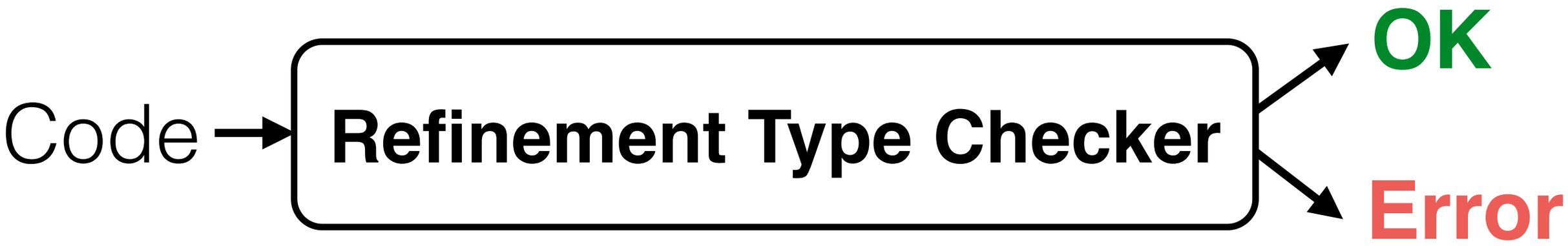
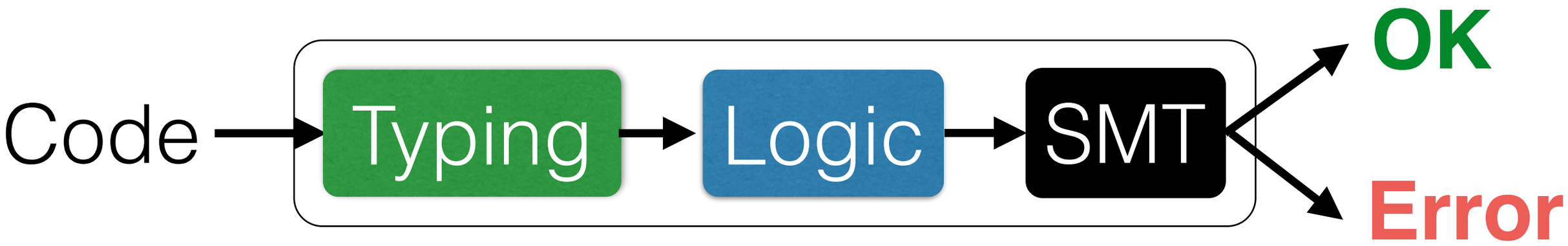
Liquid Haskell 101

Niki Vazou
University of Maryland

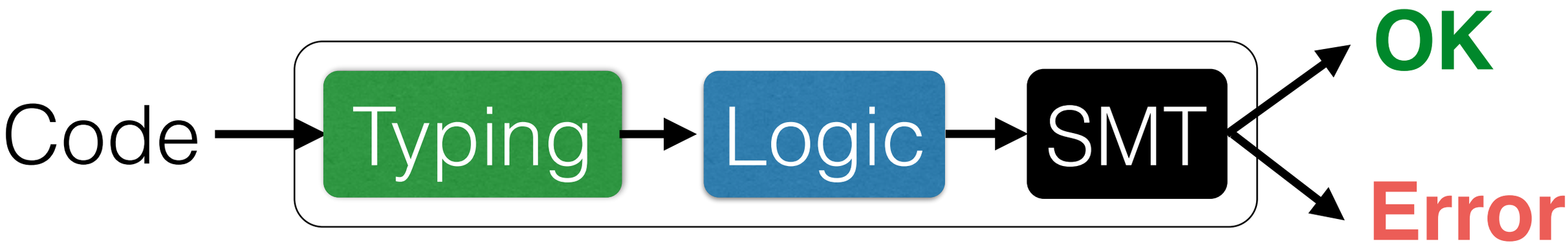
Liquid Haskell 101



Liquid Haskell 101



1. Source Code to **Type constraints**
2. **Type Constraints** to **Verification Condition (VC)**
3. Check **VC validity** with **SMT Solver**




```
take :: t:Text -> {v | v <= len t} -> Text

heartbleed = let x = "Niki"
             in  take x 8
```

Code → Typing

```
take :: t:Text->{v|v <= len t}->Text
heartbleed = let x = "Niki"
             in  take x 8
```

