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
Reduced products

Idea

Theory:

- the set of abstract domains is a lattice,
- ordered by abstraction, which is a partial order, i.e.: \((, \leq) \xrightarrow{\gamma} \alpha (, \sqsubseteq)\) denotes that \(\alpha\) is more concrete than \(\gamma\).
  (every property of \(\alpha\) can also be represented exactly in \(\gamma\))
- 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 \(1 \sqcap 2\), able to represent properties expressible in either \(1\) or \(2\).

Benefit

We can design more precise analyses by combining existing abstractions
Abstract domain lattice
### Reminder: interval abstraction

$$\mathcal{P}(\mathbb{Z}) \xleftrightarrow[\alpha_i \gamma_i]{\implies} \{ [a, \; b] | a \leq b \} \cup \{ \bot \}$$

- $$\alpha_i(\ ) \triangleq [\min, \; \max]$$
- $$\gamma_i([a, \; b]) \triangleq \{ x \in \mathbb{Z} | a \leq x \leq b \}$$
Reminder: sign abstraction

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

\[ \mathcal{P}(\mathbb{Z}) \xleftarrow{\gamma_i} \{ [\ , \ ] | \leq \} \cup \{ \bot \} \xrightarrow{\alpha'_i} \{ \bot, 0, \leq 0, \geq 0, \top \} \]

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

We can compose Galois connections:

If \(( 1, \sqsubseteq_1 ) \xleftarrow{\gamma_1} ( 2, \sqsubseteq_2 ) \xrightarrow{\gamma_2} ( 3, \sqsubseteq_3 )\), then

\(( 1, \sqsubseteq_1 ) \xleftarrow{\gamma_1 \circ \gamma_2} ( 3, \sqsubseteq_3 )\).

Proof: \(( \alpha_2 \circ \alpha_1)(c) \sqsubseteq_3 a \iff \alpha_1(c) \sqsubseteq_2 \gamma_2(a) \iff c \sqsubseteq_1 (\gamma_1 \circ \gamma_2)(a)\)
Parity domain

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

\[ \gamma_p(\bot) \overset{\text{def}}{=} \emptyset \]
\[ \gamma_p(\text{even}) \overset{\text{def}}{=} 2\mathbb{Z} \]
\[ \gamma_p(\text{odd}) \overset{\text{def}}{=} 2\mathbb{Z} + 1 \]
\[ \gamma_p(\top) \overset{\text{def}}{=} \mathbb{Z} \]

\[ \alpha_p(\ ) \overset{\text{def}}{=} \begin{cases} 
\bot & \text{if } = \emptyset \\
\text{else, if } \subseteq 2\mathbb{Z} \\
\text{else, if } \subseteq 2\mathbb{Z} + 1 \\
\top & \text{otherwise}
\end{cases} \]
Part of the abstraction lattice for \( \mathcal{P}(\mathbb{Z}) \)

- signs are more abstract than intervals;
  every sign information can be expressed as an interval

- parities and intervals are incomparable;
  no common property, except \( \bot \) and \( \top \)

- \( \mathcal{P}(\mathbb{Z}) \) is the most concrete domain;

- \( \{ \top \} \) is the most abstract domain;

- intervals \( \land \) parities is the coarsest abstract domain more precise than intervals and parities.
Reduced product construction
Simple product

**Algebraic structure:**

Given two domains \((\mathcal{D}_1^\#, \sqsubseteq_1)\) and \((\mathcal{D}_2^\#, \sqsubseteq_2)\), we use pairs of abstract elements to represent conjunctions of properties.

- \(\mathcal{D}_1^\# \times_2 \overset{\text{def}}{=} \mathcal{D}_1^\# \times \mathcal{D}_2^\#\)
- \(\gamma_1 \times_2 (\ #_1, \ #_2) \overset{\text{def}}{=} \gamma_1(\ #_1) \cap \gamma_2(\ #_2)\)
- \(\alpha_1 \times_2 (\ ) \overset{\text{def}}{=} (\alpha_1(\ ), \alpha_2(\ ))\)
- \((\ #_1, \ #_2) \sqsubseteq_1 \times_2 (\ #_1, \ #_2) \overset{\text{def}}{=} 1 \sqsubseteq_1 1 \text{ and } 2 \sqsubseteq_2 2\)

