Abstract Interpretation IV

Semantics and Application to Program Verification

Antoine Miné

École normale supérieure, Paris
year 2014–2015

Course 13
20 May 2015
Selected advanced topics (not for the exam):

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

Practical session:

- finish interval and relational analyses
- help with the project
Disjunctive domains
**Motivation**

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

\[ \bigcup \] causes a loss of precision!

**The need for non-convex invariants**

\[
\begin{align*}
X & \leftarrow \text{rand}(10, 20); \\
Y & \leftarrow \text{rand}(0, 1); \\
\text{if } Y > 0 \text{ then } X & \leftarrow -X; \\
\bullet \quad Z & \leftarrow 100/X
\end{align*}
\]

**Concrete semantics:**

At \( \bullet \), \( X \in [-20, -10] \cup [10, 20] \)

\[ \implies \text{there is no division by zero} \]

**Abstract analysis:**

Convex analyses (intervals, polyhedra) will find \( X \in [-20, 20] \)

(with intervals, \([-20, -10] \cup [10, 20] = [-20, 20])

\[ \implies \text{possible division by zero} \]
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 of each.
**Given:** \((\mathcal{E}^\#, \subseteq, \gamma, \cup^\#, \cap^\#, \nabla, S^\#[\text{stat}])\)

abstract domain \(\mathcal{E}^\#\)

ordered by \(\subseteq\), which also acts as a sound abstraction of \(\subseteq\) (i.e., \(\subseteq^\# = \subseteq\))

with concretization \(\gamma : \mathcal{E}^\# \rightarrow P(\mathcal{E})\)

sound abstractions \(\cup^\#, \cap^\#, S^\#[\text{stat}]\) of \(\cup, \cap, S[\text{stat}]\), and a widening \(\nabla\)

**Construct:** \((\hat{\mathcal{E}}^\#, \hat{\subseteq}, \hat{\gamma}, \hat{\cup}^\#, \hat{\cap}^\#, \hat{\nabla}, \hat{S}^\#[\text{stat}])\)

\[\hat{\mathcal{E}}^\# \overset{\text{def}}{=} P_{\text{finite}}(\mathcal{E}^\#)\] (finite sets of abstract elements)

\[\hat{\gamma}(A^\#) \overset{\text{def}}{=} \cup \{ \gamma(X^\#) \mid X^\# \in A^\# \}\] (join of concretizations)

**Example:** using the interval domain as \(\mathcal{E}^\#\)

\[\hat{\gamma}(\{[-10, -5], [2, 4], [0, 0], [2, 3]\}) = [-10, -5] \cup \{0\} \cup [2, 4]\]
Disjunctive domains

Ordering

**Issue:** How can we compare two elements of $\mathcal{E}^\#$?

- $\mathbf{\hat{\gamma}}$ is generally not injective
  - There is no canonical representation for $\mathbf{\hat{\gamma}}(A^\#)$
- Testing $\mathbf{\hat{\gamma}}(A^\#) = \mathbf{\hat{\gamma}}(B^\#)$ or $\mathbf{\hat{\gamma}}(A^\#) \subseteq \mathbf{\hat{\gamma}}(B^\#)$ is difficult

**Example:** Powerset completion of the interval domain

$A^\# = \{\{0\} \times \{0\}, [0, 1] \times \{1\}\}$

$B^\# = \{\{0\} \times \{0\}, \{0\} \times \{1\}, \{1\} \times \{1\}\}$

$C^\# = \{\{0\} \times [0, 1], [0, 1] \times \{1\}\}$

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

$B^\#$ is more costly to represent: it requires three abstract elements instead of two.

$C^\#$ is a covering and not a partition ($\text{red } \cap \text{ blue } = \{0\} \times \{1\} \neq \emptyset$)
**Solution:** sound approximation of $\subseteq$

$$A^\# \hat{\subseteq} B^\# \overset{\text{def}}{\iff} \forall X^\# \in A^\#: \exists Y^\# \in B^\#: X^\# \subseteq Y^\# \quad (\text{Hoare powerdomain order})$$

- $\hat{\subseteq}$ is a partial order (when $\subseteq$ is)
- $\hat{\subseteq}$ is a sound approximation of $\subseteq$ (when $\subseteq$ is)
  $$A^\# \hat{\subseteq} B^\# \implies \hat{\gamma}(A^\#) \subseteq \hat{\gamma}(B^\#) \text{ but the converse may not hold}$$
- testing $\hat{\subseteq}$ reduces to testing $\subseteq$ finitely many times

**Example:** powerset completion of the interval domain

\[ \hat{\gamma}(A^\#) = \hat{\gamma}(B^\#) = \hat{\gamma}(C^\#) \]
\[ B^\# \hat{\subseteq} A^\# \hat{\subseteq} C^\# \]
Abstract operators

- \( \hat{S}^{\#}[\text{stat}] A^{\#} \overset{\text{def}}{=} \{ S^{\#}[\text{stat}] X^{\#} \mid X^{\#} \in A^{\#} \} \)
  apply \( \text{stat} \) on each abstract element independently

- \( A^{\#} \hat{\cup}^{\#} B^{\#} \overset{\text{def}}{=} A^{\#} \cup B^{\#} \)
  keep elements from both arguments without applying any abstract operation
  \( \hat{\cup}^{\#} \) is exact

- \( A^{\#} \hat{\cap}^{\#} B^{\#} \overset{\text{def}}{=} \{ X^{\#} \cap Y^{\#} \mid X^{\#} \in A^{\#}, Y^{\#} \in B^{\#} \} \)
  \( \hat{\cap}^{\#} \) is exact if \( \cap^{\#} \) 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 \( \hat{E}^{\#} \) of the interval domain \( E^{\#} \)
given the disc \( S \overset{\text{def}}{=} \{ (x, y) \mid x^2 + y^2 \leq 1 \} \)
\( \alpha(S) = [-1, 1] \times [-1, 1] \) (optimal interval abstraction)
but there is no best abstraction in \( \hat{E}^{\#} \)
Dynamic approximation

