Abstract Interpretation IV

Semantics and Application to Program Verification

Antoine Miné

École normale supérieure, Paris

year 2013–2014

Course 12

21 May 2014

Overview

Selected advanced topics:

- Disjunctive abstract domains
- Abstracting arrays
- Inter-procedural analyses
- Backward analyses

Practical session: help with the project

Disjunctive domains

Motivation

Remark: most domains abstract **convex sets** (conjunctions of constraints)

$\Longrightarrow$ $\cup^\sharp$ causes a loss of precision!

The need for non-convex invariants

$X \leftarrow$ **rand**$(10, 20)$;
$Y \leftarrow$ **rand**$(0, 1)$;
if $Y > 0$ **then** $X \leftarrow -X$;
• $Z \leftarrow 100/X$

Concrete semantics:

At •, $X \in [-20, -10] \cup [10, 20]$
$\Longrightarrow$ there is no division by zero

Abstract analysis:

Convex analyses (intervals, polyhedra) will find $X \in [-20, 20]$
(with intervals, $[-20, -10] \cup^\sharp [10, 20] = [-20, 20]$)
$\Longrightarrow$ possible division by zero (false alarm)

Disjunctive domains

Principle:

generic constructions to lift any numeric abstract domain to a domain able to represent disjunctions exactly

Example constructions:

- powerset completion
  unordered "soup" of abstract elements
- state partitioning
  abstract elements keyed to selected subsets of environments
- decision tree abstract domains
  efficient representation of state partitioning
- path-sensitive analyses
  partition with respect to the history of execution

each construction has its strength and weakness
they can be combined during an analysis to exploit the best in each

Powerset completion

Given: $(\mathcal{E}^\sharp, \sqsubseteq, \gamma, \cup^\sharp, \cap^\sharp, \nabla, \mathsf{S}^\sharp[\![\, stat \,]\!]\,)$

abstract domain $\mathcal{E}^\sharp$
ordered by $\sqsubseteq$, which also acts as a sound abstraction of $\subseteq$ (i.e., $\subseteq^\sharp = \sqsubseteq$)
with concretization $\gamma : \mathcal{E}^\sharp \to \mathcal{P}(\mathcal{E})$
sound abstractions $\cup^\sharp$, $\cap^\sharp$, $\mathsf{S}^\sharp[\![\, stat \,]\!]$ of $\cup$, $\cap$, $\mathsf{S}[\![\, stat \,]\!]$, and a widening ∇

Construct: $(\hat{\mathcal{E}}^\sharp, \hat{\sqsubseteq}, \hat{\gamma}, \hat{\cup}^\sharp, \hat{\cap}^\sharp, \hat{\nabla}, \hat{\mathsf{S}}^\sharp[\![\, stat \,]\!]\,)$

- $\hat{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} \mathcal{P}_{finite}(\mathcal{E}^\sharp)$ (finite sets of abstract elements)
- $\hat{\gamma}(A^\sharp) \stackrel{\text{def}}{=} \cup \{\, \gamma(X^\sharp) \,|\, X^\sharp \in A^\sharp \,\}$ (join of concretizations)

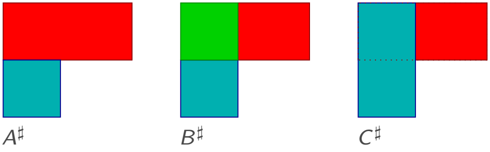
Example:
using the interval domain for $\mathcal{E}^\sharp$
$\hat{(}\gamma\{[-10,-5],[2,4],[0,0],[2,3]\}) = [-10,-5] \cup \{0\} \cup [2,4]$

Ordering

Issue: how can we compare two elements of $\hat{\mathcal{E}}^\sharp$?

- $\hat{\gamma}$ is generally not injective
  there is no canonical representation for $\hat{\gamma}(A^\sharp)$
- testing $\hat{\gamma}(A^\sharp) = \hat{\gamma}(B^\sharp)$ or $\hat{\gamma}(A^\sharp) \subseteq \hat{\gamma}(B^\sharp)$ is difficult

Example: powerset completion of the interval domain



$A^\sharp = \{\{0\} \times \{0\}, [0,1] \times \{1\}\}$
$B^\sharp = \{\{0\} \times \{0\}, \{0\} \times \{1\}, \{1\} \times \{1\}\}$
$C^\sharp = \{\{0\} \times [0,1], [0,1] \times \{1\}\}$

$\hat{\gamma}(A^\sharp) = \hat{\gamma}(B^\sharp) = \hat{\gamma}(C^\sharp)$

$B^\sharp$ is more costly to represent: it requires three abstract elements instead of two
$C^\sharp$ is a covering and not a partition (red $\cap$ blue $= \{0\} \times \{1\} \neq \emptyset$)

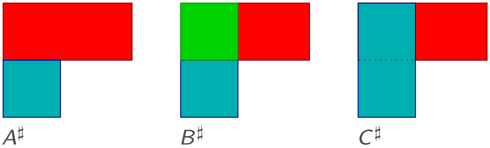
Ordering (cont.)

Solution: sound approximation of $\subseteq$

$A^\sharp \mathrel{\hat{\sqsubseteq}} B^\sharp \stackrel{\text{def}}{\iff} \forall X^\sharp \in A^\sharp : \exists Y^\sharp \in B^\sharp : X^\sharp \sqsubseteq Y^\sharp$ (Hoare powerdomain order)

- $\hat{\sqsubseteq}$ is a partial order (when $\sqsubseteq$ is)
- $\hat{\sqsubseteq}$ is a sound approximation of $\subseteq$ (when $\sqsubseteq$ is)
  $(A^\sharp \mathrel{\hat{\sqsubseteq}} B^\sharp \implies \hat{\gamma}(A^\sharp) \subseteq \hat{\gamma}(B^\sharp))$
- testing $\hat{\sqsubseteq}$ reduces to testing $\sqsubseteq$ finitely many times

Example: powerset completion of the interval domain



$\hat{\gamma}(A^\sharp) = \hat{\gamma}(B^\sharp) = \hat{\gamma}(C^\sharp)$

$B^\sharp \mathrel{\hat{\sqsubseteq}} A^\sharp \mathrel{\hat{\sqsubseteq}} C^\sharp$

Abstract operations

Abstract operators

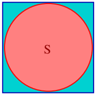
- $\hat{\mathsf{S}}^\sharp[\![\, \mathit{stat} \,]\!]\, A^\sharp \stackrel{\text{def}}{=} \{\, \mathsf{S}^\sharp[\![\, \mathit{stat} \,]\!]\, X^\sharp \mid X^\sharp \in A^\sharp \,\}$
  apply *stat* on each abstract element independently
- $A^\sharp \mathbin{\hat{\cup}^\sharp} B^\sharp \stackrel{\text{def}}{=} A^\sharp \cup B^\sharp$
  keep elements from both arguments without applying any abstract operation
  $\hat{\cup}^\sharp$ is exact
- $A^\sharp \mathbin{\hat{\cap}^\sharp} B^\sharp \stackrel{\text{def}}{=} \{\, X^\sharp \cap^\sharp Y^\sharp \mid X^\sharp \in A^\sharp,\ Y^\sharp \in B^\sharp \,\}$
  $\hat{\cap}^\sharp$ is exact if $\cap^\sharp$ is (as $\cup$ and $\cap$ are distributive)

Galois connection:

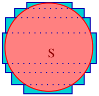
in general, there is no abstraction function $\hat{\alpha}$ corresponding to $\hat{\gamma}$

Example: powerset completion $\hat{\mathcal{E}}^\sharp$ of the interval domain $\mathcal{E}^\sharp$

given the disc $S \stackrel{\text{def}}{=} \{\, (x,y) \mid x^2 + y^2 \le 1 \,\}$
$\alpha(S) = [-1,1] \times [-1,1]$ (optimal interval abstraction)
but there is no best abstraction in $\hat{\mathcal{E}}^\sharp$



$\alpha(S)$



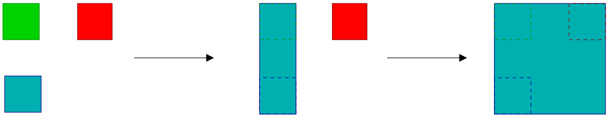
not $\hat{\alpha}(S)$

Dynamic approximation

Issue: the size $|A^\sharp|$ of elements $A^\sharp \in \hat{\mathcal{E}}^\sharp$ is unbounded
(every application of $\hat{\cup}^\sharp$ adds some more elements)
$\Longrightarrow$ efficiency and convergence problems

Solution: to reduce the size of elements

- redundancy removal
  $\textit{simplify}(A^\sharp) \stackrel{\text{def}}{=} \{\, X^\sharp \in A^\sharp \mid \forall Y^\sharp \neq X^\sharp \in A^\sharp : X^\sharp \not\sqsubseteq Y^\sharp \,\}$
  (no loss of precision: $\hat{\gamma}(\textit{simplify}(A^\sharp)) = \hat{\gamma}(A^\sharp)$)
- collapse: join elements in $\mathcal{E}^\sharp$
  $\textit{collapse}(A^\sharp) \stackrel{\text{def}}{=} \{\cup^\sharp \{\, X^\sharp \in A^\sharp \,\}\}$



  (large loss of precision, but very effective: $|\textit{collapse}(A^\sharp)| = 1$)
- partial collapse: limit $|A^\sharp|$ to a fixed size k by $\cup^\sharp$
  (but how to choose which elements to merge? no easy solution!)

