AAA616: Program Analysis

Lecture 4 — Introduction to Static Analysis

Hakjoo Oh
2022 Fall

Static Program Analysis

A general method for
automatic and sound approximation of
sw run-time behaviors
before the execution

- "before": statically, without running sw
- "automatic": sw analyzes sw
- "sound": all possibilities into account
- "approximation": cannot be exact
- "general": for any source language and property
  - C, C++, C#, F#, Java, JavaScript, ML, Scala, Python, JVM, Dalvik, x86, Excel, etc
  - "buffer-overrun?", "memory leak?", "type errors?", "x = y at line 2?", "memory use $\leq \boldsymbol{2K}$?", etc

Program Analysis is Undecidable

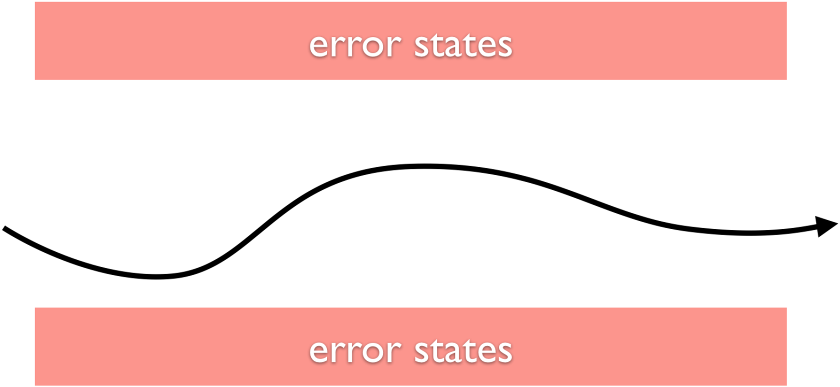
Reasoning about program behavior involves the Halting Problem: e.g.,

$$\texttt{if} \ \cdots \ \texttt{then} \ x := 1 \ \texttt{else} \ (S; x := 2); y := x$$

(S does not define x.) What are the possible values of x at the last statement?

Alan Turing (1912–1954)

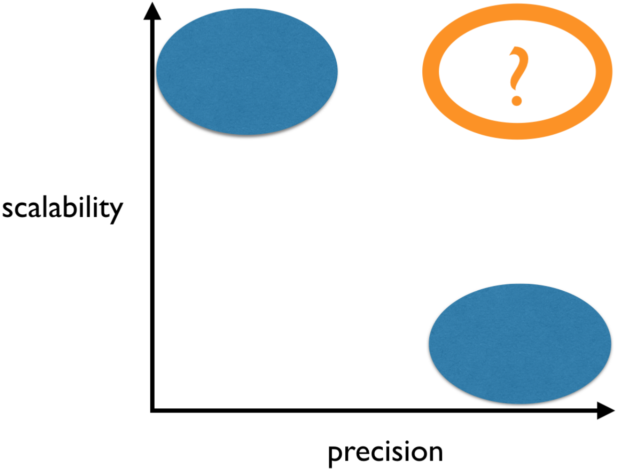
Side-Stepping Undecidability



error states

error states

Tradeoff between Precision and Scalability



The While Language

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

Example 1: Sign Analysis

- The complete lattice $(\mathbf{Sign}, \sqsubseteq)$:

$$\mathbf{Sign} = \{\top, \bot, \mathbf{Pos}, \mathbf{Neg}, \mathbf{Zero}\}$$

$$a \sqsubseteq b \iff a = b \ \lor \ a = \bot \ \lor \ b = \top$$

- The lattice is an abstraction of integers:

$$\alpha_{\mathbf{Sign}} : \wp(\mathbb{Z}) \to \mathbf{Sign}, \qquad \gamma_{\mathbf{Sign}} : \mathbf{Sign} \to \wp(\mathbb{Z})$$

