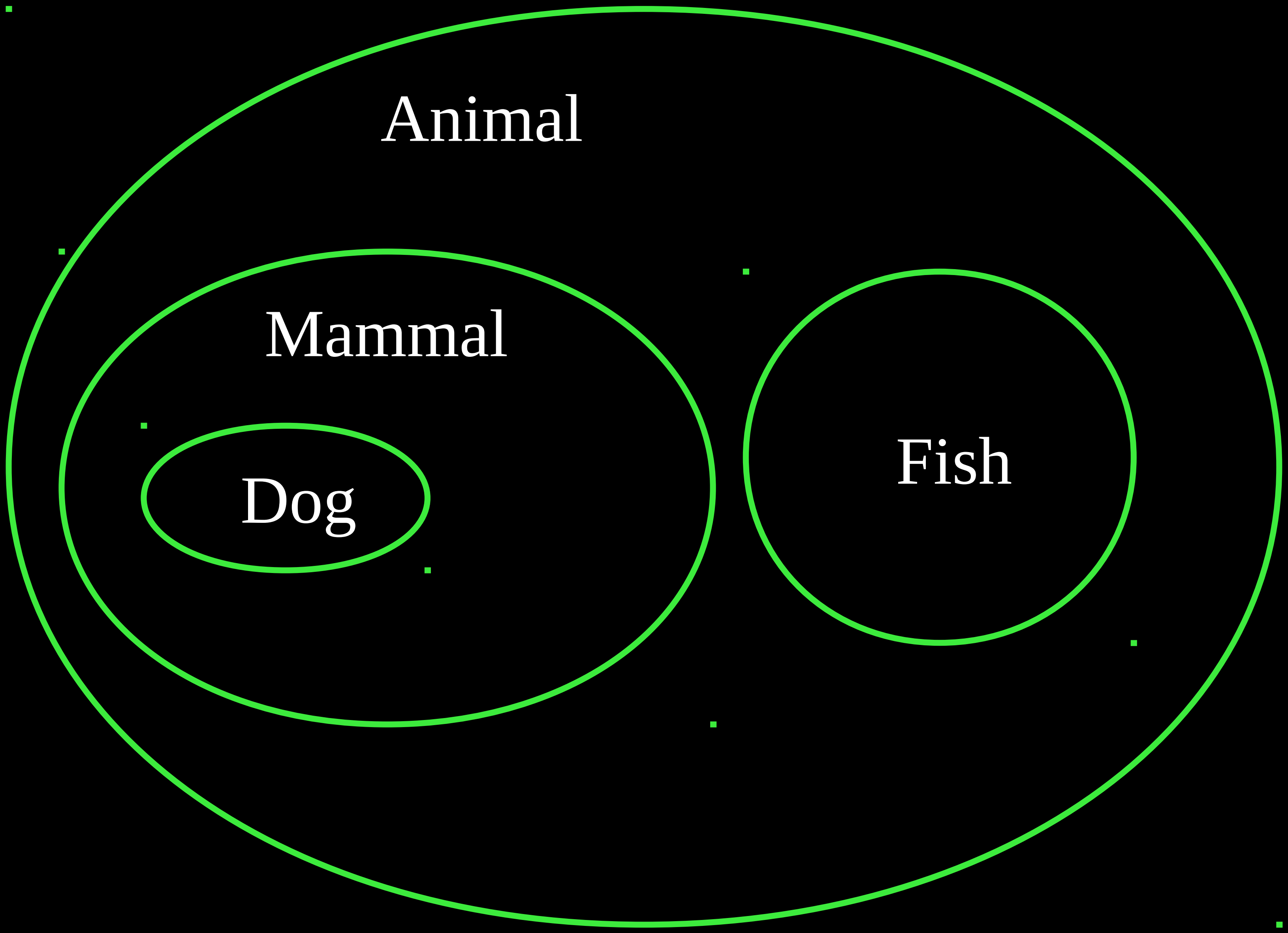
- *Type*: a set

- OOP classes also are sets.

- But with types, we're more interested

in functions *from* one type *to* another.