Widening

Issue: for loops, abstract iterations $(A^\sharp_n)_{n\in\mathbb{N}}$ may not converge

- the size of $A^\sharp_n$ may grow arbitrarily large
- even if $|A^\sharp_n|$ is stable, some elements in $A^\sharp_n$ may not converge
  (if $\mathcal{E}^\sharp$ has infinite increasing sequences)

$\implies$ we need a widening $\triangledown$

Widenings for powerset domains are difficult to design

Example widening: collapse after a fixed number N of iterations

$$A^\sharp_{n+1} \stackrel{\text{def}}{=} \begin{cases} A^\sharp_n \mathrel{\hat{\cup}^\sharp} B^\sharp_{n+1} & \text{if } n < N \\ \mathit{collapse}(A^\sharp_n) \mathrel{\triangledown} \mathit{collapse}(B^\sharp_{n+1}) & \text{otherwise} \end{cases}$$

(this is very naïve, see Bagnara et al. STTT06 for more interesting widenings)

State partitioning

Principle:

- partition *a priori* $\mathcal{E}$ into finitely many sets
- abstract each partition separately in $\mathcal{E}^\sharp$

Abstract domain:

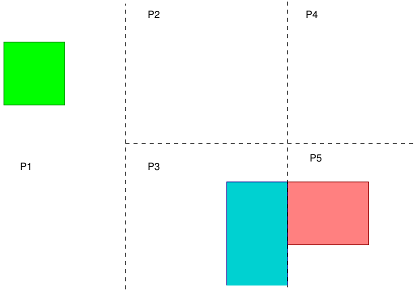
Given $P^\sharp \subseteq \mathcal{E}^\sharp$ such that:

- $P^\sharp$ is finite
- $\cup \{ \gamma(X^\sharp) \,|\, X^\sharp \in P^\sharp \} = \mathcal{E}$
  for generally, we have a covering, not a partitioning of $\mathcal{E}$
  i.e., we can have $X^\sharp \neq Y^\sharp \in P^\sharp$ with $\gamma(X^\sharp) \cap \gamma(Y^\sharp) \neq \emptyset$

Then $\tilde{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} P^\sharp \to \mathcal{E}^\sharp$

(representable in memory, as $P^\sharp$ is finite)

Ordering



Example: $\mathcal{E}^\sharp$ is the interval domain

$$P^\sharp = \{P_1, P_2, P_3, P_4, P_5\} \text{ where}$$
$$\begin{aligned} P_1 &= [-\infty, 0] \times [-\infty, +\infty] \\ P_2 &= [0, 10] \times [0, +\infty] \\ P_3 &= [0, 10] \times [-\infty, 0] \\ P_4 &= [10, +\infty] \times [0, +\infty] \\ P_5 &= [10, +\infty] \times [-\infty, 0] \end{aligned}$$

$$\begin{aligned} X^\sharp = [&P_1 \mapsto [-6, -5] \times [5, 6],\ P_2 \mapsto \bot, \\ &P_3 \mapsto [9, 10] \times [-\infty, -1],\ P_4 \mapsto \bot, \\ &P_5 \mapsto [10, 12] \times [-3, -1]] \end{aligned}$$

- $\tilde{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} P^\sharp \to \mathcal{E}^\sharp$
- $\tilde{\gamma}(A^\sharp) \stackrel{\text{def}}{=} \cup\ \{\, \gamma(A^\sharp(X^\sharp)) \cap \gamma(X^\sharp) \mid X^\sharp \in P^\sharp \,\}$
- $A^\sharp \mathrel{\tilde{\sqsubseteq}} B^\sharp \stackrel{\text{def}}{\iff} \forall X^\sharp \in P^\sharp\colon A^\sharp(X^\sharp) \sqsubseteq B^\sharp(X^\sharp)$ (point-wise order)
- $\tilde{\alpha}(S) \stackrel{\text{def}}{=} \lambda X^\sharp \in P^\sharp . \alpha(S \cap \gamma(X^\sharp))$

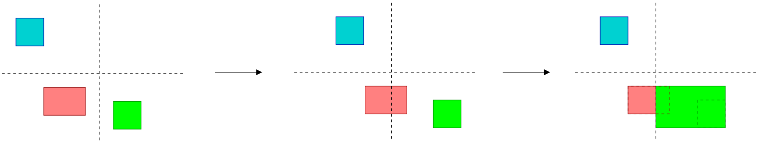
  (if $\mathcal{E}^\sharp$ enjoys a Galois connection, so does $\tilde{\mathcal{E}}^\sharp$)

Abstract operators

Abstract operators: point-wise extension from $\mathcal{E}^\sharp$ to $P^\sharp \to \mathcal{E}^\sharp$

- $A \,\tilde{\cup}^\sharp\, B \stackrel{\text{def}}{=} \lambda X^\sharp \in P^\sharp . A(X^\sharp) \cup^\sharp B(X^\sharp)$
- $A \,\tilde{\cap}^\sharp\, B \stackrel{\text{def}}{=} \lambda X^\sharp \in P^\sharp . A(X^\sharp) \cap^\sharp B(X^\sharp)$
- $A \,\tilde{\triangledown}\, B \stackrel{\text{def}}{=} \lambda X^\sharp \in P^\sharp . A(X^\sharp) \triangledown B(X^\sharp)$
- $\tilde{\mathsf{S}}^\sharp[\![\, e \leq 0?\,]\!]\, A^\sharp \stackrel{\text{def}}{=} \lambda X^\sharp \in P^\sharp . \mathsf{S}^\sharp[\![\, e \leq 0?\,]\!]\, A^\sharp(X^\sharp)$
- $\tilde{\mathsf{S}}^\sharp[\![\, V \leftarrow e\,]\!]\, A^\sharp$ is more complex
  as $\mathsf{S}^\sharp[\![\, V \leftarrow e\,]\!]\, A^\sharp(X^\sharp)$ may escape $X^\sharp$

example: $X \leftarrow X + 2$



$$\tilde{\mathsf{S}}^\sharp[\![\, V \leftarrow e\,]\!]\, A^\sharp \stackrel{\text{def}}{=} \lambda X^\sharp . \cup^\sharp \{\, X^\sharp \cap^\sharp \mathsf{S}^\sharp[\![\, V \leftarrow e\,]\!]\, A(Y^\sharp) \mid Y^\sharp \in P^\sharp \,\}$$

Example analysis

Example

$X \leftarrow \textbf{rand}(10, 20);$
$Y \leftarrow \textbf{rand}(0, 1);$
$\textbf{if } Y > 0 \textbf{ then } X \leftarrow -X;$
• $Z \leftarrow 100/X$

Analysis:

- $\mathcal{E}^\sharp$ is the interval domain
- partition with respect to the sign of X
  $P^\sharp \stackrel{\text{def}}{=} \{ X^+, X^- \}$ where
  $X^+ \stackrel{\text{def}}{=} [0, +\infty] \times \mathbb{Z} \times \mathbb{Z}$ and $X^- \stackrel{\text{def}}{=} [-\infty, 0] \times \mathbb{Z} \times \mathbb{Z}$
- at • we find:
  $X^+ \mapsto [X \in [10, 20], Y \mapsto [0, 0], Z \mapsto [0, 0]]$
  $X^- \mapsto [X \in [-20, -10], Y \mapsto [1, 1], Z \mapsto [0, 0]]$

  $\Longrightarrow$ no division by zero

Binary decision trees

Principle: data-structure to compactly represent partitions

Example: boolean partitions

- assume that variables have a type: $\mathbb{V} \stackrel{\text{def}}{=} \mathbb{V}_b \cup \mathbb{V}_n$
  - each $V \in \mathbb{V}_b$ has value in $\{0,1\}$ (boolean variable)
  - each $V \in \mathbb{V}_n$ has value in $\mathbb{Z}$ (numeric variable)
- $\mathcal{E} \simeq \{0,1\}^{|\mathbb{V}_b|} \times \mathbb{Z}^{|\mathbb{V}_n|}$

  $P^\sharp \stackrel{\text{def}}{=} \{\, \langle b_1, \ldots, b_{|\mathbb{V}_b|} \rangle \times \mathbb{Z}^{|\mathbb{V}_n|} \mid b_1, \ldots, b_{|\mathbb{V}_b|} \in \{0,1\} \,\}$

  a partition corresponds to a precise valuation of all the boolean variables and no information on the numeric variables
- assume that $\mathcal{E}_n^\sharp$ abstracts $\mathcal{P}(\mathbb{V}_n \to \mathbb{Z})$ (numeric domain)

  the boolean partitioning domain based on $\mathcal{E}_n^\sharp$ is:

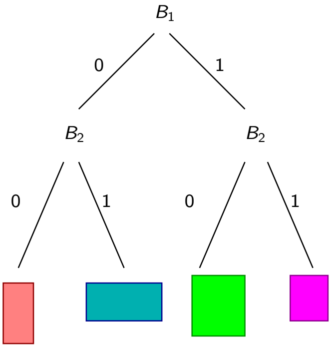
  $\tilde{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} \{0,1\}^{|\mathbb{V}_b|} \to \mathcal{E}_n^\sharp$

Binary decision trees (cont.)