- Join (least upper bound):

$$\begin{array}{l} a \sqcup b = a \ (b \sqsubseteq a) \\ a \sqcup b = b \ (a \sqsubseteq b) \\ a \sqcup b = \top \end{array}$$

Abstract States

The complete lattice of abstract states $(\widehat{\textbf{State}}, \sqsubseteq)$:

$$\widehat{\textbf{State}} = Var \to \textbf{Sign}$$

with the pointwise ordering:

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

The least upper bound of $Y \subseteq \widehat{\textbf{State}}$,

$$\bigsqcup Y = \lambda x.\ \bigsqcup_{\hat{s} \in Y} \hat{s}(x).$$

i.e., $\hat{s_1} \sqcup \hat{s_2} = \lambda x.\ s_1(x) \sqcup s_2(x)$.

Lemma

Let S be a non-empty set and $(D, \sqsubseteq)$ be a poset. Then, the poset $(S \to D, \sqsubseteq)$ with the ordering

$$f_1 \sqsubseteq f_2 \iff \forall s \in S.\ f_1(s) \sqsubseteq f_2(s)$$

is a complete lattice (resp., CPO) if D is a complete lattice (resp., CPO).

Abstract States

The complete lattice of abstract states $(\widehat{\textbf{State}}, \sqsubseteq)$:

$$\widehat{\textbf{State}} = Var \to \textbf{Sign}$$

with the pointwise ordering:

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

The least upper bound of $Y \subseteq \widehat{\textbf{State}}$,

$$\bigsqcup Y = \lambda x.\ \bigsqcup_{\hat{s} \in Y} \hat{s}(x).$$

$$\alpha : \wp(\textbf{State}) \to \widehat{\textbf{State}}$$

$$\alpha(S) = \lambda x.\ \bigsqcup_{s \in S} \alpha_{\textsf{Sign}}(\{s(x)\})$$

$$\gamma : \widehat{\textbf{State}} \to \wp(\textbf{State})$$

$$\gamma(\hat{s}) = \{s \in \textbf{State} \mid \forall x \in Var.\ s(x) \in \gamma_{\textsf{Sign}}(\hat{s}(x))\}$$

Abstract Booleans

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

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

$$\widehat{b_1} \sqsubseteq \widehat{b_2} \iff \widehat{b_1} = \widehat{b_2} \ \vee\ \widehat{b_1} = \bot \ \vee\ \widehat{b_2} = \top$$

The abstraction and concretization functions for the lattice:

$$\alpha_{\widehat{\mathbf{T}}} : \wp(\mathbf{T}) \to \widehat{\mathbf{T}}, \qquad \gamma_{\widehat{\mathbf{T}}} : \widehat{\mathbf{T}} \to \wp(\mathbf{T})$$

Abstract Semantics

$$
\begin{array}{rcl}
\widehat{\mathcal{A}}[\![a]\!] & : & \widehat{\textsf{State}} \to \textsf{Sign} \\
\widehat{\mathcal{A}}[\![n]\!](\hat{s}) & = & \alpha_{\textsf{Sign}}(\{n\}) \\
\widehat{\mathcal{A}}[\![x]\!](\hat{s}) & = & \hat{s}(x) \\
\widehat{\mathcal{A}}[\![a_1 + a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) +_{\textsf{Sign}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 \star a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \star_{\textsf{Sign}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 - a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) -_{\textsf{Sign}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s})
\end{array}
$$

Abstract Semantics

$$\begin{array}{rcl}
\widehat{\mathcal{B}}[\![b]\!] & : & \widehat{\textsf{State}} \to \widehat{\textsf{T}} \\
\widehat{\mathcal{B}}[\![\texttt{true}]\!](\hat{s}) & = & \widehat{true} \\
\widehat{\mathcal{B}}[\![\texttt{false}]\!](\hat{s}) & = & \widehat{false} \\
\widehat{\mathcal{B}}[\![a_1 = a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) =_{\textsf{Sign}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![a_1 \leq a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \leq_{\textsf{Sign}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\neg b]\!](\hat{s}) & = & \neg_{\widehat{\textsf{T}}} \widehat{\mathcal{B}}[\![b]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![b_1 \wedge b_2]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![b_1]\!](\hat{s}) \wedge_{\widehat{\textsf{T}}} \widehat{\mathcal{B}}[\![b_2]\!](\hat{s})
\end{array}$$

