Arrow-based Abstract Interpreters

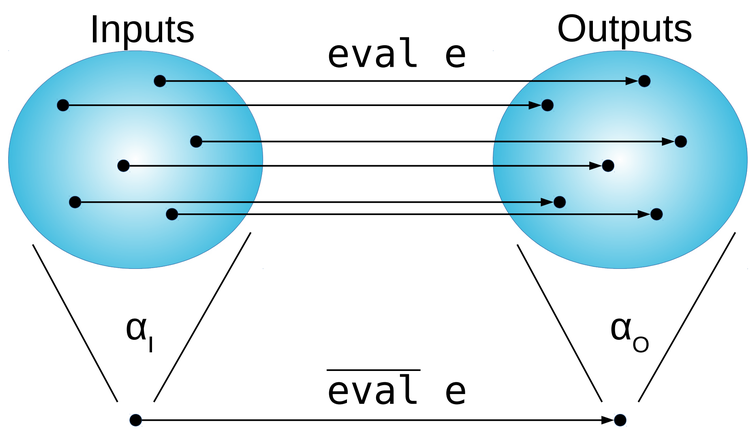
Sven Keidel, Casper Bach Poulsen, Sebastian Erdweg
Programming languages research group – University of Technology Delft

The Problem

What is abstract interpretation?

- Approximation of program properties such as termination, runtime errors, reachbility, …
- Examples: static Analyzers, type Checkers
- Used in compilers, for debugging, and verification

Soundness proofs of abstract interpreters



For all X, α_O {eval e ρ | ρ ∈ X} ⊑ $\overline{eval}$ e α_I(X)

Our Solution

Arrow-based abstract interpreters make soundness proofs more compositional

We describe concrete and abstract interpreter with a single shared parametrized interpreter in arrow style.

```
Shared Interpreter
eval :: (ArrowChoice c, IsValue v c) => c Expr v
eval = proc e → case e of
  Var x     → lookup -< x
  Add e1 e2 → plus   <<< eval *** eval -< (e1,e2)
  Div e1 e2 → divide <<< eval *** eval -< (e1,e2)
```

```
Concrete Interpreter                 sound   Abstract Interpreter
eval :: Interp Expr Val                      eval :: Interp Expr Val

arr f   = …                                  arr f   = …
f >>> g = …                                  f >>> g = …
f *** g = …                                  f *** g = …
f +++ g = …                                  f +++ g = …
f <+> g = …                                  f <+> g = …
lookup  = …                                  lookup  = …
plus    = …                                  plus    = …
divide  = …                                  divide  = …
```

sound

Soundness Lemmas

If f ⊑ $\overline{f}$ then arr f ⊑ $\overline{arr}$ $\overline{f}$
If f ⊑ $\overline{f}$ and g ⊑ $\overline{g}$ then (f >>> g) ⊑ ($\overline{f}$ $\overline{>>>}$ $\overline{g}$)
If f ⊑ $\overline{f}$ and g ⊑ $\overline{g}$ then (f *** g) ⊑ ($\overline{f}$ $\overline{***}$ $\overline{g}$)
If f ⊑ $\overline{f}$ and g ⊑ $\overline{g}$ then (f +++ g) ⊑ ($\overline{f}$ $\overline{+++}$ $\overline{g}$)
If f ⊑ $\overline{f}$ and g ⊑ $\overline{g}$ then (f <+> g) ⊑ ($\overline{f}$ $\overline{<+>}$ $\overline{g}$)

Soundness Theorem

E ::= arr F | E >>> E | E *** E | E +++ E | E <+> E

For all e ∈ E, (e :: Interp a b) ⊑ (e :: $\overline{Interp}$ $\overline{a}$ $\overline{b}$)

Prove by induction over expressions of E

Benefits

- Decompose soundness proof into smaller proof obligations
- Construction of a soundness proof of eval without manual proof effort

Soundness of pure functions

Parametricity is essential for soundness of arrow-based abstract interpreters

We require that all pure functions (arr f) uses types with non-standard orderings parametrically

```
Shared Interpreter
eval = proc e → case e of
  IfZero e1 e2 e3 → do
    v ← eval -< e1
    b ← isZero -< v
    case b of
      True  → eval -< e2
      False → eval -< e3
      Top   → (eval -< e2) <+> (eval -< e3)
```

Top / True / False

b not handled parametrically

```
Concrete Interpreter
isZero = arr $ \n →
  if n == 0
  then True
  else False
```

```
Abstract Interpreter
isZero = arr $ \(n1,n2) →
  if n1 == 0 && n2 == 0
  then True
  else if n1 > 0 || n2 < 0
  then False
  else Top
```

Parametric Solution

```
Shared Interpreter
eval = proc e → case e of
  IfZero e1 e2 e3 → do
    v ← eval -< e1
    isZero (proc () → eval -< e2)
           (proc () → eval -< e3) -<< v
```

```
Concrete Interpreter
isZero f g = proc n →
  if n == 0
  then f -< ()
  else g -< ()
```

```
Abstract Interpreter
isZero f g = proc (n1,n2) →
  if n1 == 0 && n2 == 0
  then f -< ()
  else if n1 > 0 || n2 < 0
  then g -< ()
  else f <+> g -< ()
```

Fixpoints

Calculating fixpoints of arrow-based abstract interpreters

Challenge: Finding precise and sound fixpoint combinators that requires no explicit soundness proof of eval

We adapted the technique of calculating fixpoints from Abstracting Abstract Machines (Van Horn).

Function calls are bound to a finite amount of addresses.

If the interpreter runs out of addresses, it overapproximates.

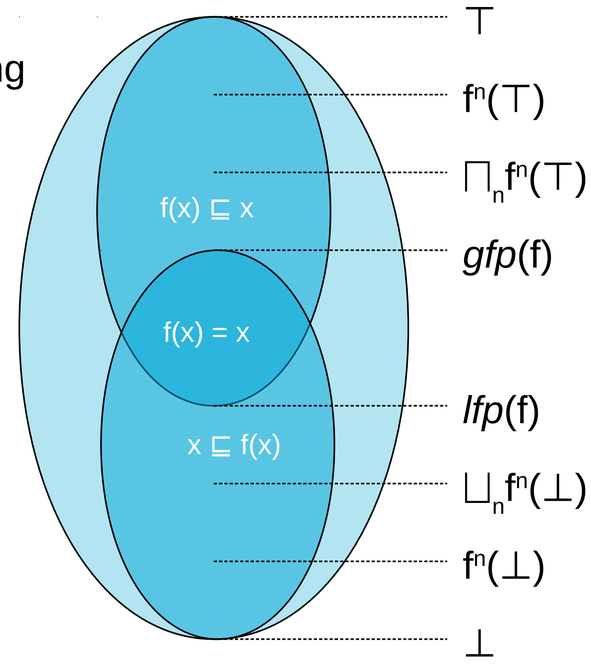


Image: Nielson & Nielson