Gradual Liquid Types

Niki Vazou[*], Éric Tanter[+], and David Van Horn[*]
[*] University of Maryland [+] University of Chile

Liquid Types

fast type checking & inference for

Refinement Types

Refinement Types

```
(/) :: Int -> {v:Int | 0 < v } -> Int
```

Basic Type

Refinement

Refinement Types

```
(/) :: Int -> {v | 0 < v } -> Int
```

Simplification

Refinement Types

```
(/)    :: Int -> {v | 0 < v } -> Int
isPos :: Int -> Bool
```

```
divIf :: Int -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

Q: What is a refinement type for divIf ?

A: Let's ask Liquid Inference!

Liquid Types

```
(/)   :: Int -> {v | 0 < v } -> Int
isPos :: Int -> Bool
```

```
divIf :: x:{Int | false } -> {Int | false }
divIf x = if isPos x then 1/x else 1/(1-x)
```

Problem: Inferred type for divIf insensible!

Liquid Types

```
divIf :: x:{Int | 0 < x } -> {Int | true  }
divIf x = if isPos x then 1/x else 1/(1-x)
```

```
client = divIf 42
```

Problem: Inferred type for divIf insensible!

Worse: Inferred type is non-modular!

Liquid Types

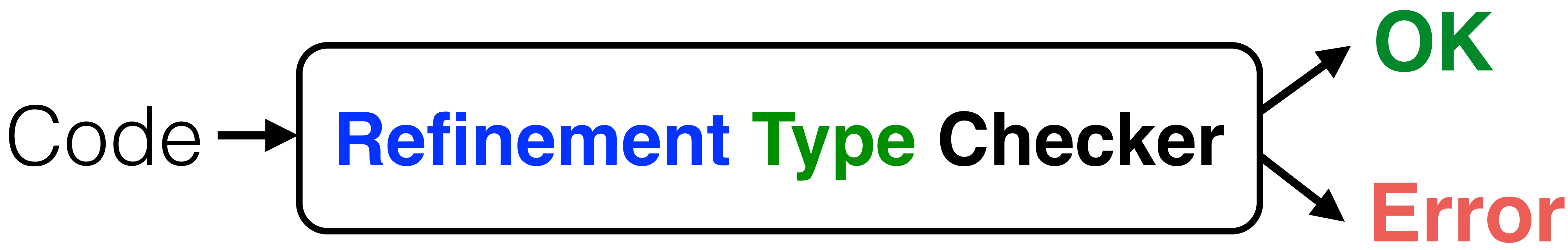
Problem: To understand errors ...
you need to know how inference works!

Liquid Types

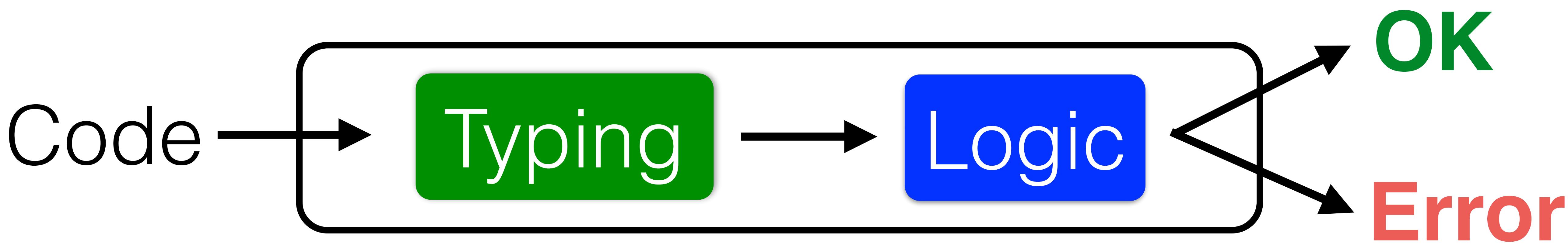
Problem: To understand errors ...
you need to know how inference works!

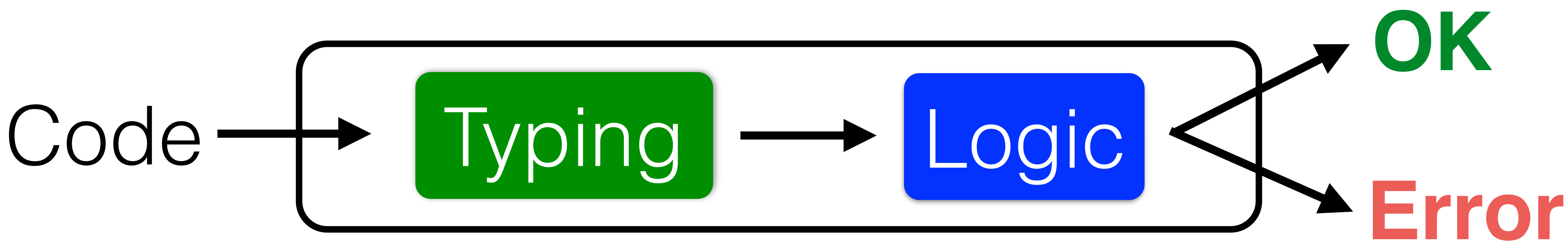
Let's start by how refinement types work!

Refinement Types



Refinement Types






