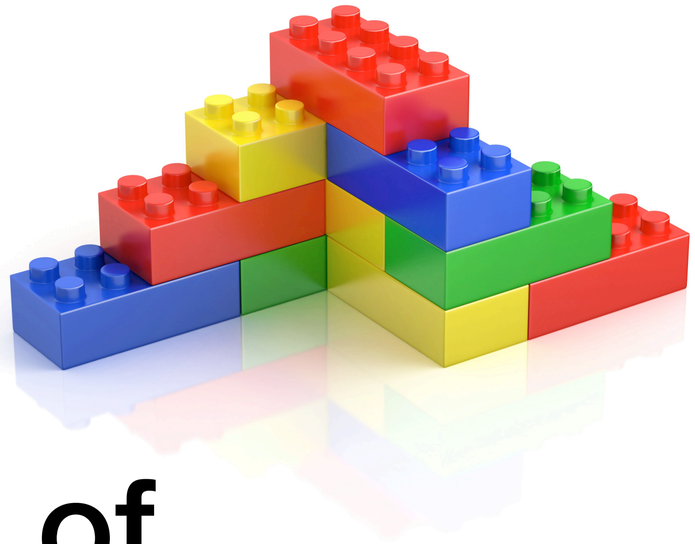
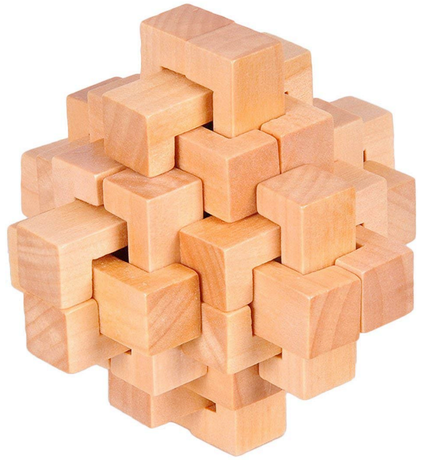


Compositional Soundness Proofs of Abstract Interpreters



Sven Keidel, Casper B. Poulsen, Sebastian Erdweg

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

JG|U

TU Delft

Abstract Interpretation
= Sound Static Analysis

Factorial

```
function fact(n)
  if n = 0
    return 1
  else
    return n * fact(n-1)
```

Interval Analysis

$$fact([1..\infty]) \subseteq [1..\infty]$$

$$\uparrow \notin$$

$$fact(1000) = -46035$$

Type Checking

```
String[] str = new String[5];
Object[] obj = str;
obj[0] = new Integer(42);
print(str[0]);
```

Instability

Exception in thread "main"
java.lang.ArrayStoreException:
java.lang.Integer

Compiler Optimisations

```
while(x <= 10) {
  y = foo(5);
  bar(y);
  x = x + 1;
}
```

⟹

```
y = foo(5);
while(x <= 10) {
  bar(y);
  x = x + 1;
}
```

Unexpected Behavior

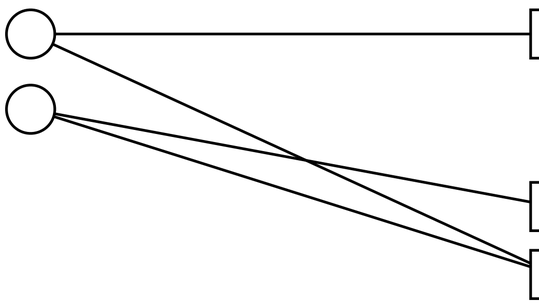
```
data Expr = IfZero Expr Expr Expr | …
```

Concrete

```
eval e = case e of
  IfZero e₁ e₂ e₃ → do
    v ← eval e₁
    if v == 0
      then eval e₂
      else eval e₃
```

Abstract

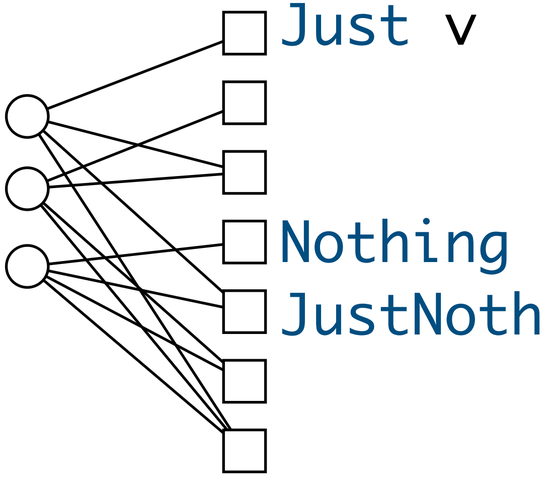
```
eval e = case e of
  IfZero e₁ e₂ e₃ → do
    v ← eval e₁
    if v == (0,0)
      then eval e₂
      else if v ⋡ (0,0)
        then eval e₃
        else eval e₂ ⊔ eval e₃
```



```
data Expr = TryZero Expr Expr Expr | …
```

