

# EC4219: Software Engineering

## Lecture 18 — Abstract Interpretation (3) *Implementation of Sign Analysis*

Sunbeom So  
2024 Spring

The full implementation can be found at:

<https://github.com/gist-pal/ec4219-software-engineering/blob/main/code-examples/sign-analysis/signAnalysis.ml>

$$a \rightarrow n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$
$$b \rightarrow \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$$
$$c \rightarrow x := a \mid \text{skip} \mid c_1; c_2 \mid \text{if } b \text{ } c_1 \text{ } c_2 \mid \text{while } b \text{ } c$$

```
1 type aexp =  
2   | Int of int  
3   | Var of var  
4   | Plus of aexp * aexp  
5   | Mul of aexp * aexp  
6   | Sub of aexp * aexp  
7  
8 and var = string  
9 ...
```

# Abstract Domain for Booleans

Recall Lecture 16:

The truth values  $\mathbf{T} = \{true, false\}$  are abstracted by the complete lattice  $(\hat{\mathbf{T}}, \sqsubseteq)$ :

$$\hat{\mathbf{T}} = \{\top, \perp, \widehat{true}, \widehat{false}\}$$

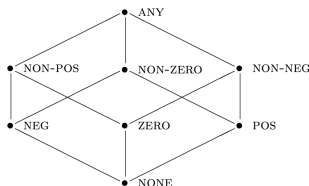
```
1 module AbsBool = struct
2   type t = Top | Bot | True | False
3
4   let porder : t -> t -> bool
5   = fun b1 b2 ->
6     if b1 = b2 then true
7     else
8       match b1,b2 with
9       | Bot,_ -> true
10      | _,Top -> true
11      | _ -> false
12   ...
13 end
```

# Abstract Domain for Arithmetics

The abstract domain is defined as a pair **(Sign,  $\sqsubseteq$ )**:

**Sign** =  $\{\top, \perp, \text{Pos}, \text{Neg}, \text{Zero}, \text{Non-Pos}, \text{Non-Neg}, \text{Non-Zero}\}$

where  $\top = \text{ANY}$ ,  $\perp = \text{NONE}$ , and the partial order ( $\sqsubseteq$ ) is defined as:



```
1 module Sign = struct
2   type t = Top | Bot | Pos | Neg | Zero | NonPos | NonNeg | NonZero
3   let porder : t -> t -> bool
4   = fun s1 s2 ->
5     match s1,s2 with
6     | _ when s1 = s2 -> true | Bot, _ -> true | _, Top -> true
7     | Neg, NonPos -> true | Neg, NonZero -> true | Zero, NonPos -> true
8     | ...
```

# Abstract Memory State

The value abstraction is extended to the memory abstraction. The complete lattice of abstract states ( $\widehat{\mathbf{State}}, \sqsubseteq$ ):

$$\widehat{\mathbf{State}} = \mathit{Var} \rightarrow \mathbf{Sign}$$

with the pointwise ordering:

$$\hat{s}_1 \sqsubseteq \hat{s}_2 \iff \forall x \in \mathit{Var}. \hat{s}_1(x) \sqsubseteq \hat{s}_2(x).$$

```
1 module AbsMem = struct
2   module Map = Map.Make(String) (* key domain: variable *)
3   type t = Sign.t Map.t (* map domain: var -> Sign.t *)
4
5   let porder : t -> t -> bool
6   = fun m1 m2 ->
7     Map.for_all (fun x v -> Sign.porder v (find x m2)) m1
8   ...
9 end
```

# Abstract Semantics for Arithmetics

$$\begin{aligned}\widehat{\mathcal{A}}[a] &: \widehat{\text{State}} \rightarrow \text{Sign} \\ \widehat{\mathcal{A}}[n](\hat{s}) &= \alpha_{\text{Sign}}(\{n\}) \\ \widehat{\mathcal{A}}[x](\hat{s}) &= \hat{s}(x) \\ \widehat{\mathcal{A}}[a_1 + a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) +_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s}) \\ \widehat{\mathcal{A}}[a_1 \star a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) \star_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s}) \\ \widehat{\mathcal{A}}[a_1 - a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) -_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s})\end{aligned}$$

```
1 let rec eval_a : aexp -> AbsMem.t -> Sign.t
2 = fun a m ->
3   match a with
4   | Int n -> Sign.alpha' n
5   | Var x -> AbsMem.find x m
6   | Plus (a1, a2) -> Sign.add (eval_a a1 m) (eval_a a2 m)
7   ...
8 module Sign = struct
9   let add : t -> t -> t
10  = fun s1 s2 ->
11    match s1,s2 with
12    | Bot,_ | _,Bot -> Bot | Top,_ | _,Top -> Top
13    | Neg,Neg -> Neg | Neg,Zero -> Neg | Neg,NonPos -> Neg ...
```