```
(/)    :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```


```
(/)   :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b} |- {v|v=x}   <: {v|0<v}
x:Int,b:{b|b⇔0<x,¬b} |- {v|v=1-x} <: {v|0<v}
```


```
(/)   :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b}|-{v|v=x}  <:{v|0<v}
x:Int,b:{b|b⇔0<x,¬b}|-{v|v=1-x}<:{v|0<v}
```


```
(/)   :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b}|-{v|v=x}  <:{v|0<v}
x:Int,b:{b|b⇔0<x,¬b}|-{v|v=1-x}<:{v|0<v}
```


```
(/)    :: Int -> {v:Int | 0 < v } -> Int
isPos  :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b}|-{v|v=x}   <: {v|0<v}
x:Int,b:{b|b⇔0<x,¬b}|-{v|v=1-x} <: {v|0<v}
```


```
(/)    :: Int -> {v:Int | 0 < v } -> Int
isPos  :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b}|-{v|v=x}    <:{v|0<v}
x:Int,b:{b|b⇔0<x,¬b}|-{v|v=1-x}<:{v|0<v}
```


```
(/)    :: Int -> {v:Int | 0 < v } -> Int
isPos  :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b}|-{v|v=x}   <: {v|0<v}
x:Int,b:{b|b⇔0<x,¬b}|-{v|v=1-x} <: {v|0<v}
```


```
(/)   :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b}|-{v|v=x}   <: {v|0<v}
x:Int,b:{b|b⇔0<x,¬b}|-{v|v=1-x}<: {v|0<v}
```

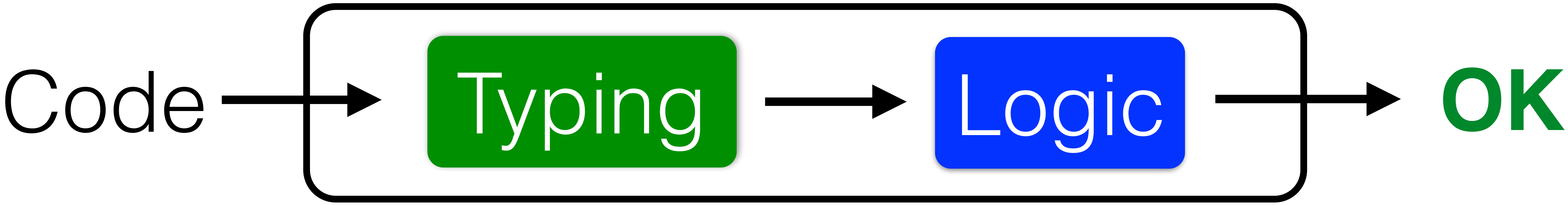

```
(/)   :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b <=> 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int,b:{b|b⇔0<x, b} |- {v|v=x}   <: {v|0<v}
x:Int,b:{b|b⇔0<x,¬b} |- {v|v=1-x} <: {v|0<v}
```


```
x:Int,b:{b|b⇔0<x, b} |- {v|v=x}   <: {v|0<v}
x:Int,b:{b|b⇔0<x,¬b} |- {v|v=1-x} <: {v|0<v}
```

```
true,     b⇔0<x, b  =>  v=x   =>  0<v
true,     b⇔0<x,¬b  =>  v=1-x =>  0<v
```




```
true,        b⇔0<x, b  =>   v=x    =>  0<v
true,        b⇔0<x,¬b  =>   v=1-x  =>  0<v
```