**Abstract operators in \(\mathcal{D}^\#\):**

Applied in parallel (independently) in each abstract domain:

- \((\ #_1, \ #_2) \cup_1 \times_2 (\ #_1, \ #_2) \overset{\text{def}}{=} (\ #_1 \cup_1 1, \ #_2 \cup_2 2)\)
- \((\ #_1, \ #_2) \triangledown_1 \times_2 (\ #_1, \ #_2) \overset{\text{def}}{=} (\ #_1 \triangledown_1 1, \ #_2 \triangledown_2 2)\)
- \(\mathcal{S}_1 \mathcal{S}_2 1 \times_2 (\ #_1, \ #_2) \overset{\text{def}}{=} (\mathcal{S}_1 \mathcal{S}_2 1(\ #_1), \mathcal{S}_1 \mathcal{S}_2 2(\ #_2))\).
Simple products: limitations

\[\begin{align*}
\leftarrow 1; \\
\text{while } V \leq 10 \text{ do} & \quad \leftarrow + 2 \text{ done} \\
\bullet \text{ if } V \geq 12 \text{ then} \bullet & \quad \leftarrow 0 \bullet
\end{align*}\]

Analysis in the product domain of intervals and parities:

<table>
<thead>
<tr>
<th>intervals</th>
<th>parities</th>
<th>product: intervals \times parities</th>
</tr>
</thead>
<tbody>
<tr>
<td>∈ [11, 12]</td>
<td>odd</td>
<td>( ∈ [11, 12]) ∧ ( odd)</td>
</tr>
<tr>
<td>= 12</td>
<td>odd</td>
<td>( = 12) ∧ ( odd)</td>
</tr>
<tr>
<td>= 0</td>
<td>even</td>
<td>( = 0) ∧ ( even)</td>
</tr>
</tbody>
</table>

Identical to two separate analyses:

- at ●, we get ( = 12) ∧ ( odd), which represents \(\emptyset\);
- at ●, we apply \(\leftarrow 0\) independently on intervals and parities, which gives ( = 0) ∧ ( even), instead of \(\emptyset\)!

\(\Rightarrow\) huge loss of precision
Reduced products

Reduced product construction

Fully reduced product

Idea: propagate information between domains

Given Galois connections \((\alpha_1, \gamma_1)\) and \((\alpha_2, \gamma_2)\) over \(D_1^\#\) and \(D_2^\#\), we define a reduction operator \(\rho\) as:

\[
\rho: D_1^\# \times D_2^\# \rightarrow D_1^\# \times D_2^#
\]

\[
\rho(\#_1, \#_2) \overset{\text{def}}{=} (\alpha_1(\gamma_1(\#_1) \cap \gamma_2(\#_2)), \alpha_2(\gamma_1(\#_1) \cap \gamma_2(\#_2)))
\]

i.e., the best representation of \(\gamma_1 \times \gamma_2(\#_1, \#_2)\) in both domains

Application:

use \(\rho\) to transfer information between domains after abstract operations:

\[
\begin{align*}
(\#_1, \#_2) \cup_{1 \times 2} \#_1 \cup_1 \#_2 & \overset{\text{def}}{=} \rho(\#_1 \cup_1 \#_1, \#_2 \cup_2 \#_2), \\
S^\#[1 \times 2] \#_1 \cup_1 \#_2 & \overset{\text{def}}{=} \rho(S^\#[1 \times 2] \#_1, S^\#[2 \times 2] \#_2).
\end{align*}
\]

Warning:

\(\rho\) should not be used on fixpoint iterates with widening \(\nabla\) \((X_{n+1} \overset{\text{def}}{=} \rho(X_n \nabla F(X_n)))\)

\[\Longrightarrow\] this could prevent the convergence in \(D_1^\# \times D_2^\#\)!
Analysis with reduction: exemple

\begin{verbatim}
\leftarrow 1;
\textbf{while } V \leq 10 \textbf{ do } V \leftarrow V + 2 \textbf{ done;}

• \textbf{if } V \geq 12 \textbf{ then } V \leftarrow 0
\end{verbatim}

Reduction $\rho$ between intervals and parities $\rho([a,b], p)$:
First refine interval bounds $[a,b]$ using parity information $p$, then refine the parity information using the refined bounds:

• let $a' = a + 1$ if $a \notin \gamma_p(\cdot)$, $a' = a$ otherwise;
• let $b' = b - 1$ if $b \notin \gamma_p(\cdot)$, $b' = b$ otherwise;
• if $a' > b'$, return $(\perp, \perp)$;
• if $a' = b'$, return $([a', b'], \alpha_p(\cdot))$;
• otherwise, return $([a', b'], \cdot)$.