**Issue:** the size $|A^\#|$ of elements $A^\# \in \hat{\mathcal{E}}^\#$ is unbounded

every application of $\hat{\cup}^\#$ adds some more elements

$\implies$ efficiency and convergence problems

**Solution:** to reduce the size of elements

- redundancy removal

  $\text{simplify}(A^\#) \overset{\text{def}}{=} \{ X^\# \in A^\# \mid \forall Y^\# \neq X^\# \in A^\#: X^\# \nsubseteq Y^\# \}$

  no loss of precision: $\hat{\gamma}(\text{simplify}(A^\#)) = \hat{\gamma}(A^\#)$

- collapse: join elements in $\mathcal{E}^\#$

  $\text{collapse}(A^\#) \overset{\text{def}}{=} \{ \cup^\# \{ X^\# \in A^\# \} \}$

  large loss of precision, but very effective: $|\text{collapse}(A^\#)| = 1$

- partial collapse: limit $|A^\#|$ to a fixed size $k$ by $\cup^\#$

  but how to choose which elements to merge? no easy solution!
Disjunctive domains

Widening

**Issue:** for loops, abstract iterations \((A_n^\#)_{n \in \mathbb{N}}\) may not converge

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

\[\Rightarrow\text{ we need a widening} \nabla\]

Widenings for powerset domains are **difficult to design**

**Example widening:** collapse after a fixed number \(N\) of iterations

\[
A_{n+1}^\# \overset{\text{def}}{=} \begin{cases} 
\text{simplify}(A_n^\# \cup^\# B_{n+1}^\#) & \text{if } n < N \\
\text{collapse}(A_n^\#) \nabla \text{collapse}(B_{n+1}^\#) & \text{otherwise}
\end{cases}
\]

this is very naïve, see Bagnara et al. STTT06 for more interesting widenings
State partitioning
State partitioning

**Principle:**

- partition *a priori* $\mathcal{E}$ into *finitely* many sets
- abstract each partition of $\mathcal{E}$ independently using an element of $\mathcal{E}^\#$

**Abstract domain:**

Given an abstract partition $P^\# \subseteq \mathcal{E}^\#$, i.e., a set such that:

- $P^\#$ is finite
- $\bigcup \{ \gamma(X^\#) \mid X^\# \in P^\# \} = \mathcal{E}$

for generality, we have in fact a covering, not a partitioning of $\mathcal{E}$ i.e., we can have $X^\# \neq Y^\# \in P^\#$ with $\gamma(X^\#) \cap \gamma(Y^\#) \neq \emptyset$

We define $\tilde{\mathcal{E}}^\# \overset{\text{def}}{=} P^\# \rightarrow \mathcal{E}^\#$

representable in memory, as $P^\#$ is finite
Example: \( \mathcal{E}^\# \) is the interval domain

\[
P^\# = \{ P_1, P_2, P_3, P_4, P_5 \} \text{ where} \\
P_1 = [-\infty, 0] \times [-\infty, +\infty] \\
P_2 = [0, 10] \times [0, +\infty] \\
P_3 = [0, 10] \times [-\infty, 0] \\
P_4 = [10, +\infty] \times [0, +\infty] \\
P_5 = [10, +\infty] \times [-\infty, 0]
\]

\[
X^\# = [P_1 \mapsto [-6, -5] \times [5, 6], \\
P_2 \mapsto \perp, \\
P_3 \mapsto [9, 10] \times [-\infty, -1], \\
P_4 \mapsto \perp, \\
P_5 \mapsto [10, 12] \times [-3, -1]]
\]

- \( \tilde{\mathcal{E}}^\# \) def = \( P^\# \rightarrow \mathcal{E}^\# \)
- \( \tilde{\gamma}(A^\#) \) def = \( \bigcup \{ \gamma(A^\#(X^\#)) \cap \gamma(X^\#) \mid X^\# \in P^\# \} \)
- \( A^\# \sqsubseteq B^\# \) def = \( \forall X^\# \in P^\#: A^\#(X^\#) \subseteq B^\#(X^\#) \) (point-wise order)
- \( \tilde{\alpha}(S) \) def = \( \lambda X^\# \in P^\#: \alpha(S \cap \gamma(X^\#)) \)

if \( \mathcal{E}^\# \) enjoys a Galois connection, so does \( \tilde{\mathcal{E}}^\# \)
Abstract operators: point-wise extension from $E^\#$ to $P^\# \rightarrow E^\#$

- $A \hat{\cup}^\# B \overset{\text{def}}{=} \lambda X^\# \in P^\#. A(X^\#) \cup^\# B(X^\#)$
- $A \hat{\cap}^\# B \overset{\text{def}}{=} \lambda X^\# \in P^\#. A(X^\#) \cap^\# B(X^\#)$
- $A \hat{\triangledown}^\# B \overset{\text{def}}{=} \lambda X^\# \in P^\#. A(X^\#) \triangledown B(X^\#)$
- $\tilde{S}^\#[ e \leq 0? ] A^\# \overset{\text{def}}{=} \lambda X^\# \in P^\#. S^\#[ e \leq 0? ] A^\#(X^\#)$
- $\tilde{S}^\#[ V \leftarrow e ] A^\#$ is more complex

any $S^\#[ V \leftarrow e ] A^\#(X^\#)$ may escape its partition $X^\#$; we must cut them at partition borders and glue the pieces falling into the same partition

example: $X \leftarrow X + 2$

\[
\tilde{S}^\#[ V \leftarrow e ] A^\# \overset{\text{def}}{=} \lambda X^\#. \cup^\# \{ X^\# \cap^\# S^\#[ V \leftarrow e ] A(Y^\#) \mid Y^\# \in P^\# \}
\]
Example analysis

Example

\[ X \leftarrow \text{rand}(10, 20); \\
Y \leftarrow \text{rand}(0, 1); \\
\text{if } Y > 0 \text{ then } X \leftarrow -X; \\
\bullet \quad Z \leftarrow 100/X \]

Analysis:

- \( \mathcal{E}^\# \) is the interval domain
- Partition with respect to the sign of \( X \)
  \( P^\# \overset{\text{def}}{=} \{ X^+, X^- \} \) where
  \( X^+ \overset{\text{def}}{=} [0, +\infty) \times \mathbb{Z} \times \mathbb{Z} \) and \( X^- \overset{\text{def}}{=} (-\infty, 0) \times \mathbb{Z} \times \mathbb{Z} \)
- At \( \bullet \) we find:
  \( X^+ \mapsto [X \in [10, 20], Y \mapsto [0, 0], Z \mapsto [0, 0]] \)
  \( X^- \mapsto [X \in [-20, -10], Y \mapsto [1, 1], Z \mapsto [0, 0]] \)
  \( \implies \) no division by zero
Binary decision trees
**Binary decision trees**

**Principle:** data-structure to compactly represent partitions

**Example:** boolean partitions

- Assume that variables have a type: $\forall \overset{\text{def}}{=} \forall_b \cup \forall_n$
  - Each $\forall \in \forall_b$ has value in $\{0, 1\}$ (boolean variable)
  - Each $\forall \in \forall_n$ has value in $\mathbb{Z}$ (numeric variable)

- $\mathcal{E} \sim \{0, 1\}^{\|\forall_b\|} \times \mathbb{Z}^{\|\forall_n\|}$

- $P^\# \overset{\text{def}}{=} \{ \langle b_1, \ldots, b_{\|\forall_b\|} \rangle \times \mathbb{Z}^{\|\forall_n\|} | b_1, \ldots, b_{\|\forall_b\|} \in \{0, 1\} \}$
  A partition corresponds to a precise valuation of all the boolean variables and no information on the numeric variables

- Assume that $\mathcal{E}_n^\#$ abstracts $P(\forall_n \rightarrow \mathbb{Z})$ (numeric domain)

  The boolean partitioning domain based on $\mathcal{E}_n^\#$ is:

  $\tilde{\mathcal{E}}^\# \overset{\text{def}}{=} \{0, 1\}^{\|\forall_b\|} \rightarrow \mathcal{E}_n^\#$
Binary decision trees (cont.)