Representation: for $\tilde{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} \{0,1\}^{|\mathbb{V}_b|} \to \mathcal{E}_n^\sharp$

binary trees:

- nodes are labelled with boolean variables $B_i \in \mathbb{V}_b$
- two children: $B_i = 0$ and $B_i = 1$
- leaves are abstract elements in $\mathcal{E}_n^\sharp$ (abstraction of $\mathcal{P}(\mathbb{V}_n \to \mathbb{Z})$)

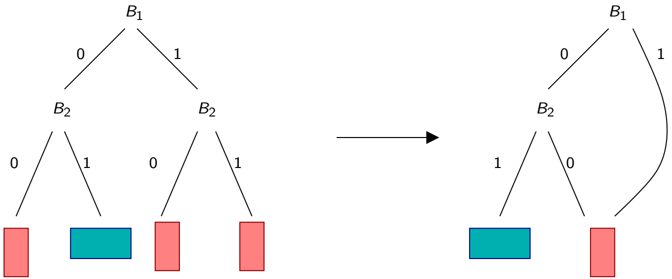


Reduced binary decision trees

Optimization: similar to Reduced Ordered Binary Decision Diagrams

- merge identical sub-trees (memory sharing)
- remove nodes if both children are identical

$\Longrightarrow$ we get a directed acyclic graphs



if $\gamma_n : \mathcal{E}_n^\sharp \to \mathbb{Z}^{|\mathbb{V}_n|}$ is injective and we use memoization
then $\tilde{\gamma}(A^\sharp) = \tilde{\gamma}(B^\sharp) \iff A^\sharp$ and $B^\sharp$ are physically equal
(i.e., == in OCaml, which is faster to test than structural equality =)

Abstract operations

- numeric operations: performed independently on each leaf
  (e.g., $\tilde{\mathsf{S}}^\sharp[\![\, V \leftarrow e \,]\!]$ reverts to applying $\mathsf{S}^\sharp[\![\, V \leftarrow e \,]\!]$ on each leaf)
- boolean operations: manipulate trees
  - $\tilde{\mathsf{S}}^\sharp[\![\, B_i \leftarrow \mathbf{rand}(0,1) \,]\!]$: merge B_i's subtrees recursively
  - $\tilde{\mathsf{S}}^\sharp[\![\, B_i = 0? \,]\!]$: set all $B_i = 1$ branches to $\perp$
  - $\cdots$
- binary operations: $\tilde{\cup}^\sharp$, $\tilde{\cap}^\sharp$, $\tilde{\triangledown}$, $\tilde{\sqsubseteq}$
  - first, unify tree structures (unshare trees and add missing nodes)
  - then, apply the operation pair-wise on leaves
- optimization needs to be performed again after each operation
  (ensures that abstract elements do not grow too large)

Example analysis

Example

$X \leftarrow \textbf{rand}(0, 100);$
$\textbf{if } X = 0 \textbf{ then } B \leftarrow 0 \textbf{ else } B \leftarrow 1;$
$\ldots$
$\bullet$ $\textbf{if } B = 1 \textbf{ then } \bullet\ Y \leftarrow 100/X$

Analysis: using the interval domain for $\mathcal{E}_n^\sharp$

at $\bullet$, we can infer the invariant:
$(B = 0 \implies X = 0) \land (B = 1 \implies X \in [1, 100])$

at $\bullet$, we deduce that $B = 1 \land X \in [1, 100]$
$\implies$ there is no division by zero

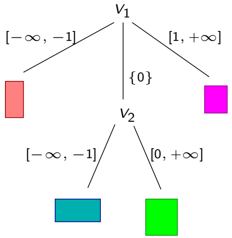
More tree-based partitioning

Other tree-based partitioning data-structure

we can extend partition trees in many ways

- allow $n-$array nodes
  and partition wrt. abstract values

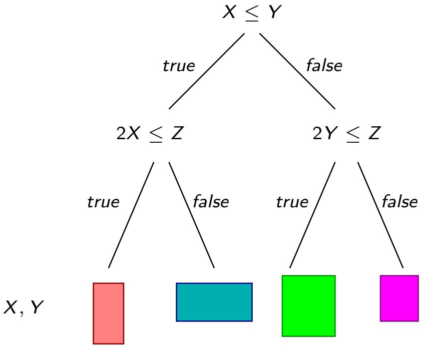
Example: partitioning integer variables in the interval domain



More tree-based partitioning

- partitioning with respect to predicates

Example: linear relations over $\mathbb{V} \stackrel{\text{def}}{=} \{X, Y, Z\}$



the same variables may appear in predicates and in the leaves
$\Longrightarrow$ $S^\sharp[\![\, stat \,]\!]$ must generally update both the nodes and the leaves

the set of node predicates may be fixed before the analysis
or chosen dynamically during the analysis

Path sensitivity

Principle: partition wrt. the history of computation

- keep different abstract elements for different execution paths
  (i.e., different branches taken, different loop iterations)
- avoid merging with $\cup^\sharp$ elements at control-flow joins
  (at the end of **if** $\cdots$ **then** $\cdots$ **else**, or at loop head)

Intuition: as a program transformation

$X \leftarrow \textbf{rand}(-50, 50);$
if $X \geq 0$ **then**
  $Y \leftarrow X + 10$
else
  $Y \leftarrow X - 10;$
assert $Y \neq 0$

$\longrightarrow$

$X \leftarrow \textbf{rand}(-50, 50);$
if $X \geq 0$ **then**
  $Y \leftarrow X + 10;$
  assert $Y \neq 0$
else
  $Y \leftarrow X - 10;$
  assert $Y \neq 0$

the **assert** is tested in the context of each branch
instead of after the control-flow join

the interval domain can prove the right assertion, but not the left one

Abstract domain

Formalization: limited hre to **if** $\cdots$ **then** $\cdots$ **else**

- $\mathcal{L}$ denote syntactic labels of **if** $\cdots$ **then** $\cdots$ **else** instructions
- history abstraction $\mathbb{H} \stackrel{\text{def}}{=} \mathcal{L} \to \{\text{true}, \text{false}, \perp\}$

  $H \in \mathbb{H}$ indicates the outcome of the last time we executed each test:
  - $H(\ell) = \text{true}$: we took the **then** branch
  - $H(\ell) = \text{false}$: we took the **else** branch
  - $H(\ell) = \perp$: we never executed the test

  Notes:

  $\mathbb{H}$ can remember the outcome of several successive tests
  ℓ_1 : **if** $\cdots$ **then** $\cdots$ **else**; ℓ_2 : **if** $\cdots$ **then** $\cdots$ **else**

  for tests in loops, H remembers only the last outcome
  while $\cdots$ **do** ℓ : **if** $\cdots$ **then** $\cdots$ **else**

  we could extend $\mathbb{H}$ to longer histories with $\mathbb{H} = (\mathcal{L} \to \{\text{true}, \text{false}, \perp\})^*$
  we could extend $\mathbb{H}$ to track loop iterations with $\mathbb{H} = \mathcal{L} \to \mathbb{N}$

- $\breve{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} \mathbb{H} \to \mathcal{E}^\sharp$

  use a different abstract element for each abstract history

Abstract operators

- $\breve{\mathcal{E}}^\sharp \stackrel{\text{def}}{=} \mathbb{H} \to \mathcal{E}^\sharp$
- $\breve{\gamma}(A^\sharp) = \cup \{\, \gamma(A^\sharp(H)) \mid H \in \mathbb{H} \,\}$
- $\breve{\sqsubseteq}$, $\breve{\cup}^\sharp$, $\breve{\cap}^\sharp$, $\breve{\triangledown}$ are point-wise
- $\breve{S}^\sharp[\![\, V \leftarrow e \,]\!]$ and $\breve{S}^\sharp[\![\, e \leq 0? \,]\!]$ are point-wise
- $\breve{S}^\sharp[\![\, \ell : \textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, A^\sharp$ is more complex
  - we merge all information about ℓ
    $C^\sharp = \lambda H. A^\sharp(H[\ell \mapsto \text{true}]) \cup^\sharp A^\sharp(H[\ell \mapsto \text{false}]) \cup^\sharp A^\sharp(H[\ell \mapsto \bot])$
  - we compute the then branch, where $H(\ell) = \text{true}$
    $T'^\sharp = \breve{S}^\sharp[\![\, s_1 \,]\!]\, (\breve{S}^\sharp[\![\, c? \,]\!]\, T^\sharp)$ where
    $T^\sharp = \lambda H. C^\sharp(H)$ if $H(\ell) = \text{true}$, $\bot$ otherwise
  - we compute the else branch, where $H(\ell) = \text{false}$
    $F'^\sharp = \breve{S}^\sharp[\![\, s_2 \,]\!]\, (\breve{S}^\sharp[\![\, \neg c? \,]\!]\, F^\sharp)$ where
    $F^\sharp = \lambda H. C^\sharp(H)$ if $H(\ell) = \text{false}$, $\bot$ otherwise
  - we join both branches: $T'^\sharp \,\breve{\cup}^\sharp\, F'^\sharp$
    the join is exact as $\forall H \in \mathbb{H}$: either $T'^\sharp(H) = \bot$ or $F'^\sharp(H) = \bot$