Example:

At $\bullet$, $\rho([11, 12], \text{odd}) = ([11, 11], \text{odd})$  
$\implies$ the “then” branch is not reachable.
Partial reduction

The optimal reduction $\rho$ is well-defined but:

- $\rho$ assumes we have Galois connections;
- there is no general effective algorithm to compute $\rho$.

(similar to the case of optimal operators, defined as $F^\# \overset{\text{def}}{=} \alpha \circ F \circ \gamma$)

**Partial reduction:**

Practical definition, when the optimal reduction is not available:

- $\rho(\#_1, \#_2) = (\#_1, \#_2)$ is a partial reduction if:
  - $\#_1 \sqsubseteq_1 \#_1$ and $\#_2 \sqsubseteq_2 \#_2$ (improvement)
  - $\gamma_{1\times2}(\#_1, \#_2) = \gamma_{1\times2}(\#_1, \#_2)$ (soundness)

**Example:**

$$\rho(\#_1, \#_2) \overset{\text{def}}{=} \begin{cases} (\bot_1, \bot_2) & \text{if } \#_1 = \bot_1 \text{ or } \#_2 = \bot_2 \\ (\#_1, \#_2) & \text{otherwise} \end{cases}$$

In practice, an analyzer contains many abstract domains (for expressiveness) with limited reductions between them (for efficiency).
Disjunctive domains
Disjunctive domains

Motivation

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

⇒ ∪♯ causes a loss of precision!

The need for non-convex invariants

```
← rand(10, 20);
← rand(0, 1);
if Y > 0 then ← − ;
• ← 100/
```

Concrete semantics:

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

⇒ there is no division by zero

Abstract analysis:

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

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

⇒ 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

- **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.
Powerset completion


**Powerset completion**

**Given:** \((\mathcal{E}^#, \sqsubseteq, \gamma, \cup^#, \cap^#, \bigtriangledown, S^#[stat])\)

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

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

- \(\hat{\mathcal{E}}^# \overset{\text{def}}{=} \mathcal{P}_{\text{finite}}(\mathcal{E}^#)\) (finite sets of abstract elements)
- \(\hat{\gamma}(\#) \overset{\text{def}}{=} \bigcup \{\gamma(\#) | \# \in \#\}\) (join of concretizations)

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

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

**Issue:** how can we compare two elements of $\hat{\gamma}^\#$?

- $\hat{\gamma}$ is generally not injective
  
  there is no canonical representation for $\hat{\gamma}(A^\#)$

- testing $\hat{\gamma}(\ #) = \hat{\gamma}(\ #)$ or $\hat{\gamma}(\ #) \subseteq \hat{\gamma}(\ #)$ is difficult

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

\[
\begin{align*}
A^\# &= \{\{0\} \times \{0\}, [0, 1] \times \{1\}\} \\
B^\# &= \{\{0\} \times \{0\}, \{0\} \times \{1\}, \{1\} \times \{1\}\} \\
C^\# &= \{\{0\} \times [0, 1], [0, 1] \times \{1\}\}
\end{align*}
\]

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

$B^\#$ 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 $\subseteq$

\[
\hat{\subseteq} \quad \text{def} \quad \forall \# \in \# : \exists \# \in \#: \# \hat{\subseteq} \# \quad \text{(Hoare powerdomain order)}
\]

- \(\hat{\subseteq}\) is a partial order (when \(\subseteq\) is)
- \(\hat{\subseteq}\) is a sound approximation of \(\subseteq\) (when \(\subseteq\) is)
   \[A^\# \hat{\subseteq} B^\# \implies \hat{\gamma}(A^\#) \subseteq \hat{\gamma}(B^\#)\]
   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 operations