**Representation:**

for $\mathcal{E}^\# \overset{\text{def}}{=} \{0, 1\}^{\mathcal{V}_b} \rightarrow \mathcal{E}_n^\#$

binary trees:

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

⇒ we get a directed acyclic graph

\[
\begin{align*}
\gamma_n : \mathcal{E}_n^\# &\to \mathbb{Z}^{\lvert \mathcal{V}_n \rvert} \\
\text{is injective and we use memoization} &
\end{align*}
\]

then \(\tilde{\gamma}(A^\#) = \tilde{\gamma}(B^\#) \iff A^\# \text{ and } B^\# \text{ occupy the same address in memory} \)
e.g., == in OCaml, which is faster to test than structural equality =
Abstract operations

- **numeric operations**: performed independently on each leaf
  e.g., \( \tilde{S}^\#[V \leftarrow e] \) reverts to applying \( S^\#[V \leftarrow e] \) on each leaf

- **boolean operations**: manipulate trees
  - \( \tilde{S}^\#[B_i \leftarrow \text{rand}(0, 1)] \): merge \( B_i \)'s subtrees recursively
  - \( \tilde{S}^\#[B_i = 0?] \): set all \( B_i = 1 \) branches to false
  - \( \ldots \)

- **binary operations**: \( \tilde{\cup}^\#, \tilde{\cap}^\#, \tilde{\triangledown}, \tilde{\sqsubseteq} \)
  - 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

\[
X \leftarrow \text{rand}(0, 100);
\]
\[
\text{if } X = 0 \text{ then } B \leftarrow 0 \text{ else } B \leftarrow 1;
\]
\[
\cdots
\]
\[
\bullet \text{ if } B = 1 \text{ then } \bullet \ Y \leftarrow 100/X
\]

Analysis: using the interval domain for \( \mathcal{E}^n \)

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 \text{there is no division by zero}
Disjunctive domains

Binary decision trees

More tree-based partitioning structures

**Other tree-based partitioning data-structure**

we can extend partition trees in many ways

- allow \( n \)-array nodes
  - partition wrt. abstract values in a non-relational domain

  Example: partitioning integer variables in the interval domain

  \[
  V_1 \quad \begin{aligned}
  &{[\neg \infty, -1]} \\
  &{[1, +\infty]}
  \end{aligned}
  \]

  \[
  V_2 \quad \begin{aligned}
  &{[\neg \infty, -1]} \\
  &{[0, +\infty]}
  \end{aligned}
  \]
partitioning with respect to predicates

Example: linear relations over $\forall \overset{\text{def}}{=} \{X, Y, Z\}$

\[
\begin{align*}
X & \leq Y \\
2X & \leq Z \\
2Y & \leq Z
\end{align*}
\]

the same variables may appear in predicates and in the leaves
\[\implies S^{\sharp}[\text{stat}]\] 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 partitioning
Path partitioning

**Principle:** partition wrt. the history of computation

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

**Intuition:** as a program transformation

```
X ← rand(−50, 50);
if X ≥ 0 then
    Y ← X + 10
else
    Y ← X − 10;
assert Y ≠ 0
```

```
X ← rand(−50, 50);
if X ≥ 0 then
    Y ← X + 10;
    assert Y ≠ 0
else
    Y ← X − 10;
    assert Y ≠ 0
```

the **assert** is tested in the context of each branch
instead of after the control-flow join
the interval domain can prove the assertion on the right, but not on the left
Abstract domain

Formalization: we consider here only if · · · then · · · else

- $\mathcal{L}$ denote **syntactic labels** of if · · · then · · · else instructions

- **History abstraction** $\mathcal{H} \overset{\text{def}}{=} \mathcal{L} \to \{\text{true, false, } \perp\}$
  - $H \in \mathcal{H}$ 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:
- $\mathcal{H}$ can remember the outcome of several successive tests
  - $\ell_1 : \text{if } \cdots \text{then } \cdots \text{else}; \ell_2 : \text{if } \cdots \text{then } \cdots \text{else}$
- For tests in loops, $\mathcal{H}$ remembers only the last outcome
  - while · · · do $\ell : \text{if } \cdots \text{then } \cdots \text{else}$
- We could extend $\mathcal{H}$ to longer histories with $\mathcal{H} = (\mathcal{L} \to \{\text{true, false, } \perp\})^*$
- We could extend $\mathcal{H}$ to track loop iterations with $\mathcal{H} = \mathcal{L} \to \mathbb{N}$

- $\mathcal{E}^\#$ $\overset{\text{def}}{=} \mathcal{H} \to \mathcal{E}^\#$
  - use a different abstract element for each abstract history
Abstract operators

- $\tilde{E}^\# \overset{\text{def}}{=} \mathcal{H} \rightarrow \mathcal{E}^\#

- \tilde{\gamma}(A^\#) = \bigcup \{ \gamma(A^\#(H)) \mid H \in \mathcal{H} \}

- \sqsubseteq, \sqcup^\#, \sqcap^\#, \sqtriangledown^\# \text{ are point-wise}

- $\tilde{S}^\#[ V \leftarrow e ]$ and $\tilde{S}^\#[ e \leq 0? ]$ are point-wise

