type-of :: Expr, TEnv → Type

TEnv = Dict<Type>

```
data Type:
  | t-num
  | t-bool
  | t-arrow(
      a :: Type,
      r :: Type)
```

(lam (x) 4)

((lam (x) (if x 4 5)) true) :: t-num

t-clos

((lam (x) (if x 4 5)) 2) raise t-not-a-bool

(let (x 4)
  x) :: t-num

Num → Num

x raises t-unbound-id

((lam (x Num) x) 4) :: t-num t-arrow(t-num, t-num)

((lam (x Bool) x) true) :: t-bool t-arrow(t-bool, t-bool)

```
fun type-of(e :: Expr, tenv :: TEnv) -> Type:
  cases(Expr) e:
    | e-let(x, expr, body) =>
      new-te = tenv.add(x, type-of(expr, tenv))
      type-of(body, new-te)
    | e-id(x) => tenv.get(x)   # check unbound
    | e-lam(x, t, body) =>
      t-arrow(t, type-of(body, tenv.set(x, t)))
    | e-app(f, arg) =>
      ft = type-of(f, tenv)
      at = type-of(arg, tenv)
      when not(is-t-arrow(ft)): raise(t-not-a-fun) end
      argtyp-from-ft = ft.a
      rettyp = ft.r
      when not(argtype-from-ft == at): raise(t-bad-app) end
      rettyp
```

ASSUME → t-arrow(t , type-of(body, tenv.set(x, t))

GUARANTEE → rettyp

```
(let (f (lam (x Bool) (if x 4 5)))
  (let (dontcare1 (f true))
    (f false)))
```

ONLY TYPE-CHECKED ONCE

---

op := + | < | ÷

(÷ <exp> <exp>) :: t-num

↑ t-num ↑ t-num (t-zero)

(+ <exp> <exp>)

t-zero	t-zero	t-z
t-z	t∧z	t∧z
t∧z	t∧z	number

DESIGN DECISION

data Error:

| not-a-num	✓
| not-a-bool	✓
| not-a-fun	✓
| div-0	

data Type

| t-zero
| t-∧zero

((lam (x (→)) (x x))
 (lam (x) (x x)))

(((→) →) →) →

STRONG NORMALIZATION

You can't type-check infinite loops.

Every type-checked prog terminates.

(Today's TC)