```
data AE where
        Num Int
        Plus AE AE

t ::= Num
        | True
        | False
        | t+t
        | t-t
        | bind id = t in t
        | id
        | if t then t else t
        | t <= t
        | t && t
        | isZero t

v ::= Num
        | True
        | False
```

If … then error “abc” else …

```
eval :: AE -> Maybe Int
eval (Num x) = if x<0 then Nothing else (Just x)
eval (Plus l r) =  do { (Num l') <- eval l;
                        (Num r') <- eval r;
                                                return (Num (l'+r')) }

typeof :: ABE -> Maybe TABE
typeof (Plus l r) = do {TNum <- typeof l;
                                                TNum <- typeof r;
                                                return TNum}
```

Formal systems: Syntax, Inference System, Semantics
New language:
(C)oncrete Syntax
(A)bstract Syntax
Inference (R)ules
(E)valuation

$$\frac{X, Y}{X \wedge Y} \qquad \frac{}{c} \qquad \frac{eval\ t_1 = v_1, eval\ t_2 = v_2}{eval\ t_1 \underline{+} t_2 = v_1 + v_2}$$

$$\frac{t_1 \Downarrow v_1, t_2 \Downarrow v_2, v_1 \geq v_2}{t_1 \underline{-} t_2 \Downarrow\ v_1 - v_2} \qquad \frac{t \Downarrow v, v = 0}{isZero\ t \Downarrow\ true}$$

$$\frac{t \Downarrow v, v \neq 0}{isZero\ t \Downarrow\ false} \qquad \frac{t_1 \Downarrow\ true, t_2 \Downarrow\ v_2}{if\ t_1\ then\ t_2\ else\ t_3 \Downarrow\ v_2}$$

$$\frac{t_1 \Downarrow\ false, t_3 \Downarrow\ v_3}{if\ t_1\ then\ t_2\ else\ t_3 \Downarrow\ v_3}$$

optimize :: ABE -> ABE
optimize (Num n) = (Num n)
optimize (Plus l (Num 0)) = (optimize l)

$$\frac{n >= 0}{(Num\ n) :\ TNum} \qquad \frac{c : TBool, t : T, e : T}{if\ c\ then\ t\ else\ e : T}$$

bind id = 1+2 in
id + id - 6

lookupId :: Eq a => a -> [(a, t)] -> (Maybe t)
lookupId search [] = Nothing
lookupId search ((identifier, expression):ids) = if search == identifier then return expression else
lookupId search ids

<instance, free instance> Bind <binding instance> = <bound value> = (scope:)
<instance, bound instance> + 3
Definitions
instance - any occurrence of an identifier
binding instance - identifier instance where the identifier is declared and given a value (occurs immediately following bind keyword)
bound value - value given to an identifier (occurs following the =)
scope - code region where an identifier is defined (term immediately following in)
bound instance - identifier instance in its scope
free instance - identifier instance outside its scope

$$[x \mapsto 3]x + 3 == 3 + 3 == 6$$

$$\frac{t \Downarrow v_t, [i \mapsto v_t]b \Downarrow v_s}{bind\ i = t\ in\ b \Downarrow v_s}$$

```
subst :: string -> BAE -> BAE -> BAE
subst i v (Bind i' v' b') = if i==i'
                    then (Bind i' (subst i v v') b')
                    else (Bind i' (subst i v v') (subst i v b'))

type Env = [(string, BAE)]
eval :: Env -> BAE -> (Maybe BAE)
eval e (Id i) = (lookup i e)
eval e (Bind i v b) = do {v' <- eval e v;
                                                (eval (i,v'):e b))}
```

$$(x, T) : \Gamma \quad \Gamma \vdash t : T$$

$$\frac{n \geq 0}{\Gamma \vdash (Num\ n) :\ TNum}$$

$$\frac{\Gamma \vdash v : T_v,\ ((i, T_v) : \Gamma) \vdash b : T_b}{\Gamma \vdash bind\ i = v\ in\ b : T_b}$$

$$\frac{(i, T) \in \Gamma}{\Gamma \vdash (Id\ i) : T}$$