$\Longrightarrow$ we get a semantic by induction on the syntax of the original program

Complex example

Linear interpolation

$X \leftarrow$ **rand**$(TX[0], TX[N])$;
$I \leftarrow 0$;
while $I < N$ && $X > TX[I+1]$ **do** $I \leftarrow I + 1$;
$Y \leftarrow TY[I] + (X - TX[I]) \times TS[I]$

Concrete semantics: table-based interpolation based on the value of X

- look-up index I in the interpolation table: $TX[I] \leq X \leq TX[I+1]$
- interpolate from value $TY[I]$ when $X = TX[I]$ with slope $TS[I]$

Analysis: in the interval domain

- without partitioning:
  $Y \in [\min\ TY, \max\ TY] + (X - [\min\ TX, \max\ TX]) \times [\min\ TS, \max\ TS]$
- partitioning with respect to the number of loop iterations:
  $Y \in \cup_{I \in [0,N]}\ TY[I] + ([0, TX[I+1] - TX[I]]) \times TS[I]$
  (more precise as it keeps the relation between table indices)

Abstracting arrays

Example

Example: increasing subsequence

$p[0] \leftarrow 0; B[0] \leftarrow A[0];$
$i \leftarrow 1; k \leftarrow 1;$
while $i < N$ **do**
  if $A[i] > B[k-1]$ **then**
    $B[k] \leftarrow A[i];$
    $p[k] \leftarrow i;$
    $k \leftarrow k+1;$
  $i \leftarrow i+1$

Given an array $A[0], \ldots, A[N-1]$
the program computes an increasing sub-array $B[0], \ldots, B[k-1]$
and the index sequence $p[0], \ldots, p[k-1]$

Invariants:

$1 \leq k \leq i \leq N$ $\quad \forall x < k\colon B[x] = A[p[x]]$
$\forall x\colon 0 \leq p[x] < N$ $\quad \forall x < k-1\colon B[x+1] > B[x]$

Overview

- Syntax and concrete semantics
- Non-relational abstract semantics

  e.g., $\forall i: A[i] \leq \textit{constant}$

  - application to interval analysis
- Relational (uniform) abstract semantics

  e.g., $\forall i: A[i] \leq V$

  - expand and fold operations
  - application to polyhedral analysis
- Non-uniform abstraction

  e.g., $\forall i: A[i] \leq i$

Syntax extension

Modified expressions and statements

$$
\begin{array}{lcll}
\mathit{expr} & ::= & V & (\mathit{scalar\ access},\ V \in \mathbb{V}) \\
 & | & A[\mathit{expr}] & (\mathit{array\ access},\ A \in \mathbb{A}) \\
 & | & \ldots & \\
\mathit{stat} & ::= & V \leftarrow \mathit{expr} & (\mathit{scalar\ update},\ V \in \mathbb{V}) \\
 & | & A[\mathit{expr}] \leftarrow \mathit{expr} & (\mathit{array\ update},\ A \in \mathbb{A}) \\
 & | & \ldots &
\end{array}
$$

Our language now has two ways to access the memory

- $\mathbb{V}$: scalar integer variables (as before)
- $\mathbb{A}$: arrays of integer values (new)
  - arrays are indiced by positive integers
  - arrays are unbounded (to simplify, we ignore overflows)

$\Longrightarrow$ an array A is similar to a map $A : \mathbb{N} \to \mathbb{Z}$

Concrete semantics

Concrete environments: $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{P}((\mathbb{V} \cup (\mathbb{A} \times \mathbb{N})) \to \mathbb{Z})$

$\rho \in \mathcal{E}$ assigns an integer value to "memory cells" as follows:

- $\rho(V)$ for every scalar variable $V \in \mathbb{V}$
- $\rho(A, i)$ for every array position $A \in \mathbb{A}$, $i \geq 0$

Concrete semantics:

$$
\begin{array}{lcl}
\mathsf{E}[\![\, V \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\rho(V)\} \\
\mathsf{E}[\![\, A[e] \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, \rho(A, i) \mid i \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\} \\
\mathsf{S}[\![\, V \leftarrow e \,]\!]\, R & \stackrel{\text{def}}{=} & \{\, \rho[V \mapsto v] \mid \rho \in R,\ v \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\} \\
\mathsf{S}[\![\, A[f] \leftarrow e \,]\!]\, R & \stackrel{\text{def}}{=} & \{\, \rho[(A, i) \mapsto v] \mid \rho \in R,\ v \in \mathsf{E}[\![\, e \,]\!]\, \rho,\ i \in \mathsf{E}[\![\, f \,]\!]\, \rho,\ i \geq 0 \,\} \\
\ldots & &
\end{array}
$$

Summarization abstraction

Goal: reuse existing numeric abstract domains

issue: numeric domains only abstract subsets of $\mathbb{Z}^n$, for finite n

solution: reduce $\mathcal{E}$ to maps on finite set of abstract variables

Abstract variables: $\mathbb{V}^\sharp \stackrel{\text{def}}{=} \mathbb{V} \cup \mathbb{A}$

- scalar variables in $\mathbb{V}$ are exactly represented in $\mathbb{V}^\sharp$
- the contents of an array $A \in \mathbb{A}$ is abstracted with a single summary variable A (modeling the contents of the whole array)
- $\mathbb{V}^\sharp$ is finite

Summarization Galois Connection:

$$(\mathcal{P}(\mathcal{E}), \subseteq) \xrightleftharpoons[\alpha_s]{\gamma_s} (\mathcal{P}(\mathbb{V}^\sharp \to \mathbb{Z}), \subseteq)$$

- $\alpha_s(R) \stackrel{\text{def}}{=} \{\, [V \mapsto \rho(V), A \mapsto \rho(A, \iota(A))] \mid \rho \in R,\ \iota \in \mathbb{A} \to \mathbb{N} \,\}$
  (folds all array elements (A, i) into the abstract variable A)
- $\gamma_s(S) \stackrel{\text{def}}{=} \{\, \rho \mid \forall \iota \in \mathbb{A} \to \mathbb{N}\colon [V \mapsto \rho(V), A \mapsto \rho(A, \iota(A))] \in S \,\}$
  (indeed, $\gamma_s(S) = \{\, \rho \mid \alpha_s(\{\rho\}) \subseteq S \,\} = \cup \{\, R \mid \alpha_s(R) \subseteq S \,\}$)

Non-relational abstraction

Reminder: Interval abstraction

- $\mathcal{P}(\mathbb{V}^\sharp \to \mathbb{Z})$ is abstracted into $\mathbb{V}^\sharp \to \mathcal{P}(\mathbb{Z})$ (Cartesian abstraction)
- $\mathcal{P}(\mathbb{Z})$ is abstracted as an interval in $\mathbb{I}$

(Note: the Cartesian and summarization abstractions commute)

Abstract semantics: in $\mathcal{E}^\sharp \stackrel{\text{def}}{=} \mathbb{V}^\sharp \to \mathbb{I}$

- $E^\sharp[\![\, V \,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp(V)$
  $E^\sharp[\![\, A[e] \,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp(A)$ (e is ignored)
- $S^\sharp[\![\, V \leftarrow e \,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp[V \mapsto E^\sharp[\![\, e \,]\!]\, X^\sharp]$
  $S^\sharp[\![\, A[f] \leftarrow e \,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp[A \mapsto X^\sharp(A) \cup^\sharp E^\sharp[\![\, e \,]\!]\, X^\sharp]$
  (f is ignored, we perform a weak update that accumulates values)
- assuming $X^\sharp(V) = X^\sharp(A) = [a, b]$:
  $S^\sharp[\![\, V \leq c \,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp[V \mapsto [a, \min(b, c)]]$ if $a \leq c$, $\perp$ otherwise
  $S^\sharp[\![\, A[e] \leq c \,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp$ if $a \leq c$, $\perp$ otherwise
  (we test for satisfability but do not refine $X^\sharp(A)$; the case $A[e] \leq A[f]$ is similar)
- other operations are unchanged, including $\cap^\sharp$, $\cup^\sharp$, ...