Abstract Semantics

$$
\begin{array}{rcl}
\widehat{\mathcal{C}}[\![c]\!] & : & \widehat{\textbf{State}} \rightarrow \widehat{\textbf{State}} \\
\widehat{\mathcal{C}}[\![x := a]\!] & = & \lambda \hat{s}.\hat{s}[x \mapsto \widehat{\mathcal{A}}[\![a]\!](\hat{s})] \\
\widehat{\mathcal{C}}[\![\texttt{skip}]\!] & = & \textbf{id} \\
\widehat{\mathcal{C}}[\![c_1; c_2]\!] & = & \widehat{\mathcal{C}}[\![c_2]\!] \circ \widehat{\mathcal{C}}[\![c_1]\!] \\
\widehat{\mathcal{C}}[\![\texttt{if}\ b\ c_1\ c_2]\!] & = & \widehat{\textbf{cond}}(\widehat{\mathcal{B}}[\![b]\!], \widehat{\mathcal{C}}[\![c_1]\!], \widehat{\mathcal{C}}[\![c_2]\!]) \\
\widehat{\mathcal{C}}[\![\texttt{while}\ b\ c]\!] & = & \mathit{fix}\,\widehat{F} \\
 & & \text{where } \widehat{F}(g) = \widehat{\textbf{cond}}(\widehat{\mathcal{B}}[\![b]\!], g \circ \widehat{\mathcal{C}}[\![c]\!], \textbf{id})
\end{array}
$$

$$
\widehat{\textbf{cond}}(f, g, h)(\hat{s}) = \begin{cases} \bot & \cdots f(\hat{s}) = \bot \\ f(\hat{s}) & \cdots f(\hat{s}) = \widehat{\mathit{true}} \\ g(\hat{s}) & \cdots f(\hat{s}) = \widehat{\mathit{false}} \\ f(\hat{s}) \sqcup g(\hat{s}) & \cdots f(\hat{s}) = \top \end{cases}
$$

Implementation

```
type aexp =
  | Const of int
  | Var of string
  | Plus of aexp * aexp
  | Mult of aexp * aexp
  | Sub of aexp * aexp

type bexp =
  | True  | False
  | Equal of aexp * aexp
  | Le of aexp * aexp
  | Ge of aexp * aexp
  | Not of bexp
  | And of bexp * bexp

type cmd =
  | Assign of string * aexp
  | Seq of cmd list
  | If of bexp * cmd * cmd
  | While of bexp * cmd
```

Implementation

```
module AbsBool = struct
  type t = Top | Bot | True | False
  let not b = ...
  let band b1 b2 = ...
end

module Sign = struct
  type t = Top | Bot | Neg | Zero | Pos
  let order a b = ...
  let alpha n = ...
  let join a b = ...
  let add a b = ...
  let sub a b = ...
  let mul a b = ...
  let equal a b = ...
  let le a b = ...
  let ge a b = ...
end
```

Implementation

```
module AbsMem = struct
  module LocMap = Map.Make(String)
  type t = Sign.t LocMap.t
  let empty = LocMap.empty
  let add = LocMap.add
  let find x m = try LocMap.find x m with _ -> Sign.Bot
  let join m1 m2 =
    LocMap.fold (fun x v m’ -> add x (Sign.join v (find x m’)) m’) m1 m2
  let order m1 m2 =
    LocMap.for_all (fun x v -> Sign.order v (find x m2)) m1
  let print m =
    LocMap.iter (fun x v -> print_endline (x ˆ " |-> " ˆ Sign.to_string v))
end
```