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

Code → Typing

```
take :: t:Text->{v|v <= len t}->Text
heartbleed = let x = "Niki"
             in  take x 8
```

x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}

Code → Typing

```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "Niki"
             in take x 8
```

x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}

Code → Typing

```
take :: t:Text -> {v | v <= len t} -> Text

heartbleed = let x = "Niki"
             in  take x 8
```

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

Code → Typing

```
take :: t:Text->{v | v <= len t}->Text
heartbleed = let x = "Niki"
             in  take x 8
```

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

Encode Subtyping as Logical VC

If VC valid then Subtyping holds

Encode Subtyping as Logical VC

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

`x:{v|p}`

`{v|p} <: {v|q}`

x : {v | p}

Means*: *If* x reduces to a value *then* p[x/v]

Encoded as: “x has a value” => p[x/v]

* Flanagan “Hybrid Type Checking” (POPL ’06)

$$\{v \mid p\} <: \{v \mid q\}$$

Means: *if* $y : \{v \mid p\}$ *then* $y : \{v \mid q\}$

Encoded as: $p \Rightarrow q$

Encode Subtyping …

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

… as Logical VC

```
("x has a value" => len x = 4)
                 => (v = 8) => (v <= len x)
```

Encode Subtyping …

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

… as Logical VC

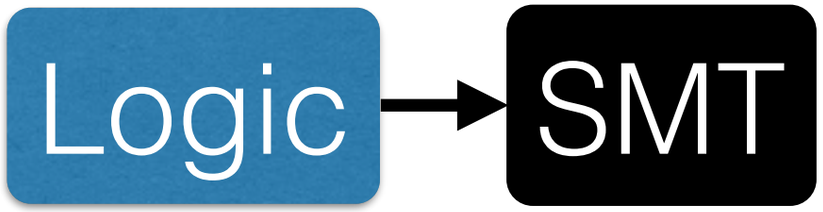
```
("x has a value" => len x = 4)
                 => (v = 8) => (v <= len x)
```

Encode Subtyping …

```
x:{v|len v=4} |- {v|v=8} <: {v|v<=len x}
```

… as Logical VC

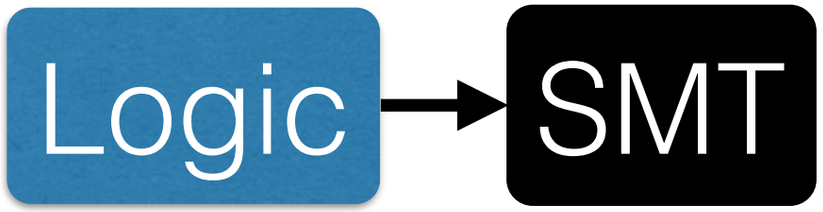
```
("x has a value" => len x = 4)
                 => (v = 8) => (v <= len x)
```




```
("x has a value" => len x = 4)
                 => (v = 8) => (v <= len x)
```

How to encode "x has a value" ?

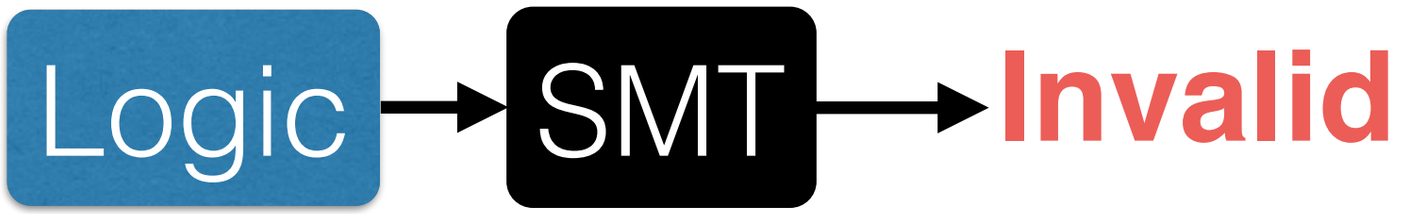
(In a decidable manner)




```
("x has a value" => len x = 4)
                 => (v = 8) => (v <= len x)
```

CBV: Binders Are Values!