Interval analysis example

Example: increasing subsequence

$p[0] \leftarrow 0$; $B[0] \leftarrow A[0]$;
$i \leftarrow 1$; $k \leftarrow 1$;
while $i < N$ **do**
    if $A[i] > B[k-1]$ **then**
        $B[k] \leftarrow A[i]$;
        $p[k] \leftarrow i$;
        $k \leftarrow k+1$;
    $i \leftarrow i+1$

Analysis result:

Assuming that $N \in [N_\ell, N_h]$, $\forall x$: $A[x] \in [A_\ell, A_h]$, we get:

- $\forall x$: $p[x] \in [0, N_h - 1]$
- $\forall x$: $B[x] \in [\min(0, A_\ell), \max(0, A_h)]$

Variable duplication and fold

Reminders: adding and removing regular variables

$$S[\![\, \textbf{add } V \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho[V \mapsto v] \mid \rho \in R,\ v \in \mathbb{Z} \,\}$$

$$S[\![\, \textbf{del } V \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho_{|\mathrm{dom}(\rho) \setminus \{V\}} \mid \rho \in R \,\}$$

Expanding and folding: model dynamic summarization

$$S[\![\, \textbf{expand } V \to V' \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho[V' \mapsto v] \mid \rho \in R \wedge \rho[V \mapsto v] \in R \,\}$$

$$S[\![\, \textbf{fold } V \hookleftarrow V' \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho \mid \exists v\colon \rho[V' \mapsto v] \in R \vee \rho[V' \mapsto \rho(V), V \mapsto v] \in R \,\}$$

- expand duplicates a variable and its constraints
  ($1 \le V \le X \implies 1 \le V \le X \wedge 1 \le V' \le X$; but $V = V'$ does not hold!)
- fold summarizes V and V' into V
  ($1 \le V \le X \wedge 2 \le V' \le Y \implies 1 \le V \le X \vee 2 \le V \le Y$)
- fold is an abstraction, expand is its associated concretization:

$$\mathcal{P}(\mathbb{V} \to \mathbb{Z}) \underset{S[\![\, \textbf{fold } V \hookleftarrow V' \,]\!]}{\overset{S[\![\, \textbf{expand } V \to V' \,]\!]}{\leftleftarrows\!\!\!\rightrightarrows}} \mathcal{P}((\mathbb{V} \setminus \{V'\}) \to \mathbb{Z})$$

(we have a Galois insertion)

Relational expand and join

Polyhedral abstraction:

- expand can be exactly modeled by copying constraints:

  $$\mathsf{S}^\sharp[\![\, \textbf{expand } V_a \to V_b \,]\!] \{ \sum_i \alpha_{ij} V_i \geq \beta_j \} \stackrel{\text{def}}{=} \{ \sum_i \alpha_{ij} V_i \geq \beta_j \} \cup \{ \sum_{i \neq a} \alpha_{ij} V_i + \alpha_{aj} V_b \geq \beta_j \}$$

- join can be approximated using a weak copy:

  $$\mathsf{S}^\sharp[\![\, \textbf{fold } V \hookleftarrow V' \,]\!] X^\sharp \stackrel{\text{def}}{=} \mathsf{S}^\sharp[\![\, \textbf{del } V' \,]\!] (X^\sharp \cup^\sharp \mathsf{S}^\sharp[\![\, V \leftarrow V' \,]\!] X^\sharp)$$

  (assignment that keeps new and old values, instead of replacing old by new)

  example: $0 \leq V \leq 3 \wedge 10 \leq V' \leq 13 \Longrightarrow 0 \leq V \leq 13$
  which over-approximates $0 \leq V \leq 3 \vee 10 \leq V \leq 13$

- $\mathsf{S}^\sharp[\![\, \textbf{add } V \,]\!]$ keeps the constraint set unchanged

- $\mathsf{S}^\sharp[\![\, \textbf{del } V \,]\!]$ projects out V

Relational array abstraction

Goal: abstract $\mathcal{P}(\mathcal{E})$ using polyhedra over $\mathbb{V}^\sharp \stackrel{\text{def}}{=} \mathbb{V} \cup \mathbb{A}$

Principle: use temporary variables, join and expand

Abstract assignment: $S^\sharp[\![\, A[f] \leftarrow e \,]\!] X^\sharp$

- replace each array expression $A[expr]$ in e with a fresh copy of A
  we get a new expression e' and environment $X_1^\sharp$

  e.g., replace $B[expr]$ in $X^\sharp$, with B' in $X_1^\sharp \stackrel{\text{def}}{=} S^\sharp[\![\, \textbf{expand } B \to B' \,]\!] X^\sharp$

- create a new copy A' of A to hold the result

  $X_2^\sharp \stackrel{\text{def}}{=} S^\sharp[\![\, \textbf{expand } A \to A' \,]\!] X_1^\sharp$

- assign e' into A'

  $X_3^\sharp \stackrel{\text{def}}{=} S^\sharp[\![\, A' \leftarrow e' \,]\!] X_2^\sharp$

- fold A' back into A

  $X_4^\sharp \stackrel{\text{def}}{=} S^\sharp[\![\, \textbf{fold } A \hookleftarrow A' \,]\!] X_3^\sharp$

- remove all fresh copies of arrays:

  $S^\sharp[\![\, \textbf{del } B' \,]\!] X_4^\sharp$

The cases for $S^\sharp[\![\, V \leftarrow e \,]\!]$ and $S^\sharp[\![\, c? \,]\!]$ are similar, and a bit simpler

Polyhedral analysis example

Example: increasing subsequence

$p[0] \leftarrow 0; B[0] \leftarrow A[0];$
$i \leftarrow 1; k \leftarrow 1;$
while $i < N$ **do**
    if $A[i] > B[k-1]$ **then**
        $B[k] \leftarrow A[i];$
        $p[k] \leftarrow i;$
        $k \leftarrow k+1;$
    $i \leftarrow i+1$

Analysis result:

Assuming that $\forall x\colon A[x] \in [A_\ell, A_h]$, we get:

- $\forall x\colon 0 \leq p[x] < N$
  (which is stronger than $\forall k\colon 0 \leq p[k] < N_h$)
- $\forall x\colon B[x] \in [\min(0, A_\ell), \max(0, A_h)]$
  ($B \leq A$ would mean $\forall i, j\colon B[i] \leq A[j]$, which does not hold)

Beyond uniform abstractions

The summarization $\alpha_s : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathbb{V}^\sharp \to \mathbb{Z})$ is uniform:
it forgets relations between array element indices and element values

Non-uniform abstraction example: array segmentation

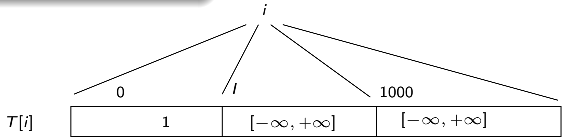
Initialization loop

$I \leftarrow 0$;
while • $I < 1000$ **do**
  $T[I] \leftarrow 1$;
  $I \leftarrow I + 1$

we wish to analyze the loop without unrolling

at • we need to express the loop invariant:
$\forall i < I \colon T[i] = 1$

$\Longrightarrow$ at loop exit, T is initialized until 1000



abstract domain: partition the array contents into uniform segments
segments have constant or symbolic bounds (0, I, 1000,...)
segments have a contents in an abstract domain (intervals,...)

Inter-procedural analyses

Overview

- Syntax and concrete semantics

- Semantic inlining

  simple and precise
  but not efficient and may not terminate

- Call-site and call-stack abstraction

  terminates even for recursive programs
  parametric cost-precision trade-off

- Tabulated abstraction

  optimal reuse of analysis partial results

- Summary-based abstraction

  modular bottom-up analysis
  leverage relational domains

in general, these different abstractions give incomparable results
(there is no clear winner)

Procedures

Syntax:

- $\mathcal{F}$ finite set of procedure names
- $body : \mathcal{F} \to stat$: procedure bodies
- $main \in stat$: entry point body
- $\mathbb{V}_G$: set of global variables
- $\mathbb{V}_f$: set of local variables for procedure $f \in \mathcal{F}$
  procedure f can only access $\mathbb{V}_f \cup \mathbb{V}_G$
  main has no local variable and can only access $\mathbb{V}_G$
- $stat ::= f(expr_1, \ldots, expr_{|\mathbb{V}_f|}) \mid \cdots$

  procedure call, $f \in \mathcal{F}$, setting all its local variables

  local variables double as procedure arguments
  no special mechanism to return a value (a global variable can be used)

Concrete environments

Notes:

- when f calls g, we must remember the value of f's locals $\mathbb{V}_f$ in the semantics of g and restore them when returning
- several copies of each $V \in \mathbb{V}_f$ may exist at a given time
  (due to recursive calls, cycles in the call graph)

$\Longrightarrow$ concrete environments use per-variable stacks

Stacks: $\mathcal{S} \stackrel{\text{def}}{=} \mathbb{Z}^*$ (finite sequences of integers)

- **push**$(v, s) \stackrel{\text{def}}{=} v \cdot s$ $\qquad (v, v' \in \mathbb{Z},\ s, s' \in \mathcal{S})$
- **pop**$(s) \stackrel{\text{def}}{=} s'$ when $\exists v\colon s = v \cdot s'$, undefined otherwise
- **peek**$(s) \stackrel{\text{def}}{=} v$ when $\exists s'\colon s = v \cdot s'$, undefined otherwise
- **set**$(v, s) \stackrel{\text{def}}{=} v \cdot s'$ when $\exists v'\colon s = v' \cdot s'$, undefined otherwise

Environments: $\mathcal{E} \stackrel{\text{def}}{=} (\cup_{f \in \mathcal{F}} \mathbb{V}_f \cup \mathbb{V}_G) \to \mathcal{S}$

for $\mathbb{V}_G$, stacks are not necessary but simplify the presentation

traditionally, there is a single global stack for all local variables
using per-variable stacks instead will make the analysis presentation simpler

Concrete semantics

Concrete semantics: on $\mathcal{E} \stackrel{\text{def}}{=} (\cup_{f \in \mathcal{F}} \mathbb{V}_f \cup \mathbb{V}_G) \to \mathcal{S}$

variable read and update only consider the top of the stack
procedure calls push and pop local variables