Abstract operators

\[ \hat{S} \# \] \# def = \{ S \# \} \# | \# \in \#

apply \textit{stat} on each abstract element independently

\[ \# \hat{\cup} \# \text{ def } = \# \cup \#

\text{keep elements from both arguments without applying any abstract operation}\n\hat{\cup} \# \text{ is exact}\n
\[ \# \hat{\cap} \# \text{ def } = \{ \# \cap \# | \# \in \#, \# \in \# \}

\hat{\cap} \# \text{ is exact if} \cap \# \text{ is (as} \cup \text{ and} \cap \text{ are distributive)}

Galois connection:

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

Example: powerset completion \hat{\mathcal{E}} \# of the interval domain \mathcal{E} \#

given the disc \( S \text{ def } = \{ (x, y) | x^2 + y^2 \leq 1 \}\)
\( \alpha(S) = [-1, 1] \times [-1, 1] \) (optimal interval abstraction)
but there is no best abstraction in \hat{\mathcal{E}} \#
Dynamic approximation

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

  $$(\#) \overset{\text{def}}{=} \{ \# \in \# | \forall \# \neq \# \in \#: \# \not\subseteq \# \}$$

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

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

  $$(\#) \overset{\text{def}}{=} \{ \cup\# \{ \# \in \# \} \}$$

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

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

  but how to choose which elements to merge? no easy solution!
**Issue:** for loops, abstract iterations \((\#_n)_{n \in \mathbb{N}}\) may not converge
- the size of \(#_n\) may grow arbitrarily large
- even if \(|#_n|\) is stable, some elements in \(#_n\) may not converge
  if \(E^\#\) has infinite increasing sequences

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

Widenings for powerset domains are difficult to design

**Example widening:** collapse after a fixed number of iterations

\[
\begin{align*}
\#_{n+1} & \overset{\text{def}}{=} #_n \hat{\triangledown} #_n \overset{\text{def}}{=} \begin{cases} 
(\#_n \bigtriangleup^\# #_n) & \text{if } < \\
(#_n) \triangledown (\#_n) & \text{otherwise}
\end{cases}
\end{align*}
\]

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

Principle:

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

Abstract domain:

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

- $\#$ is finite
- $\bigcup \{ \gamma(\#) | \# \in \# \} = \mathcal{E}$

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

representable in memory, as $P^\#$ is finite
Disjunctive domains

State partitioning

Ordering

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

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

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

\[
\tilde{\gamma}(\#) \overset{\text{def}}{=} \bigcup \{ \gamma(\#(\#)) \cap \gamma(\#) \mid \# \in \# \}
\]

\[
\tilde{\alpha}(\#) \overset{\text{def}}{=} \lambda \# \in \#. \alpha(\# \cap \gamma(\#))
\]

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

- $\mathcal{A}^\# \cup^\# \overset{\text{def}}{=} \lambda \# \in \#. (\#) \cup^\# (\#)$
- $\mathcal{A}^\# \cap^\# \overset{\text{def}}{=} \lambda \# \in \#. (\#) \cap^\# (\#)$
- $\mathcal{A}^\# \triangleleft \overset{\text{def}}{=} \lambda \# \in \#. (\#) \triangleleft (\#)$
- $\mathcal{S}^\#[\leq 0?] \# \overset{\text{def}}{=} \lambda \# \in \#. \mathcal{S}^\#[\leq 0?] \# (\#)$

$\mathcal{S}^\#[\leftarrow ] \#$ is more complex

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

Example: $X \leftarrow X + 2$

\[
\mathcal{S}^\#[\leftarrow ] \# \overset{\text{def}}{=} \lambda \# \cup^\# \{ \# \cap^\# \mathcal{S}^\#[\leftarrow ] (\#) | \# \in \# \}
\]
Example analysis

Example

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

Analysis:

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

$\implies$ no division by zero
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

\[
\begin{align*}
X & \leftarrow \text{rand}(-50, 50); \\
\text{if } X & \geq 0 \text{ then } \\
Y & \leftarrow X + 10 \\
\text{else } \\
Y & \leftarrow X - 10; \\
\text{assert } Y & \neq 0
\end{align*}
\]

\[
\begin{align*}
X & \leftarrow \text{rand}(-50, 50); \\
\text{if } X & \geq 0 \text{ then } \\
Y & \leftarrow X + 10; \\
\text{assert } Y & \neq 0 \\
\text{else } \\
Y & \leftarrow X - 10; \\
\text{assert } Y & \neq 0
\end{align*}
\]