Implementation

```
let rec eval_aexp : aexp -> AbsMem.t -> Sign.t
=fun a m ->
  match a with
  | Const n -> Sign.alpha n
  | Var x -> AbsMem.find x m
  | Plus (a1, a2) -> Sign.add (eval_aexp a1 m) (eval_aexp a2 m)
  | Mult (a1, a2) -> Sign.mul (eval_aexp a1 m) (eval_aexp a2 m)
  | Sub (a1, a2) -> Sign.sub (eval_aexp a1 m) (eval_aexp a2 m)

let rec eval_bexp : bexp -> AbsMem.t -> AbsBool.t
=fun b m ->
  match b with
  | True -> AbsBool.True
  | False -> AbsBool.False
  | Equal (a1, a2) -> Sign.equal (eval_aexp a1 m) (eval_aexp a2 m)
  | Le (a1, a2) -> Sign.le (eval_aexp a1 m) (eval_aexp a2 m)
  | Ge (a1, a2) -> Sign.ge (eval_aexp a1 m) (eval_aexp a2 m)
  | Not b -> AbsBool.not (eval_bexp b m)
  | And (b1, b2) -> AbsBool.band (eval_bexp b1 m) (eval_bexp b2 m)
```

Implementation

```
let rec eval_cmd : cmd -> AbsMem.t -> AbsMem.t
=fun c m ->
  match c with
  | Assign (x, a) -> AbsMem.add x (eval_aexp a m) m
  | Seq cs -> List.fold_left (fun m c -> eval_cmd c m) m cs
  | If (b, c1, c2) -> begin
      match eval_bexp b m with
      | AbsBool.Top -> AbsMem.join (eval_cmd c1 m) (eval_cmd c2 m)
      | AbsBool.True -> eval_cmd c1 m
      | AbsBool.False -> eval_cmd c2 m
      | AbsBool.Bot -> AbsMem.empty
    end
  | While (b, c) ->
    let rec iter b c m =
      match eval_bexp b m with
      | AbsBool.True | AbsBool.Top ->
        if AbsMem.order (eval_cmd c m) m then m
        else iter b c (AbsMem.join m (eval_cmd c m))
      | _ -> m
    in iter b c m
```

Implementation

```
let pgm =
  Seq [
    Assign ("q", Const 1);
    Assign ("r", Var "a");
    While (Ge (Var "r", Var "b"),
        Seq [
          Assign ("r", Sub (Var "r", Var "b"));
          Assign ("q", Plus (Var "q", Const 1))
        ])
  ]

let mem = (AbsMem.add "b" Sign.Pos (AbsMem.add "a" Sign.Pos AbsMem.empty))
let _ = AbsMem.print (eval_cmd pgm mem)
```

Example 2: Taint Analysis (Information Flow Analysis)

Can the information from the untrustworthy source be transferred to the sink?

```
x:=source(); ...; sink(y)
```

Applications to sw security:

- privacy leak
- SQL injection
- buffer overflow
- integer overflow
- XSS
- ...

Abstract Domain

- The complete lattice of the abstract values $(\widehat{\mathbf{T}}, \sqsubseteq)$:

$$\widehat{\mathbf{T}} = \{\mathrm{LOW}, \mathrm{HIGH}\}$$

with the ordering $\mathrm{LOW} \sqsubseteq \mathrm{HIGH}$, $\mathrm{LOW} \sqsubseteq \mathrm{LOW}$, and $\mathrm{HIGH} \sqsubseteq \mathrm{HIGH}$.

