Combination of Abstract Domains

MPRI — Cours 2.6 “Interprétation abstraite : application à la vérification et à l’analyse statique”

Xavier Rival

INRIA, ENS, CNRS

Nov, 2nd. 2012

Overview of the lecture

- **Construction of abstract semantics**
  a **step-by-step process** from basic abstractions
  - **numerical abstractions**
  - **conjunctions** of abstract properties: product
  - **disjunctions** of abstract properties: disjunctive completion, partitioning
- **Decomposing abstraction** has **many advantages**:
  - **modular** design of static analyzers:
    split into several different abstractions
  - **flexibility** of the resulting tools:
    better scalability, extensibility to broader analysis setups
- Also, we will get a **better understanding** of abstract domain properties: **reduction**

An example

How to verify the following program ?

```
int i = 0;                     // integer variable
bool b;                        // boolean variable
while(i < 10){
   i = i + 2;
   b = brand();
   if(b){
      break;
   }
}
assert(b ∨ i == 10);           // assertion to prove
```

- We want to do an abstract interpretation of the code
- First, we need to **construct an abstract domain**

Hoare proof and choice of an abstract domain

```
  int i = 0;
{i = 0}
  bool b;
{i = 0}
  while(i < 10){
{0 ≤ i ≤ 8 ∧ i ≡ 0(2)}
    i = i + 2;
{2 ≤ i ≤ 10 ∧ i ≡ 0(2)}
    b = brand();
{2 ≤ i ≤ 10 ∧ i ≡ 0(2)}
    if(b){
{2 ≤ i ≤ 10 ∧ i ≡ 0(2) ∧ b = TRUE}
      break;
{}
    }
{2 ≤ i ≤ 10 ∧ i ≡ 0(2) ∧ b = FALSE}
  }
{b = TRUE ∨ i = 10}
  assert(b ∨ i == 10);
```

Abstract interpretation

Which abstract domain ?

We need:

- interval constraints
- congruences constraints
- conjunctions
- disjunctions

- This lecture shows **how to build such a domain** using **combinations of basic abstract domains**

A first (de)composition: function composition

Flashback: composition of Galois connections

Let $(\mathbb{D}_0, \sqsubseteq_0)$, $(\mathbb{D}_1, \sqsubseteq_1)$ and $(\mathbb{D}_2, \sqsubseteq_2)$ be three abstract domains, and let us assume the Galois connections below are defined:

$$(\mathbb{D}_0, \sqsubseteq_0) \underset{\alpha_{01}}{\overset{\gamma_{10}}{\leftrightarrows}} (\mathbb{D}_1, \sqsubseteq_1) \qquad (\mathbb{D}_1, \sqsubseteq_1) \underset{\alpha_{12}}{\overset{\gamma_{21}}{\leftrightarrows}} (\mathbb{D}_2, \sqsubseteq_2)$$

Then, we have a third Galois connection

$$(\mathbb{D}_0, \sqsubseteq_0) \underset{\alpha_{12} \circ \alpha_{01}}{\overset{\gamma_{10} \circ \gamma_{21}}{\leftrightarrows}} (\mathbb{D}_2, \sqsubseteq_2)$$

We can generalize this principle:

Composition of concretization functions

If $\gamma_{21} : \mathbb{D}_2 \rightarrow \mathbb{D}_1$ (resp., $\gamma_{10} : \mathbb{D}_1 \rightarrow \mathbb{D}_0$) describe concretization functions from $(\mathbb{D}_2, \sqsubseteq_2)$ to $(\mathbb{D}_1, \sqsubseteq_1)$ (resp., from $(\mathbb{D}_1, \sqsubseteq_1)$ to $(\mathbb{D}_0, \sqsubseteq_0)$), then $\gamma_{20} = \gamma_{10} \circ \gamma_{21}$ describes a concretization from $(\mathbb{D}_2, \sqsubseteq_2)$ to $(\mathbb{D}_0, \sqsubseteq_0)$

Decomposition of abstract domains

We inspect the predicates needed in the Hoare proof:

- **One invariant per control point**:
  - already seen informally in previous lectures
  - different control states need be abstracted separately
  - **partitioning** abstraction
- $\{0 \leq \mathtt{i} \leq 8 \wedge \mathtt{i} \equiv 0(2)\}$:
  - **conjunction** of an **interval** constraint and of a **congruence** constraint
  - expressible in a **product** of abstractions
- $\{\mathtt{b} = \mathtt{TRUE} \vee \mathtt{i} = 10\}$:
  - **disjunction** of constraints
  - several ways to express this:
    state partitioning, **trace partitioning**

Notations and definitions: concrete level

Concrete states

Concrete states are of the form $\mathbb{S} = \mathbb{L} \times \mathbb{M}$

- $\mathbb{L}$ is the set of *labels* or *control states*
- $\mathbb{M}$ is the set of *memory states*

Moreover, $\mathbb{M} = \mathbb{X} \to \mathbb{V}$, where:

- $\mathbb{X}$ is the *set of variables*
- $\mathbb{V}$ is the *set of values*

We will use several concrete semantics during this lecture:

- **finite traces semantics** $[\![\mathbb{S}]\!]^\star \in \mathcal{P}(\mathbb{S}^\star)$
- **reachable states semantics** $[\![\mathbb{S}]\!]_{\mathcal{R}} \in \mathcal{P}(\mathbb{S})$

Notations and definitions: abstract level

We shall use abstract-domains to over-approximate sets of **concrete values**, sets of **states**, sets of **traces**

Abstract domain definitions

An abstract domain will comprise a set of abstract values $\mathbb{D}^\sharp$ and:

- a concretization function γ and optionnally an abstraction α
- an abstract order $\sqsubseteq^\sharp$, an abstract infimum $\perp$
- an abstract upper bound $\sqcup^\sharp$, and a widening operator ∇
- abstract transfer functions $\mathfrak{f}^\sharp, \mathfrak{g}^\sharp, \ldots$ associated to common concrete operations

- These allow defining static analyses computing abstract least-fixpoints or abstract post-fixpoints

When we build composite abstract domains from basic ones, we will assume / ensure such elements

Outline

Partitioning of an abstraction

Partitioning abstraction

Given set E and partition $\mathfrak{P}$ of E, we let the **partitioning abstraction** over E be defined by:

$$\begin{array}{rlcl}
\alpha_{\mathrm{part}} : & \mathcal{P}(E) & \longrightarrow & (\mathfrak{P} \to \mathcal{P}(E)) \\
& X & \longmapsto & \lambda(p \in \mathfrak{P}) \cdot (p \cap X) \\
\gamma_{\mathrm{part}} : & (\mathfrak{P} \to \mathcal{P}(E)) & \longrightarrow & \mathcal{P}(E) \\
& \Phi & \longmapsto & \bigcup_{p \in \mathfrak{P}} \Phi(p)
\end{array}$$

It indeed forms a Galois connection:

$$(\mathcal{P}(E), \subseteq) \underset{\alpha_{\mathrm{part}}}{\overset{\gamma_{\mathrm{part}}}{\leftrightarrows}} (\mathfrak{P} \to \mathcal{P}(E), \dot{\subseteq})$$

Proof: $\alpha_{\mathrm{part}}(X) \mathrel{\dot{\subseteq}} \Phi \iff X \subseteq \gamma_{\mathrm{part}}(\Phi)$

Example: control state partitioning

How to abstract separately memory states associated to different control states ?

Control state partitioning

We apply the partitioning abstraction with:

- $E = \mathbb{S}$
- $\mathfrak{P} = \{\{(l, m) \mid m \in \mathbb{M}\} \mid l \in \mathbb{L}\}$

We note that $\mathfrak{P} \equiv \mathbb{L}$ and that, for all $l \in \mathbb{L}$, $\{(l, m) \mid m \in \mathbb{M}\} \equiv \mathbb{M}$, therefore, the partitioning abstraction is:

$$\begin{array}{rlcl} \alpha_{\mathrm{part}} : & \mathcal{P}(E) & \longrightarrow & (\mathbb{L} \rightarrow \mathcal{P}(E)) \\ & X & \longmapsto & \lambda(l \in \mathbb{L}) \cdot \{m \in \mathbb{M} \mid (l, m) \in X\} \\ \gamma_{\mathrm{part}} : & (\mathbb{L} \rightarrow \mathcal{P}(E)) & \longrightarrow & \mathcal{P}(E) \\ & \Phi & \longmapsto & \bigcup_{l \in \mathbb{L}} \{(l, m) \mid m \in \Phi(l)\} \end{array}$$

Example: control state partitioning

We can compose this abstraction with any other abstraction over memory states:

Abstraction over a partitioned system

Let $(\mathbb{D}^\sharp_{\mathrm{num}}, \sqsubseteq^\sharp_{\mathrm{num}})$ be an abstraction of $(\mathcal{P}(\mathbb{M}), \subseteq)$, with a Galois connection $(\mathcal{P}(\mathbb{M}), \subseteq) \underset{\alpha_{\mathrm{num}}}{\overset{\gamma_{\mathrm{num}}}{\leftrightarrows}} (\mathbb{D}^\sharp_{\mathrm{num}}, \sqsubseteq^\sharp_{\mathrm{num}})$.

