Abstract Interpretation IV
Semantics and Application to Program Verification

Antoine Miné

École normale supérieure, Paris
year 2015–2016

Course 13
27 May 2016
Overview

Selected advanced topics:

- reduced products of abstract domains
- disjunctive abstract domains
- inter-procedural analysis

Practical session:

- implement a reduced product
- help with the project
Reduced products
Idea

Theory:
- the set of abstract domains is a lattice,
- ordered by abstraction, which is a partial order, i.e.:
  \[(C, \leq) \xleftarrow{\gamma} (A, \sqsubseteq)\] denotes that \(C\) is more concrete than \(A\).
  (every property of \(A\) can also be represented exactly in \(C\))
- there is a least upper bound \(\sqcup\) for arbitrary sets of domains and a greatest lower bound \(\sqcap\).

Application: reduced product
Effective construction for the least upper bound \(A_1 \sqcap A_2\), able to represent properties expressible in either \(A_1\) or \(A_2\)

Benefit
We can design more precise analyses by combining existing abstractions
Abstract domain lattice
Reduced products

Abstract domain lattice

Reminder: interval abstraction

\[ \mathcal{P}(\mathbb{Z}) \quad \overset{\alpha_i}{\leftrightarrow} \quad \{ [a, b] \mid a \leq b \} \cup \{ \bot \} \]

- \( \alpha_i(S) \overset{\text{def}}{=} [\min S, \max S] \)
- \( \gamma_i([a, b]) \overset{\text{def}}{=} \{ x \in \mathbb{Z} \mid a \leq x \leq b \} \)
Reminder: sign abstraction

\[ \mathcal{P}(\mathbb{Z}) \]

\[ \gamma_s(\bot) \overset{\text{def}}{=} \emptyset \]
\[ \gamma_s(0) \overset{\text{def}}{=} \{0\} \]
\[ \gamma_s(\geq 0) \overset{\text{def}}{=} \mathbb{N} \]
\[ \gamma_s(\leq 0) \overset{\text{def}}{=} -\mathbb{N} \]
\[ \gamma_s(\top) \overset{\text{def}}{=} \mathbb{Z} \]

\[ \alpha_s(S) \overset{\text{def}}{=} \begin{cases} 
\bot & \text{if } S = \emptyset \\
0 & \text{if } S = \{0\} \\
\geq 0 & \text{else, if } \forall s \in S, s \geq 0 \\
\leq 0 & \text{else, if } \forall s \in S, s \leq 0 \\
\top & \text{otherwise}
\end{cases} \]
Composing abstractions

\[ \mathcal{P}(\mathbb{Z}) \xleftrightarrow{\alpha_i, \gamma_i} \{ [a, b] \mid a \leq b \} \cup \{ \bot \} \xleftrightarrow{\alpha'_s, \gamma'_s} \{ \bot, 0, \leq 0, \geq 0, \top \} \]

where:
- \( \gamma'_s(\bot) \overset{\text{def}}{=} \bot \)
- \( \gamma'_s(\top) \overset{\text{def}}{=} [-\infty, +\infty] \)
- \( \gamma'_s(\geq 0) \overset{\text{def}}{=} [0, +\infty] \)
- \( \gamma'_s(\leq 0) \overset{\text{def}}{=} [-\infty, 0] \)
- \( \gamma'_s(0) \overset{\text{def}}{=} [0, 0] \)

We can compose Galois connections:

If \((X_1, \sqsubseteq_1) \xleftrightarrow{\gamma_1, \alpha_1} (X_2, \sqsubseteq_2) \xleftrightarrow{\gamma_2, \alpha_2} (X_3, \sqsubseteq_3)\), then

\((X_1, \sqsubseteq_1) \xleftrightarrow{\alpha_2 \circ \alpha_1} (X_3, \sqsubseteq_3)\).