- $\tilde{S}^\#[ \ell : \text{if } c \text{ then } s_1 \text{ else } s_2 ] A^\#$ is more complex
  - we merge all information about $\ell$
    $$C^\# = \lambda H. A^\#(H[\ell \mapsto \text{true}]) \cup^\# A^\#(H[\ell \mapsto \text{false}]) \cup^\# A^\#(H[\ell \mapsto \bot])$$
  - we compute the then branch, where $H(\ell) = \text{true}$
    $$T'^\# = \tilde{S}^\#[ s_1 ] (\tilde{S}^\#[ c? ] T^\#)$$
    $$T^\# = \lambda H. C^\#(H) \text{ if } H(\ell) = \text{true}, \bot \text{ otherwise}$$
  - we compute the else branch, where $H(\ell) = \text{false}$
    $$F'^\# = \tilde{S}^\#[ s_2 ] (\tilde{S}^\#[ \neg c? ] F^\#)$$
    $$F^\# = \lambda H. C^\#(H) \text{ if } H(\ell) = \text{false}, \bot \text{ otherwise}$$
  - we join both branches: $T'^\# \sqcup^\# F'^\#
    \text{ the join is exact as } \forall H \in \mathcal{H}: \text{ either } T'^\#(H) = \bot \text{ or } F'^\#(H) = \bot$

$\implies$ we get a semantic by induction on the syntax of the original program
Complex example

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:
  \[ Y \in \bigcup_{I \in [0,N]} TY[I] + ([0, TX[I+1] - TX[I]] \times TS[I] \]
  more precise as it keeps the relation between table indices

Linear interpolation

\[
X \leftarrow \text{rand}(TX[0], TX[N]);
I \leftarrow 0;
\text{while } I < N \land X > TX[I+1] \text{ do}
I \leftarrow I + 1;
\text{done};
Y \leftarrow TY[I] + (X - TX[I]) \times TS[I]
\]
Inter-procedural analyses
Overview

- **Analysis on the control-flow graph**
  reduce function calls and returns to *gotos*
  useful for the project!

- **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)
Analysis on the control-flow graph
Inter-procedural control-flow graphs

Extend control-flow graphs:

- one subgraph for each function
- additional arcs to denote function calls and returns

We get one big graph without procedures nor calls, only gotos

\[ \implies \text{reduced to a classic analysis based on equation systems} \]

but difficult to use in a denotational-style analysis by induction on the syntax

Note: to simplify, we assume here no local variables and no function arguments:

- locals and arguments are transformed into locals
- only possible if there are no recursive calls

This will be fixed in the following
Example: Control-flow graph

Example

main :
\[ R \leftarrow -1; \]
\[ X \leftarrow \text{rand}(5, 10); f(); \]
\[ X \leftarrow 80; f(); \]

f :
\[ R \leftarrow 2 \times X; \]
\[ \text{if } R > 100 \text{ then } R \leftarrow 0 \]

create one control-flow graph for each function
Example: Control-flow graph

Example

\textbf{main}:
\begin{align*}
R & \leftarrow -1; \\
X & \leftarrow \text{rand}(5, 10); f(); \\
X & \leftarrow 80; f()
\end{align*}

\textbf{f}:
\begin{align*}
R & \leftarrow 2 \times X; \\
\text{if } R > 100 \text{ then } R & \leftarrow 0
\end{align*}

replace call instructions with gotos
Example: Equation system

- each variable $S_i$ denotes a set of environments at a control location $i$
- we can derive an abstract version of the system
  
  
  \[ S_{main,1} = \top \]
  
  \[ S_{main,2} = S[R \leftarrow 1] S_{main,1} \]
  
  \[ S_{main,3} = S[X \leftarrow \text{rand}(5, 10)] S_{main,2} \]
  
  \[ S_{main,4} = S_f,6 \]
  
  \[ S_{main,5} = S[X \leftarrow 80] S_{main,4} \]
  
  \[ S_{main,6} = S_f,6 \]

\[
S_f,1 = S_{main,3} \cup S_{main,5} \\
S_f,2 = S[R \leftarrow 2X] S_f,1 \\
S_f,3 = S[R > 100] S_f,2 \\
S_f,4 = S[R \leftarrow 0] S_f,3 \\
S_f,5 = S[R \leq 100] S_f,2 \\
S_f,6 = S_f,4 \cup S_f,5
\]

- we can solve the abstract system, using widenings to terminate
  
  c.f. project
Example: Equation system

using intervals we get the following solution:

\[
S_{\text{main,1}} = X, R \in \mathbb{Z}
\]
\[
S_{\text{main,2}} = X \in \mathbb{Z}, R = -1
\]
\[
S_{\text{main,3}} = X \in [5, 10], R = -1
\]
\[
S_{\text{main,4}} = X \in [5, 80], R \in [0, 100]
\]
\[
S_{\text{main,5}} = X = 80, R \in [0, 100]
\]
\[
S_{\text{main,6}} = X \in [5, 80], R \in [0, 100]
\]

\[
S_{f,1} = X \in [5, 80], R \in [-1, 100]
\]
\[
S_{f,2} = X \in [5, 80], R \in [10, 160]
\]
\[
S_{f,3} = X \in [5, 80], R \in [101, 160]
\]
\[
S_{f,4} = X \in [5, 80], R \in [0]
\]
\[
S_{f,5} = X \in [5, 80], R \in [10, 100]
\]
\[
S_{f,6} = X \in [5, 80], R \in [0, 100]
\]
Imprecision

In fact, in our example, \( R = 0 \) holds at the end of the program! \( \rightarrow \) the analysis is imprecise

**Explanation:** the control-flow graph adds impossible executions paths
General case: concrete semantics
Procedures

Syntax:

- $\mathcal{F}$ finite set of procedure names
- $\textit{body} : \mathcal{F} \rightarrow \textit{stat}$: procedure bodies
- $\textit{main} \in \textit{stat}$: entry point body
- $\mathcal{V}_G$: set of global variables
- $\mathcal{V}_f$: set of local variables for procedure $f \in \mathcal{F}$
  - procedure $f$ can only access $\mathcal{V}_f \cup \mathcal{V}_G$
  - $\textit{main}$ has no local variable and can only access $\mathcal{V}_G$

- $\textit{stat} ::= f(e_{x_1}, \ldots, e_{|\mathcal{V}_f|}) | \cdots$

  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 \( \forall_f \) in the semantics of \( g \) and restore them when returning

- several copies of each \( V \in \forall_f \) may exist at a given time due to recursive calls, i.e.: cycles in the call graph

\[ \implies \text{concrete environments use per-variable stacks} \]

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

- \( \text{push}(v, s) \overset{\text{def}}{=} v \cdot s \) \((v, v' \in \mathbb{Z}, s, s' \in S)\)

- \( \text{pop}(s) \overset{\text{def}}{=} s' \) when \( \exists v: s = v \cdot s' \), undefined otherwise

- \( \text{peek}(s) \overset{\text{def}}{=} v \) when \( \exists s': s = v \cdot s' \), undefined otherwise