- The lattice of states:

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

Abstract Semantics

$$\begin{array}{rcl}
\widehat{\mathcal{A}}[\![a]\!] & : & \widehat{\textsf{State}} \to \widehat{\textsf{T}} \\
\widehat{\mathcal{A}}[\![n]\!](\hat{s}) & = & \left\{ \begin{array}{ll} \mathrm{LOW} & \cdots n \text{ is public} \\ \mathrm{HIGH} & \cdots n \text{ is private} \end{array} \right. \\
\widehat{\mathcal{A}}[\![x]\!](\hat{s}) & = & \hat{s}(x) \\
\widehat{\mathcal{A}}[\![a_1 + a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \sqcup \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 \star a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \sqcup \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 - a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \sqcup \widehat{\mathcal{A}}[\![a_2]\!](\hat{s})
\end{array}$$

Abstract Semantics

$$\begin{array}{rcl}
\widehat{\mathcal{B}}[\![b]\!] & : & \widehat{\textsf{State}} \to \hat{\textsf{T}} \\
\widehat{\mathcal{B}}[\![\texttt{true}]\!](\hat{s}) & = & \mathrm{LOW} \\
\widehat{\mathcal{B}}[\![\texttt{false}]\!](\hat{s}) & = & \mathrm{LOW} \\
\widehat{\mathcal{B}}[\![a_1 = a_2]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![a_1]\!](\hat{s}) \sqcup \widehat{\mathcal{B}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![a_1 \leq a_2]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![a_1]\!](\hat{s}) \sqcup \widehat{\mathcal{B}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\neg b]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![b]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![b_1 \land b_2]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![b_1]\!](\hat{s}) \sqcup \widehat{\mathcal{B}}[\![b_2]\!](\hat{s})
\end{array}$$

Abstract Semantics

$$
\begin{array}{rcl}
\widehat{\mathcal{C}}[\![c]\!] & : & \widehat{\textsf{State}} \to \widehat{\textsf{State}} \\
\widehat{\mathcal{C}}[\![x := a]\!] & = & \lambda\hat{s}.\hat{s}[x \mapsto \widehat{\mathcal{A}}[\![a]\!](\hat{s})] \\
\widehat{\mathcal{C}}[\![\texttt{skip}]\!] & = & \textsf{id} \\
\widehat{\mathcal{C}}[\![c_1; c_2]\!] & = & \widehat{\mathcal{C}}[\![c_2]\!] \circ \widehat{\mathcal{C}}[\![c_1]\!] \\
\widehat{\mathcal{C}}[\![\texttt{if}\ b\ c_1\ c_2]\!] & = & \lambda\hat{s}.\widehat{\mathcal{C}}[\![c_1]\!](\hat{s}) \sqcup \widehat{\mathcal{C}}[\![c_2]\!](\hat{s}) \\
\widehat{\mathcal{C}}[\![\texttt{while}\ b\ c]\!] & = & \mathit{fix}\widehat{F} \\
 & & \text{where } \widehat{F}(g) = \lambda\hat{s}.\hat{s} \sqcup (g \circ \widehat{\mathcal{C}}[\![c]\!])(\hat{s})
\end{array}
$$

Example 3: Interval Analysis

The While language:

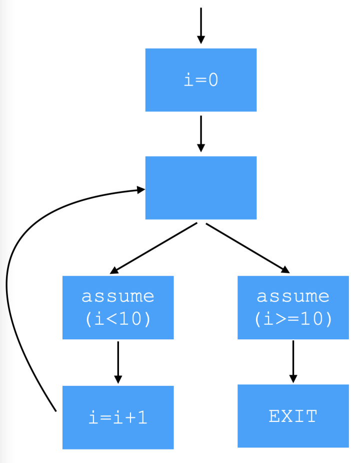
$$\begin{array}{rcl} a & \rightarrow & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\ b & \rightarrow & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ c & \rightarrow & x := a \mid \texttt{skip} \mid c_1; c_2 \mid \texttt{if}\ b\ c_1\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