- $\mathsf{E}[\![\, V \,]\!]\, \rho \stackrel{\text{def}}{=} \mathbf{peek}(\rho(V))$
- $\mathsf{S}[\![\, V \leftarrow e \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho[V \mapsto \mathbf{set}(x, \rho(V))] \mid \rho \in R, x \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$
- $\mathsf{S}[\![\, f(e_{V_1}, \ldots, e_{V_n}) \,]\!]\, R = R_3$, where:

  $R_1 \stackrel{\text{def}}{=} \{\, \rho[\forall V \in \mathbb{V}_f \colon V \mapsto \mathbf{push}(x_V, \rho(V))] \mid \rho \in R, \forall V \in \mathbb{V}_f \colon x_V \in \mathsf{E}[\![\, e_V \,]\!]\, \rho \,\}$
  (evaluate each argument e_V and push its value x_V on the stack $\rho(V)$)

  $R_2 \stackrel{\text{def}}{=} \mathsf{S}[\![\, body(f) \,]\!]\, R_1$ (evaluate the procedure body)

  $R_3 \stackrel{\text{def}}{=} \{\, \rho[\forall V \in \mathbb{V}_f \colon V \mapsto \mathbf{pop}(\rho(V))] \mid \rho \in R_2 \,\}$ (pop local variables)

- initial environment: $\rho_0 \stackrel{\text{def}}{=} \lambda V \in \mathbb{V}_G . 0$

(other statements are unchanged)

Semantic inlining

Naïve abstract procedure call: mimic the concrete semantics

- assign abstract variables to stack positions:
  $\mathbb{V}^\sharp \stackrel{\text{def}}{=} \mathbb{V}_G \cup (\cup_{f\in\mathcal{F}} \mathbb{V}_f \times \mathbb{N})$
  $\mathbb{V}^\sharp$ is infinite, but each abstract environment uses finitely many variables
- $\mathcal{E}^\sharp_{\mathbb{V}}$ abstracts $\mathcal{P}(\mathbb{V} \to \mathbb{Z})$, for any finite $\mathbb{V} \subseteq \mathbb{V}^\sharp$
  $V \in \mathbb{V}_f$ denotes $(V,0)$ in $\mathbb{V}^\sharp$
  push V: shift variables, replacing (V,i) with $(V,i+1)$, then add $(V,0)$
  pop V: remove $(V,0)$ and shift each (V,i) to $(V,i-1)$
- $\mathsf{S}^\sharp[\![\, f(e_1,\ldots,e_n) \,]\!]\, X^\sharp$ is then reduced to:

$$\begin{array}{ll}
X_1^\sharp = \mathsf{S}^\sharp[\![\, \textbf{push } V_1; \ldots; \textbf{push } V_n \,]\!]\, X^\sharp & \text{(add fresh variables for } \mathbb{V}_f\text{)} \\
X_2^\sharp = \mathsf{S}^\sharp[\![\, V_1 \leftarrow e_1; \ldots; V_n \leftarrow e_n \,]\!]\, X_1^\sharp & \text{(bind arguments to locals)} \\
X_3^\sharp = \mathsf{S}^\sharp[\![\, body(f) \,]\!]\, X_2^\sharp & \text{(execute the procedure body)} \\
X_4^\sharp = \mathsf{S}^\sharp[\![\, \textbf{pop } V_1; \ldots; \textbf{pop } V_n \,]\!]\, X_3^\sharp & \text{(delete local variables)}
\end{array}$$

Limitations:

- does not terminate in case of unbounded recursivity
- requires many abstract variables to represent the stacks
- procedures must be re-analyzed for every call
  (full context-sensitivity: precise but costly)

Example

Example

main :
  $R \leftarrow -1$;
  $f(\textbf{rand}(5, 10))$;
  $f(80)$

$f(X)$:
  $R \leftarrow 2 \times X$;
  if $R > 100$ **then** $R \leftarrow 0$

Analysis using intervals

- after the first call to f, we get $R \in [10, 20]$
- after the second call to f, we get $R = 0$

Call-site abstraction

Abstracting stacks: into a fixed, bounded set $\mathbb{V}^\sharp$ of variables

- $\mathbb{V}^\sharp \stackrel{\text{def}}{=} \cup_{f \in \mathcal{F}} \{ V, \hat{V} \mid V \in \mathbb{V}_f \} \cup \mathbb{V}_G$
  two copies of each local variable
  V abstracts the value at the top of the stack (current call)
  $\hat{V}$ abstracts the rest of the stack
- $\mathsf{S}^\sharp[\![\, \textbf{push } V \,]\!] X^\sharp \stackrel{\text{def}}{=} X^\sharp \cup^\sharp \mathsf{S}^\sharp[\![\, \hat{V} \leftarrow V \,]\!] X^\sharp$
  $\mathsf{S}^\sharp[\![\, \textbf{pop } V \,]\!] X^\sharp \stackrel{\text{def}}{=} X^\sharp \cup^\sharp \mathsf{S}^\sharp[\![\, V \leftarrow \hat{V} \,]\!] X^\sharp$
  weak updates, similar to array manipulation
  no need to create and delete variables dynamically
- assignments and tests always access V, not $\hat{V}$
  $\Longrightarrow$ strong update (precise)

Note: when there is no recursivity, $\hat{V}$, **push** and **pop** can be omitted

Call-site abstraction

Principle: merge all the contexts in which each function is called

- we maintain two global maps $\mathcal{F} \to \mathcal{E}^\sharp$:

  $C^\sharp(f)$: abstracts the environments when calling f
  $R^\sharp(f)$: abstracts the environments when returning from f
  (gather environments from all possible calls to f, disregarding the call sites)

- during the analysis, when encountering a call $\mathsf{S}^\sharp[\![\, body(f) \,]\!] X^\sharp$:
  we return $R^\sharp(f)$
  but we also replace $C^\sharp$ with $C^\sharp[f \mapsto C^\sharp(f) \cup^\sharp X^\sharp]$

- $R^\sharp(f)$ is computed from $C^\sharp(f)$ as

  $R^\sharp(f) = \mathsf{S}^\sharp[\![\, body(f) \,]\!]\,(C^\sharp(f))$

Call-site abstraction

Fixpoint:

there may be circular dependencies between $C^\sharp$ and $R^\sharp$

e.g., in $f(2); f(3)$, the input for $f(3)$ depends on the output from $f(2)$

$\Longrightarrow$ we compute a fixpoint for $C^\sharp$ by iteration:

- initially, $\forall f\colon C^\sharp(f) = R^\sharp(f) = \bot$
- analyze *main*
- while $\exists f\colon C^\sharp(f)$ not stable
  apply widening $\triangledown$ to the iterates of $C^\sharp(f)$
  update $R^\sharp(f) = \mathsf{S}^\sharp[\![\, body(f)\,]\!]\, C^\sharp(f)$
  analyze *main* and all the procedures again
  (this may modify some $C^\sharp(g)$)

$\Longrightarrow$ using $\triangledown$, the analysis always terminates in finite time

we can be more efficient and avoid re-analyzing procedures when not needed
e.g., use a workset algorithm, track procedure dependencies, etc.

Example

Example

$main$:
$R \leftarrow -1$;
$f(\textbf{rand}(5,10))$;
$f(80)$

$f(X)$:
$R \leftarrow 2 \times X$;
if $R > 100$ **then** $R \leftarrow 0$

Analysis: using intervals (without widening as there is no dependency)

- first analysis of *main*: we get $\perp$ (as $R^\sharp(f) = \perp$)
  but $C^\sharp(f) = [R \mapsto [-1,-1], X \mapsto [5,10]]$
- first analysis of f: $R^\sharp(f) = [R \mapsto [10,20], X \mapsto [5,10]]$
- second analysis of *main*: we get
  $C^\sharp(f) = [R \mapsto [-1,20], X \mapsto [5,80]]$
- second analysis of f: $R^\sharp(f) = [R \mapsto [0,100], X \mapsto [5,80]]$
- final analysis of *main*, we find $R \in [0,100]$ at the program end
  (less precise than $R = 0$ found by semantic inlining!)