- \( \text{set}(v, s) \overset{\text{def}}{=} v \cdot s' \) when \( \exists v': s = v' \cdot s' \), undefined otherwise

Environments: \( E \overset{\text{def}}{=} (\bigcup_{f \in F} \forall_f \cup \forall_G) \rightarrow S \)

for \( \forall_G \), stacks are not necessary but simplify the presentation

traditionally, there is a single global stack for all local variables using per-variable stacks instead also makes the presentation simpler
Concrete semantics: on $\mathcal{E} \overset{\text{def}}{=} (\bigcup_{f \in \mathcal{F}} \mathcal{V}_f \cup \mathcal{V}_G) \rightarrow \mathcal{S}$

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

- $E[ V ] \rho \overset{\text{def}}{=} \text{peek}(\rho(V))$
- $S[ V \leftarrow e ] R \overset{\text{def}}{=} \{ \rho[V \mapsto \text{set}(x, \rho(V))] | \rho \in R, x \in E[ e ] \rho \} \overset{\text{def}}{=} S[ \text{body}(f) ] R_1$
- $S[ f(e_{v_1}, \ldots, e_{v_n}) ] R = R_3$, where: $R_1 \overset{\text{def}}{=} \{ \rho[\forall V \in \mathcal{V}_f: V \mapsto \text{push}(x_V, \rho(V))] | \rho \in R, \forall V \in \mathcal{V}_f: x_V \in E[ e_V ] \rho \}$ (evaluate each argument $e_V$ and push its value $x_V$ on the stack $\rho(V)$)
- $R_2 \overset{\text{def}}{=} S[ \text{body}(f) ] R_1$ (evaluate the procedure body)
- $R_3 \overset{\text{def}}{=} \{ \rho[\forall V \in \mathcal{V}_f: V \mapsto \text{pop}(\rho(V))] | \rho \in R_2 \}$ (pop local variables)

initial environment: $\rho_0 \overset{\text{def}}{=} \lambda V \in \mathcal{V}_G.0$

other statements are unchanged
Semantic inlining
Semantic inlining

**Naïve abstract procedure call:** mimic the concrete semantics

- assign abstract variables to stack positions:
  \[
  \mathbb{V}^\# \overset{\text{def}}{=} \mathbb{V}_G \cup (\bigcup_{f \in \mathcal{F}} \mathbb{V}_f \times \mathbb{N})
  \]
  \[
  \mathbb{V}^\# \text{ is infinite, but each abstract environment uses finitely many variables}
  \]

- \(\mathcal{E}^\_\mathbb{V}\) abstracts \(\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})\), for any finite \(\mathbb{V} \subseteq \mathbb{V}^\#\)
  \[
  \mathbb{V} \in \mathbb{V}_f \text{ denotes } (\mathbb{V}, 0) \text{ in } \mathbb{V}^\#
  \]
  \[\text{push } \mathbb{V}: \text{ shift variables, replacing } (\mathbb{V}, i) \text{ with } (\mathbb{V}, i + 1), \text{ then add } (\mathbb{V}, 0)\]
  \[\text{pop } \mathbb{V}: \text{ remove } (\mathbb{V}, 0) \text{ and shift each } (\mathbb{V}, i) \text{ to } (\mathbb{V}, i - 1)\]

- \(S^\#[f(e_1, \ldots, e_n)] X^\#\) is then reduced to:
  \[
  X^\#_1 = S^\#[\text{push } \mathbb{V}_1; \ldots; \text{push } \mathbb{V}_n] X^\#
  \]
  \[
  X^\#_2 = S^\#[\mathbb{V}_1 \leftarrow e_1; \ldots; \mathbb{V}_n \leftarrow e_n] X^\#_1
  \]
  \[
  X^\#_3 = S^\#[\text{body}(f)] X^\#_2
  \]
  \[
  X^\#_4 = S^\#[\text{pop } \mathbb{V}_1; \ldots; \text{pop } \mathbb{V}_n] X^\#_3
  \]
  \[\text{(add fresh variables for } \mathbb{V}_f)\]
  \[\text{(bind arguments to locals)}\]
  \[\text{(execute the procedure body)}\]
  \[\text{(delete local variables)}\]

**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

\begin{align*}
\text{main} : & R \leftarrow -1; \\
& f(\text{rand}(5, 10)); \\
& f(80) \\
\text{f(X)} : & R \leftarrow 2 \times X; \\
& \text{if } R > 100 \text{ then } R \leftarrow 0
\end{align*}

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
Call-site abstraction

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

- $\mathbb{V}^\# \overset{\text{def}}{=} \bigcup_{f \in F} \{ \mathbb{V}, \mathbb{\hat{V}} \mid \mathbb{V} \in \mathbb{V}_f \} \cup \mathbb{V}_G$
  - two copies of each local variable
  - $\mathbb{V}$ abstracts the value at the top of the stack (current call)
  - $\mathbb{\hat{V}}$ abstracts the rest of the stack

- $S^\#[\text{push } \mathbb{V}] X^# \overset{\text{def}}{=} X^# \cup S^#[\mathbb{V} \leftarrow \mathbb{\hat{V}}] X^#$
- $S^\#[\text{pop } \mathbb{V}] X^# \overset{\text{def}}{=} X^# \cup S^#[\mathbb{\hat{V}} \leftarrow \mathbb{V}] X^#$
  - weak updates, similar to array manipulation
  - no need to create and delete variables dynamically

- assignments and tests always access $\mathbb{V}$, not $\mathbb{\hat{V}}$
  $\implies$ strong update (precise)

Note: when there is no recursivity, $\mathbb{\hat{V}}$, push and pop can be omitted
**Principle:** merge all the contexts in which each function is called

- we maintain two global maps $\mathcal{F} \rightarrow \mathcal{E}^\#:$
  
  $C^\#(f)$: abstracts the environments when calling $f$
  $R^\#(f)$: 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 $S^\#[body(f)] X^#$:
  
  we return $R^\#(f)$
  but we also replace $C^#$ with $C^# [f \mapsto C^#(f) \cup^# X^#]$ 

- $R^\#(f)$ is computed from $C^#(f)$ as 
  
  $R^#(f) = S^\#[body(f)] (C^#(f))$
Call-site abstraction

**Fixpoint:**
there may be circular dependencies between $C^\#$ and $R^\#$
e.g., in $f(2); f(3)$, the input for $f(3)$ depends on the output from $f(2)$

$\implies$ we compute a fixpoint for $C^\#$ by iteration:

- initially, $\forall f: C^\#(f) = R^\#(f) = \bot$
- analyze *main*
- while $\exists f: C^\#(f)$ not stable
  - apply widening $\nabla$ to the iterates of $C^\#(f)$
  - update $R^\#(f) = S^\#\left[ body(f) \right] C^\#(f)$
  - analyze *main* and all the procedures again
    (this may modify some $C^\#(g)$)

$\implies$ using $\nabla$, the analysis always terminates in finite time

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

**main**: 

\[
\begin{align*}
R &\leftarrow -1; \\
f &\leftarrow \text{rand}(5, 10)); \\
f &\leftarrow (80) 
\end{align*}
\]

\[
\begin{align*}
\text{f(X)} : & \quad R \leftarrow 2 \times X; \\
& \quad \text{if } R > 100 \text{ then } R \leftarrow 0
\end{align*}
\]

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

- first analysis of **main**: we get \( \bot \) (as \( R^\#$\text{(f)} = \bot \))
  but \( C^\#$\text{(f)} = [R \mapsto [-1, -1], X \mapsto [5, 10]] \)

- first analysis of **f**: \( R^\#$\text{(f)} = [R \mapsto [10, 20], X \mapsto [5, 10]] \)

- second analysis of **main**: we get
  \( C^\#$\text{(f)} = [R \mapsto [-1, 20], X \mapsto [5, 80]] \)

- second analysis of **f**: \( R^\#$\text{(f)} = [R \mapsto [0, 100], X \mapsto [5, 80]] \)

- 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: \( c^\# \in \mathcal{L}^k \)
  the \( C^\# \) and \( R^\# \) maps now distinguish abstract call stacks
  \( C^\#, R^\# : \mathcal{L}^k \rightarrow \mathcal{E}^\# \)
  more precise than a partitioning by function only

larger \( k \) give more precision but less efficiency
### Example: context-sensitivity

#### Example

<table>
<thead>
<tr>
<th>main</th>
<th>( f(X) : )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R \leftarrow -1; )</td>
<td>( R \leftarrow 2 \times X; )</td>
</tr>
<tr>
<td>( \ell_1 : f(\text{rand}(5, 10)); )</td>
<td>( \text{if } R &gt; 100 \text{ then } R \leftarrow 0 )</td>
</tr>
<tr>
<td>( \ell_2 : f(80) )</td>
<td></td>
</tr>
</tbody>
</table>

#### Analysis:

using intervals and \( k = 1 \)

- \( C^\#(\ell_1) = \left[ R \mapsto [-1, 1], X \mapsto [5, 10] \right] \)
  \[ \implies R^\#(\ell_1) = \left[ R \mapsto [10, 20], X \mapsto [5, 10] \right] \]

- \( C^\#(\ell_2) = \left[ R \mapsto [10, 20], X \mapsto [80, 80] \right] \)
  \[ \implies R^\#(\ell_2) = \left[ R \mapsto [0, 0], X \mapsto [80, 80] \right] \]

at the end of the analysis, we get \( R = 0 \)
more precise than \( R \in [0, 100] \) found without context-sensitivity
Tabulation abstraction
Cardinal power

**Principle:**

the semantic of a function is $S[\text{body}(f)] : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$

$\implies$ abstract it as an abstract function in $E^\# \rightarrow E^\#$

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

**Analysis:** tabulated analysis

- use a global partial map $F^\# : \mathcal{F} \times E^\# \rightarrow E^\#$
- $F^\#$ is initially empty, and is filled on-demand
- when encountering $S^\#[\text{body}(f)] X^\#$
  - return $F^\#(f, X^\#)$ if defined
  - else, compute $S^\#[\text{body}(f)] X^\#$, store it in $F^\#(f, X^\#)$ and return it

**Optimizations:** trade precision for efficiency

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

**Example**

```plaintext
main:
R ← −1;
f(rand(5, 10));
f(80)
```

```plaintext
f(X):
R ← 2 × X;
if R > 100 then R ← 0
```

---

**Analysis using intervals**

- \( F^\# = \)
  
  \[
  [ (f, [R ↔ [−1, −1], X ↔ [5, 10]]) \mapsto [R ↔ [10, 20], X ↔ [5, 10]],
  (f, [R ↔ [10, 20], X ↔ [80, 80]]) \mapsto [R ↔ [0, 0], X ↔ [80, 80]] ]
  \]

- 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

\[ \text{main} : \]
\[ Mc(\text{rand}(0, +\infty)) \]

\[ Mc(n) : \]
\[ \text{if } n > 100 \text{ then } r \leftarrow n - 10 \]
\[ \text{else } Mc(n + 11); Mc(r) \]

- 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 \( F^\#(f, X^\#) \)
  we find:
  \[ n \in [0, 72] \quad \Rightarrow \quad r = 91 \]
  \[ n \in [73, 90] \quad \Rightarrow \quad r \in [91, 101] \]
  \[ n \in [91, 101] \quad \Rightarrow \quad r = 91 \]
  \[ n \in [102, 111] \quad \Rightarrow \quad r \in [91, 101] \]
  \[ n \in [112, +\infty] \quad \Rightarrow \quad r \in [91, +\infty] \]

(source: Bourdoncle, JFP 1992)
Summary-based abstraction
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
  - \[ \Rightarrow \text{completely modular analysis} \]
  - for recursive calls, we still need to iterate the analysis of call cycles, with \( \nabla \)

**Analysis:**
- analyze \( f \) with abstract variables \( \forall_f \overset{\text{def}}{=} \{ V, V' | V \in \forall_G \cup \forall_f \} \)
  - \( V' \) 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 \( \forall V \in \forall_G \cup \forall_f : V = V' \)
  - 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 \( T^\#(f) \)
- \( S^\#[body(f)] X^\# \) can be computed using \( T^\#(f) \) and variable substitution
  \[ S^\#[\forall i: \text{del} \ V_i'''] (X^\#[\forall i: V_i''' / V_i'] \cap^\# T^\#(f)[\forall i: V_i''' / V_i]) \]
Example

```
max(a, b) :
  if a > b then r ← a;
  else r ← b;  c ← c + 1;

main :
  x ← [0, 10];  y ← [0, 10];
  c ← 0;  max(x, y);
  r ← r - x
```

Analysis using polyhedra

- the analysis of `max` gives:
  \[ r' \geq a \land r' \geq b \land c' \geq c \land c' \leq c + 1 \land a = a' \land b = b' \land x = x' \land y = y' \]