Code → **Refinement Type Checker** → **OK**

```
(/)    :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b ⇔ 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

Code → **Refinement Type Checker** → **OK**

```
(/)   :: Int -> {v:Int | 0 < v } -> Int
isPos :: x:Int -> {b | b ⇔ 0 < x }

divIf x = if isPos x then 1/x else 1/(1-x)
```

x:{0<x}   x:{x≤0}

What if `isPos` is not verified?

Code → **Refinement Type Checker** → **OK**

```
(/)    :: Int -> {v:Int | 0 < v } -> Int


divIf x = if isPos x then 1/x else 1/(1-x)
```

`x:{0<x}`   `x:{x≤0}`

What if `isPos` is not verified?

Code → **Refinement Type Checker** → **OK**

```
(/)   :: Int -> {v:Int | 0 < v } -> Int
divIf :: x:{Int | ? } -> Int

divIf x = if isPos x then 1/x else 1/(1-x)
```

`x:{0<x}`   `x:{x≤0}`

What if `isPos` is not verified?

Is there a type for `x` that makes `divIf` OK?

Is there a type for x that makes divIf OK?

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

x:{0<x}   x:{x≤0}

For every occurrence of x,

there exists a predicate p, so that

x:{Int | p x }

Gradual Refinement Types

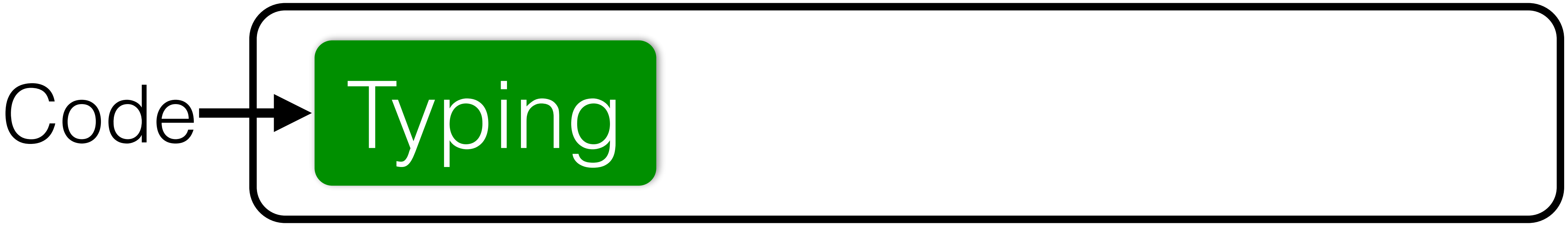
`x:{Int | ? }`

For every occurrence of `x`,

there exists a predicate `p`, so that

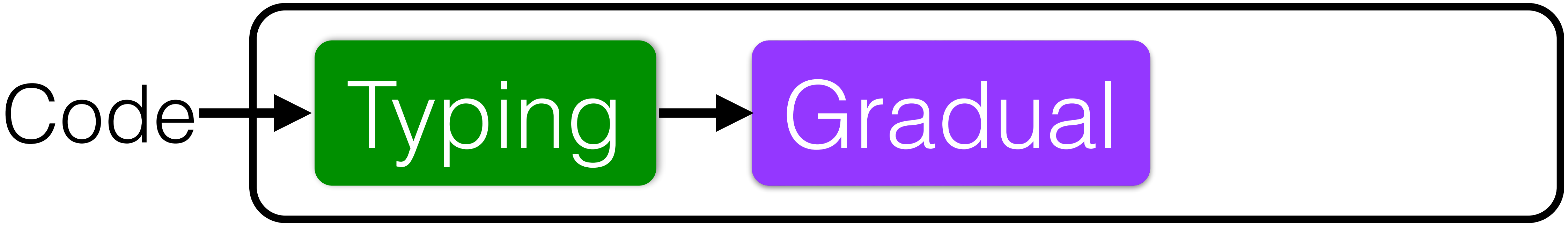
`x:{Int | p x }`