Assume the program is represented by control flow graph $(\mathbb{C}, \rightarrow)$, where $\mathbb{C}$ denotes the set of nodes and $(\rightarrow) \subseteq \mathbb{C} \times \mathbb{C}$ edges. Each node $c \in \mathbb{C}$ is associated with a command, denoted $\mathbf{cmd}(c)$:

$$cmd \rightarrow skip \mid x := a \mid assume(b)$$

Example

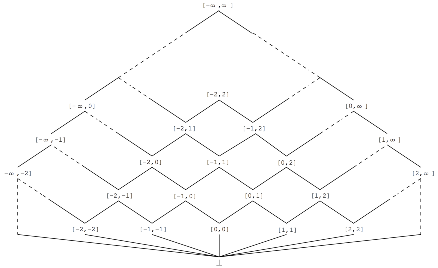
```
i = 0;
while (i<10)
  i++;
```



Interval Domain

The complete lattice $(\hat{\mathbb{Z}}, \sqsubseteq)$:

$$\hat{\mathbb{Z}} = \{\bot\} \cup \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \wedge l \leq u\}$$



Abstraction/concretization functions:

$$\alpha_{\hat{\mathbb{Z}}} : \wp(\mathbb{Z}) \to \hat{\mathbb{Z}}, \qquad \gamma_{\hat{\mathbb{Z}}} : \hat{\mathbb{Z}} \to \wp(\mathbb{Z})$$

Join/Meet:

Abstract States

The complete lattice of abstract states $(\widehat{\textbf{State}}, \sqsubseteq)$:

$$\widehat{\textbf{State}} = \mathit{Var} \to \hat{\mathbb{Z}}$$

with the pointwise ordering:

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

The least upper bound of $Y \subseteq \widehat{\textbf{State}}$,

$$\bigsqcup Y = \lambda x.\ \bigsqcup_{\hat{s} \in Y} \hat{s}(x).$$

$$\alpha : \wp(\textbf{State}) \to \widehat{\textbf{State}}$$

$$\alpha(S) = \lambda x.\ \bigsqcup_{s \in S} \alpha_{\hat{\mathbb{Z}}}(\{s(x)\})$$

$$\gamma : \widehat{\textbf{State}} \to \wp(\textbf{State})$$

$$\gamma(\hat{s}) = \{s \in \textbf{State} \mid \forall x \in \mathit{Var}.\ s(x) \in \gamma_{\hat{\mathbb{Z}}}(\hat{s}(x))\}$$

Abstract Booleans

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

$$\widehat{\mathbf{T}} = \{\top, \bot, \widehat{\mathit{true}}, \widehat{\mathit{false}}\}$$

$$\widehat{b_1} \sqsubseteq \widehat{b_2} \iff \widehat{b_1} = \widehat{b_2} \lor \widehat{b_1} = \bot \lor \widehat{b_2} = \top$$

The abstraction and concretization functions for the lattice:

$$\alpha_{\widehat{\mathbf{T}}} : \wp(\mathbf{T}) \to \widehat{\mathbf{T}}, \qquad \gamma_{\widehat{\mathbf{T}}} : \widehat{\mathbf{T}} \to \wp(\mathbf{T})$$

Abstract Semantics

$$\begin{array}{rcl}
\widehat{\mathcal{A}}[\![a]\!] & : & \widehat{\textsf{State}} \to \hat{\mathbb{Z}} \\
\widehat{\mathcal{A}}[\![n]\!](\hat{s}) & = & \alpha_{\hat{\mathbb{Z}}}(\{n\}) \\
\widehat{\mathcal{A}}[\![x]\!](\hat{s}) & = & \hat{s}(x) \\
\widehat{\mathcal{A}}[\![a_1 + a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) +_{\hat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 \star a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \star_{\hat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 - a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) -_{\hat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s})
\end{array}$$

Abstract Semantics

$$\begin{array}{rcl}
\widehat{\mathcal{B}}[\![b]\!] & : & \widehat{\textsf{State}} \to \widehat{\textsf{T}} \\
\widehat{\mathcal{B}}[\![\texttt{true}]\!](\hat{s}) & = & \widehat{true} \\
\widehat{\mathcal{B}}[\![\texttt{false}]\!](\hat{s}) & = & \widehat{false} \\
\widehat{\mathcal{B}}[\![a_1 = a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) =_{\hat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![a_1 \leq a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \leq_{\hat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\neg b]\!](\hat{s}) & = & \neg_{\widehat{\textsf{T}}} \widehat{\mathcal{B}}[\![b]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![b_1 \wedge b_2]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![b_1]\!](\hat{s}) \wedge_{\widehat{\textsf{T}}} \widehat{\mathcal{B}}[\![b_2]\!](\hat{s})
\end{array}$$