Then, we define the abstract domain $(\mathbb{D}^\sharp_{\mathrm{part}}, \sqsubseteq^\sharp_{\mathrm{part}}) = (\mathbb{L} \rightarrow \mathbb{D}^\sharp_{\mathrm{num}}, \dot{\sqsubseteq}^\sharp_{\mathrm{num}})$, with the abstraction and concretization defined by:

$$\begin{array}{rlcl}
\dot{\alpha}_{\mathrm{num}} \circ \alpha_{\mathrm{part}} : & \mathcal{P}(\mathbb{S}) & \longrightarrow & (\mathbb{L} \rightarrow \mathbb{D}^\sharp_{\mathrm{num}}) \\
 & \mathcal{S} & \longmapsto & \lambda(l \in \mathbb{L}) \cdot \alpha_{\mathrm{num}}(\{m \in \mathbb{M} \mid (l, m) \in \mathcal{S}\}) \\
\gamma_{\mathrm{part}} \circ \dot{\gamma}_{\mathrm{num}} : & (\mathbb{L} \rightarrow \mathbb{D}^\sharp_{\mathrm{num}}) & \longrightarrow & \mathcal{P}(\mathbb{S}) \\
 & \Phi & \longmapsto & \{(l, m) \mid \exists l \in \mathbb{L},\ m \in \gamma_{\mathrm{num}}(\Phi(l))\}
\end{array}$$

- Case with only a γ_{num} (no α_{num}): similar defintions

Example: context sensitive abstraction

We consider a language with procedures (set of procedures $\mathbb{P}$)

Semantics with procedures

The set of states is of the form $\mathbb{S} = \mathbb{K} \times \mathbb{L} \times \mathbb{M}$, where $\mathbb{K}$ is the set of contexts defined by:

$$\begin{array}{rcll} k \in \mathbb{K} & ::= & \epsilon & \text{empty call stack} \\ & | & f \cdot k & \text{call to } f \text{ from stack } k \end{array}$$

Context sensitive abstraction

$$\mathfrak{P} = \{\{(k, l, m) \mid m \in \mathbb{M}\} \mid k \in \mathbb{K}, l \in \mathbb{M}\}$$

- one invariant per calling context
- infinite if recursion

Context insensitive abstraction

$$\mathfrak{P} = \{\{(f \cdot k, l, m) \mid m \in \mathbb{M}, k \in \mathbb{K}\} \mid f \in \mathbb{P}, l \in \mathbb{M}\}$$

- merges different calling contexts to a same procedure
- coarser abstraction

Fixpoint form of a partitioned semantics

- We consider a transition system $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$
- The reachable states are computed as $[\![\mathcal{S}]\!]_{\mathcal{R}} = \mathbf{lfp}_{\mathbb{S}_{\mathcal{I}}} F$ where

$$\begin{array}{rccl} F : & \mathcal{P}(\mathbb{S}) & \longrightarrow & \mathcal{P}(\mathbb{S}) \\ & X & \longmapsto & \{s \in \mathbb{S} \mid \exists s' \in X,\ s' \rightarrow s\} \end{array}$$

Semantic function over the partitioned system

We let F_{part} be defined over $\mathbb{D}^{\sharp}_{\mathrm{part}} = \mathfrak{P} \rightarrow \mathcal{P}(\mathbb{S})$ by:

$$\begin{array}{rccl} F_{\mathrm{part}} : & \mathbb{D}^{\sharp}_{\mathrm{part}} & \longrightarrow & \mathbb{D}^{\sharp}_{\mathrm{part}} \\ & \Phi & \longmapsto & \lambda(p \in \mathfrak{P}) \cdot \{s \in p \mid \exists p' \in \mathfrak{P},\ \exists s' \in \Phi(p'),\ s' \rightarrow s\} \end{array}$$

Then $F_{\mathrm{part}} \circ \alpha_{\mathrm{part}} = \alpha_{\mathrm{part}} \circ F$, and

$$\alpha_{\mathrm{part}}([\![\mathcal{S}]\!]_{\mathcal{R}}) = \mathbf{lfp}_{\alpha_{\mathrm{part}}(\mathbb{S}_{\mathcal{I}})} F_{\mathrm{part}}$$

Abstract equations form of a partitioned semantics

- We look for a set of equivalent abstract equations
- We consider the case of a system partitioned by control states $\mathbb{L} = \{l_1, \ldots, l_s\}$
- Let us consider the system of semantic equations over sets of states $\mathcal{E}_1, \ldots, \mathcal{E}_s \in \mathcal{P}(\mathbb{M})$:

$$\left\{ \begin{array}{rcl} \mathcal{E}_1 & = & \bigcup_i \{ m \in \mathbb{M} \mid \exists m' \in \mathcal{E}_i,\ (l_i, m') \rightarrow (l_1, m) \} \\ & \vdots & \\ \mathcal{E}_s & = & \bigcup_i \{ m \in \mathbb{M} \mid \exists m' \in \mathcal{E}_i,\ (l_i, m') \rightarrow (l_s, m) \} \end{array} \right.$$

So, if we let
$F_i : (\mathcal{E}_1, \ldots, \mathcal{E}_s) \mapsto \bigcup_i \{ m \in \mathbb{M} \mid \exists m' \in \mathcal{E}_i,\ (l_i, m') \rightarrow (l_i, m) \}$, then:

$\alpha_{\mathrm{part}}(\llbracket \mathcal{S} \rrbracket_{\mathcal{R}})$ is the least solution of the system $\left\{ \begin{array}{rcl} \mathcal{E}_1 & = & F_1(\mathcal{E}_1, \ldots, \mathcal{E}_s) \\ & \vdots & \\ \mathcal{E}_s & = & F_s(\mathcal{E}_1, \ldots, \mathcal{E}_s) \end{array} \right.$

Partitioned systems and fixpoint computation

How to compute an abstract invariant for a partitioned systme described by a set of abstract equations ?

(for now, we assume no convergence issue, i.e., that the abstract lattice is of finite height)

- In practice F_i depends **only on a few of its arguments**
  i.e., $\mathcal{E}_k$ depends only on the predecessors of l_k in the control flow graph of the program being analyzed
- **Example** of a simple system of abstract equations:

$$\left\{ \begin{array}{rcl} \mathcal{E}_0 & = & \mathcal{I} \cup F_0(\mathcal{E}_3) \\ \mathcal{E}_1 & = & F_1(\mathcal{E}_0) \\ \mathcal{E}_2 & = & F_2(\mathcal{E}_0) \\ \mathcal{E}_3 & = & F_3(\mathcal{E}_1, \mathcal{E}_2) \end{array} \right.$$

where $\alpha_{\mathrm{part}}(\mathbb{S}_{\mathcal{I}}) = (\mathbb{S}_{\mathcal{I}}, \perp, \perp, \perp)$ (i.e., init states are at point l_0)

Partitioned systems and fixpoint computation

Following the fixpoint transfer, we obtain the following abstract iterates $(\mathcal{E}^\sharp_n)_{n \in \mathbb{N}}$:

$$\begin{array}{rclllll}
\mathcal{E}^\sharp_0 & = & (\mathbb{I}, & \bot, & \bot, & \bot) \\
\mathcal{E}^\sharp_1 & = & (\mathbb{I}, & F^\sharp_1(\mathbb{I}), & F^\sharp_2(\mathbb{I}), & \bot) \\
\mathcal{E}^\sharp_2 & = & (\mathbb{I}, & F^\sharp_1(\mathbb{I}), & F^\sharp_2(\mathbb{I}), & F^\sharp_3(F^\sharp_1(\mathbb{I}), F^\sharp_2(\mathbb{I}))) \\
\mathcal{E}^\sharp_3 & = & (\mathbb{I} \sqcup F^\sharp_0(F^\sharp_3(F^\sharp_1(\mathbb{I}), F^\sharp_2(\mathbb{I}))), & F^\sharp_1(\mathbb{I}), & F^\sharp_2(\mathbb{I}), & F^\sharp_3(F^\sharp_1(\mathbb{I}), F^\sharp_2(\mathbb{I})))
\end{array}$$

- Each iteration causes **the recomputation of all components**
- Though, each iterate differs from the previous one **in only a few components**

Chaotic iterations: principle

Fairness

Let K be a finite set. A sequence $(k_n)_{n\in\mathbb{N}}$ of elements of K is fair if and only if, for all $k \in K$, the set $\{n \in \mathbb{N} \mid k_n = k\}$ is infinite.