Proof: \((\alpha_2 \circ \alpha_1)(c) \sqsubseteq_3 a \iff \alpha_1(c) \sqsubseteq_2 \gamma_2(a) \iff c \sqsubseteq_1 (\gamma_1 \circ \gamma_2)(a)\)
**Solution:** sound approximation of $\subseteq$

$A^\# \subseteq B^\# \iff \forall X^\# \in A^\#: \exists Y^\# \in B^\#: X^\# \subseteq Y^\#$ (Hoare powerdomain order)

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

Abstract operators

- $\hat{S}^\# \left[ \text{stat} \right] A^\# \overset{\text{def}}{=} \{ S^\# \left[ \text{stat} \right] X^\# \mid X^\# \in A^\# \}$
  
  apply 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^\#

\begin{align*}
\text{given the disc } S & \overset{\text{def}}{=} \{ (x, y) \mid x^2 + y^2 \leq 1 \} \\
\alpha(S) & = [-1, 1] \times [-1, 1] \quad (\text{optimal interval abstraction}) \\
\text{but there is no best abstraction in } \hat{E}^#
\end{align*}
**Dynamic approximation**

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

  \[ \text{simplify}(A^\#) \overset{\text{def}}{=} \{ X^\# \in A^\# \mid \forall Y^\# \neq X^\# \in A^\#: X^\# \not\subseteq Y^\# \} \]

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

- collapse: join elements in $E^\#$

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

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

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

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

Powerset completion

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 \(E^\#\) has infinite increasing sequences

\(\implies\) we need a widening \(\hat{\Diamond}\)

Widenings for powerset domains are difficult to design

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

\[A_{n+1}^\# \equiv A_n^\# \hat{\Diamond} B_n^\# \equiv \begin{cases} \text{simplify}(A_n^\# \hat{\Diamond} B_n^\#) & \text{if } n < N \\ \text{collapse}(A_n^\#) \hat{\Diamond} \text{collapse}(B_n^\#) & \text{otherwise} \end{cases} \]

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

Principle:
- partition \textit{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
Ordering

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

$P^\# = \{P_1, P_2, P_3, P_4, P_5\}$ where

- $P_1 = \mathbb{R}_{\leq 0} \times \mathbb{R}_{+\infty}$
- $P_2 = [0, 10] \times [0, +\infty]$ 
- $P_3 = [0, 10] \times (-\infty, 0]$ 
- $P_4 = [10, +\infty] \times [0, +\infty]$ 
- $P_5 = [10, +\infty] \times (-\infty, 0]$

$X^\# = \{P_1 \mapsto (-6, -5) \times [5, 6], P_2 \mapsto \bot, P_3 \mapsto [9, 10] \times (-\infty, -1], P_4 \mapsto \bot, P_5 \mapsto [10, 12] \times [-3, -1]\}$

- $\tilde{\mathcal{E}}^\# \overset{\text{def}}{=} P^\# \to \mathcal{E}^\#$
- $\tilde{\gamma}(A^\#) \overset{\text{def}}{=} \bigcup \{ \gamma(A^\#(X^\#)) \cap \gamma(X^\#) | X^\# \in P^\# \}$
- $A^\# \subseteq B^\# \overset{\text{def}}{\iff} \forall X^\# \in P^\#: A^\#(X^\#) \subseteq B^\#(X^\#)$ (point-wise order)
- $\tilde{\alpha}(S) \overset{\text{def}}{=} \lambda X^\# \in P^\#. \alpha(S \cap \gamma(X^\#))$

if $\mathcal{E}^\#$ enjoys a Galois connection, so does $\tilde{\mathcal{E}}^\#$
Abstract operators

Abstract operators: point-wise extension from $E^\#$ to $P^\# \rightarrow E^\#$

- $A \uparrow^\# B \overset{\text{def}}{=} \lambda X^\# \in P^\#.A(X^#) \cup^\# B(X^#)$
- $A \wedge^\# B \overset{\text{def}}{=} \lambda X^\# \in P^#.A(X^#) \cap^\# B(X^#)$
- $A \triangledown^\# B \overset{\text{def}}{=} \lambda X^\# \in P^#.A(X^#) \triangledown B(X^#)$
- $\tilde{S}^\#[ e \leq 0? ] A^# \overset{\text{def}}{=} \lambda X^# \in P^#.S^#[ e \leq 0? ] A^#(X^#)$
- $\tilde{S}^#[ V \leftarrow e ] A^#$ 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 ← rand(10, 20);
Y ← rand(0, 1);
if Y > 0 then X ← −X;
Z ← 100/X