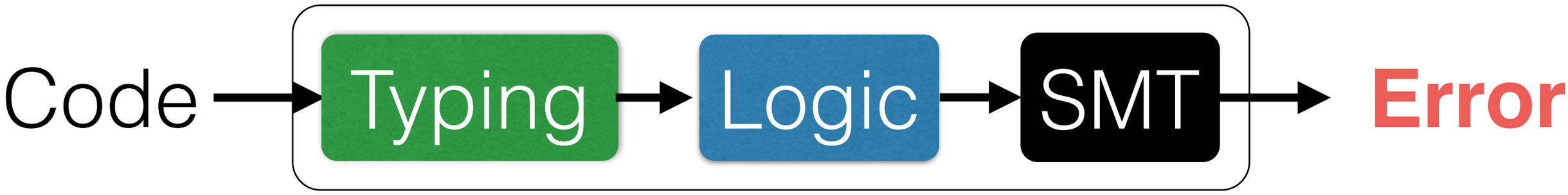
i.e. x is guaranteed to be a value




```
len x = 4
=> (v = 8) => (v <= len x)
```

CBV: Binders Are Values!

i.e. x is guaranteed to be a value




```
take :: t:Text -> {v | v <= len t} -> Text

heartbleed = let x = "Niki"
             in  take x 8
```

CBV: Checker soundly reports **Error**

CBV: Binders must be values

Ignoring “has a value” is sound!

CBN: Binders may not be values

Ignoring “has a value” is unsound!

Ignoring “has a value” is unsound!

```
spin :: Int -> Int
spin x = spin x
```

Ignoring "has a value" is unsound!

```
spin :: Int -> {v:Int|false}
spin x = spin x
```

OK

As `spin` does not return any value

Ignoring "has a value" is unsound!

```
take :: t:Text->{v | v <= len t}->Text
spin :: Int -> {v:Int | false}

heartbleed = let x = "Niki"
             y = spin 0
         in  take x 8
```

OK? or **Error?**

Ignoring "has a value" is unsound!

```
take :: t:Text->{v | v <= len t}->Text
spin :: Int -> {v:Int | false}

heartbleed = let x = "Niki"
                 y = spin 0
             in  take x 8
```

OK under **CBV** evaluation

Ignoring "has a value" is unsound!

```
take :: t:Text->{v | v <= len t}->Text
spin :: Int -> {v:Int | false}

heartbleed = let x = "Niki"
             y = spin 0
         in  take x 8
```

Error under **CBN** evaluation

Ignoring “has a value” is unsound!

CBV-style typing is unsound under CBN!

Reports **Erroneous** code as **OK**

Ignoring “has a value” is unsound!

How to encode “has a value” ?

How to encode “has a value” ?

Most expressions provably reduce to a value

How to encode “has a value” ?

Most expressions provably reduce to a value

If x reduces to a value,
Then encode “x has a value” by `true`

Solution: Stratified Types

$$x{:}\{v{:}\mathtt{Int} \mid p\}$$

Must reduce to a Value

$$x{:}\{v{:}\mathtt{Int}^{\uparrow} \mid p\}$$

May-not reduce to a Value

Stratified Types to Logic

`x:{v:Int |p}` encoded as `p[x/v]`

`x:{v:Int`$^{\uparrow}$`|p}` encoded as `true`

Code → Typing

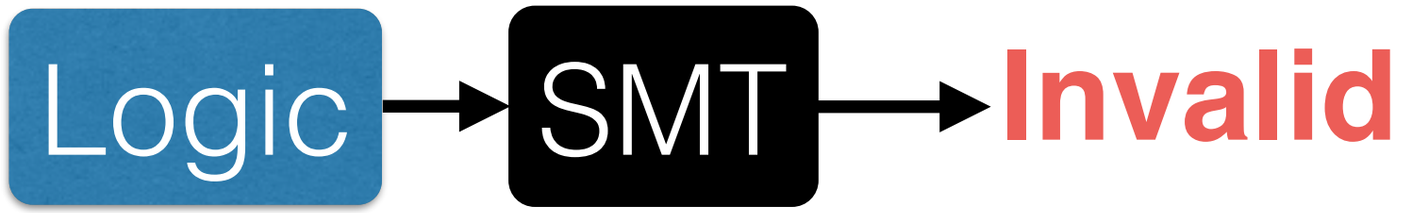
```
take :: t:Text -> {v | v <= len t} -> Text
spin :: Int -> {v:Int↑ | false}

heartbleed = let x = "Niki"
                 y = spin 0
             in  take x 8
```

```
x:{v|len v=4}
y:{v:Int↑|false} |- {v|v=8} <: {v|v<=len x}
```