OOP



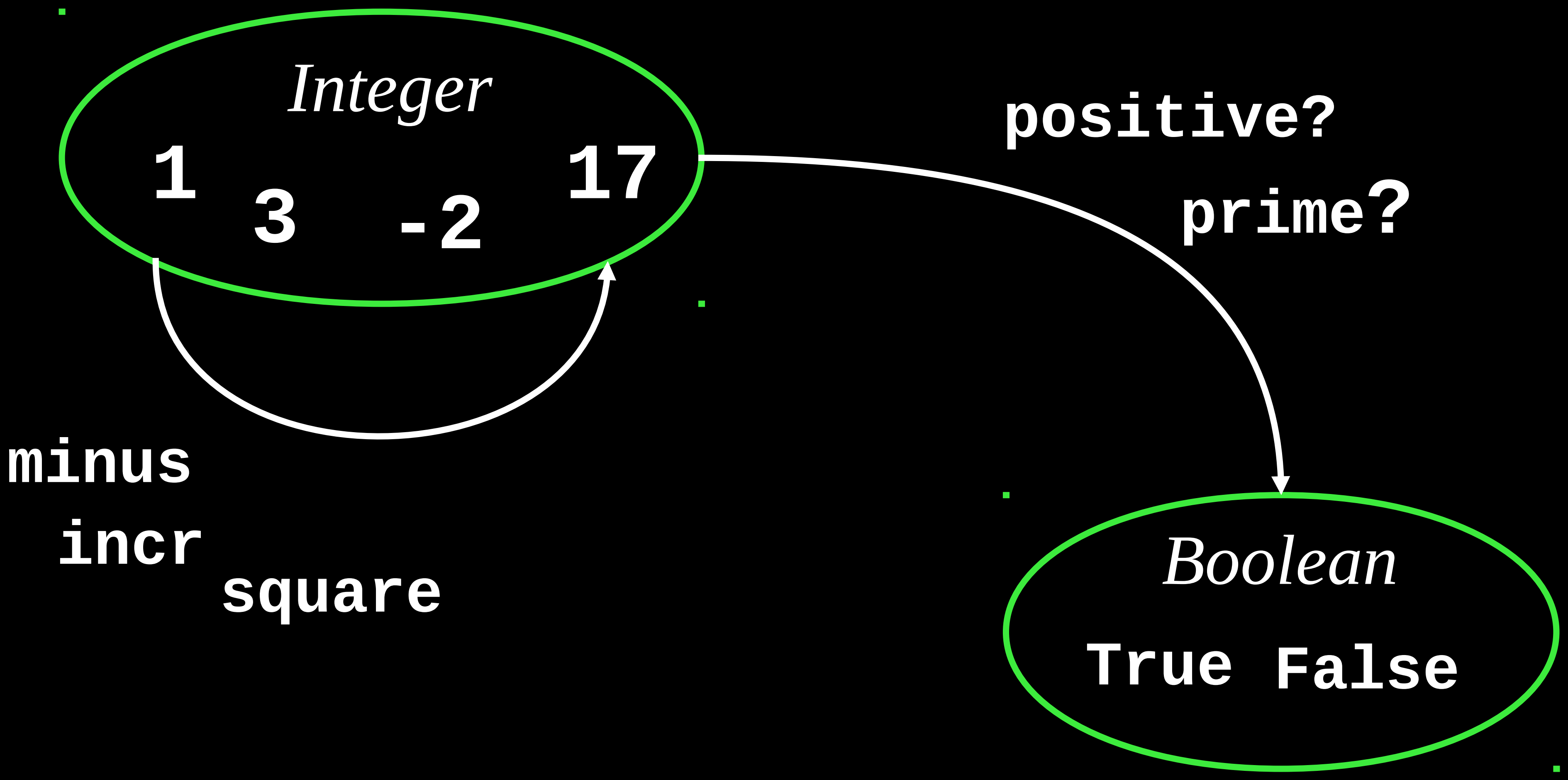
Animal

Mammal

Dog

Fish

Types



Integer

1 3 -2 17

positive?

prime?