Analysis:

• $\mathcal{E}^\#$ is the interval domain

• partition with respect to the sign of $X$
  \[
  P^\# \overset{\text{def}}{=} \{ X^+, X^- \} \text{ where } \\
  X^+ \overset{\text{def}}{=} [0, +\infty] \times \mathbb{Z} \times \mathbb{Z} \text{ and } X^- \overset{\text{def}}{=} [−\infty, 0] \times \mathbb{Z} \times \mathbb{Z}
  \]

• at • we find:
  \[
  X^+ \mapsto [X \in [10, 20], Y \mapsto [0, 0], Z \mapsto [0, 0]] \\
  X^- \mapsto [X \in [−20, −10], Y \mapsto [1, 1], Z \mapsto [0, 0]]
  \]

⇒ no division by zero
Path partitioning
Disjunctive domains

Path partitioning

Path sensitivity

**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** \( H \overset{\text{def}}{=} \mathcal{L} \rightarrow \{ \text{true}, \text{false}, \bot \} \)

  \( H \in H \) 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
  - \( H(\ell) = \bot \): we never executed the test

**Notes:**
- \( H \) 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, \( H \) remembers only the last outcome
  while · · · do \( \ell : \text{if} \cdots \text{then} \cdots \text{else} \)

- we could extend \( H \) to longer histories with \( H = (\mathcal{L} \rightarrow \{ \text{true}, \text{false}, \bot \})^* \)

- we could extend \( H \) to track loop iterations with \( H = \mathcal{L} \rightarrow \mathbb{N} \)

- \( \check{E}^\# \overset{\text{def}}{=} H \rightarrow E^\# \)

  use a different abstract element for each abstract history
Abstract operators

- $\tilde{\mathcal{E}}^\# \overset{\text{def}}{=} \mathcal{H} \rightarrow \mathcal{E}^\#$
- $\tilde{\gamma}(A^\#) = \bigcup \{ \gamma(A^\#(H)) \mid H \in \mathcal{H} \}$
- $\sqsubseteq, \sqcup^\#, \sqcap^\#, \sqtriangledown$ are point-wise
- $\tilde{\mathcal{S}}^\#[V \leftarrow e]$ and $\tilde{\mathcal{S}}^\#[e \leq 0?]$ are point-wise
- $\tilde{\mathcal{S}}^\#[\ell : \text{if } c \text{ then } s_1 \text{ else } s_2 ] A^\#$ is more complex
  - we merge all information about $\ell$
    $C^\# = \lambda H.A^\#(H[\ell \mapsto \text{true}]) \sqcup^\# A^\#(H[\ell \mapsto \text{false}]) \sqcup^\# A^\#(H[\ell \mapsto \bot])$
  - we compute the then branch, where $H(\ell) = \text{true}$
    $T'^\# = \tilde{\mathcal{S}}^\#[s_1] (\tilde{\mathcal{S}}^\#[c?] T^\#)$ where
    $T^\# = \lambda H.C^\#(H)$ if $H(\ell) = \text{true}$, $\bot$ otherwise
  - we compute the else branch, where $H(\ell) = \text{false}$
    $F'^\# = \tilde{\mathcal{S}}^\#[s_2] (\tilde{\mathcal{S}}^\#[\neg c?] F^\#)$ where
    $F^\# = \lambda H.C^\#(H)$ if $H(\ell) = \text{false}$, $\bot$ otherwise
  - we join both branches: $T'^\# \sqcup^\# F'^\#
  - the join is exact as $\forall H \in \mathcal{H}$: either $T'^\#(H) = \bot$ or $F'^\#(H) = \bot$