```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```



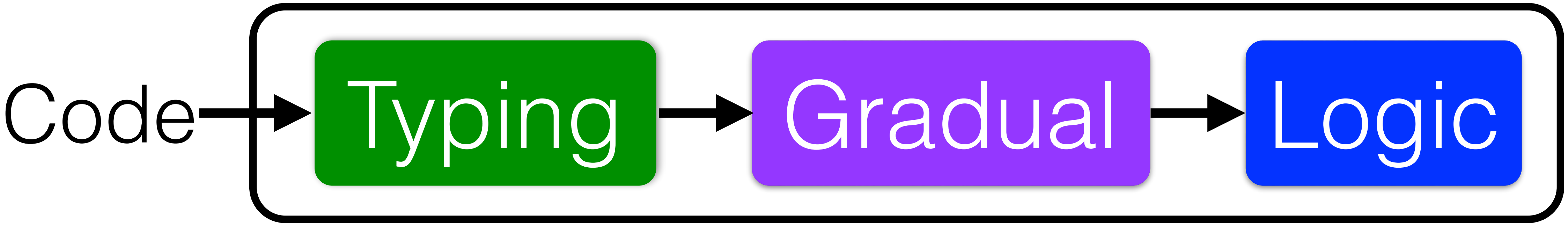

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:{x| ? },b:{b|b } |- {v|v=x}  <: {v|0<v}
x:{x| ? },b:{b|¬b} |- {v|v=1-x} <: {v|0<v}
```



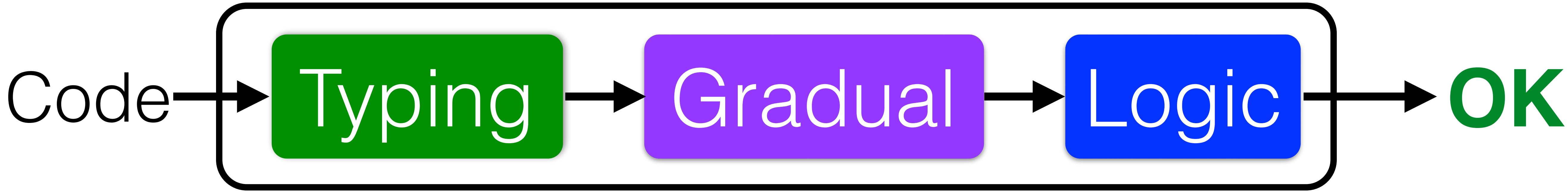

```
x:{x| ? },b:{b|b } |- {v|v=x}  <: {v|0<v}
x:{x| ? },b:{b|¬b} |- {v|v=1-x} <: {v|0<v}
```

```
∃p. p x,  b        =>   v = x    =>  0<v
∃p. p x, ¬b        =>   v = 1-x  =>  0<v
```

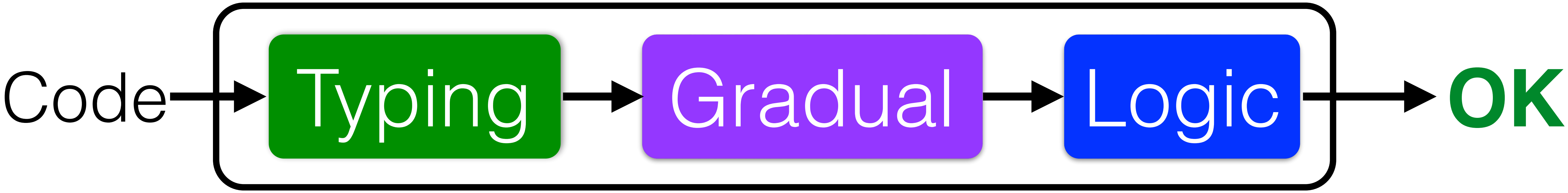



```
∃p. p x,  b         =>   v = x    =>  0<v
∃p. p x, ¬b         =>   v = 1-x  =>  0<v
```

```
0<x,  b       =>   v = x    =>  0<v
x≤0, ¬b       =>   v = 1-x  =>  0<v
```




```
0<x,  b        =>   v = x    =>  0<v
x≤0, ¬b        =>   v = 1-x  =>  0<v
```




```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

x:{0<x}   x:{x≤0}

Code → Typing → Gradual → Logic → OK

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

x:{0<x}   x:{x≤0}

How do we solve existential over predicates?