- at `main`'s call to `max`
  before `max`: \[ c' = 0 \land x' \in [0, 10] \land y' \in [0, 10] \]
  applying the summary: \[ c' \in [0, 1] \land x' \in [0, 10] \land y' \in [0, 10] \land r' \geq x' \land r' \geq y' \]
  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
Abstracting arrays
Example: increasing subsequence

\begin{align*}
p[0] & \leftarrow 0; B[0] \leftarrow A[0]; \\
i & \leftarrow 1; k & \leftarrow 1; \\
\textbf{while} & \ i < N \ \textbf{do} \\
& \quad \textbf{if} \ A[i] > B[k - 1] \ \textbf{then} \\
& \quad \quad B[k] \leftarrow A[i]; \\
& \quad \quad p[k] \leftarrow i; \\
& \quad \quad k \leftarrow k + 1; \\
& \quad i \leftarrow i + 1
\end{align*}

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 \( p[0], \ldots, p[k - 1] \)

\textbf{Invariants:}
\begin{align*}
1 & \leq k \leq i \leq N \\
\forall x & : 0 \leq p[x] < N \\
\forall x & : B[x] = A[p[x]] \\
\forall x & : k - 1 < B[x + 1] > B[x]
\end{align*}
Abstracting arrays

Overview

- **Syntax and concrete semantics**

- **Non-relational** abstract semantics
  - e.g., $\forall i: A[i] \leq \text{constant}$
  - 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

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 indexed by positive integers
  - arrays are unbounded (to simplify, we ignore overflows)

\( \mathbb{A} \) is similar to a map \( \mathbb{A} : \mathbb{N} \rightarrow \mathbb{Z} \)
Concrete semantics:  \( \mathcal{E} \overset{\text{def}}{=} (\mathbb{V} \cup (\mathbb{A} \times \mathbb{N})) \rightarrow \mathbb{Z} \)

\( \rho \in \mathcal{E} \) 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:

\[
\begin{align*}
E[V] \rho & \overset{\text{def}}{=} \{ \rho(V) \} \\
E[A[e]] \rho & \overset{\text{def}}{=} \{ \rho(A, i) \mid i \in E[e] \rho \} \\
S[V \leftarrow e] R & \overset{\text{def}}{=} \{ \rho[V \mapsto v] \mid \rho \in R, \ v \in E[e] \rho \} \\
S[A[f] \leftarrow e] R & \overset{\text{def}}{=} \{ \rho[(A, i) \mapsto v] \mid \rho \in R, \ v \in E[e] \rho, \ i \in E[f] \rho, \ i \geq 0 \}
\end{align*}
\]

\[
\cdots
\]
Non-relational abstractions
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: $\mathbb{V}^\# \overset{\text{def}}{=} \mathbb{V} \cup \mathbb{A}$

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

Summarization Galois Connection:

\[(\mathcal{P}(\mathcal{E}), \subseteq) \xrightleftharpoons[\gamma_s]{\alpha_s} (\mathcal{P}(\mathbb{V}^\# \rightarrow \mathbb{Z}), \subseteq)\]

- $\alpha_s(R) \overset{\text{def}}{=} \{ [V \mapsto \rho(V), A \mapsto \rho(A, \iota(A))] | \rho \in R, \iota \in \mathbb{A} \rightarrow \mathbb{N} \}$
  (folds all array elements $(A, i)$ into the abstract variable $A$)
- $\gamma_s(S) \overset{\text{def}}{=} \{ \rho | \forall \iota \in \mathbb{A} \rightarrow \mathbb{N}: [V \mapsto \rho(V), A \mapsto \rho(A, \iota(A))] \in S \}$
  (indeed, $\gamma_s(S) = \{ \rho | \alpha_s(\{\rho\}) \subseteq S \} = \bigcup \{ R | \alpha_s(R) \subseteq S \}$)
Non-relational abstraction

Reminder: Interval abstraction

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

(Note: the Cartesian and summarization abstractions commute)

Abstract semantics:

\[
\begin{align*}
\text{in } E^\# & \overset{\text{def}}{=} \forall^\# \to \mathbb{I} \\
E^\#[A[e]] X^\# & \overset{\text{def}}{=} X^\#(A) \\
S^\#[V \leftarrow e] X^\# & \overset{\text{def}}{=} X^#[V \mapsto E^\#[e] X^#] \\
S^#[A[f] \leftarrow e] X^# & \overset{\text{def}}{=} X^#[A \mapsto X^#(A) \cup E^\#[e] X^#] \\
\end{align*}
\]

\( e \) is ignored, we perform a weak update that accumulates values

\[
\begin{align*}
S^\#[V \leq c] X^\# & \overset{\text{def}}{=} X^#[V \mapsto [a, \min(b, c)]] \text{ if } a \leq c, \bot \text{ otherwise} \\
S^#[A[e] \leq c] X^# & \overset{\text{def}}{=} X^# \text{ if } a \leq c, \bot \text{ otherwise} \\
\end{align*}
\]

we test for satisfiability but do not refine \( X^#(A) \); the case \( A[e] \leq A[f] \) is similar

- other operations are unchanged, including \( \cap^\#, \cup^\#, \ldots \)
Abstracting arrays

Non-relational abstractions

Interval analysis example

**Example:** increasing subsequence

\[
\begin{align*}
p[0] &\leftarrow 0; B[0] \leftarrow A[0]; \\
i &\leftarrow 1; k &\leftarrow 1; \\
\textbf{while } i < N \textbf{ do} \\
&\quad \textbf{if } A[i] > B[k - 1] \textbf{ then} \\
&\quad \quad B[k] \leftarrow A[i]; \\
&\quad \quad p[k] \leftarrow i; \\
&\quad \quad k \leftarrow k + 1; \\
&\quad i \leftarrow i + 1
\end{align*}
\]

Analysis result:
Assuming that \( N \in [N_\ell, N_h] \), \( \forall x: A[x] \in [A_\ell, A_h] \), we get:

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

Variable addition and removal

**Concrete semantics:**
The set $\mathbb{V}$ of variables is not always fixed during program execution:
e.g., local variables

Now $\mathcal{E} \overset{\text{def}}{=} \bigcup_{\mathbb{V} \text{ finite}} \mathbb{V} \rightarrow \mathbb{Z}$

- $S[\text{add } \mathbb{V}] R \overset{\text{def}}{=} \{ \rho[\mathbb{V} \mapsto v] \mid \rho \in R, \, v \in \mathbb{Z} \}$
  add an uninitialized variable

- $S[\text{del } \mathbb{V}] R \overset{\text{def}}{=} \{ \rho |_{\text{dom}(\rho) \setminus \{\mathbb{V}\}} \mid \rho \in R \}$
  remove a variable