- Other alternate definition: $\forall k \in K, \forall n_0 \in \mathbb{N}, \exists n \in \mathbb{N}, n > n_0 \wedge k_n = k$
- i.e., all elements of K is encountered infintely often

Chaotic iterations

A chaotic sequence of iterates is a sequence of abstract iterates $(X_n^\sharp)_{n\in\mathbb{N}}$ in $\mathbb{D}^\sharp_{\mathrm{part}}$ such that there exists a sequence $(k_n)_{n\in\mathbb{N}}$ of elements of $\{1, \ldots s\}$ such that:

$$X_{n+1}^\sharp = \lambda(l_i \in \mathbb{L}) \cdot \begin{cases} X_n^\sharp(l_i) & \text{if } i \neq k_n \\ X_n^\sharp(l_i) \sqcup F^\sharp(X_n^\sharp(l_1), \ldots, X_n^\sharp(l_s)) & \text{if } i = k_n \end{cases}$$

Chaotic iterations: soundness

Soundness

Assuming the abstract lattice satisfies the ascending chain condition, any sequence of chaotic iterates computes the abstract fixpoint:

$$\lim\,(X^\sharp_n)_{n\in\mathbb{N}} = \alpha_{\mathrm{part}}([\![\mathcal{S}]\!]_{\mathcal{R}})$$

Proof: exercise

- **Applications**: we can recompute only what is necessary
- **Back to the example**, where only the **recomputed components** are **colored**:

$$\begin{array}{rcllll}
\mathcal{E}^\sharp_0 & = & (\mathbb{I}, & \bot, & \bot, & \bot) \\
\mathcal{E}^\sharp_1 & = & (\mathbb{I}, & F^\sharp_1(\mathbb{I}), & \bot, & \bot) \\
\mathcal{E}^\sharp_2 & = & (\mathbb{I}, & F^\sharp_1(\mathbb{I}), & F^\sharp_2(\mathbb{I}), & \bot) \\
\mathcal{E}^\sharp_3 & = & (\mathbb{I}, & F^\sharp_1(\mathbb{I}), & F^\sharp_2(\mathbb{I}), & F^\sharp_3(F^\sharp_1(\mathbb{I}), F^\sharp_2(\mathbb{I}))) \\
\mathcal{E}^\sharp_4 & = & (\mathbb{I} \sqcup F^\sharp_0(F^\sharp_3(F^\sharp_1(\mathbb{I}), F^\sharp_2(\mathbb{I}))), & F^\sharp_1(\mathbb{I}), & F^\sharp_2(\mathbb{I}), & F^\sharp_3(F^\sharp_1(\mathbb{I}), F^\sharp_2(\mathbb{I})))
\end{array}$$

Chaotic iterations: worklist algorithm

Worklist algorithms

Principle:

- maintain a queue of partitions to update
- initialize the queue with the entry label of the program and the local invariant at that point at $\alpha_{\mathrm{num}}(\mathbb{S}_{\mathcal{I}})$
- for each iterate, update the first partition in the queue (after removing it), and add to the queue all its successors *unless* the updated invariant is equal to the former one
- terminate when the queue is empty

This algorithm implements a **chaotic iteration** strategy, thus it is sound

- **Application**: only partitions that need be updated are recomputed
- **Implemented in many static analyzers**

Selection of a set of widening points for a partitioned system

- We do not assume anymore that $\mathbb{D}^\sharp_{\mathrm{num}}$ satisfies the ascending chain condition
- We assume $\mathbb{D}^\sharp_{\mathrm{num}}$ provides widening operator $\triangledown$

How to adapt the chaotic iteration strategy, i.e. guarantee termination and soundness ?

Enforcing termination of chaotic iterates

Let $K \subseteq \{1, \ldots, s\}$ such that each cycle in the control flow graph of the program contains at least one point in K; we define the chaotic abstract iterates with widening as follows:

$$X^\sharp_{n+1} = \lambda(l_i \in \mathbb{L}) \cdot \begin{cases} X^\sharp_n(l_i) & \text{if } i \neq k_n \\ X^\sharp_n(l_i) \sqcup F^\sharp(X^\sharp_n(l_1), \ldots, X^\sharp_n(l_s)) & \text{if } i = k_n \wedge l_i \notin K \\ X^\sharp_n(l_i) \triangledown F^\sharp(X^\sharp_n(l_1), \ldots, X^\sharp_n(l_s)) & \text{if } i = k_n \wedge l_i \in K \end{cases}$$

Selection of a set of widening points for a partitioned system

Soundness and termination

Under the assumption of a fair iteration strategy, sequence $(X_n^\sharp)_{n\in\mathbb{N}}$ terminates and computes a sound abstract post-fixpoint:

$$\exists n_0 \in \mathbb{N}, \left\{ \begin{array}{l} \forall n \geq n_0,\, X_{n_0}^\sharp = X_n^\sharp \\ [\![\mathcal{S}]\!]_{\mathcal{R}} \subseteq \gamma_{\mathrm{part}}(X_{n_0}) \end{array} \right.$$

Proof: exercise

Outline

Product abstraction

Definition

Let $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$ and $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1)$ be two abstract domains:

$$(\mathbb{D}, \subseteq) \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} (\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0) \qquad \text{and} \qquad (\mathbb{D}, \subseteq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1)$$

The product abstract domain $(\mathbb{D}^\sharp_\times, \sqsubseteq^\sharp_\times)$ is defined by:

- $\mathbb{D}^\sharp_\times = \mathbb{D}^\sharp_0 \times \mathbb{D}^\sharp_1$
- $(x_0, x_1) \sqsubseteq^\sharp_\times (y_0, y_1) \iff x_0 \sqsubseteq^\sharp_0 y_0 \wedge x_1 \sqsubseteq^\sharp_1 y_1$

The product abstraction is defined by:

$$(\mathbb{D}, \subseteq) \underset{\alpha_\times}{\overset{\gamma_\times}{\leftrightarrows}} (\mathbb{D}^\sharp_\times, \sqsubseteq^\sharp_\times) \qquad \text{where}$$

$$\begin{array}{rcccl} \alpha_\times : & \mathbb{D} & \longrightarrow & \mathbb{D}^\sharp_\times \\ & a & \longmapsto & (\alpha_0(a), \alpha_1(a)) \end{array} \qquad \begin{array}{rcccl} \gamma_\times : & \mathbb{D}^\sharp & \longrightarrow & \mathbb{D} \\ & (x_0, x_1) & \longmapsto & \gamma_0(x_0) \cap \gamma_1(x_1) \end{array}$$

Product abstraction

Proof, following the usual principle:

$$\begin{array}{rcl}
\alpha(a)\sqsubseteq^\sharp_\times(x_0, x_1) & \Longleftrightarrow & (\alpha_0(a), \alpha_1(a))\sqsubseteq^\sharp_\times(x_0, x_1) \\
& \Longleftrightarrow & \alpha_0(a)\sqsubseteq^\sharp_0 x_0 \wedge \alpha_1(a)\sqsubseteq^\sharp_1 x_1 \\
& \Longleftrightarrow & a \subseteq \gamma_0(x_0) \wedge a \subseteq \gamma_1(x_1) \\
& \Longleftrightarrow & a \subseteq \gamma_0(x_0) \cap \gamma_1(x_1) \\
& \Longleftrightarrow & a \subseteq \gamma_\times(x_0, x_1)
\end{array}$$

Conjunctions of abstract properties

Elements of the product abstract domain stand for conjunctions of abstract properties of $\mathbb{D}^\sharp_0$ and of $\mathbb{D}^\sharp_1$.