Transfer Function

$$\hat{f}_c : \widehat{\textsf{State}} \to \widehat{\textsf{State}}$$

$$\begin{array}{rcll}
\hat{f}_c(\hat{s}) & = & \hat{s} & c = skip \\
\hat{f}_c(\hat{s}) & = & \hat{s}[x \mapsto \widehat{\mathcal{A}}[\![a]\!](\hat{s})] & c = x := a \\
\hat{f}_c(\hat{s}) & = & \hat{s}[x \mapsto \hat{s}(x) \sqcap [-\infty, n-1]] & c = assume(x < n) \\
\hat{f}_c(\hat{s}) & = & \hat{s} & c = assume(b), \\
 & & & \widehat{true} \sqsubseteq \widehat{\mathcal{B}}[\![b]\!](\hat{s}) \\
\hat{f}_c(\hat{s}) & = & \bot & c = assume(b), \\
 & & & \widehat{false} \sqsupseteq \widehat{\mathcal{B}}[\![b]\!](\hat{s})
\end{array}$$

Fixed Point Equation

The analysis is to compute the least fixed point of the function, i.e., $\boldsymbol{fix}\hat{\boldsymbol{F}}$:

$$\hat{F} : (\mathbb{C} \to \widehat{\textsf{State}}) \to (\mathbb{C} \to \widehat{\textsf{State}})$$

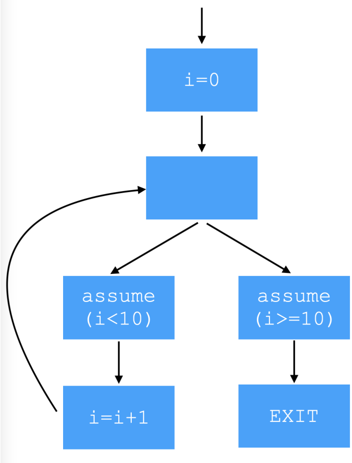
$$\hat{F}(X) = \lambda c.\ \hat{f}_c(\bigsqcup_{c' \to c} X(c'))$$

Fixed Point Computation

The tabulation algorithm naively computes $\bigsqcup_{i\in\mathbb{N}} \hat{F}^i$:

```
$T := T' := \bot (= \lambda c.\bot)$
repeat
    $T' := T$
    $T := T \sqcup \hat{F}(T)$
until $T \sqsubseteq T'$
return $T'$
```

Example



Widening/Narrowing

A simple widening operator for the interval domain:

$$\begin{array}{rcll} [a,b] & \bigtriangledown & \bot & = [a,b] \\ \bot & \bigtriangledown & [c,d] & = [c,d] \\ [a,b] & \bigtriangledown & [c,d] & = [(c < a ? -\infty : a), (b < d ? +\infty : b)] \end{array}$$

A simple narrowing operator:

$$\begin{array}{rcll} [a,b] & \bigtriangleup & \bot & = \bot \\ \bot & \bigtriangleup & [c,d] & = \bot \\ [a,b] & \bigtriangleup & [c,d] & = [(a = -\infty ? c : a), (b = +\infty ? d : b)] \end{array}$$

Point-wise extensions for states and tables:

$$\begin{array}{rcl} \hat{s_1} \bigtriangledown \hat{s_2} & = & \lambda x.\ s_1(x) \bigtriangledown s_2(x) \\ X_1 \bigtriangledown X_2 & = & \lambda c.\ X_1(c) \bigtriangledown X_2(c) \end{array}$$