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

Linear interpolation

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

**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
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

- We also mentioned summary-based abstractions last week,
  leveraging relational domains for modular bottom-up analysis

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

\[ \Rightarrow \] 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 variable and no function argument:

- locals and arguments are transformed into globals
- only possible if there are no recursive calls
## Example: Control-flow graph

**Example**

### main:

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

### f:

- \( R \leftarrow 2 \times X; \)
- \( \text{if } R > 100 \text{ then } R \leftarrow 0 \)

---

```
main:
R = -1
X = rand(5,10)
f()
X = 80 
f()

f:
R = 2X
R > 100
R <= 100
R = 0
```

---

create one control-flow graph for each function
Inter-procedural analyses

Analysis on the control-flow graph

Example: Control-flow graph

Example

Let

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

Then

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

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
  e.g.: $S_{f,2}^\# = S_{f,1}^\#[ R \leftarrow 2X ]$, $S_{f,6}^\# = S_{f,4}^\# \cup S_{f,5}^\#$, etc.
- we can solve the abstract system, using widenings to terminate
c.f. project
Example: Equation system

using intervals we get the following solution:

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

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

In fact, in our example, \( R = 0 \) holds at the end of the program but we find \( R \in [0, 100]! \)
\[ \Rightarrow \text{the analysis is imprecise} \]

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

Syntax:

- $\mathcal{F}$ finite set of procedure names
- body : $\mathcal{F} \rightarrow \text{stat}$: procedure bodies
- main $\in \text{stat}$: 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 $\mathcal{V}_f \cup \mathcal{V}_G$
  main has no local variable and can only access $\mathcal{V}_G$

- \text{stat} ::= f(\text{expr}_1, \ldots, \text{expr}_{|\mathcal{V}_f|}) | \ldots

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

- \( \text{push}(v, s) \overset{\text{def}}{=} v \cdot s \quad (v, v' \in \mathbb{Z}, s, s' \in S) \)
- \( \text{pop}(s) \overset{\text{def}}{=} s' \) 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:** \[ E \overset{\text{def}}{=} (\bigcup_{f \in \mathcal{F}} \forall_f \cup \forall_G) \rightarrow S \]

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 $E \overset{\text{def}}{=} (\bigcup_{f \in F} V_f \cup V_G) \rightarrow S$
variable reads and updates only consider the top of the stack;
procedure calls push and pop local variables

- $E[V][\rho] \overset{\text{def}}{=} \text{peek}(\rho(V))$
- $S[V \gets e]R \overset{\text{def}}{=} \{ \rho[V \mapsto \text{set}(x, \rho(V))] | \rho \in R, x \in E[e] \rho \}$
- $S[f(e_{V_1}, \ldots, e_{V_n})]R = R_3$, where:
  - $R_1 \overset{\text{def}}{=} \{ \rho[\forall V \in V_f: V \mapsto \text{push}(x_V, \rho(V))] | \rho \in R, \forall V \in V_f: x_V \in E[e_{V}] \rho \}$ (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)
  - $R_3 \overset{\text{def}}{=} \{ \rho[\forall V \in V_f: V \mapsto \text{pop}(\rho(V))] | \rho \in R_2 \}$ (pop local variables)

- initial environment: $\rho_0 \overset{\text{def}}{=} \lambda V \in V_G.0$
other statements are unchanged
Semantic inlining
**Semantic inlining**

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

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

- \(E^*_\mathcal{V}\) abstracts \(\mathcal{P}(\forall \rightarrow \mathbb{Z})\), for any finite \(\forall \subset \forall^*\)

  - \(\mathcal{V} \in \forall_f\) denotes \((\mathcal{V}, 0)\) in \(\forall^*\)
  - **push** \(\mathcal{V}\): shift variables, replacing \((\mathcal{V}, i)\) with \((\mathcal{V}, i + 1)\), then add \((\mathcal{V}, 0)\)
  - **pop** \(\mathcal{V}\): remove \((\mathcal{V}, 0)\) and shift each \((\mathcal{V}, i)\) to \((\mathcal{V}, i - 1)\)