Example: conjunctions of constraints

Assumptions:

- $\mathbb{D}$ is $\mathcal{P}(\mathbb{Z})$ and $\subseteq$ the set inclusion
- $\mathbb{D}_0^\sharp$ is $\mathbb{Z} \cup \{-\infty, +\infty\}$, $\sqsubseteq_0^\sharp$ is $\leq$ and $\alpha_0(E) = \inf\ E$
- $\mathbb{D}_1^\sharp$ is $\mathbb{Z} \cup \{-\infty, +\infty\}$, $\sqsubseteq_0^\sharp$ is $\leq$ and $\alpha_1(E) = \sup\ E$

Product abstraction:

- Then:

$$\begin{array}{rcl} \alpha_\times(\mathbb{Z}) & = & (-\infty, +\infty) \\ \alpha_\times(\emptyset) & = & (+\infty, -\infty) \end{array} \qquad \begin{array}{rcl} \alpha_\times(\{0,2,4,6,8\}) & = & (0,8) \\ \alpha_\times(\{1,2,3\}) & = & (1,3) \end{array}$$

- Moreover:

$$\gamma_\times(x_0, x_1) \quad = \quad \{x \in \mathbb{Z} \mid x_0 \leq x \wedge x \leq x_1\}$$

Therefore $\mathbb{D}_\times^\sharp$ is the **interval abstraction**, where an interval is viewed as a conjunction of two constraints

Example: intervals and congruences

Assumptions:

- $\mathbb{D}$ is $\mathcal{P}(\mathbb{Z})$ and $\subseteq$ the set inclusion
- $\mathbb{D}^\sharp_0$ is the interval abstract domain (an abstract values is either $\perp$ or a pair of elements of $\mathbb{Z} \cup \{-\infty, +\infty\}$)
- $\mathbb{D}^\sharp_1$ is the congruences abstract domain:
  - abstract values are either $\perp$, or of the form $\langle a, b \rangle$ with $0 \leq a < b$ or $b = 0$
  - $\gamma_1(\perp) = \emptyset$ and $\gamma_1(\langle a, b \rangle) = \{a + k \cdot b \mid k \in \mathbb{Z}\}$

Product abstraction:

- Then:

$$\begin{array}{rclrcl} \alpha_\times(\emptyset) & = & (\perp, \perp) & \alpha_\times(\{1, 3, \ldots\}) & = & ([1, +\infty[, \langle 1, 2 \rangle \\ \alpha_\times(\mathbb{Z}) & = & (]-\infty, +\infty[, \langle 0, 1 \rangle) & \alpha_\times(\{1, 3, 7\}) & = & ([1, 7], \langle 1, 2 \rangle) \end{array}$$

- Moveover:

$$\begin{array}{rclrcl} \gamma_\times([1, 7], \langle 1, 2 \rangle) & = & \{1, 3, 5, 7\} & \gamma_\times([0, 10], \langle 3, 6 \rangle) & = & \{3, 9\} \\ \gamma_\times([1, 8], \langle 1, 2 \rangle) & = & \{1, 3, 5, 7\} & \gamma_\times([0, +\infty[, \langle 3, 6 \rangle) & = & \{3, 9, \ldots \end{array}$$

Operations in the product domain

- **Least element**: if $\bot_0$ (resp., $\bot_1$) is the least element of $\mathbb{D}_0^\sharp$ (resp. of $\mathbb{D}_1^\sharp$), then $\bot_\times = (\bot_0, \bot_1)$ is the least element of $\mathbb{D}_\times^\sharp$
- **Upper bound**: if $\sqcup_0$ (resp., $\sqcup_1$) is a sound upper bound operator on $\mathbb{D}_0^\sharp$ (resp., $\mathbb{D}_1^\sharp$), then $\sqcup_\times$ defined by $(x_0, x_1) \sqcup_\times (y_0, y_1) = (x_0 \sqcup_0 y_0, x_1 \sqcup_1 y_1)$ is a sound upper bound operator on $\mathbb{D}_\times^\sharp$
- **Widening**: if $\sqcup_0$ (resp. $\sqcup_1$) is a widening on $\mathbb{D}_0^\sharp$ (resp. $\mathbb{D}_1^\sharp$), then $\sqcup_\times$ defined by $(x_0, x_1) \sqcup_\times (y_0, y_1) = (x_0 \sqcup_0 y_0, x_1 \sqcup_1 y_1)$ is a widening on $\mathbb{D}_\times^\sharp$

Proofs: exercise!

Operations in the product domain

- **Transfer functions**:
  We assume that:
  - $\mathfrak{f} : \mathbb{D} \rightarrow \mathbb{D}$ is a concrete transfer function (e.g., describing the effect of a test or of an assignment)
  - $\mathfrak{f}_0^\sharp : \mathbb{D}_0^\sharp \rightarrow \mathbb{D}_0^\sharp$ is a sound transfer function with respect to $\mathfrak{f}$, that is such that $\mathfrak{f} \circ \gamma_0 \subseteq \gamma_0 \circ \mathfrak{f}_0^\sharp$
  - $\mathfrak{f}_1^\sharp : \mathbb{D}_1^\sharp \rightarrow \mathbb{D}_1^\sharp$ achieves the same condition in $\mathbb{D}_1^\sharp$

  Then, we let $\mathfrak{f}_\times^\sharp$ be defined by:

$$\begin{array}{rccl} \mathfrak{f}_\times^\sharp : & \mathbb{D}_\times^\sharp & \longrightarrow & \mathbb{D}_\times^\sharp \\ & (x_0, x_1) & \longmapsto & (\mathfrak{f}_0^\sharp(x_0), \mathfrak{f}_1^\sharp(x_1)) \end{array}$$

  Then $\mathfrak{f}_\times^\sharp$ is sound with respect to $\mathfrak{f}$

Transfer functions in the product abstraction

We consider **the interval abstraction** as a **product of constraints**

- $\mathbb{D}$ is $\mathcal{P}(\mathbb{Z})$ and $\subseteq$ the set inclusion
- $\mathbb{D}^\sharp_0$ is $\mathbb{Z} \cup \{-\infty, +\infty\}$, $\sqsubseteq^\sharp_0$ is $\leq$ and $\alpha_0(E) = \inf\ E$
- $\mathbb{D}^\sharp_1$ is $\mathbb{Z} \cup \{-\infty, +\infty\}$, $\sqsubseteq^\sharp_0$ is $\leq$ and $\alpha_1(E) = \sup\ E$

We consider the concrete function $\mathfrak{f} : x \mapsto -x$

- The lower bound before gives no information on the lower bound after: $\mathfrak{f}^\sharp_0 : x_0 \mapsto -\infty$
- The same goes for the upper bounds: $\mathfrak{f}^\sharp_1 : x_1 \mapsto +\infty$
- Hence, $\mathfrak{f}^\sharp_\times(x_0, x_1) =]-\infty, +\infty[= \top$
- Though, we would like the more precise: $(x_0, x_1) \longmapsto (-x_1, -x_0)$

- Decomposed transfer function may lose precision
- Decomposing the interval abstract domain in a product abstraction does not make sense for the computation of transfer functions

Transfer functions in the product abstraction

We now consider the product of intervals and congruences, with transfer functions:

- $\mathbb{D}$ is $\mathcal{P}(\mathbb{Z})$ and $\subseteq$ the set inclusion
- **Test**: $\mathfrak{f}(t, \mathcal{E}) = \{z \in \mathbb{Z} \mid [\![t]\!](v \mapsto z) = \texttt{TRUE}\}$ returns the values that satisfy condition t on variable v
- **Random add**: $\mathfrak{g}(\mathcal{E}) = \{x + k \mid x \in \mathcal{E} \wedge -1 \leq k \leq 1\}$

- $x^\sharp ::= ([0, 10], \langle 0, 2 \rangle)$
- $y^\sharp ::= \mathfrak{p}^\sharp_\times(v = 5, x^\sharp) = ([5, 5], \perp)$
- $\gamma_\times(y^\sharp) = \emptyset$
- why not $y^\sharp = (\perp, \perp)$ then ?