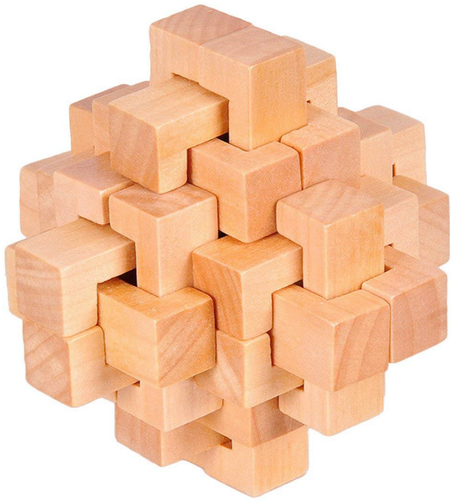
Concrete

```
eval e = case e of
 TryZero e₁ e₂ e₃ →
  case eval e₁ of
   Just 0   → eval e₂
   Just _   → eval e₃
   Nothing  → eval e₃
```



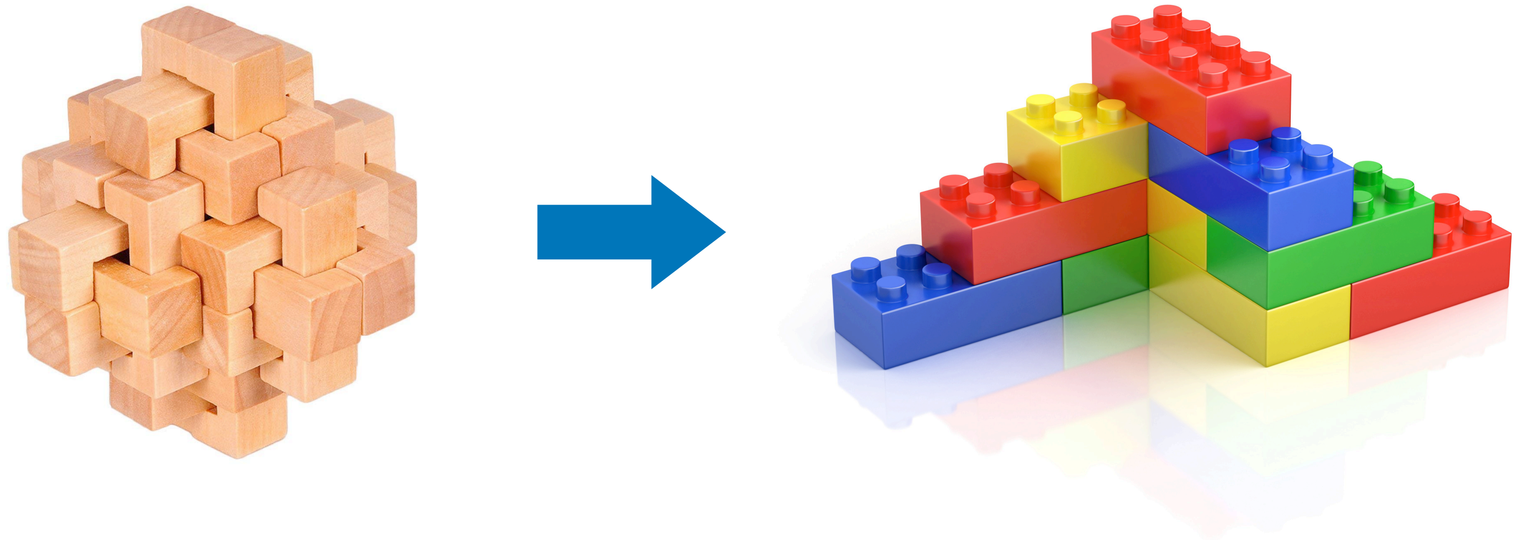
Abstract

```
eval e = case e of
 TryZero e₁ e₂ e₃ → case eval e₁ of
   Just v                | v == (0,0) → eval e₂
                         | v ⋡ (0,0) → eval e₃
                         | v ⊃ (0,0) → eval e₂ ⊔ eval e₃
   Nothing                           → eval e₃
   JustNothing v         | v == (0,0) → eval e₂ ⊔ eval e₃
                         | v ⋡ (0,0) → eval e₃
                         | v ⊃ (0,0) → eval e₂ ⊔ eval e₃
```

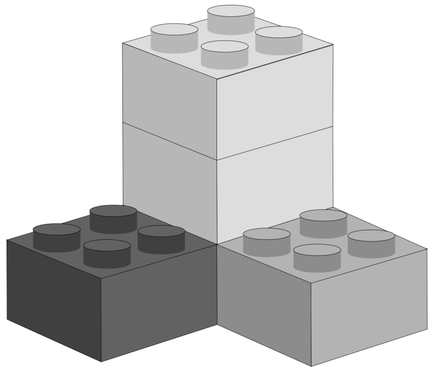


Hard to <u>decompose</u> into smaller lemmas

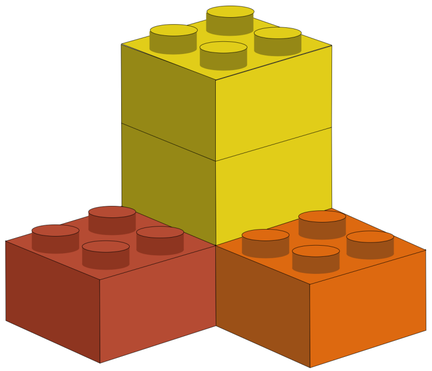
Compositional Soundness Proofs of Abstract Interpreters