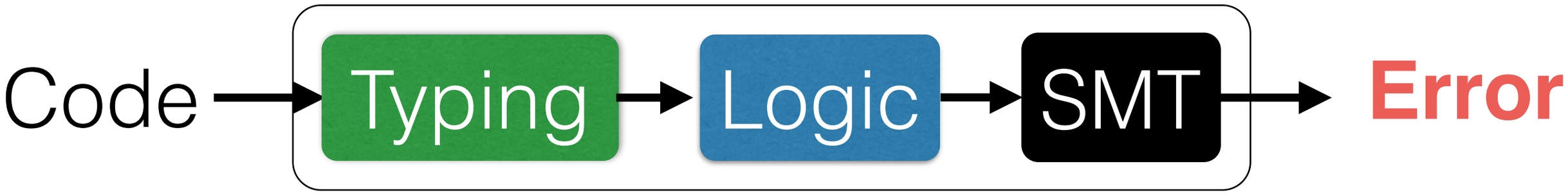
Typing → Logic

```
x:{v|len v=4}
y:{v:Int↑|false}|-{v|v=8} <: {v|v<=len x}
```

```
len x=4
true                    => v=8 => v<=len x
```




```
len x = 4
true                    => v = 8 => v <= len x
```




```
take :: t:Text -> {v | v <= len t} -> Text
spin :: Int -> {v:Int↑ | false}

heartbleed = let x = pack "Niki"
                 y = spin 0
             in  take x 8
```

How to enforce stratification?

How to enforce stratification?

`x:{v:Int|p}`

Must have a Value

Terminating expressions must have a value

Solution

Check termination with Refinement Types!

Soundness Under Lazy Evaluation

Liquid Haskell101

1. Soundness

Under Lazy Evaluation

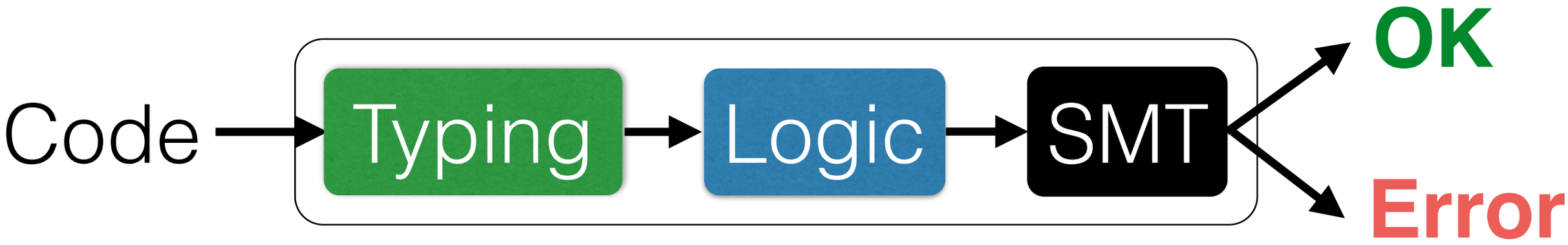
2. Expressiveness

Refinement Reflection

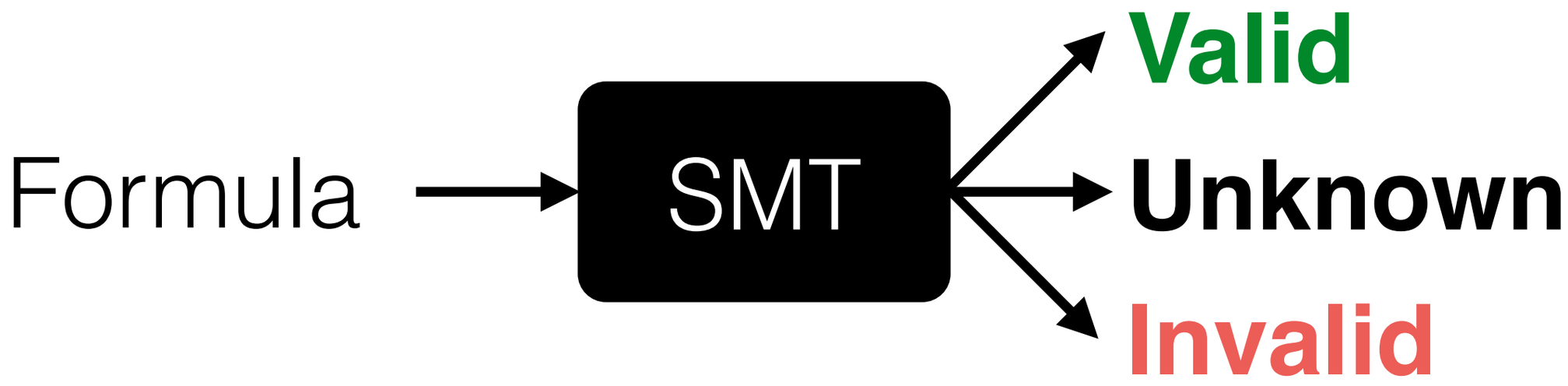
With decidable & predictable verification.