# Abstract Semantics for Booleans

$$\begin{aligned}\widehat{\mathcal{B}}[b] &: \widehat{\text{State}} \rightarrow \widehat{\mathbf{T}} \\ \widehat{\mathcal{B}}[\text{true}](\hat{s}) &= \widehat{\text{true}} \\ \widehat{\mathcal{B}}[\text{false}](\hat{s}) &= \widehat{\text{false}} \\ \widehat{\mathcal{B}}[a_1 = a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) =_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s}) \\ \widehat{\mathcal{B}}[a_1 \leq a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) \leq_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s}) \\ \widehat{\mathcal{B}}[\neg b](\hat{s}) &= \neg_{\widehat{\mathbf{T}}} \widehat{\mathcal{B}}[b](\hat{s}) \\ \widehat{\mathcal{B}}[b_1 \wedge b_2](\hat{s}) &= \widehat{\mathcal{B}}[b_1](\hat{s}) \wedge_{\widehat{\mathbf{T}}} \widehat{\mathcal{B}}[b_2](\hat{s})\end{aligned}$$

```
1 let rec eval_b : bexp -> AbsMem.t -> AbsBool.t
2 = fun b m ->
3   match b with
4   | True -> AbsBool.True | False -> AbsBool.False
5   | Eq (a1, a2) -> Sign.eq (eval_a a1 m) (eval_a a2 m)
6   | Leq (a1, a2) -> Sign.leq (eval_a a1 m) (eval_a a2 m)
7   | Not b -> AbsBool.not (eval_b b m)
8   | And (b1, b2) -> AbsBool.band (eval_b b1 m) (eval_b b2 m)
9   ...
10 module Sign = struct ... let eq = ... end
11 module AbsBool = struct ... let not = ... end
```

# Abstract Semantics for Commands

$$\widehat{\mathcal{C}}[c] : \widehat{\text{State}} \rightarrow \widehat{\text{State}}$$

$$\widehat{\mathcal{C}}[x := a] = \lambda \hat{s}. \hat{s}[x \mapsto \widehat{\mathcal{A}}[a](\hat{s})]$$

$$\widehat{\mathcal{C}}[\text{skip}] = \text{id}$$

$$\widehat{\mathcal{C}}[c_1; c_2] = \widehat{\mathcal{C}}[c_2] \circ \widehat{\mathcal{C}}[c_1]$$

$$\widehat{\mathcal{C}}[\text{if } b \text{ } c_1 \text{ } c_2] = \widehat{\text{cond}}(\widehat{\mathcal{B}}[b], \widehat{\mathcal{C}}[c_1], \widehat{\mathcal{C}}[c_2])$$

$$\widehat{\mathcal{C}}[\text{while } b \text{ } c] = \lambda \hat{m}. \text{filter}(\neg b)(\text{fix}(\lambda \hat{x}. m \cup \widehat{\mathcal{C}}[c](\text{filter}(b)(\hat{x}))))(\hat{m})$$

$$\text{where } \widehat{F}(g) = \widehat{\text{cond}}(\widehat{\mathcal{B}}[b], g \circ \widehat{\mathcal{C}}[c], \text{id})$$

$$\widehat{\text{cond}}(f, g, h)(\hat{s}) = \begin{cases} \perp & \dots f(\hat{s}) = \perp \\ g(\hat{s}) & \dots f(\hat{s}) = \widehat{\text{true}} \\ h(\hat{s}) & \dots f(\hat{s}) = \widehat{\text{false}} \\ g(\hat{s}) \sqcup h(\hat{s}) & \dots f(\hat{s}) = \top \end{cases}$$



# Abstract Semantics for Commands (Cont'd)

```
1 let rec eval_c : cmd -> AbsMem.t -> AbsMem.t
2 = fun c m ->
3   match c with
4   | Assign (x,a) -> AbsMem.add x (eval_a a m) m
5   | Seq (c1,c2) -> eval_c c2 (eval_c c1 m)
6   | If (b, c1, c2) -> cond (eval_b b, eval_c c1, eval_c c2) m
7   | While (b, c) ->
8     let filter p x =
9       if AbsBool.porder AbsBool.True (eval_b p x) then x
10      else AbsMem.empty in
11     let onestep x = AbsMem.join m (eval_c c (filter b x)) in
12     let rec fix f x i =
13       let x' = f x in
14       if AbsMem.porder x' x then x
15       else fix f x' (i+1)
16     in
17     filter (Not b) (fix onestep m 1)
18
19 and cond (f,g,h) m =
20   match f m with
21   | AbsBool.Bot -> AbsMem.empty | AbsBool.True -> g m
22   | AbsBool.False -> h m | AbsBool.Top -> AbsMem.join (g m) (h m)
```