- $x^\sharp ::= ([0, 10], \langle 0, 2 \rangle)$
- $y^\sharp ::= \mathfrak{p}^\sharp_\times(v \leq 5, x^\sharp) = ([0, 5], \langle 0, 2 \rangle)$
- $z^\sharp ::= \mathfrak{p}^\sharp_\times(v \geq 5, y^\sharp) = ([5, 5], \langle 0, 2 \rangle)$
- $\gamma_\times(z^\sharp) = \emptyset$
- why not $z^\sharp = (\perp, \perp)$ then ?

Improving transfer functions

We consider the program:

```
assume($x \in [0, 10]$, even);
if($x \leq 5$){
    if($x \geq 5$){
        $x + rand([-1, 1])$;
        assert(FALSE);
    }
}
```

- analysis, from state $x^\sharp ::= ([0, 10], \langle 0, 2 \rangle)$
- $y^\sharp ::= \mathfrak{p}^\sharp_\times(v \leq 5, x^\sharp) = ([0, 5], \langle 0, 2 \rangle)$
- $z^\sharp ::= \mathfrak{p}^\sharp_\times(v \geq 5, y^\sharp) = ([5, 5], \langle 0, 2 \rangle)$
- $v^\sharp ::= \mathfrak{g}^\sharp(z^\sharp) = ([4, 6], \langle 0, 1 \rangle)$

Then, we notice that:

- In the concrete, the body of the second **if** is **unreachable**
- In the abstract, $\gamma_\times(v^\sharp) = \{4, 5, 6\} \neq \emptyset$
- The product abstraction misses the fact that:

$$x = 5 \wedge x \equiv 0 \mod (2) \Longrightarrow x \in \emptyset$$

Limitations of product abstraction

- It does not allow information be sent from one domain to the other
- This is the source of a **loss of precision** in the analysis

How to overcome this ?

Outline

Injective concretization

We consider the loss of information in the interval + congruences example:

- $\gamma_\times([5,5],\langle 0,2\rangle) = \emptyset = \gamma_\times(\perp,\perp)$
- $\mathfrak{g}([5,5],\langle 0,2\rangle) = ([4,6],\langle 0,1\rangle)$
- $\mathfrak{g}(\perp,\perp) = (\perp,\perp)$, which means that $(\perp,\perp)$ is **much more useful** for the rest of the analysis than $([5,5],\langle 0,2\rangle)$
- converting $([5,5],\langle 0,2\rangle)$ into $(\perp,\perp)$ amounts to applying the mathematical result:

$$x = 5 \wedge x \equiv 0 \mod (2) \Longrightarrow x \in \emptyset$$

- Some product elements are semantically "equivalent"
  for computing other transfer functions, proving semantic assertions...
- Some semantically equivalent product elements are "better"
  Computing those "better" elements is **reduction**

Galois surjection (or Galois insertion)

Definition

Let us consider an abstraction defined by a Galois connection

$$(\mathbb{D}, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{D}^\sharp, \sqsubseteq^\sharp)$$

Then, the following properties are equivalent:

- α is surjective (onto)
- γ is injective (into)
- $\alpha \circ \gamma = \lambda(x \in \mathbb{D}^\sharp) \cdot x$

When they hold, the Galois connection is said to be a **Galois insertion**

Intuition:

- there is no pair of distinct abstract elements with the same meaning
- less chance of losing precision by taking the "wrong" abstraction of concrete property x

Galois surjection (or Galois insertion)

Proof:

- Let us assume α surjective, i.e. $\forall y \in \mathbb{D}^\sharp, \exists x \in \mathbb{D}, \alpha(x) = y$.
  If $\gamma(x) = \gamma(y)$,
  - as α is surjective, there exist $x', y' \in \mathbb{D}$, such that $\alpha(x') = x$ and $\alpha(y') = y$
  - thus, $\gamma(\alpha(x')) = \gamma(\alpha(y'))$, which implies $x' \subseteq \gamma(\alpha(y'))$, and thus $\alpha(x') \sqsubseteq^\sharp \alpha(y')$ $(\alpha \circ \gamma \circ \alpha = \alpha)$
  - similarly $\alpha(y') \sqsubseteq^\sharp \alpha(x')$, thus $x = y$
- Let us assume γ is injective:
  Let $y \in \mathbb{D}^\sharp$; as $\gamma \circ \alpha \circ \gamma = \gamma$, we get that $\gamma \circ \alpha \circ \gamma(y) = \gamma(y)$, thus $\alpha \circ \gamma(y) = y$
- Let us assume that $\alpha \circ \gamma$ is the identity, and let $y \in \mathbb{D}^\sharp$. Then, $\alpha \circ \gamma(y) = y$, which means there exists $x \in \mathbb{D}$ such that $\alpha(x) = y$. Thus α is surjective.

Reduction of an abstraction

Quotient abstract domain

Let us consider an abstraction defined by a Galois connection

$$(\mathbb{D}, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{D}^\sharp, \sqsubseteq^\sharp)$$

We let $\equiv$ be the equivalence relation over $\mathbb{D}^\sharp$ defined by:

$$\forall x, y \in \mathbb{D}^\sharp,\ x \equiv y \iff \gamma(x) = \gamma(y)$$

We define the **quotient abstract domain** $(\mathbb{D}^\sharp_\equiv, \sqsubseteq^\sharp_\equiv)$ by:

- $\mathbb{D}^\sharp_\equiv$ is the set of equivalence classes of $\mathbb{D}^\sharp$ for $\equiv$
- $\bar{x} \sqsubseteq^\sharp_\equiv \bar{y} \iff x \sqsubseteq^\sharp y$

Proof:

- $\equiv$ is an equivalence relation, so the quotient is well-defined
- well-definedness of $\sqsubseteq^\sharp_\equiv$: exercise

Reduction of an abstraction

Reduced abstraction (sing the same notations)

The reduced abstraction is defined by the Galois connection

$$(\mathbb{D}, \subseteq) \xleftrightarrow[\alpha_{\equiv}]{\gamma_{\equiv}} (\mathbb{D}^{\sharp}_{\equiv}, \sqsubseteq^{\sharp}_{\equiv})$$

where

$$\begin{array}{rccc} \alpha_{\equiv} : & \mathbb{D} & \longrightarrow & \mathbb{D}^{\sharp}_{\equiv} \\ & x & \longmapsto & \overline{\alpha(x)} \end{array} \qquad \begin{array}{rccc} \gamma_{\equiv} : & \mathbb{D}^{\sharp}_{\equiv} & \longrightarrow & \mathbb{D} \\ & \bar{x} & \longmapsto & \gamma(x) \end{array}$$

The above Galois connection is a Galois insertion.

Proof:

- well-definedness of γ, Galois insertion property: exercises

Notes:

- the construction works even with no α
- representation of abstract element: **use representants** of equivalence classes, i.e. elements of $\mathbb{D}^{\sharp}_{\equiv}$ are **selected** elements of $\mathbb{D}^{\sharp}$

Reduction operator

Definition (using the same notations)

A reduction operator over $\mathbb{D}^\sharp$ is an operator $\rho_\equiv$ such that:

- $\forall x \in \mathbb{D}^\sharp,\ \gamma(\rho_\equiv(x)) = \gamma(x)$;
- $\forall x, y \in \mathbb{D}^\sharp,\ \gamma(x) = \gamma(y) \Longrightarrow \rho_\equiv(x) = \rho_\equiv(y)$

Such an operator allows to construct the quotient abstraction, using elements of $\mathbb{D}^\sharp$ to represent equivalence classes, thanks to the following definitions:

- $\mathbb{D}^\sharp_\equiv = \mathbb{D}^\sharp$;
- $\alpha_\equiv(x) = \rho_\equiv(\alpha(x))$
- $\gamma_\equiv(x) = \gamma(x)$

Note:

- the construction works even with no α

Example: reduction of intervals as a product

We still use:

- $\mathbb{D}$ is $\mathcal{P}(\mathbb{Z})$ and $\subseteq$ the set inclusion
- $\mathbb{D}_0^\sharp$ is $\mathbb{Z} \cup \{-\infty, +\infty\}$, $\sqsubseteq_0^\sharp$ is $\leq$ and $\alpha_0(E) = \sup E$
- $\mathbb{D}_1^\sharp$ is $\mathbb{Z} \cup \{-\infty, +\infty\}$, $\sqsubseteq_0^\sharp$ is $\leq$ and $\alpha_1(E) = \inf E$