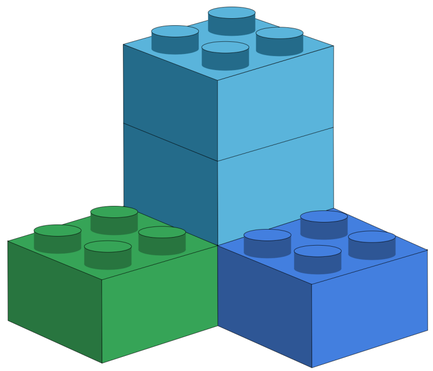
Shared

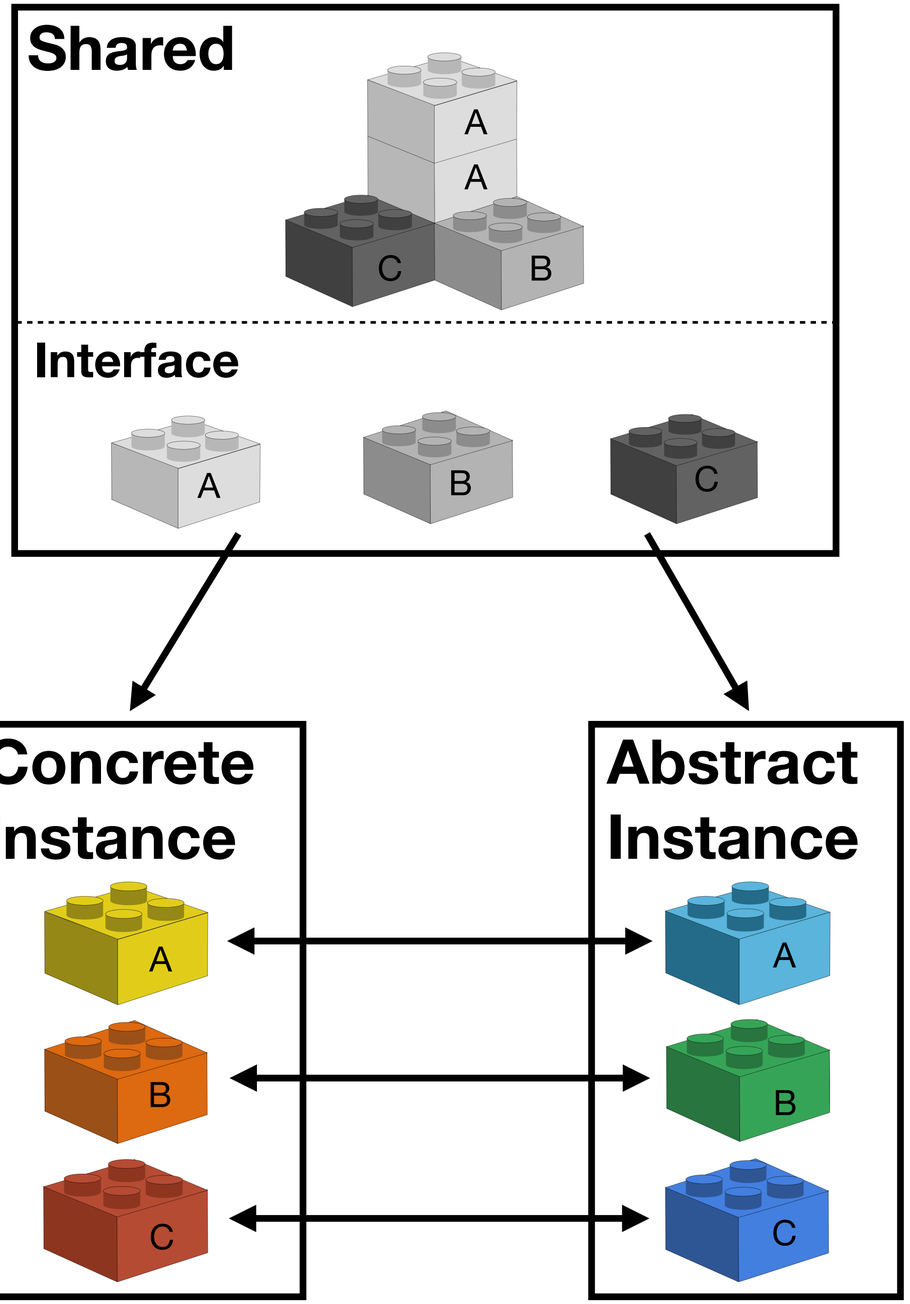


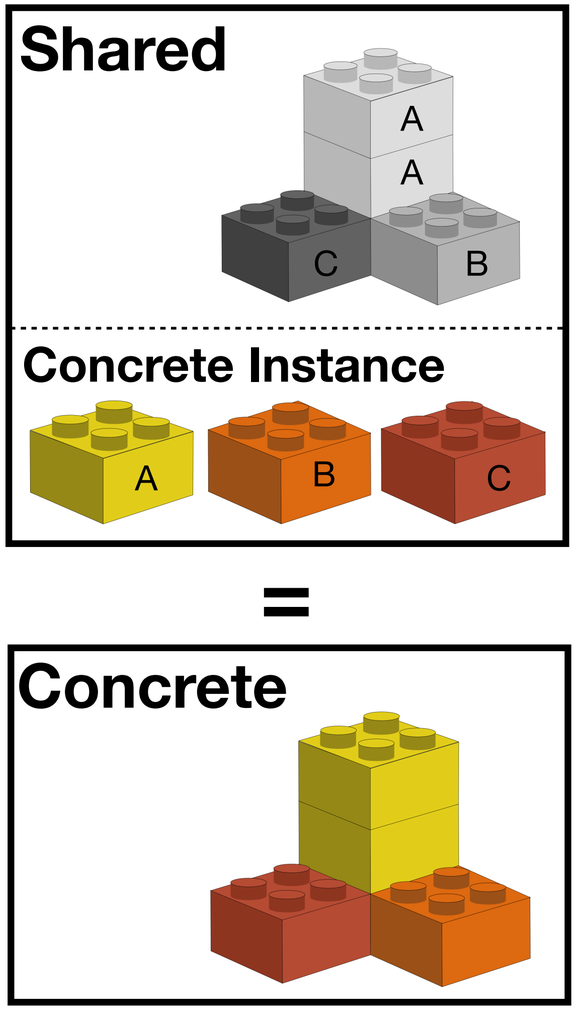
Concrete

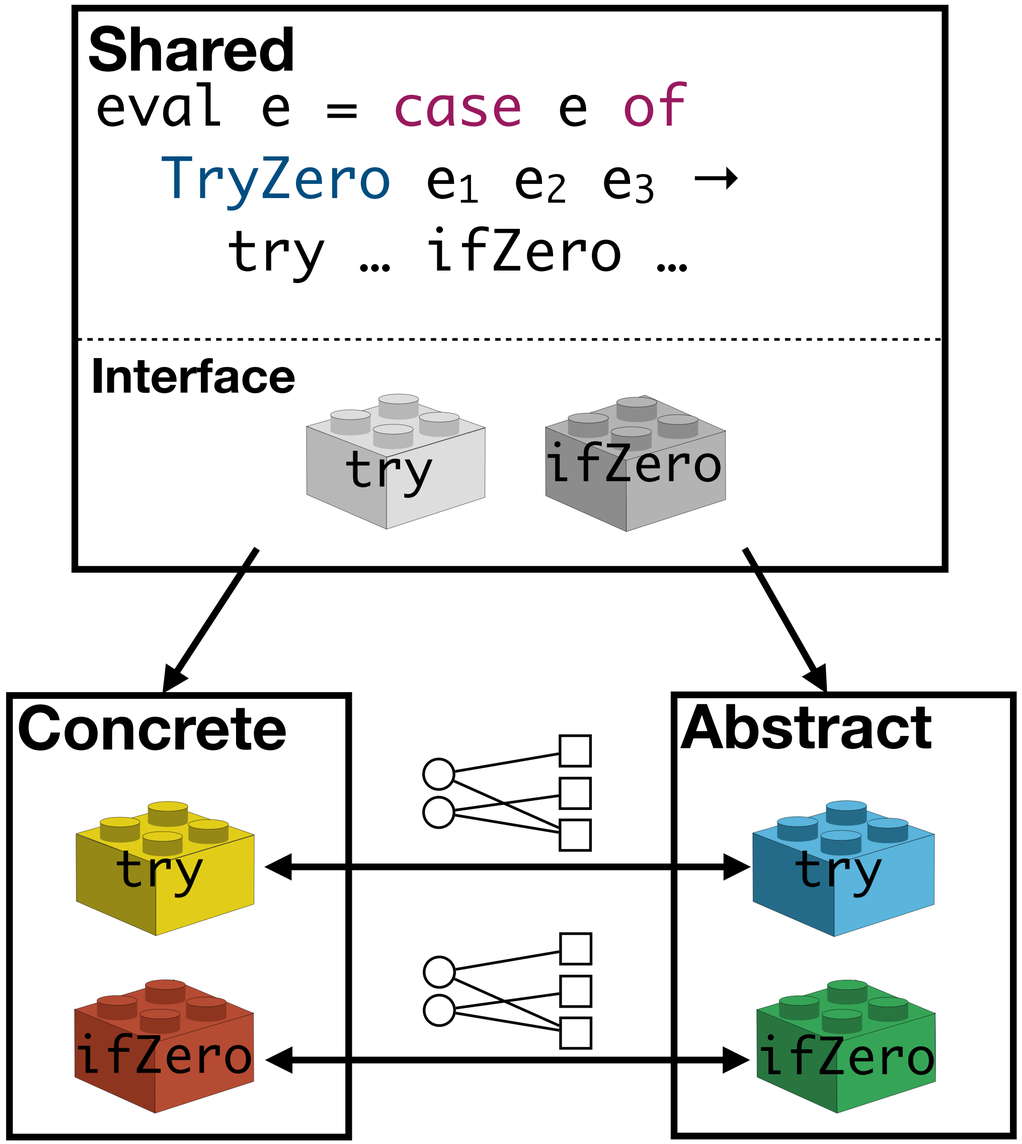