Fixed Point Algorithm

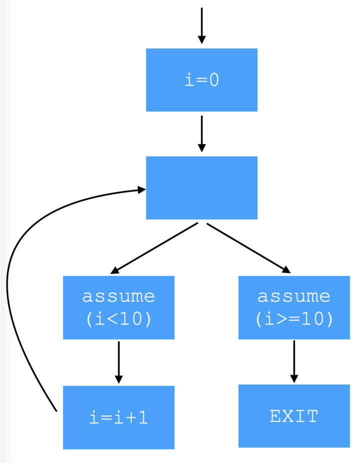
$$\hat{F}(X) = \lambda c.\ \hat{f}_c(\bigsqcup_{c' \to c} X(c'))$$

The tabulation algorithm algorithm naively computes $\bigsqcup \hat{F}$:

```
$T := T' := \bot (= \lambda c. \bot)$
repeat
    $T' := T$
    $T := T \bigtriangledown \hat{F}(T)$
until $T \sqsubseteq T'$

$T := T'$
repeat
    $T' := T$
    $T := T \bigtriangleup \hat{F}(T)$
until $T' \sqsubseteq T$
return $T'$
```

Example



Worklist Algorihtm

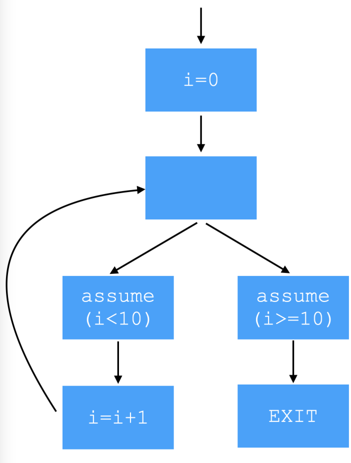
$$\hat{F}(X) = \lambda c.\ \hat{f}_c(\bigsqcup_{c' \to c} X(c'))$$

Worklist algorithm:

```
$W := \mathbb{C}$
$X := \bot$
repeat
    $c :=$ choose$(W)$
    $W := W \setminus \{c\}$
    $s := \hat{f}_c(\bigsqcup_{c' \to c} X(c'))$
    if $s \not\sqsubseteq X(c)$
        $X(c) := X(c) \triangledown s$
        $W := W \cup \{c \mid c \to c'\}$
until $W = \emptyset$
```

```
$W := \mathbb{C}$
repeat
    $c :=$ choose$(W)$
    $W := W \setminus \{c\}$
    $s := \hat{f}_c(\bigsqcup_{c' \to c} X(c'))$
    if $X(c) \not\sqsubseteq s$
        $X(c) := X(c) \vartriangle s$
        $W := W \cup \{c \mid c \to c'\}$
until $W = \emptyset$
```

Example

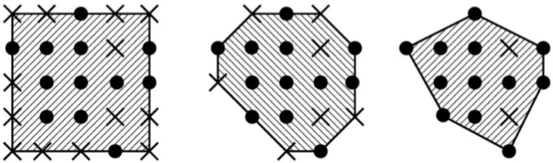


Strength and Weakness

-
  ```
  x = 0;
  while (x < 10) {
    assert (x < 10);
    x++;
  }
  assert (x == 10);
  ```

-
  ```
  x = 0;
  y = 0;
  while (x < 10) {
    assert (y < 10);
    x++; y++;
  }
  assert (y == 10);
  ```

Other Numerical Abstract Domains



Intervals Octagons Polyhedra

```
i = 0;
p = 0;

while (i < 12) {
    i = i + 1;
    p = p + 1;
}
assert(i==p)
```

Interval analysis

i	[12,12]
p	[0,+oo]

Octagon analysis

i	[12,12]
p	[12,12]
p-i	[0,0]
p+i	[24,24]

Summary

Introduction to

- sign analysis, taint analysis, interval analysis
- abstract domain, abstract semantics, fixed point algorithm, widening/narrowing
- implementation of static analysis

Next: abstract interpretation framework