**Abstract semantics:**
$\mathcal{E}^\# \overset{\text{def}}{=} \bigcup_{\mathbb{V} \text{ finite}} \mathcal{E}^\#_{\mathbb{V}}$
one abstract $|\mathbb{V}|$-dimensional abstract domain for each $\mathbb{V}$, e.g.: $\mathcal{E}^\#_{\mathbb{V}} = \text{polyhedra of } \mathbb{R}^{|\mathbb{V}|}$

Example, in the interval domain:

- $S^\#[\text{add } \mathbb{V}] X^\# \overset{\text{def}}{=} X^#[\mathbb{V} \mapsto [\infty, +\infty]]$

- $S^\#[\text{del } \mathbb{V}] X^\# \overset{\text{def}}{=} X^#|_{\text{dom}(X^\#) \setminus \{\mathbb{V}\}}$
Variable duplication and fold

Expanding and folding: model dynamic summarization

\[ S[\text{expand } \mathbf{V} \rightarrow \mathbf{V}' ] R \overset{\text{def}}{=} \{ \rho[\mathbf{V}' \mapsto v] \mid \rho \in R \land \rho[\mathbf{V} \mapsto v] \in R \} \]

\[ S[\text{fold } \mathbf{V} \leftarrow \mathbf{V}' ] R \overset{\text{def}}{=} \{ \rho \mid \exists v : \rho[\mathbf{V}' \mapsto v] \in R \lor \rho[\mathbf{V}' \mapsto \rho(\mathbf{V}), \mathbf{V} \mapsto v] \in R \} \]

- **expand** duplicates a variable and its constraints
  
  \( 1 \leq \mathbf{V} \leq X \implies 1 \leq \mathbf{V} \leq X \land 1 \leq \mathbf{V}' \leq X \); but \( \mathbf{V} = \mathbf{V}' \) does not hold!

- **fold** summarizes \( \mathbf{V} \) and \( \mathbf{V}' \) into \( \mathbf{V} \)
  
  \( 1 \leq \mathbf{V} \leq X \land 2 \leq \mathbf{V}' \leq Y \implies 1 \leq \mathbf{V} \leq X \lor 2 \leq \mathbf{V} \leq Y \)

- **fold** is an abstraction, **expand** is its associated concretization:

\[ \mathcal{P}(\mathbf{V} \rightarrow \mathbb{Z}) \leftarrow S[\text{expand } \mathbf{V} \rightarrow \mathbf{V}'] \overset{\text{def}}{\longrightarrow} \mathcal{P}(\mathbf{V} \setminus \{ \mathbf{V}' \}) \rightarrow \mathbb{Z}) \]

we have a Galois insertion
Relational expand and join

Polyhedral abstraction:

- **expand** can be exactly modeled by copying constraints:
  \[ S^\#[\text{expand } V_a \rightarrow V_b] \{ \sum_i \alpha_{ij} V_i \geq \beta_j \} \overset{\text{def}}{=} \{ \sum_i \alpha_{ij} V_i \geq \beta_j \} \cup \{ \sum_{i \neq a} \alpha_{ij} V_i + \alpha_{aj} V_b \geq \beta_j \} \]

- **join** can be approximated using a weak copy:
  \[ S^\#[\text{fold } V \leftarrow V'] X^\# \overset{\text{def}}{=} S^\#[\text{del } V'] (X^\# \cup^\# S^\#[ V \leftarrow V'] X^\#) \]

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

  **example:** \( 0 \leq V \leq 3 \land 10 \leq V' \leq 13 \implies 0 \leq V \leq 13 \)

  which over-approximates \( 0 \leq V \leq 3 \lor 10 \leq V \leq 13 \)

- **add** \( V \) keeps the constraint set unchanged
- **del** \( V \) projects out \( V \)
Relational array abstraction

**Goal:** abstract $\mathcal{P}(\mathcal{E})$ using polyhedra over $\mathcal{V}^\# \overset{\text{def}}{=} \mathcal{V} \cup \mathcal{A}$

**Principle:** use temporary variables, join and expand

**Abstract assignment:** $S^\#[A[f] \leftarrow e] X^\#$

- replace each array expression $A[\text{expr}]$ in $e$ with a fresh copy of $A$
  - we get a new expression $e'$ and environment $X_1^\#$
    - e.g., replace $B[\text{expr}]$ in $X^\#$, with $B'$ in $X_1^\#$ $\overset{\text{def}}{=} S^\#[\text{expand } B \rightarrow B'] X^\#$

- create a new copy $A'$ of $A$ to hold the result
  - $X_2^\# \overset{\text{def}}{=} S^\#[\text{expand } A \rightarrow A'] X_1^\#$

- assign $e'$ into $A'$
  - $X_3^\# \overset{\text{def}}{=} S^\#[A' \leftarrow e'] X_2^\#$

- fold $A'$ back into $A$
  - $X_4^\# \overset{\text{def}}{=} S^\#[\text{fold } A \leftrightarrow A'] X_3^\#$

- remove all fresh copies of arrays:
  - $S^\#[\text{del } B'] X_4^\#$

The cases for $S^\#[V \leftarrow e]$ and $S^\#[c?]$ are similar, and a bit simpler
Polyhedral analysis example

Example: increasing subsequence

\[
p[0] \leftarrow 0; B[0] \leftarrow A[0]; \\
i \leftarrow 1; k \leftarrow 1; \\
\textbf{while } i < N \textbf{ do} \\
\quad \textbf{if } A[i] > B[k - 1] \textbf{ then} \\
\qquad B[k] \leftarrow A[i]; \\
\qquad p[k] \leftarrow i; \\
\qquad k \leftarrow k + 1; \\
\quad i \leftarrow i + 1
\]

Analysis result:

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

- \( \forall x: 0 \leq p[x] < N \)
  which is stronger than \( \forall k: 0 \leq p[k] < N_h \)

- \( \forall x: B[x] \in [\min(0, A_\ell), \max(0, A_h)] \)
  \( B \leq A \) would mean \( \forall i, j: B[i] \leq A[j], \) which does not hold
Non-uniform abstractions
Beyond uniform abstractions

The summarization \( \alpha_s : \mathcal{P}(E) \to \mathcal{P}(\forall \# \to \mathbb{Z}) \) is uniform: it forgets relations between array element indices and element values.

**Non-uniform abstraction example:** array segmentation

**Initialization loop**

\[
\begin{align*}
I & \leftarrow 0; \\
\textbf{while} \bullet I < 1000 & \textbf{do} \\
T[I] & \leftarrow 1; \\
I & \leftarrow I + 1
\end{align*}
\]

we wish to analyze the loop without unrolling

at \( \bullet \) we need to express the loop invariant:

\[
\forall i < I : T[i] = 1
\]

\( \implies \) at loop exit, \( T \) is initialized until 1000

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