We write $\bot = (+\infty, -\infty)$, and we let:

$$\begin{array}{rrcl} \rho_\equiv : & \mathbb{D}_\times^\sharp & \longrightarrow & \mathbb{D}_\times^\sharp \\ & (x, y) & \longmapsto & \left\{ \begin{array}{ll} (x, y) & \text{if } x \leq y \\ \bot & \text{if } x > y \end{array} \right. \end{array}$$

- $\rho_\equiv$ defines a reduction operator over $\mathbb{D}_\times^\sharp$
- this does not solve the issue of the transfer function for $x \mapsto -x$

Proof: exercise

Example: reduction of interval + congruences

We still use:

- $\mathbb{D}$ is $\mathcal{P}(\mathbb{Z})$ and $\subseteq$ the set inclusion
- $\mathbb{D}_0^\sharp$ is the interval abstract domain (an abstract values is either $\perp$ or a pair of elements of $\mathbb{Z} \cup \{-\infty, +\infty\}$)
- $\mathbb{D}_1^\sharp$ is the congruences abstract domain:
  - abstract values are $\perp$, or of the form $\langle a, b\rangle$ with $0 \leq a < b$ or $b = 0$
  - $\gamma_1(\perp) = \emptyset$ and $\gamma_1(\langle a, b\rangle) = \{a + k \cdot b \mid k \in \mathbb{Z}\}$

Exercise: define $\rho_{\equiv}$

1. reduce to $(\perp, \perp)$ when the concretization is empty:
   $\rho_{\equiv}([1, 4], \langle 0, 5\rangle) = (\perp, \perp)$
2. reduce interval bounds to match the congruence constraint
   $\rho_{\equiv}([0, 10], \langle 3, 6\rangle) = ([3, 9], \langle 3, 6\rangle)$
3. build a congruence constraint when there is none and the interval contains only one value $\rho_{\equiv}([5, 5], \langle 0, 1\rangle) = ([5, 5], \langle 5, 0\rangle)$

This solves the imprecision in the example

Example: reduction of non relational abstractions

Assumptions:

- $\mathbb{D} = \mathcal{P}(\mathbb{X} \to \mathbb{V})$, and $\subseteq$ is the inclusion order
- $\mathbb{D}^\sharp = \mathbb{X} \to \mathcal{P}(\mathbb{V})$, and $\sqsubseteq^\sharp$ is the pointwise inclusion
- α, γ define the non relational abstraction, by

$$\begin{array}{rcl} \alpha(\mathcal{E}) & = & \lambda(x \in \mathbb{X}) \cdot \{\phi(x) \mid \phi \in \mathcal{E}\} \\ \gamma(\phi^\sharp) & = & \{\phi : \mathbb{X} \to \mathbb{V} \mid \forall x \in \mathbb{X},\, \phi(x) \in \phi^\sharp(x)\} \end{array}$$

Then, for all $x \in \mathbb{X}$, if $\phi^\sharp \in \mathbb{D}^\sharp$ is such that $\phi^\sharp(x) = \emptyset$, then $\gamma(\phi^\sharp) = \emptyset$

- we let $\bot = \lambda(x \in \mathbb{X}) \cdot \emptyset$
- the reduction operator $\rho_\equiv$ is defined by (Proof: exercise):

$$\begin{array}{rccl} \rho_\equiv : & \mathbb{D}^\sharp & \longrightarrow & \mathbb{D}^\sharp \\ & \phi^\sharp & \longmapsto & \left\{ \begin{array}{ll} \phi^\sharp & \text{if } \forall x \in \mathbb{X},\, \phi^\sharp(x) \neq \emptyset \\ \bot & \text{if } \exists x \in \mathbb{X},\, \phi^\sharp(x) = \emptyset \end{array} \right. \end{array}$$

Thus, we can view non relational abstraction as **a reduced product over $\mid \mathbb{X} \mid$ instances of $(\mathcal{P}(\mathbb{V}), \subseteq)$**

Operations in the reduced domain

We define abstract operations on $\mathbb{D}^\sharp_\equiv$ from operations on $\mathbb{D}^\sharp$:

- **Least element**: if $\bot$ is the least element of $\mathbb{D}^\sharp$, then $\rho_\equiv(\bot)$ is the least element of $\mathbb{D}^\sharp_\equiv$;
- **Upper bound**: if $\sqcup$ is a sound upper bound operator on $\mathbb{D}^\sharp$ then $\sqcup_\equiv$ defined by $x \sqcup_\equiv y = \rho_\equiv(x \sqcup y)$ is a sound upper bound operator on $\mathbb{D}^\sharp_\equiv$
- **Transfer functions**:
  We assume that:
  - $\mathfrak{f} : \mathbb{D} \to \mathbb{D}$ is a concrete transfer function (e.g., describing the effect of a test or of an assignment)
  - $\mathfrak{f}^\sharp : \mathbb{D}^\sharp \to \mathbb{D}^\sharp$ is a sound transfer function with respect to $\mathfrak{f}$, that is such that $\mathfrak{f} \circ \gamma \subseteq \gamma \circ \mathfrak{f}^\sharp$

  Then, $\mathfrak{f}^\sharp_\equiv$ defined below is sound with respect to $\mathfrak{f}$:

$$\begin{array}{rccc} \mathfrak{f}^\sharp_\equiv : & \mathbb{D}^\sharp_\equiv & \longrightarrow & \mathbb{D}^\sharp_\equiv \\ & x & \longmapsto & \rho_\equiv(\mathfrak{f}^\sharp(x)) \end{array}$$

Caveat 1: widening

This construction does not work for widening

- Termination condition of ∇ on $\mathbb{D}^\sharp$:
  for all sequence $(x_n^\sharp)_{n\in\mathbb{N}}$, the sequence $(y_n^\sharp)_{n\in\mathbb{N}}$ defined below is ultimately stationary:

$$y_0^\sharp = x_0^\sharp \qquad \forall n \in \mathbb{N},\ y_{n+1}^\sharp = y_n^\sharp \nabla x_{n+1}^\sharp$$

- Applying $\rho_\equiv$ to the widening output would boil down to:

$$y_0^\sharp = \rho_\equiv(x_0^\sharp) \qquad \forall n \in \mathbb{N},\ y_{n+1}^\sharp = \rho_\equiv(y_n^\sharp \nabla x_{n+1}^\sharp)$$

  Thus the termination condition of ∇ **does not apply here**

Solution

- Simply use ∇ on $\mathbb{D}^\sharp$
- Apply reduction in the body of loops (whenever we like)

Caveat 2: reduction cost

The optimal reduction function may be computationally very costly

Approximate reduction function

An **approximate reduction operator** is an operator $\rho_{\equiv} : \mathbb{D}^\sharp \rightarrow \mathbb{D}^\sharp$ which preserves concretization:

$$\forall x^\sharp \in \mathbb{D}^\sharp,\ \gamma(\rho_{\equiv}(x^\sharp)) = \gamma(x^\sharp)$$

We can require additional conditions such as:

- idempotence: $\forall x^\sharp \in \mathbb{D}^\sharp,\ \rho_{\equiv} \circ \rho_{\equiv}(x^\sharp) = \rho_{\equiv}(x^\sharp)$
- contraction: $\forall x^\sharp \in \mathbb{D}^\sharp,\ \rho_{\equiv}(x^\sharp) \sqsubseteq^\sharp x^\sharp$

In all cases, **we may not obtain the reduced abstraction**

Reduced product abstraction

Definition

The reduced product abstraction is obtained by applying the reduction to the product abstraction

- **Examples**: as seen previously
  - intervals as products of constraints
  - intervals and congruences
  - non relational abstraction
- Abstract operators and transfer functions are defined by composition with reduction
- In many cases, only a partial reduction can be applied
  i.e., an approximation of reduced product is used

Reduced product: implementation

The modularity of the abstraction

- The whole point of reduced product is to keep the domain implementations separate
- The reduction operator should reflect that

To achieve this, we typically use a separate constraint language:

Reduced product interface

- $\mathcal{C}$ is a set of constraints with a concretization function $\gamma_{\mathcal{C}} : \mathcal{C} \to \mathbb{D}$
- $\mathbf{read}_i : \mathbb{D}_i^\sharp \to \mathcal{C}$, such that $\gamma_i(x_i^\sharp) \subseteq \gamma(\mathbf{read}_i(x_i^\sharp))$
- $\mathbf{constr}_i : \mathbb{D}_i^\sharp \times \mathcal{C} \to \mathbb{D}_i^\sharp$ such that $\gamma_i(x_i^\sharp) \cap \gamma_{\mathcal{C}}(c) \subseteq \gamma_i(\mathbf{constr}_i(x_i^\sharp, c))$

Then, a simple reduction is: $\rho_{\equiv}(x_0^\sharp, x_1^\sharp) = (x_0^\sharp, \mathbf{constr}_1(x_1^\sharp, \mathbf{read}_0(x_0^\sharp)))$

- **Example**, non relational abstraction: **read** = “is empty”
- Already demonstrated in the previous lecture

Outline

Example

We consider the program and the basic abstractions below **[CC'79]**:

```
int x = 100;
bool b = TRUE;
while(b){
    x = x - 1;
    b = x > 0;
}
```

Basic abstractions:

- possible values for b: $\{\emptyset, \{\mathcal{T}\}, \{\mathcal{F}\}, \{\mathcal{T}, \mathcal{F}\}\}$
- sign abstraction of x: $(\perp, = 0, < 0, > 0, \neq 0, \geq 0, \leq 0)$

Properties:

loop head

$$\mathtt{b} \Longrightarrow \mathtt{x} > 0$$

loop end

$$\begin{cases} \mathtt{b} \Rightarrow \mathtt{x} > 0 \\ \neg \mathtt{b} \Rightarrow \mathtt{x} = 0 \end{cases}$$

Property to establish:
$\mathtt{x} = 0$ at the end

Cardinal power abstraction

Definition

We assume $\mathbb{D} = \mathcal{P}(\mathcal{E})$, and that two abstractions are given by their concretization functions:

$$\gamma_0 : \mathbb{D}_0^\sharp \longrightarrow \mathbb{D} \qquad \gamma_1 : \mathbb{D}_1^\sharp \longrightarrow \mathbb{D}$$

We let:

- $\mathbb{D}_\rightarrow^\sharp = \mathbb{D}_0^\sharp \overset{\mathcal{M}}{\rightarrow} \mathbb{D}_1^\sharp$, set of monotone functions from $\mathbb{D}_0^\sharp$ into $\mathbb{D}_1^\sharp$
- $\sqsubseteq_\rightarrow^\sharp$ be the pointwise extension of $\sqsubseteq_1^\sharp$
- $\gamma_\rightarrow$ is defined by:

$$\begin{array}{rccl} \gamma_\rightarrow : & \mathbb{D}_\rightarrow^\sharp & \longrightarrow & \mathbb{D} \\ & \phi & \longmapsto & \{x \in \mathcal{E} \mid \forall y \in \mathbb{D}_0^\sharp, x \in \gamma_0(y) \Longrightarrow x \in \gamma_1(\phi(y))\} \end{array}$$

Then $\gamma_\rightarrow$ defines a **cardinal power abstraction**

Example

Back to the example:

- $\mathbb{D}_0^\sharp$: abstraction of the values of $\mathtt{b}$;
- $\mathbb{D}_1^\sharp$: sign abstraction of the values of $\mathtt{x}$;
- the properties needed to establish the condition on the exit states are all expressible in the cardinal power abstraction

Intuition:

- cardinal power allows to express properties of the form $\bigwedge_{i \in I}(A_i \Rightarrow B_i)$
- exercise: prove that partitioning is a cardinal power abstraction

Reduction

- In general, the cardinal power is not a reduced abstraction ($\gamma_{\rightarrow}$ not injective)
- Reduced cardinal power is obtained by composing the reduction construction

Application: control state partitioning abstraction

Assumptions:

- $\mathbb{D} = \mathcal{P}(\mathbb{S})$ where $\mathbb{S} = \mathbb{L} \times \mathbb{M}$
- $\mathbb{D}_0^\sharp = \mathbb{L} \uplus \{\bot, \top\}$
- $\mathbb{D}_1^\sharp = \mathcal{P}(\mathbb{M})$, ordered with the inclusion

Then, if Φ is an element of the reduced cardinal power,

- By reduction, $\Phi(\bot) = \emptyset$ and $\Phi(\top) = \bigcup\{\Phi(l) \mid l \in \mathbb{L}\}$
- Moreover:

$$\begin{array}{rcl} \gamma_{\rightarrow}(\Phi) & = & \{s \in \mathbb{S} \mid \forall x \in \mathbb{D}_0^\sharp,\ s \in \gamma_0(x) \Longrightarrow s \in \gamma_1(\Phi(x))\} \\ & = & \{(l, m) \in \mathbb{S} \mid m \in \gamma_1(\Phi(l))\} \end{array}$$

- Thus is the control state partitioning abstraction
- This property also holds for partitioning abstraction in general

Outline

Disjunctions in static analysis

Unusual computation of the absolute value:

```
int x ∈ ℤ;
int s;
int y;
if(x ≥ 0){
      s = 1;
} else {
      s = −1;
}
y = x/s;
```

- Interval abstraction:
  - after the **if**, $s \in [-1, 1]$
  - **possible division by** 0
- Same with polyedra, octagons (convex abstractions)
- Interval + congruences would work

What if we want to use intervals only ?
Disjunctions are needed

Disjunctive completion

Definition

We consider an abstraction defined by a concretization function $\gamma : (\mathbb{D}^\sharp, \sqsubseteq^\sharp) \longrightarrow (\mathbb{D}, \subseteq)$.
The disjunctive completion abstraction is defined by:

- $\mathbb{D}^\sharp_\vee = \mathcal{P}(\mathbb{D}^\sharp)$
- $\sqsubseteq^\sharp_\vee$ is defined by:

$$\mathcal{E}^\sharp \sqsubseteq^\sharp_\vee \mathcal{F}^\sharp \iff \forall e^\sharp \in \mathcal{E}^\sharp, \exists f^\sharp \in \mathcal{F}^\sharp, e^\sharp \sqsubseteq^\sharp f^\sharp$$

- $\forall \mathcal{E}^\sharp \in \mathbb{D}, \gamma_\vee(\mathcal{E}^\sharp) = \bigcup \{\gamma(e^\sharp) \mid e^\sharp \in \mathcal{E}^\sharp\}$
- $\forall x \in \mathbb{D}, \alpha_\vee(x) = \{e^\sharp \in \mathbb{D}^\sharp \mid x \subseteq \gamma(e^\sharp)\}$

These define a Galois connection $(\mathbb{D}, \subseteq) \underset{\alpha_\vee}{\overset{\gamma_\vee}{\leftrightarrows}} (\mathbb{D}^\sharp_\vee, \sqsubseteq^\sharp_\vee)$

- **Proof**: exercise

State partitioning

- **Disjunctive completion** has **several severe limitations**:
  - analyses may manipulate huge abstract states
  - no obvious widening: has to be defined on a per case basis
    it may be non trivial to define one
  - this abstraction ignores properties of the system to analyze
- **Partitioning allows to express disjunctions too**

Flashback: partitioning abstraction

Given set E and a partition $\mathfrak{P}$ of E, we let the **partitioning abstraction** over E be defined by:

$$\begin{array}{rlcl} \gamma_{\rm part}: & (\mathfrak{P} \rightarrow \mathcal{P}(E)) & \longrightarrow & \mathcal{P}(E) \\ & \Phi & \longmapsto & \bigcup_{p \in \mathfrak{P}} \Phi(p) \end{array}$$

- Advantages:
  - the size of disjunctions is bounded by $\mathfrak{P}$
  - the choice of $\mathfrak{P}$ can exploit problem properties

State partitioning based on values

Back to our example, **we design a cardinal power abstraction**:

```
int x ∈ ℤ;
int s;
int y;
if(x ≥ 0){
      s = 1;
}else{
      s = −1;
}
y = x/s;
```

- $\mathbb{D}_0^\sharp$: interval of x
- $\mathbb{D}_1^\sharp$: intervals for all variables
- Property at the end of the **if**:

$$\left\{ \begin{array}{rcl} \mathrm{x} \in [0, +\infty[& \Rightarrow & \mathrm{s} = 1 \wedge \ldots \\ \mathrm{x} \in]-\infty, -1] & \Rightarrow & \mathrm{s} = -1 \wedge \ldots \end{array} \right.$$