How do we solve existential over predicates?

Gradual Refinement Types

In Theory *

How do we solve existential over predicates?

In Practise

* Lehmann & Tanter [POPL’16]

How do we solve existential over predicates?

Problem: Domain of predicates is infinite

$$p \in \{\ 0<x,\ x\leq 0,\ b \Leftrightarrow 0<x,\ b,\ \neg b,\ \ldots\ \}$$

How do we solve existential over predicates?

Problem: Domain of predicates is infinite

Solution: Predicates over finite domain

$$p \in \{\ 0<x,\ x\leq 0,\ b \Leftrightarrow 0<x,\ b,\ \neg b\ \}$$

Predicates over finite domain

Liquid Types

Application: Type Inference

Liquid Types

```
isPos :: x:Int-> {b:Bool | p b }
isPos x = 0 < x

divIf x = if isPos x then 1/x else 1/(1-x)
```

Solution for p so that divIf is OK?

Liquid Types

```
isPos :: x:Int-> {b:Bool | p b }
isPos x = 0 < x

divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:Int                    |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,  b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

Liquid Types

```
x:Int                      |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,  b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

```
p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }
```

Liquid Types

```
x:Int                       |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,  b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

```
p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }
```

Liquid Types

```
x:Int                    |- {v|v=0<x}   <: {v|p v}
x:Int,b:{b|p b,  b} |- {v|v =x }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

```
p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }
```

Liquid Types

```
x:Int                  |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

$p\ v \in \{\boxed{0<v,}\ v \le 0,\ v \Leftrightarrow 0<x,\ v,\ \neg v\}$

Liquid Types

```
x:Int                   |- {v|v=0<x}   <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

$p\ v \in \{\ 0<v,\ \boxed{v \leq 0,}\ v \Leftrightarrow 0<x,\ v,\ \neg v\ \}$

Liquid Types

```
x:Int                   |- {v|v=0<x}   <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x  }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

p v ∈ { 0<v, $\boxed{\text{v≤0,}}$ v⇔0<x, v, ¬v }

Liquid Types

```
x:Int                    |- {v|v=0<x}   <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }

Liquid Types

```
x:Int                      |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

$p\ v \in \{ 0<v,\ v\leq 0,\ v \Leftrightarrow 0<x,\ \boxed{v,}\ \neg v \}$

Liquid Types

```
x:Int                  |- {v|v=0<x}   <: {v|p v}
x:Int,b:{b|p b,  b} |- {v|v =x }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

$$p\ v \in \{\ 0<v,\ v\le 0,\ v \Leftrightarrow 0<x,\ \boxed{v,}\ \neg v\ \}$$

Liquid Types

```
x:Int                        |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }

Liquid Types

```
x:Int                    |- {v|v=0<x}   <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x  }   <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

?

```
p v ∈ {0<v, v≤0, v⇔0<x, v, ¬v}
```

Liquid Types

```
x:Int                   |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

```
p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }
```

Liquid Types

```
x:Int                      |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

```
p v ∊ { 0<v, v≤0, v⇔0<x, v,  ¬v }
```

Liquid Types

```
x:Int                   |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,   b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

p v ∈ { 0<v, v≤0, v⇔0<x, v, ¬v }

Liquid Types

```
x:Int                      |- {v|v=0<x}  <: {v|p v}
x:Int,b:{b|p b,  b} |- {v|v =x }  <: {v|0<v}
x:Int,b:{b|p b, ¬b} |- {v|v =1-x} <: {v|0<v}
```

```
p v =  0<v, v≤0, v⇔0<x  v, ¬v
```

Liquid Types

```
isPos :: x:Int-> {b:Bool | p b }
isPos x = 0 < x

divIf x = if isPos x then 1/x else 1/(1-x)
```

$$p\ v = v \Leftrightarrow 0 < x$$

Liquid Types

```
isPos :: x:Int-> {b:Bool | p b }
isPos x = 0 < x

divIf x = if isPos x then 1/x else 1/(1-x)
```

$$p\ v = v \Leftrightarrow 0 < x$$

there exists a predicate p, so that

for every occurrence of b,

`b:{Bool | p b }`

Liquid Types