Partial context-sensitivity

Variants: $k-$limiting, k is a constant

- stack:

  assign a distinct variable for the k highest levels of V
  abstract the lower (unbounded) stack part with $\hat{V}$
  (more precise than keeping only the top of the stack separately)

- context-sensitivity:

  each syntactic call has a unique call-site $\ell \in \mathcal{L}$
  a call stack is a sequence of nested call sites: $c \in \mathcal{L}^*$
  an abstract call stack remembers the last k call sites: $c^\sharp \in \mathcal{L}^k$
  the $C^\sharp$ and $R^\sharp$ maps now distinguish abstract call stacks
  $C^\sharp, R^\sharp : \mathcal{L}^k \to \mathcal{E}^\sharp$
  (more precise than a partitioning by function only)

larger k give more precision but less efficiency

Example: context-sensitivity

Example

main :
  $R \leftarrow -1$;
  $\ell_1 : f(\textbf{rand}(5,10))$;
  $\ell_2 : f(80)$

$f(X)$:
  $R \leftarrow 2 \times X$;
  if $R > 100$ **then** $R \leftarrow 0$

Analysis: using intervals and $k = 1$

- $C^\sharp(\ell_1) = [R \mapsto [-1,1], X \mapsto [5,10]]$
  $\implies R^\sharp(\ell_1) = [R \mapsto [10,20], X \mapsto [5,10]]$
- $C^\sharp(\ell_2) = [R \mapsto [10,20], X \mapsto [80,80]]$
  $\implies R^\sharp(\ell_2) = [R \mapsto [0,0], X \mapsto [80,80]]$
- at the end of the analysis, we get $R = 0$
  (more precise than $R \in [0,100]$ found without context-sensitivity)

Cardinal power

Principle:

the semantic of a function is $\mathsf{S}[\![\, body(f) \,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$

abstract it as an abstract function in $\mathcal{E}^\sharp \rightharpoonup \mathcal{E}^\sharp$

(we use a partial function as the image of most abstract elements is not useful)

Analysis: tabulated analysis

- use a global partial map $F^\sharp : \mathcal{F} \times \mathcal{E}^\sharp \rightharpoonup \mathcal{E}^\sharp$
- $F^\sharp$ is initially empty, and is filled on-demand
- when encountering $\mathsf{S}^\sharp[\![\, body(f) \,]\!]\, X^\sharp$

  return $F^\sharp(f, X^\sharp)$ if defined

  else, compute $\mathsf{S}^\sharp[\![\, body(f) \,]\!]\, X^\sharp$, store it in $F^\sharp(f, X^\sharp)$ and return it

Optimizations: trade precision for efficiency

- if $X^\sharp \sqsubseteq Y^\sharp$ and $F^\sharp(f, X^\sharp)$ is not defined, we can use $F^\sharp(f, Y^\sharp)$ instead
- if the size of $F^\sharp$ grows too large, use $F^\sharp(f, \top)$ instead
  (sound, and ensures that the analysis terminates in finite time)

Example

Example

main :
$R \leftarrow -1;$
$f(\mathbf{rand}(5,10));$
$f(80)$

$f(X)$:
$R \leftarrow 2 \times X;$
if $R > 100$ **then** $R \leftarrow 0$

Analysis using intervals

- $F^\sharp =$
$$[\ (f, [R \mapsto [-1,-1], X \mapsto [5,10]]) \mapsto [R \mapsto [10,20], X \mapsto [5,10]],$$
$$(f, [R \mapsto [10,20], X \mapsto [80,80]]) \mapsto [R \mapsto [0,0], X \mapsto [80,80]]]$$
- at the end of the analysis, we get again $R = 0$

(here, the function partitioning gives the same result as the call-site partitioning)

Dynamic partitioning: complex example

Example: McCarthy's 91 function

main :
  $Mc(\textbf{rand}(0, +\infty))$

$Mc(n)$:
  if $n > 100$ **then** $r \leftarrow n - 10$
  else $Mc(n + 11)$; $Mc(r)$

- in the concrete, when terminating:
  $r = n - 10$ when $n > 101$, and $r = 91$ wen $n \in [0, 101]$
- using a widening $\triangledown$ to choose tabulated abstract values $F^\sharp(f, X^\sharp)$
  we find:

$$\begin{array}{lcl} n \in [0, 72] & \Rightarrow & r = 91 \\ n \in [73, 90] & \Rightarrow & r \in [91, 101] \\ n \in [91, 101] & \Rightarrow & r = 91 \\ n \in [102, 111] & \Rightarrow & r \in [91, 101] \\ n \in [112, +\infty] & \Rightarrow & r \in [91, +\infty] \end{array}$$

(source: Bourdoncle, JFP 1992)

Summary-based analyses

Principle:

- abstract the input-output relation using a relational domain
- analyze each procedure out of context
  no information about its possible arguments
- analyze a procedure given the analysis of the procedures it calls
  bottom-up analysis, from leaf functions to *main*

  $\Longrightarrow$ completely modular analysis
  (for recursive calls, we still need to iterate the analysis of call cycles, with $\triangledown$)

Analysis:

- analyze f with abstract variables $\mathbb{V}_f^\sharp \stackrel{\text{def}}{=} \{\, V, V' \mid V \in \mathbb{V}_G \cup \mathbb{V}_f \,\}$
  V' denotes the current value of the variable
  V denotes the value of the variable at the function entry
- at the beginning of the procedure, start with $\forall V \in \mathbb{V}_G \cup \mathbb{V}_f \colon V = V'$
  the analysis updates only V', never V
  at the end of the procedure, the invariant gives an input-output relation
  it summarizes the effect of the procedure, store it as $T^\sharp(f)$
- $\mathsf{S}^\sharp[\![\, body(f) \,]\!]\, X^\sharp$ can be computed using $T^\sharp(f)$ and variable substitution

$$\mathsf{S}^\sharp[\![\, \forall i \colon \mathbf{del}\ V_i'' \,]\!]\,(X^\sharp[\forall i \colon V_i''/V_i'] \cap^\sharp T^\sharp(f)[\forall i \colon V_i''/V_i])$$

Example

Example

$max(a,b)$:
  if $a > b$ **then** $r \leftarrow a$;
  else $r \leftarrow b$; $c \leftarrow c+1$;

$main$:
  $x \leftarrow [0,10]; y \leftarrow [0,10]$;
  $c \leftarrow 0; max(x,y)$;
  $r \leftarrow r - x$

Analysis using polyhedra

- the analysis of *max* gives:
  $r' \geq a \wedge r' \geq b \wedge c' \geq c \wedge c' \leq c+1 \wedge a = a' \wedge b = b' \wedge x = x' \wedge y = y'$
- at *main*'s call to *max*
  before *max*: $c' = 0 \wedge x' \in [0,10] \wedge y' \in [0,10]$
  applying the summary: $c' \in [0,1] \wedge x' \in [0,10] \wedge y' \in [0,10] \wedge r' \geq x' \wedge r' \geq y'$
  at the end of the program, $x \in [0,10]$, $y \in [0,10]$, $r \in [0,10]$, $c \in [0,1]$

the method requires a relational domain to infer interesting input-output relations
it compensates for the lack of information about the entry point

Backward analysis

Forward versus backward analysis

Example

$Y \leftarrow 0;$
while $Y \leq X$ **do** $Y \leftarrow Y + 1$

Forward analysis:

- given $X \in [-10, 10]$ at the beginning of the program
  $Y \in [0, 11]$ at the end of the program

Backward analysis:

- to have $Y \in [10, 20]$ at the end of the program
  we must have $X \in [9, 19]$ at the beginning of the program

Concrete semantics: forward

$\mathsf{S}[\![\, \mathit{stat} \,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$

$$
\begin{array}{lcl}
\mathsf{S}[\![\, \textbf{skip} \,]\!]\, R & \stackrel{\text{def}}{=} & R \\
\mathsf{S}[\![\, s_1;\ s_2 \,]\!]\, R & \stackrel{\text{def}}{=} & \mathsf{S}[\![\, s_2 \,]\!]\, (\mathsf{S}[\![\, s_1 \,]\!]\, R) \\
\mathsf{S}[\![\, V \leftarrow e \,]\!]\, R & \stackrel{\text{def}}{=} & \{\, \rho[V \mapsto v] \mid \rho \in R,\ v \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\} \\
\mathsf{S}[\![\, c? \,]\!]\, R & \stackrel{\text{def}}{=} & \{\, \rho \in R \mid \mathsf{true} \in \mathsf{C}[\![\, c \,]\!]\, \rho \,\} \\
\mathsf{S}[\![\, \textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, R & \stackrel{\text{def}}{=} & \mathsf{S}[\![\, s_1 \,]\!]\, (\mathsf{S}[\![\, c? \,]\!]\, R) \cup \mathsf{S}[\![\, s_2 \,]\!]\, (\mathsf{S}[\![\, \neg c? \,]\!]\, R) \\
\mathsf{S}[\![\, \textbf{while } c \textbf{ do } s \,]\!]\, R & \stackrel{\text{def}}{=} & \mathsf{S}[\![\, \neg c? \,]\!]\, (\mathsf{lfp}\ \lambda I.R \cup \mathsf{S}[\![\, s \,]\!]\, (\mathsf{S}[\![\, c? \,]\!]\, I))
\end{array}
$$

Concrete semantics: backward

$\overleftarrow{S}[\![\, stat \,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$

$$
\begin{array}{lcl}
\overleftarrow{S}[\![\, \textbf{skip} \,]\!]\, F & \stackrel{\text{def}}{=} & F \\
\overleftarrow{S}[\![\, s_1;\ s_2 \,]\!]\, F & \stackrel{\text{def}}{=} & \overleftarrow{S}[\![\, s_1 \,]\!]\,(\overleftarrow{S}[\![\, s_2 \,]\!]\, F) \\
\overleftarrow{S}[\![\, V \leftarrow e \,]\!]\, F & \stackrel{\text{def}}{=} & \{\, \rho \mid \exists v \in \mathsf{E}[\![\, e \,]\!]\, \rho \colon \rho[V \mapsto v] \in F \,\} \\
\overleftarrow{S}[\![\, c? \,]\!]\, F & \stackrel{\text{def}}{=} & \{\, \rho \in F \mid \text{true} \in \mathsf{C}[\![\, c \,]\!]\, \rho \,\} \\
\overleftarrow{S}[\![\, \textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, F & \stackrel{\text{def}}{=} & \overleftarrow{S}[\![\, c? \,]\!]\,(\overleftarrow{S}[\![\, s_1 \,]\!]\, F) \cup \overleftarrow{S}[\![\, \neg c? \,]\!]\,(\overleftarrow{S}[\![\, s_2 \,]\!]\, F) \\
\overleftarrow{S}[\![\, \textbf{while } c \textbf{ do } s \,]\!]\, F & \stackrel{\text{def}}{=} & \text{lfp}\ \lambda I.\, \overleftarrow{S}[\![\, \neg c? \,]\!]\, F \cup \overleftarrow{S}[\![\, c? \,]\!]\,(\overleftarrow{S}[\![\, s \,]\!]\, I))
\end{array}
$$