Abstract

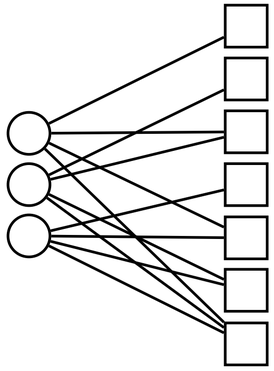


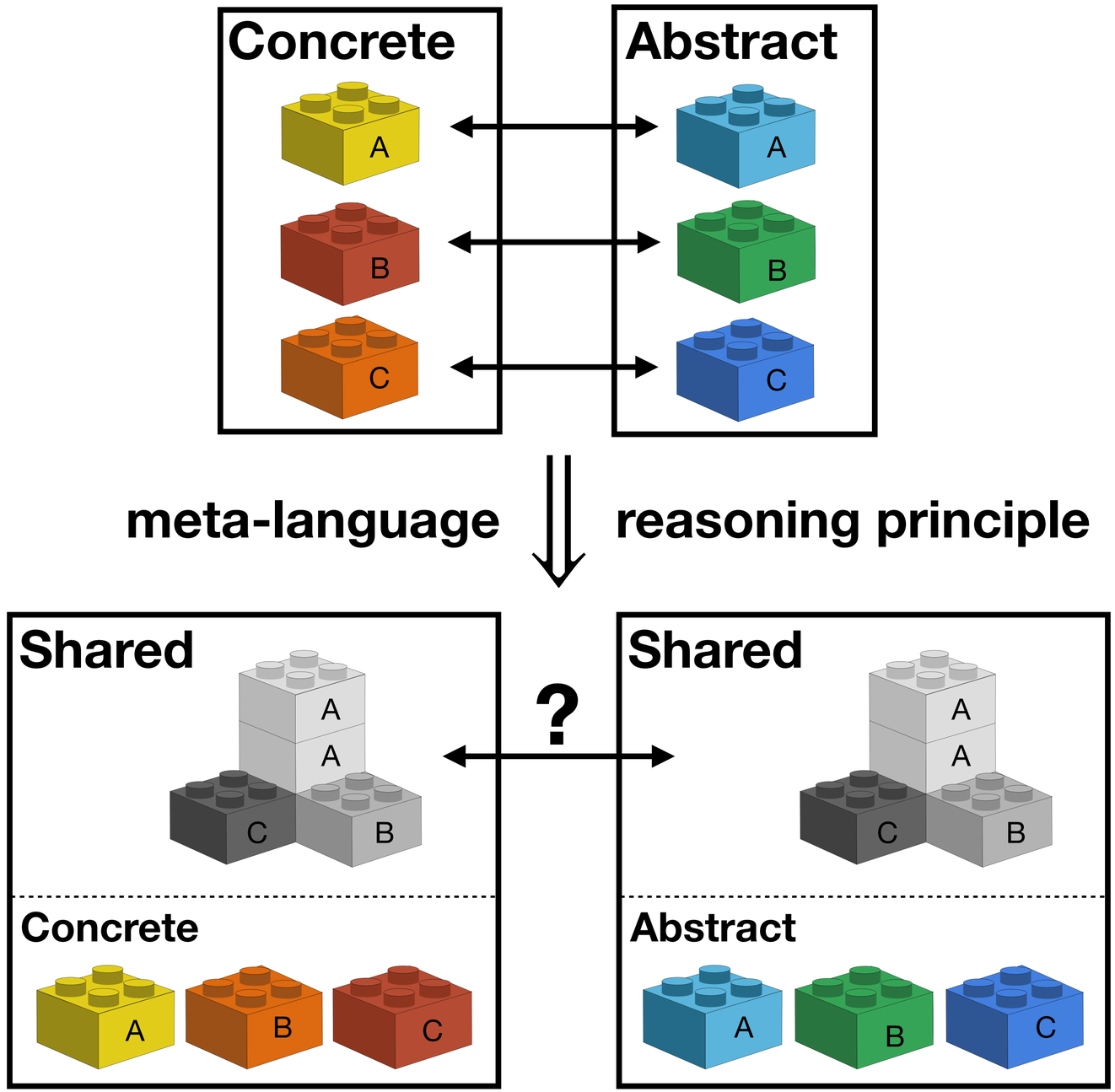

Shared
A
A
C
B
Interface
A
B
C
Concrete Instance
A
B
C
Abstract Instance
A
B
C


Shared
A
A
C
B
Concrete Instance
A
B
C
=
Concrete



Non-Compositional




Concrete
A
B
C
Abstract
A
B
C
meta-language
reasoning principle
Shared
A
A
C
B
Concrete
A
B
C
?
Shared
A
A
C
B
Abstract
A
B
C