```
isPos :: x:Int-> {b:Bool | p b }
isPos x = 0 < x

divIf x = if isPos x then 1/x else 1/(1-x)
```

there exists a predicate p, so that

for every occurrence of b,

```
b:{Bool | p b }
```

Gradual Liquid Types

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

there exists a predicate p, so that

for every occurrence of x,

```
x:{Int | p x }
```

Gradual Liquid Types

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

for every occurrence of x,

there exists a predicate p, so that

```
x:{Int | p x }
```

Gradual Liquid Types

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

```
x:{x| ? },b:{b|b } |- {v|v=x}  <: {v|0<v}
x:{x| ? },b:{b|¬b} |- {v|v=1-x} <: {v|0<v}
```

Gradual Liquid Types

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

$$x{:}\{x \mid p_1\}, b{:}\{b \mid b\} \vdash \{v \mid v = x\} <: \{v \mid 0 < v\}$$
$$x{:}\{x \mid p_2\}, b{:}\{b \mid \neg b\} \vdash \{v \mid v = 1 - x\} <: \{v \mid 0 < v\}$$

Gradual Liquid Types

```
x:{x| p1 },b:{b|b } |-{v|v=x}  <:{v|0<v}
x:{x| p2 },b:{b|¬b} |-{v|v=1-x}<:{v|0<v}
```

$p_1\ v \in \{0<v,\ v\leq 0,\ v\Leftrightarrow 0<x,\ v,\ \neg v\}$

$p_2\ v \in \{0<v,\ v\leq 0,\ v\Leftrightarrow 0<x,\ v,\ \neg v\}$

Gradual Liquid Types

$$x:\{x \mid p_1\}, b:\{b \mid b\} \vdash \{v \mid v = x\} <: \{v \mid 0 < v\}$$

$$x:\{x \mid p_2\}, b:\{b \mid \neg b\} \vdash \{v \mid v = 1 - x\} <: \{v \mid 0 < v\}$$

$$p_1\ v \in \{0<v,\ v\leq 0,\ v \Leftrightarrow 0<x,\ v,\ \neg v\}$$

$$p_2\ v \in \{0<v,\ v\leq 0,\ v \Leftrightarrow 0<x,\ v,\ \neg v\}$$

Gradual Liquid Types

```
divIf :: x:{Int | ? } -> Int
divIf x = if isPos x then 1/x else 1/(1-x)
```

x:{0<x}   x:{x≤0}

For every occurrence of x,
there exists a predicate p, so that
x:{Int | p x }

Gradual Liquid Types

In Theory

How do we solve existential over predicates?

Exhaustive search over finite domain.

In Practise

Gradual Liquid Types

Exhaustive search over finite domain.

In Practise

Advantage: Type Inference

Gradual Liquid Types

Type Inference

```
x1:{?},…,xn:{0<xn} |- {v|true} <: {v|r1}
y1:{r1},…,yn:{ ? } |- {v| ?  } <: {v|v<0}
```

For every gradual solution

If there exists a static solution

then return OK

return Error

Gradual Liquid Types

Exhaustive search over finite domain.

In Practise

Advantage: Type Inference

Gradual Liquid Types

Exhaustive search over finite domain.

In Practise

Advantage: Type Inference

Disadvantage: A lot of extra work

Gradual Liquid Types

How do we solve existential over predicates?
Exhaustive search over finite domain.

Disadvantage: A lot of extra work
Side-Effect: Certificate Generation
Application: Error Reporting

Application: Error Reporting

```
(!!) :: xs:[a]-> {Int| ? } -> a

(x:xs) !! 0 = x
(x:xs) !! i = xs !! (i-1)
_      !! _ = error "Out of bounds!"
```

Application: Error Reporting

```
(!!) :: xs:[a]-> {Int| ? } -> a

(x:xs) !! 0 = x
(x:xs) !! i = xs !! (i-1)
_      !! _ = error “Out of bounds!”
```

Q: Give me all the certificates

A: Demo

Gradual Liquid Types

Application: Error Reporting

Gradual Liquid Types

In Theory

How do we solve existential over predicates?

Exhaustive search over finite domain.

In Practise

Advantage: Type Inference

Application: Error Reporting

Thanks!