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

\( H \) indicates the outcome of the last time we executed each test:
- \( (\ell) = \text{true} \): we took the then branch
- \( (\ell) = \text{false} \): we took the else branch
- \( (\ell) = \bot \): 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 \cdots 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 \( H = \mathcal{L} \rightarrow \mathbb{N} \)

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

use a different abstract element for each abstract history
Abstract operators

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

- $\hat{\gamma}(\#) = \bigcup \{ \gamma(\#(\ )) \mid \in \mathcal{H} \}$

- $\sqsubseteq, \sqcup^\#, \sqcap^\#, \sqtriangledown$ are point-wise

- $\check{\cdot}^\[ \leftarrow ]$ and $\check{\cdot}$ are point-wise

- $\check{\cdot}^\lfloor \ell : \text{if} \ 1 \ \text{else} \ 2 \rfloor^\#$ 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 $H(\ell) = \text{true}$
    \[ T'^\# = \hat{\mathcal{S}}^\#[s_1](\hat{\mathcal{S}}^\#[c?] T^\#) \]
    \[ T^\# = \lambda H. C^\#(H) \text{ if } H(\ell) = \text{true}, \bot \text{ otherwise} \]
  - we compute the else branch, where $H(\ell) = \text{false}$
    \[ F'^\# = \hat{\mathcal{S}}^\#[s_2](\hat{\mathcal{S}}^\#[\neg c?] F^\#) \]
    \[ F^\# = \lambda H. C^\#(H) \text{ if } H(\ell) = \text{false}, \bot \text{ otherwise} \]
  - 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
- look-up index in the interpolation table: \([ \ ] \leq X \leq [\ ] + 1\)
- interpolate from value \([\ ]\) when \(X = [\ ]\) with slope \([\ ]\)

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

\[\]

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

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

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

Example

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

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
- we can derive an abstract version of the system
  e.g.: $S^\#_2 = S^\# R \leftarrow 2X \]_{S^\#_1}$, $S^\#_6 = S^\#_4 \cup S^\#_5$, etc.
- we can solve the abstract system, using widenings to terminate

c.f. project
Example: Equation system

using intervals we get the following solution:

\[
\begin{align*}
S_{\text{main},1}^{\#} & : X, R \in \mathbb{Z} \\
S_{\text{main},2}^{\#} & : X \in \mathbb{Z}, R = -1 \\
S_{\text{main},3}^{\#} & : X \in [5, 10], R = -1 \\
S_{\text{main},4}^{\#} & : X \in [5, 80], R \in [0, 100] \\
S_{\text{main},5}^{\#} & : X = 80, R \in [0, 100] \\
S_{\text{main},6}^{\#} & : X \in [5, 80], R \in [0, 100] \\
S_{\text{f},1}^{\#} & : X \in [5, 80], R \in [-1, 100] \\
S_{\text{f},2}^{\#} & : X \in [5, 80], R \in [10, 160] \\
S_{\text{f},3}^{\#} & : X \in [5, 80], R \in [101, 160] \\
S_{\text{f},4}^{\#} & : X \in [5, 80], R = 0 \\
S_{\text{f},5}^{\#} & : X \in [5, 80], R \in [10, 100] \\
S_{\text{f},6}^{\#} & : X \in [5, 80], R \in [0, 100]
\end{align*}
\]
Imprecision

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

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

Syntax:

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

\[
::= (1, \ldots, |\mathbb{V}_f|) | \ldots
\]

procedure call, $f \in F$, 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 \( V_f \) in the semantics of \( g \) and restore them when returning.
- several copies of each \( V \in V_f \) may exist at a given time due to recursive calls, i.e.: cycles in the call graph.