- Some of the issues of disjunctive completion remain:
  in particular, no obvious widening...
- Representing the full cardinal power is too costly:
  limit the number of partitions

Transfer functions

```
int $x \in \mathbb{Z}$;
int $s$;
int $y$;
if($x \geq 0$){
    $s = -1$;
} else {
    $s = 1$;
}
①$x = -x$;
②$y = x/s$;
```

- At ①:

$$\begin{cases} x \in [0, +\infty[& \Rightarrow \quad s = -1 \wedge \ldots \\ x \in]-\infty, -1] & \Rightarrow \quad s = 1 \wedge \ldots \end{cases}$$

- At ②:

$$\begin{cases} x \in [1, +\infty[& \Rightarrow \quad s = 1 \wedge \ldots \\ x \in]-\infty, 0] & \Rightarrow \quad s = -1 \wedge \ldots \end{cases}$$

Most abstract transfer functions may modify both sides of the cardinal power:

- The assignment to x modifies the abstraction in the left hand side of the cardinal power
- Thus partitions need to be recomputed: costly operation

Trace partitioning abstraction: example

Alternate way to look at the example:

```
int x ∈ ℤ;
int s;
int y;
if(x ≥ 0){
      s = 1;
} else {
      s = −1;
}
①y = x/s;
```

At ①:

- if the execution went through the `TRUE` branch of the **if**:

$$x \in [0, +\infty[\wedge s = 1 \wedge$$

- if the execution went through the `FALSE` branch of the **if**:

$$x \in]-\infty, -1] \wedge s = -1 \wedge$$

- This abstraction should be formalized as an abstraction of traces, not states

Trace partitioning abstraction: formalization

```
$l_0$ :  int $x \in \mathbb{Z}$;
$l_1$ :  int $s$;
$l_2$ :  int $y$;
$l_3$ :  if($x \geq 0$){
$l_4$ :        $s = 1$;
$l_5$ :  } else {
$l_6$ :        $s = -1$;
$l_7$ :  }
$l_8$ :  $y = x/s$;
```

- **Trace domain** $\mathbb{D}_0^\sharp$:

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
{[l_4]} \arrow[dr, dash] & & {[l_6]} \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

- **Concretization** γ_0:

$$\gamma_0 : \begin{array}{rcl} [l_4] & \mapsto & \{\langle \ldots, (l_4, m), \ldots \rangle \in \mathbb{S}^\star\} \\ [l_6] & \mapsto & \{\langle \ldots, (l_6, m), \ldots \rangle \in \mathbb{S}^\star\} \end{array}$$

- **Right hand side abstraction**: $(\mathcal{P}(\mathbb{S}), \subseteq)$, with abstraction defined by $(\alpha_{\mathcal{R}}, \gamma_{\mathcal{R}})$

Trace partitioning abstraction: definition

Definition: static trace partitioning

Let $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$ be a *finite* abstraction of sets of traces, defined by a Galois connection:

$$(\mathcal{P}(\mathbb{S}^\star), \subseteq) \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} (\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$$

It defines a **static trace partitioning abstraction** by reduced cardinal power over the reachability abstraction.

There are many ways to instantiate $\mathbb{D}^\sharp_0$:

Trace partitioning criteria

- control flow based criteria:
  - branch taken in a **if** statement
  - number of times a **while** body was executed
- value of some variable at a given point
- conjunctions of such criteria

Trace partitioning transfer functions

We assume $\mathbb{D}_0^\sharp$ is finite (case $\mathbb{D}_0^\sharp$ is infinite: dynamic partitioning, see later)

Static partitioning composed with state abstraction

By composing a state abstraction $(\mathcal{P}(\mathbb{S}), \subseteq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$, and applying the same reduced cardinal power abstraction, we get a new instance of the static trace partitioning abstraction

- **Least element**: $\lambda(x^\sharp \in \mathbb{D}_0^\sharp) \cdot \bot_1$
- **Upper bound**: $\phi^\sharp \sqcup \psi^\sharp ::= \lambda(x^\sharp \in \mathbb{D}_0^\sharp) \cdot (\phi^\sharp(x^\sharp) \sqcup_1 \psi^\sharp(x^\sharp))$
- **Widening operator**: similar definition
- **Transfer functions** with **no partition change**: We assume that:
  - $\mathfrak{f} : \mathbb{D} \to \mathbb{D}$ is a concrete transfer function (e.g., describing the effect of a test or of an assignment)
  - $\mathfrak{f}_1^\sharp : \mathbb{D}_1^\sharp \to \mathbb{D}_1^\sharp$ is a sound transfer function with respect to $\mathfrak{f}$, that is such that $\mathfrak{f} \circ \gamma \subseteq \gamma \circ \mathfrak{f}_1^\sharp$

  Then, $\lambda(x^\sharp \in \mathbb{D}_0^\sharp) \cdot \mathfrak{f}_1^\sharp$ is sound with respect to $\mathfrak{f}$

Transfer functions in the trace partitioning domain

Control history based partitioning:

Abstract partition matching

A sound abstract partition matching is a family of relations $(\rightarrow^\sharp_{l,l'})_{l,l' \in \mathbb{L}}$ where $\rightarrow^\sharp_{l,l'} \subseteq (\mathbb{D}^\sharp_0)^2$, such that:

$$\left.\begin{array}{l} \langle (l_0, m_0), \ldots, (l_n, m_n) \rangle \in \gamma_0(x^\sharp) \\ \wedge\ x^\sharp \rightarrow^\sharp_{l_n, l_{n+1}} y^\sharp \end{array}\right\} \Rightarrow \langle (l_0, m_0), \ldots, (l_{n+1}, m_{n+1}) \rangle \in \gamma_0(y^\sharp)$$

Analysis of a transition

Given a sound abstract partition matching $\rightarrow_{l,l'}$, and sound transfer function $\mathfrak{f}_{l,l'} : \mathbb{D}^\sharp_1 \rightarrow \mathbb{D}^\sharp_1$ in the underlying domain, the transfer function below in the trace partitioning domain is sound:

$$\phi^\sharp \longmapsto \lambda(x^\sharp \in \mathbb{D}^\sharp_0) \cdot \sqcup_1 \{ \mathfrak{f}_{l,l'}(\phi^\sharp(y^\sharp)) \mid y^\sharp \rightarrow_{l,l'} x^\sharp \}$$

Creation and fusion of trace partitions

- **Proof** of soundness: exercise
- **Typical choice for the abstract partition matching**:
  - at most points, the partitions are unchanged
    i.e., $\rightarrow_{l,l'}$ is the identity relation
  - at points where partitions should be merged, it reflects creation of partitions or fusion of partitions
- **Other partitioning criteria**: should provide similar operations on partitions

Dynamic partitioning

Principle:

- the domain of partitions depends on the context
- can be applied to state partitioning, trace partitioning...
  - in trace partitioning, this corresponds to cases where $\mathbb{D}_0^\sharp$ is infinite
  - indeed, only a finite number of partitions can be represented at any point in the analysis; this set is dynamic (i.e., also determined as a result of the analysis)

Formalization: cofibered abstract domain **[AV]**, **[MR'05]**

Outline

Main points of the lecture

There exists many techniques to combine abstract domains into more interesting ones

- **Product, reduced product**:
  conjunctions of abstract properties
- **Partitioning, disjunctive completion**:
  disjunctions of abstrct properties
- The list is not exhaustive

Advantages

- **Modular** design of static analyzers
- A same construction **may be used in many contexts**

Bibliography: abstract domain combination

- **[CC'79]**: **Systematic design of program analysis frameworks**.
  Patrick Cousot and Radhia Cousot. In POPL, 1979.
- **[B'90]**: **Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity**.
  Fran cois Bourdoncle. In PLILP, 1990.
- **[CC'92]**: **Abstract interpretation and application to logic programs**.
  Patrick cousot and Radhia Cousot. In Journal of Logic Programming, 1992.
- **[AV]**: **Abstract cofibered domains: application to the alias analysis of untyped programs**.
  Arnaud Venet. In SAS, 1996.
- **[MR'05]**: **Trace partitioning in abstract interpretation static analyzers**.
  Laurent Mauborgne and Xavier Rival. In ESOP, 2005.