- \(S^*[f(e_1, \ldots, e_n)] X^*\) is then reduced to:
  \[
  X_1^* = S^*[\text{push } \mathcal{V}_1; \ldots; \text{push } \mathcal{V}_n] X^* \quad \text{(add fresh variables for } \forall_f)\
  X_2^* = S^*[\mathcal{V}_1 \leftarrow e_1; \ldots; \mathcal{V}_n \leftarrow e_n] X_1^* \quad \text{(bind arguments to locals)}\
  X_3^* = S^*[\text{body}(f)] X_2^* \quad \text{(execute the procedure body)}\
  X_4^* = S^*[\text{pop } \mathcal{V}_1; \ldots; \text{pop } \mathcal{V}_n] X_3^* \quad \text{(delete local variables)}\
  \]

**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

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

#### $f(X)$:
- $R \leftarrow 2 \times X$
- 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
Call-site abstraction

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

- $\mathcal{V}^\# \overset{\text{def}}{=} \bigcup_{f \in \mathcal{F}} \{ V, \hat{V} \mid V \in \mathcal{V}_f \} \cup \mathcal{V}_G$
  - 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

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

- assignments and tests always access $V$, not $\hat{V}$
  $\implies$ 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 $\mathcal{F} \rightarrow \mathcal{E}^\#$:
  - $C^\#(f)$: 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 $S^\#[\text{body}(f)] X^\#$:
  - 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
  $$R^\#(f) = S^\#[\text{body}(f)] (C^#(f))$$
**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 \( f(2) \)

\[ \implies \text{we compute a fixpoint for } C^\# \text{ by iteration:} \]

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

\[ \implies \text{using } \nabla, \text{ the analysis always terminates in finite time} \]

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

Example

**Example**

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

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

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

- first analysis of `main`: we get ⊥ (as $R^\#(f) = \bot$)
  but $C^\#(f) = [R \mapsto [-1, -1], X \mapsto [5, 10]]$
- first analysis of `f`: $R^\#(f) = [R \mapsto [10, 20], X \mapsto [5, 10]]$
- second analysis of `main`: we get
  $C^\#(f) = [R \mapsto [-1, 20], X \mapsto [5, 80]]$
- second analysis of `f`: $R^\#(f) = [R \mapsto [0, 100], X \mapsto [5, 80]]$
- 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
  \( C^\#, R^\# : \mathcal{L}^k \rightarrow E^\# \)
  more precise than a partitioning by function only

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

Example

\[
\begin{align*}
\text{main} : & \quad R \leftarrow -1; \\
& \quad \ell_1 : f(\text{rand}(5, 10)); \\
& \quad \ell_2 : f(80) \\
\text{f}(X) : & \quad R \leftarrow 2 \times X; \\
& \quad \text{if } R > 100 \text{ then } R \leftarrow 0
\end{align*}
\]

Analysis: using intervals and \( k = 1 \)

\[
\begin{align*}
\mathcal{C}^\#(\ell_1) &= [R \mapsto [-1, 1], X \mapsto [5, 10]] \\
\implies R^\#(\ell_1) &= [R \mapsto [10, 20], X \mapsto [5, 10]] \\
\mathcal{C}^\#(\ell_2) &= [R \mapsto [10, 20], X \mapsto [80, 80]] \\
\implies R^\#(\ell_2) &= [R \mapsto [0, 0], X \mapsto [80, 80]]
\end{align*}
\]

at the end of the analysis, we get \( R = 0 \)

more precise than \( R \in [0, 100] \) found without context-sensitivity
Tabulation abstraction
**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

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

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

Analysis using intervals

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

- 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**

```plaintext
main:
    Mc(rand(0, +\infty))

Mc(n):
    if n > 100 then r ← n - 10
    else Mc(n + 11); Mc(r)
```

- in the concrete, when terminating:
  
  \[ r = n - 10 \] when \( n > 101 \), and \( r = 91 \) when \( n \in [0, 101] \)

- 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)