minus

incr

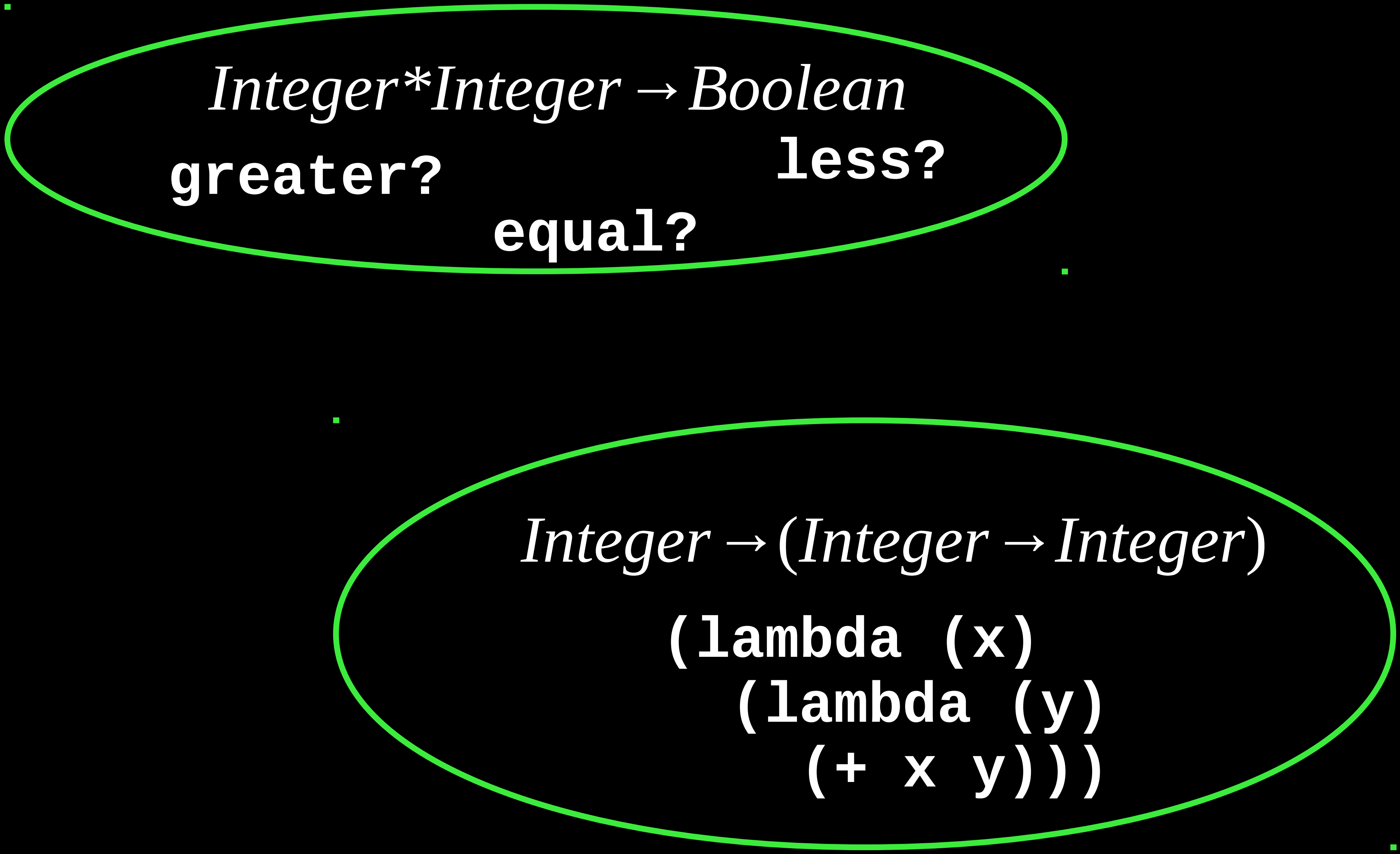
square

Boolean

True False

```
square : int -> int
prime? : int -> boolean
```

Types



$Integer{*}Integer \rightarrow Boolean$

```
greater?        less?
       equal?
```

$Integer \rightarrow (Integer \rightarrow Integer)$

```
(lambda (x)
  (lambda (y)
    (+ x y)))
```