Arrows [Hughes 2000]

```
eval :: Env → Expr → Maybe Val

type Interp a b = Env → a → Maybe b

eval :: Interp Expr Val
eval = proc e → case e of
  Add e₁ e₂ → do
    v₁ ← eval ⤙ e₁
    v₂ ← eval ⤙ e₂
    plus ⤙ (v₁,v₂)
```

```
class Category c where
  id  :: c x x
  (.) :: c y z → c x y → c x z
```

```
f . g ⟸ y ← f ⇁ x
        z ← g ⇁ y
```

```
class Category c => Arrow c where
  arr    :: (x → y) → c x y
  first  :: c x y → c (x,z) (y,z)
  second :: c x y → c (z,x) (z,y)
  (***)  :: c x y → c u v → c (x,u) (y,v)
  (&&&)  :: c x y → c x z → c x (y,z)
```

```
f . arr (+ 1) ⟸ f ⇁ x + 1
```

```
f *** g ⟸ y ← f ⇁ x
          v ← g ⇁ u
```

```
class Arrow c => ArrowChoice c where
  left  :: c x y → c (Either x z) (Either y z)
  right :: c x y → c (Either z x) (Either z y)
  (+++) :: c x y → c u v → c (Either x u) (Either y v)
  (|||) :: c x y → c x z → c x (Either y z)
```

```
f ||| g ⟸ case e of
           Foo → f ⇁ x
           Bar → g ⇁ y
```

Pretty notation in GHC

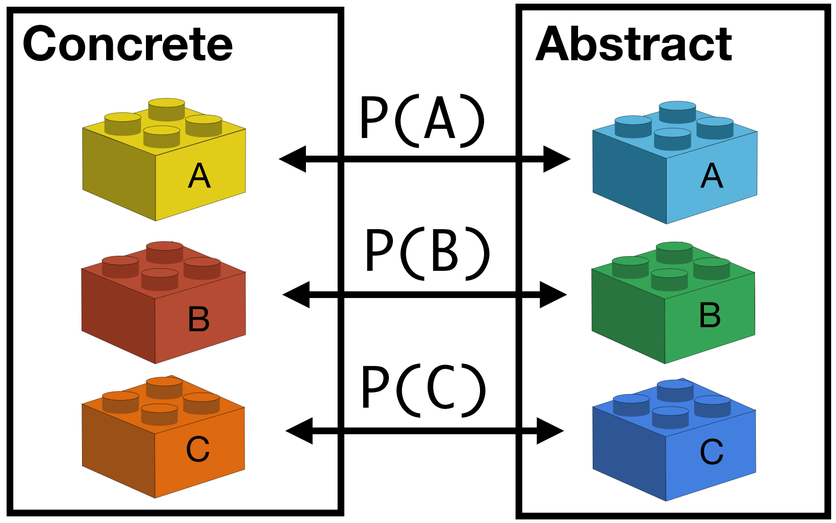
```
eval = proc e → case e of
  Add e₁ e₂ → do
    v₁ ← eval ⤙ e₁
    v₂ ← eval ⤙ e₂
    plus ⤙ (v₁,v₂)
```