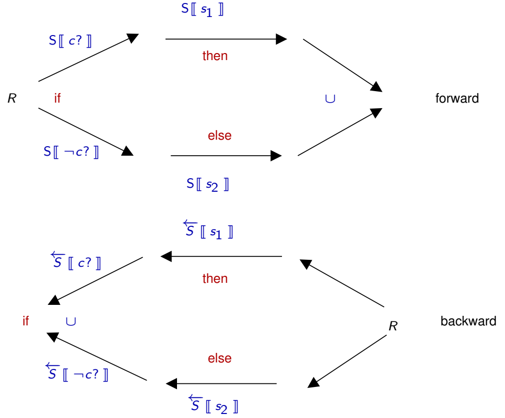
note:

statement order is inverted (s_2 before s_1, s_1 before $c?$, etc.)

$\overleftarrow{S}[\![\, c? \,]\!]$ is unchanged

Concrete semantics: flow intuition

Intuition: information propagation for **if** $\cdots$ **then** $\cdots$ **else**



$$\mathsf{S}[\![\, \textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, R = \mathsf{S}[\![\, s_1 \,]\!]\, (\mathsf{S}[\![\, c? \,]\!]\, R) \cup \mathsf{S}[\![\, s_2 \,]\!]\, (\mathsf{S}[\![\, \neg c? \,]\!]\, R)$$

$$\overleftarrow{S}\,[\![\, \textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, F = \overleftarrow{S}\,[\![\, c? \,]\!]\, (\overleftarrow{S}\,[\![\, s_1 \,]\!]\, F) \cup \overleftarrow{S}\,[\![\, \neg c? \,]\!]\, (\overleftarrow{S}\,[\![\, s_2 \,]\!]\, F)$$

Core property

Executions

- $\mathsf{S}[\![\, stat \,]\!]\, R$

  set of all possible states at the program end
  when starting in a state in R

- $\overleftarrow{S}[\![\, stat \,]\!]\, F$

  set of all the states at the program entry
  such that at least one execution ends in a state in F

Correspondence: $\iota \in \overleftarrow{S}[\![\, stat \,]\!]\{\phi\} \iff \phi \in \mathsf{S}[\![\, stat \,]\!]\{\iota\}$

Note: trace semantics and trace abstractions
the notion of "program execution" can be formalized as trace semantics:
$T \stackrel{\text{def}}{=} \text{lfp}\, \lambda X.I \cup \{\, \langle \rho_1, \ldots, \rho_{n+1} \rangle \mid \langle \rho_1, \ldots, \rho_n \rangle \in X \wedge \rho_n \to \rho_{n+1} \,\}$
$\mathsf{S}[\![\ ]\!]$ and $\overleftarrow{S}[\![\ ]\!]$ are abstractions that only remember the end or beginning of traces
$\mathsf{S}[\![\, stat \,]\!]\{\rho\} \simeq \{\, \rho' \mid \exists \langle \rho_1, \ldots, \rho_n \rangle \in X \wedge \rho_n : \in T,\ \rho = \rho_1,\ \rho' = \rho_n \,\}$
$\overleftarrow{S}[\![\, stat \,]\!]\{\rho'\} \simeq \{\, \rho \mid \exists \langle \rho_1, \ldots, \rho_n \rangle \in X \wedge \rho_n : \in T,\ \rho = \rho_1,\ \rho' = \rho_n \,\}$

Abstraction semantics

Goal: construct $\overleftarrow{S}^\sharp[\![\, stat \,]\!]$ that soundly approximates $\overleftarrow{S}[\![\, stat \,]\!]$

We can define, by induction:

$\overleftarrow{S}^\sharp[\![\, \textbf{skip} \,]\!]\, F^\sharp \stackrel{\text{def}}{=} F^\sharp$

$\overleftarrow{S}^\sharp[\![\, s_1;\ s_2 \,]\!]\, F^\sharp \stackrel{\text{def}}{=} \overleftarrow{S}^\sharp[\![\, s_1 \,]\!]\,(\overleftarrow{S}^\sharp[\![\, s_2 \,]\!]\, F^\sharp)$

$\overleftarrow{S}^\sharp[\![\, c? \,]\!]\, F^\sharp \stackrel{\text{def}}{=} \mathsf{S}^\sharp[\![\, c? \,]\!]\, F^\sharp$

$\overleftarrow{S}^\sharp[\![\, \textbf{if}\ c\ \textbf{then}\ s_1\ \textbf{else}\ s_2 \,]\!]\, F^\sharp \stackrel{\text{def}}{=} \overleftarrow{S}^\sharp[\![\, c? \,]\!]\,(\overleftarrow{S}^\sharp[\![\, s_1 \,]\!]\, F^\sharp) \cup^\sharp \overleftarrow{S}^\sharp[\![\, \neg c? \,]\!]\,(\overleftarrow{S}^\sharp[\![\, s_2 \,]\!]\, F^\sharp)$

$\overleftarrow{S}^\sharp[\![\, \textbf{while}\ c\ \textbf{do}\ s \,]\!]\, F^\sharp \stackrel{\text{def}}{=} \lim \lambda I^\sharp. I^\sharp \mathbin{\triangledown} (\overleftarrow{S}^\sharp[\![\, \neg c? \,]\!]\, F^\sharp \cup^\sharp \overleftarrow{S}^\sharp[\![\, c? \,]\!]\,(\overleftarrow{S}^\sharp[\![\, s \,]\!]\, I^\sharp))$

Abstract operators:

- we can reuse $\cup^\sharp$, $\triangledown$ and $\mathsf{S}^\sharp[\![\, c? \,]\!]$
- only $\mathsf{S}^\sharp[\![\, V \leftarrow e \,]\!]$ needs to be defined on a per-domain basis

Backward assignment

Concrete assignment:

$$\overleftarrow{S}[\![V \leftarrow e]\!] F \stackrel{\text{def}}{=} \{ \rho \mid \exists v \in \mathrm{E}[\![e]\!] \rho \colon \rho[V \mapsto v] \in F \}$$

Abstract assignment examples:

- affine assignment in polyhedra
  $\overleftarrow{S}^\sharp[\![V_j \leftarrow \sum_i \alpha_i V_i + \beta]\!] X^\sharp$
  $\Longrightarrow$ substitute V_j with $\sum_i \alpha_i V_i + \beta$ in each constraint
  (similar to the computation of weakest preconditions $wlp(X \leftarrow e, P) = P[e/X]$)

- intervals
  $\overleftarrow{S}^\sharp[\![V \leftarrow V + \mathbf{rand}(a, b)]\!] X^\sharp = \mathrm{S}^\sharp[\![V \leftarrow V - \mathbf{rand}(a, b)]\!] X^\sharp$
  using substitution is also possible but does not always give interval constraints
  we then need to solve or approximate an optimization problem: min V, max V

- fall-back (e.g., non-affine assignments in polyhedra)
  $\overleftarrow{S}^\sharp[\![V \leftarrow e]\!] X^\sharp \stackrel{\text{def}}{=} \mathrm{S}^\sharp[\![V \leftarrow [-\infty, +\infty]]\!] X^\sharp$
  (same fall-back operation as for forward assignment)

Backward-forward combination

Goal: given initial states I and finial states F
consider only executions that start in I and end in F

Application: analysis specialization to remove false alarms

Example

$X \leftarrow \textbf{rand}(-100, 100);$
$\textbf{if } X = 0 \textbf{ then } X \leftarrow 1;$
• $Y \leftarrow 100/X$

Analysis: using the interval domain

- a forward analysis finds $X \in [-100, 100]$ at •
  $\Longrightarrow$ false alarm for division by zero
- backward analysis from • assuming $X = 0$
  we find $\perp$ at the program entry
  $\Longrightarrow$ no execution can trigger the division by zero
  (we have removed the false alarm)

more complex combinations exist, such as iterated forward and backward analyses

Necessary versus sufficient conditions

Example

$Y \leftarrow 0; I \leftarrow 0;$
$\textbf{while } I \leq X \textbf{ do } Y \leftarrow Y + \textbf{rand}(1,2); I \leftarrow I + 1$
$\textbf{assert}(Y \in [10,30])$

In case of non-determinism, $\overleftarrow{S}[\![\ ]\!] F$ gives the initial states such that at least one execution terminates in F: it is a necessary conditions

We can also consider sufficient conditions
initial states such that all executions terminate in F

Examples: preconditions ensuring the assertion

(strongest) necessary precondition: $X \in [5,30]$
(weakest) sufficient precondition: $X \in [10,15]$

Note:
strongest necessary conditions can be over-approximated
weakest sufficient conditions must be under-approximated
$\Longrightarrow$ leads to very different abstract operations