\( \implies \) concrete environments use per-variable stacks

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

- \( \text{push}(\ , \) \( ) \overset{\text{def}}{=} \cdot \) \( (v, v' \in \mathbb{Z}, s, s' \in S) \)
- \( \text{pop}(\ ) \overset{\text{def}}{=} ' \) when \( \exists \ : = \cdot ' \), undefined otherwise
- \( \text{peek}(\ ) \overset{\text{def}}{=} ' \) when \( \exists \ ' : = \cdot ' \), undefined otherwise
- \( \text{set}(\ , \) \( ) \overset{\text{def}}{=} \cdot ' \) when \( \exists \ ' : = ' \cdot ' \), undefined otherwise

**Environments:** \( \mathcal{E} \overset{\text{def}}{=} (\bigcup_{f \in \mathcal{F}} V_f \cup V_G) \rightarrow S \)

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

traditionally, there is a single global stack for all local variables using per-variable stacks instead 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 reads and updates only consider the top of the stack; procedure calls push and pop local variables

- $E[\ ] \rho \overset{\text{def}}{=} \text{peek}(\rho(\ ))$
- $S[\ ] \overset{\text{def}}{=} \{ \rho[\ \mapsto \text{set}(\ , \rho(\ ))] \mid \rho \in \mathcal{E}[\ ] \}$
- $S[\ v_1, \ldots, v_n] = 3$, where:
  - $R_1 \overset{\text{def}}{=} \{ \rho[\forall V \in \mathcal{V}_f: V \mapsto \text{push}(x_V, \rho(V))] \mid \rho \in R, \forall V \in \mathcal{V}_f: x_V \in \mathcal{E}[e_V] \rho \}$
    (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)
  - $R_3 \overset{\text{def}}{=} \{ \rho[\forall V \in \mathcal{V}_f: V \mapsto \text{pop}(\rho(V))] \mid \rho \in R_2 \}$
    (pop local variables)
- initial environment: $\rho_0 \overset{\text{def}}{=} \lambda \in \mathcal{V}_G.0$

other statements are unchanged
Semantic inlining
Semantic inlining

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

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

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

- \( S^\#[ (1, \ldots, n) ] \) \( ^\# \) is then reduced to:
  \[ X_1^\# = S^\#[ \text{push } \forall_1; \ldots; \text{push } \forall_n ] X^\# \] (add fresh variables for \( \forall_f \))
  \[ X_2^\# = S^\#[ \forall_1 \leftarrow e_1; \ldots; \forall_n \leftarrow e_n ] X_1^\# \] (bind arguments to locals)
  \[ X_3^\# = S^\#[ \text{body}(f) ] X_2^\# \] (execute the procedure body)
  \[ X_4^\# = S^\#[ \text{pop } \forall_1; \ldots; \text{pop } \forall_n ] X_3^\# \] (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

<table>
<thead>
<tr>
<th>:</th>
<th>( ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>← −1; (rand(5, 10));</td>
<td>← 2 × ;</td>
</tr>
<tr>
<td>(80)</td>
<td>if &gt; 100 then ← 0</td>
</tr>
</tbody>
</table>

---

**Analysis using intervals**

- after the first call to `(rand(5, 10))`, we get $R \in [10, 20]$
- after the second call to `(80)`, we get $R = 0$
Call-site abstraction
Call-site abstraction

Abstracting stacks: into a fixed, bounded set $\forall^\#$ of variables

- $\forall^\# \overset{\text{def}}{=} \bigcup_{f \in F} \{ V, \hat{V} \mid V \in \forall_f \} \cup \forall_G$
  
  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

- $S^\#[\text{push}] \# \overset{\text{def}}{=} \# \bigcup^\# S^\#[\hat{V} \leftarrow] \#$
- $S^\#[\text{pop}] \# \overset{\text{def}}{=} \# \bigcup^\# S^\#[\hat{V} \leftarrow ^\hat{\text{}}] \#$

  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 $F \rightarrow E^\#$:
  - $(\ )$: abstracts the environments when **calling**
  - $(\ )$: abstracts the environments when **returning** from

  gather environments from all possible calls to $f$, disregarding the call sites

- during the analysis, when encountering a call $S^\#[ ( ) ]^\#$:
  - we return $( )$
  - but we also replace $( )$ with $[ \mapsto ( ) \cup ^\# ]$

- $( )$ is computed from $( )$ as
  $$ ( ) = S^\#[ ( ) ] ( ( ) ) $$
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 \( f(2) \)

\[ \Rightarrow \] we compute a fixpoint for \( C^{\#} \) by iteration:

- initially, \( \forall : \, \#(\ ) = \#(\ ) = \perp \)
- analyze

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

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

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

Example

: 
← −1;
(rand(5, 10));
(80)

( ):
← 2 × ;
if > 100 then ← 0

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

- first analysis of : we get ⊥ (as R♯(f) = ⊥)
  but #( ) = [ −1, −1], [5, 10]
- first analysis of : # ( ) = [ 10, 20], [5, 10]
- second analysis of : we get
  # ( ) = [ −1, 20], [5, 80]
- second analysis of : # ( ) = [ 0, 100], [5, 80]
- final analysis of , we find ∈ [0, 100] at the program end
  less precise than R = 0 found by semantic inlining
Partial context-sensitivity

**Variants:**  
- limiting, is a constant

- **stack:**
  assign a distinct variable for the highest levels of abstract the lower (unbounded) stack part with 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: \( \in \mathcal{L}^* \)
  an abstract call stack remembers the last call sites: \( \sharp \in \mathcal{L}^k \)
  the \( \sharp \) and \( \# \) maps now distinguish abstract call stacks
  \( \sharp, \#: \mathcal{L}^k \rightarrow \mathcal{E}^\sharp \)
  more precise than a partitioning by function only

larger give more precision but less efficiency
Example: context-sensitivity

Example

\[
\begin{align*}
\ell_1 &: \texttt{(rand}(5, 10)) ; \\
\ell_2 &: (80)
\end{align*}
\]

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

- \(\#(\ell_1) = [\rightarrow [-1, 1], \rightarrow [5, 10]]\)
  \[\Rightarrow \quad \#(\ell_1) = [\rightarrow [10, 20], \rightarrow [5, 10]]\]

- \(\#(\ell_2) = [\rightarrow [10, 20], \rightarrow [80, 80]]\)
  \[\Rightarrow \quad \#(\ell_2) = [\rightarrow [0, 0], \rightarrow [80, 80]]\]

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 $S[\ ] : \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

$
\implies$ abstract it as an abstract function in $\mathcal{E}^\# \rightarrow \mathcal{E}^\#
$
we use a partial function as the image of most abstract elements is not useful

**Analysis:** tabulated analysis

- use a global partial map $\# : \mathcal{F} \times \mathcal{E}^\# \rightarrow \mathcal{E}^\#
$

  - $\#$ is initially empty, and is filled on-demand

  - when encountering $S^\#[\ ] \#$

    - return $F^\#(f, X^\#)$ if defined
    - else, compute $S^\#[\ body(f) ] \# X^\#$, store it in $F^\#(f, X^\#)$ and return it

**Optimizations:** trade precision for efficiency

- if $X^\# \subseteq Y^\#$ 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

```plaintext
Example

:  
  ← −1;
  (rand(5, 10));
  (80)  
  
( ):
  ← 2 × ;
  if  > 100 then  ← 0
```

Analysis using intervals

- \# =
  
  [ ( , [ ← [−1, −1],  ← [5, 10]]) ← [ ] ← [10, 20],  ← [5, 10]],
  ( , [ ← [10, 20],  ← [80, 80]]) ← [ ] ← [0, 0],  ← [80, 80]]

- at the end of the analysis, we get again = 0

here, the function partitioning gives the same result as the call-site partitioning
Dynamic partitioning: complex example

Example: McCarthy’s 91 function

\[
\begin{align*}
\text{Mc} & : \quad (\text{rand}(0, +\infty)) \\
(\quad) & : \\
\text{if} & \quad > 100 \text{ then } \quad \leftarrow \quad -10 \\
\text{else} & \quad ( \quad +11); \quad (\quad)
\end{align*}
\]

- in the concrete, when terminating:
  \( n \in [0, 72] \Rightarrow r = 91 \)
  \( n \in [73, 90] \Rightarrow r \in [91, 101] \)
  \( n \in [91, 101] \Rightarrow r = 91 \)
  \( n \in [102, 111] \Rightarrow r \in [91, 101] \)
  \( n \in [112, +\infty] \Rightarrow r \in [91, +\infty] \)

- using a widening \( \triangledown \) to choose tabulated abstract values \( \#(\quad, \quad) \)

(source: Bourdoncle, JFP 1992)