First-Order Language

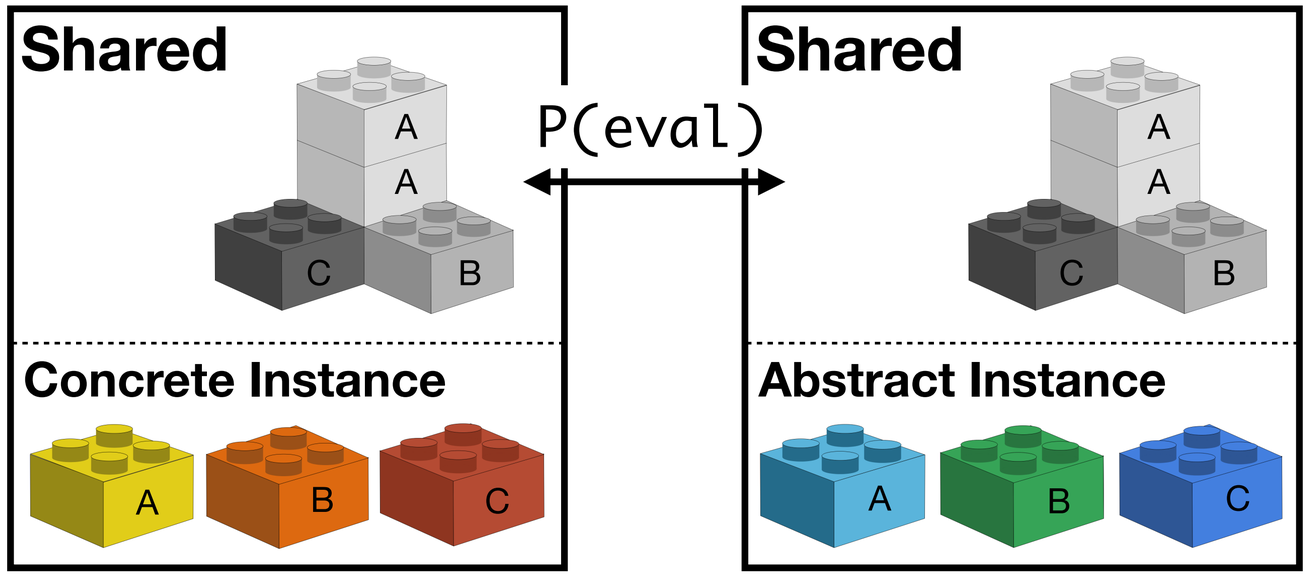
```
E ::= plus
    | E >>> E
    | E *** E
    | E +++ E
    | arr F
```

Induction Principle

$$\frac{P(\mathit{plus}) \quad P(f) \wedge P(g) \Rightarrow P(f \gg\!> g) \quad P(f) \wedge P(g) \Rightarrow P(f \mathbin{***} g) \quad P(f) \wedge P(g) \Rightarrow P(f +\!+\!+ g) \quad \forall f \in F.\ P(\mathit{arr}\ f)}{\forall e \in E.\ P(e)}$$



$$\frac{P(A) \quad P(B) \quad P(C)}{P(f) \wedge P(g) \Rightarrow P(f \ggg g)}$$
$$P(f) \wedge P(g) \Rightarrow P(f \mathbin{***} g)$$
$$P(f) \wedge P(g) \Rightarrow P(f \mathbin{+++} g)$$
$$\frac{\forall f \in F.\ P(arr\ f)}{\forall e \in E.\ P(e)}$$



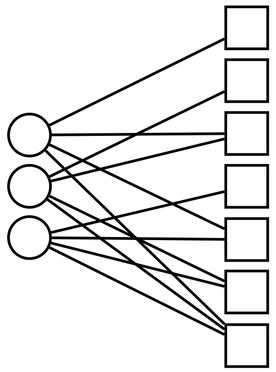
P = sound

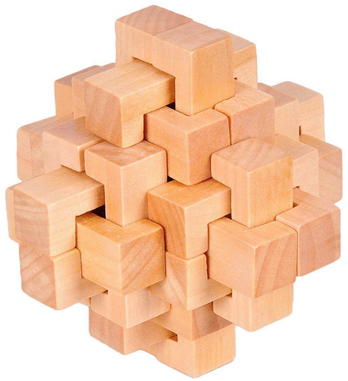
Case Studies

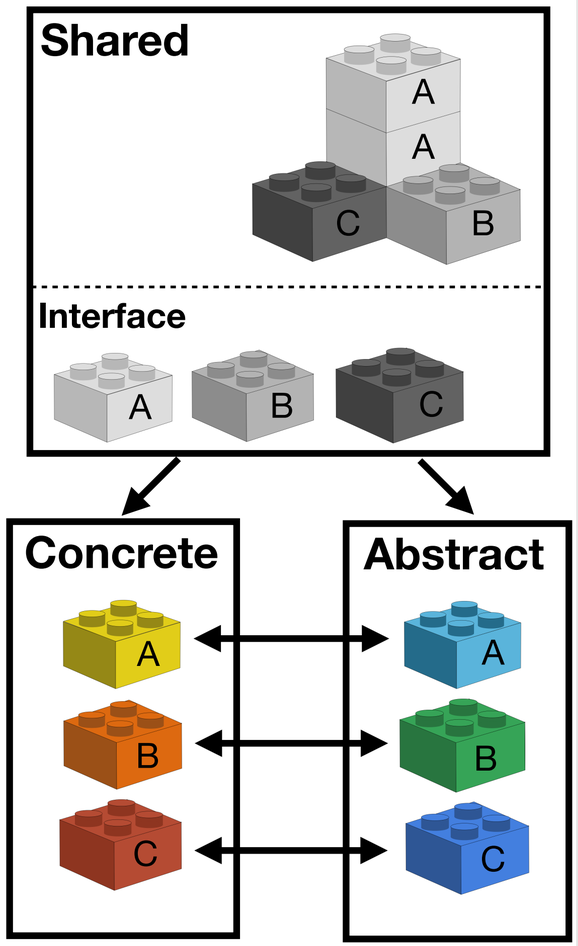
- k-CFA Analysis for PCF
- Interval Analysis for a WHILE language
- Tree Shape Analysis and Type Checker for Stratego
- Type Checker for JavaScript
- WIP: Analyses for Java and WebAssembly