With decidable & predictable verification.

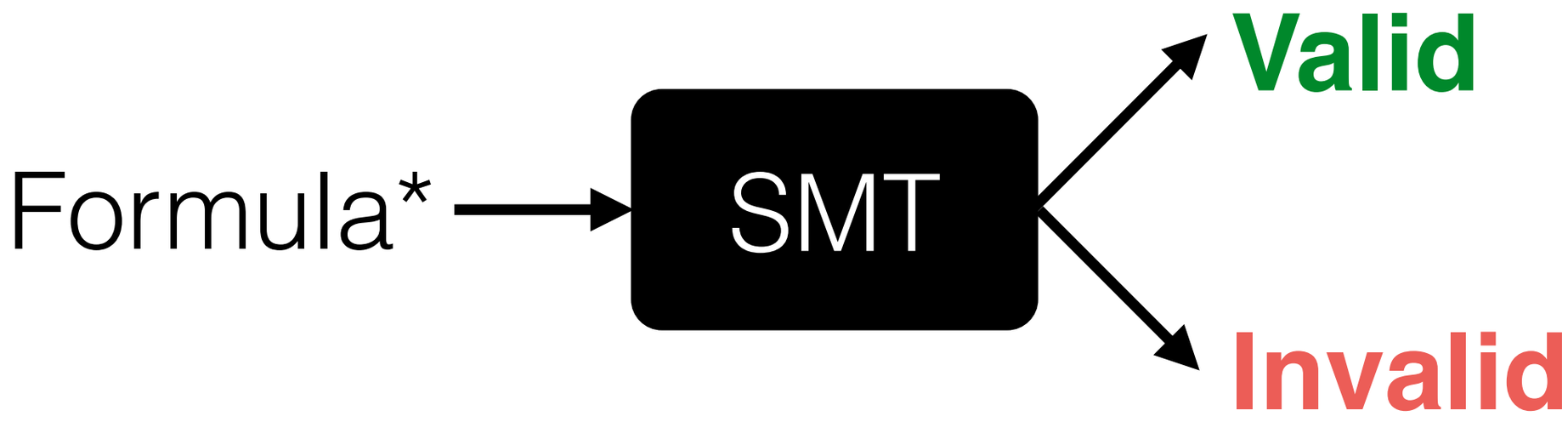
With decidable & predictable verification.



With decidable & predictable verification.



With decidable & predictable verification.



*From Decidable Logic

Encode Subtyping …

$x_1{:}\{v \mid p_1\}, \ldots, x_n{:}\{v \mid p_n\} \vdash \{v \mid q_1\} <: \{v \mid q_2\}$

… as Logical Verification Condition

$p_1 \wedge \ldots \wedge p_n \Rightarrow q_1 \Rightarrow q_{12}$

For decidability, p from decidable theories

$$\{ v : a \mid p \}$$

… as Logical Verification Condition

$$p_1 \wedge \ldots \wedge p_n \Rightarrow q_1 \Rightarrow q_{12}$$

For decidability, p from decidable theories

$$\{ v : a \mid p \}$$

Boolean Logic

(QF) Linear Arithmetic

Uninterpreted Functions ...

For decidable & predictable verification.

Liquid Haskell101

1. Soundness

Under Lazy Evaluation

2. Expressiveness

Refinement Reflection

With decidable & predictable verification.