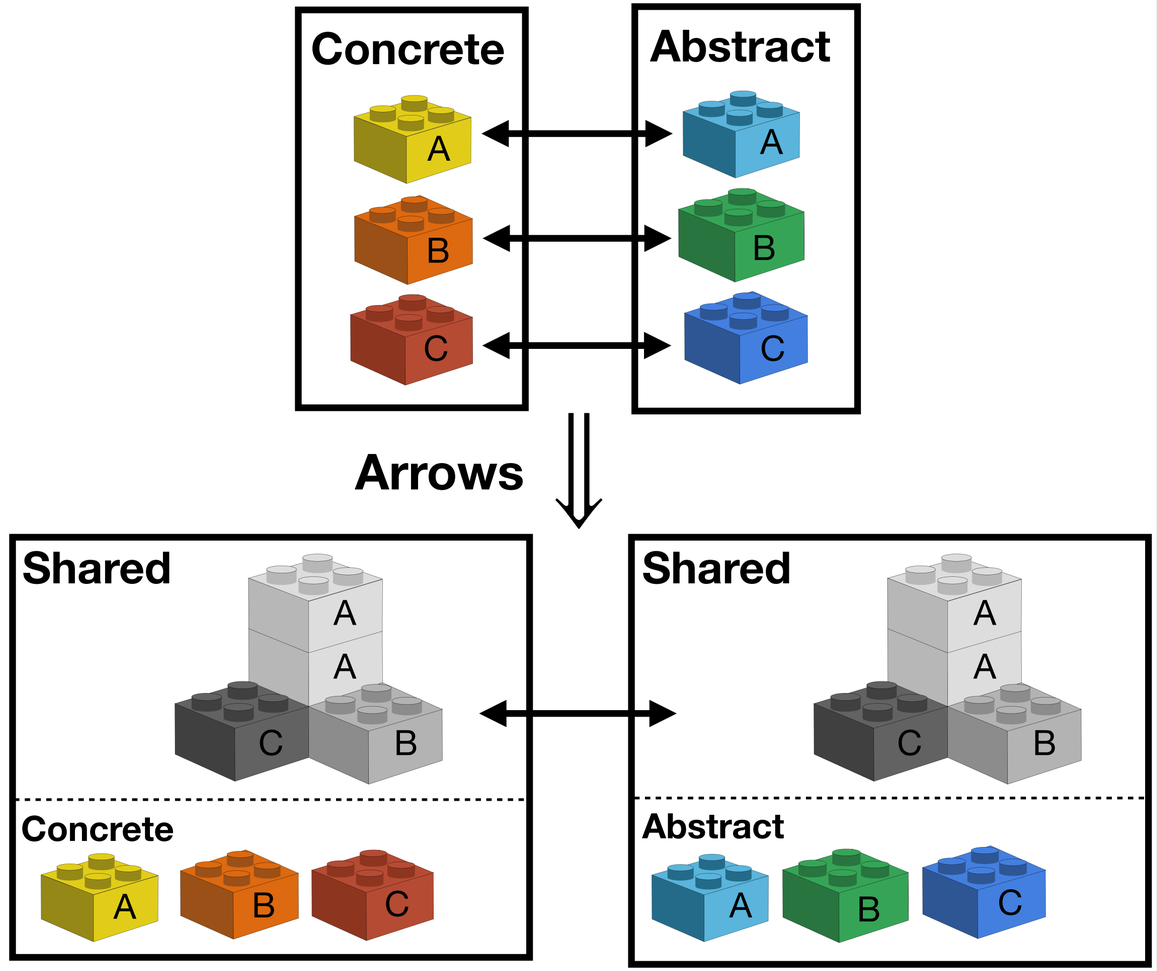
Compositional Soundness Proofs of Abstract Interpreters

Non-Compositional









github.com/svenkeidel/sturdy

Arrows

Pretty notation

```
eval = proc e → case e of
  Add e₁ e₂ → do
    v₁ ← eval ⤙ e₁
    v₂ ← eval ⤙ e₂
    plus ⤙ (v₁,v₂)
```

First-Order

```
E ::= plus
    | E >>> E
    | E *** E
    | E +++ E
    | arr F
```

Induction Principle

$$\frac{\begin{array}{c} P(plus) \\ P(f) \wedge P(g) \Rightarrow P(f >>> g) \\ P(f) \wedge P(g) \Rightarrow P(f *** g) \\ P(f) \wedge P(g) \Rightarrow P(f +++ g) \\ \forall f \in F.\ P(arr\ f) \end{array}}{\forall e \in E.\ P(e)}$$

Monads

Pretty notation

```
eval e = case e of
  Add e₁ e₂ → do
    v₁ ← eval e₁
    v₂ ← eval e₂
    plus v₁ v₂
```

Higher-Order

```
>>= :: m a → (a → m b) → m b
```

?