Partitioning abstractions

MPRI — Cours 2.6 "Interprétation abstraite : application à la vérification et à l'analyse statique"

Xavier Rival

INRIA, ENS, CNRS

Dec, 18th. 2015

Towards disjunctive abstractions

Extending the expressiveness of abstract domains

- **disjunctions** are **often needed**...
- ... but **potentially costly**

In this lecture, we will discuss:

- **precision issues** that motivate the use of abstract domains able to **express disjunctions**
- **several ways** to **express disjunctions** using **abstract domain combiners**
  - disjunctive completion
  - cardinal power
  - state partitioning
  - trace partitioning

Domain combinators (or combiners)

General combination of abstract domains

- takes one or more abstract domains as **inputs**
- produces a **new abstract domain**

Input and output abstract domains are **characterized by an "interface"**: concrete domain, abstraction relation, abstract elements and operators

Advantages:

- **general definition**, formalized and proved once
- can be **implemented** in a separate way, e.g., in ML:
  - abstract domain: **module**

    ```
    module D = (struct ... end: Interface)
    ```
  - abstract domain combinator: **functor**

    ```
    module C = functor (D: Interface) ->
      (struct ... end: Interface)
    ```

Example: product abstraction

Set notations:

- $\mathbb{V}$: values
- $\mathbb{X}$: variables
- $\mathbb{M}$: stores
  $\mathbb{M} = \mathbb{X} \to \mathbb{V}$

Assumptions:

- concrete domain $(\mathcal{P}(\mathbb{M}), \subseteq)$ with $\mathbb{M} = \mathbb{X} \to \mathbb{V}$
- we assume an abstract domain $\mathbb{D}^\sharp$ that provides
  - **concretization function** $\gamma : \mathbb{D}^\sharp \to \mathcal{P}(\mathbb{M})$
  - **element** $\bot$ **with empty concretization** $\gamma(\bot) = \emptyset$

Product combinator (implemented as a functor)

Given abstract domains $(\mathbb{D}^\sharp_0, \gamma_0, \bot_0)$ and $(\mathbb{D}^\sharp_1, \gamma_1, \bot_1)$, the **product abstraction** is $(\mathbb{D}^\sharp_\times, \gamma_\times, \bot_\times)$ where:

- $\mathbb{D}^\sharp_\times = \mathbb{D}^\sharp_0 \times \mathbb{D}^\sharp_1$
- $\gamma_\times(x^\sharp_0, x^\sharp_1) = \gamma_0(x^\sharp_0) \cap \gamma_1(x^\sharp_1)$
- $\bot_\times = (\bot_0, \bot_1)$

This amounts to expressing conjunctions of elements of $\mathbb{D}^\sharp_0$ and $\mathbb{D}^\sharp_1$

Example: product abstraction, coalescent product

The product abstraction is not very precise and **needs a reduction**:

$$\forall x_0^\sharp \in \mathbb{D}_0^\sharp, x_1^\sharp \in \mathbb{D}_1^\sharp,\ \gamma_\times(\bot_0, x_1^\sharp) = \gamma_\times(x_0^\sharp, \bot_1) = \emptyset = \gamma_\times(\bot_\times)$$

Coalescent product

Given abstract domains $(\mathbb{D}_0^\sharp, \gamma_0, \bot_0)$ and $(\mathbb{D}_1^\sharp, \gamma_1, \bot_1)$, the **coalescent product abstraction** is $(\mathbb{D}_\times^\sharp, \gamma_\times, \bot_\times)$ where:

- $\mathbb{D}_\times^\sharp = \{\bot_\times\} \uplus \{(x_0^\sharp, x_1^\sharp) \in \mathbb{D}_0^\sharp \times \mathbb{D}_1^\sharp \mid x_0^\sharp \neq \bot_0 \wedge x_1^\sharp \neq \bot_1\}$
- $\gamma_\times(\bot_\times) = \emptyset$, $\gamma_\times(x_0^\sharp, x_1^\sharp) = \gamma_0(x_0^\sharp) \cap \gamma_1(x_1^\sharp)$

In many cases, this is **not enough to achieve reduction**:

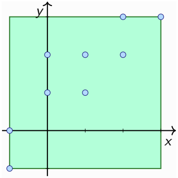
- let $\mathbb{D}_0^\sharp$ be the interval abstraction, $\mathbb{D}_1^\sharp$ be the congruences abstraction
- $\gamma_\times(\{\mathtt{x} \in [3, 4]\}, \{\mathtt{x} \equiv 0 \mod 5\}) = \emptyset$

- how to define abstract domain combiners to **add disjunctions** ?

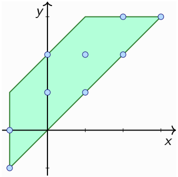
Outline

Convex abstractions

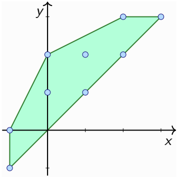
Many numerical abstractions describe **convex sets of points**



interval domain

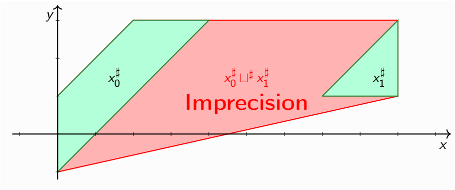


octagon domain



polyedra domain

Imprecisions inherent in the **convexity**, and when computing **abstract join** (over-approximation of concrete union):



Such imprecisions may impact analysis results

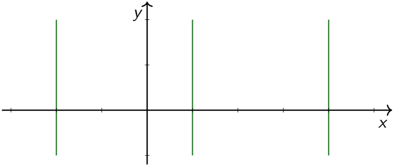
Non convex abstractions

We consider abstractions of $\mathbb{D} = \mathcal{P}(\mathbb{Z})$

Congruences:

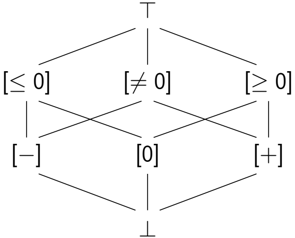
- $\mathbb{D}^\sharp = \mathbb{Z} \times \mathbb{N}$
- $\gamma(n, k) = \{n + k \cdot p \mid p \in \mathbb{Z}\}$
- $-2 \in \gamma(1, 2)$ and $1 \in \gamma(1, 2)$ but $0 \notin \gamma(1, 2)$

Non relational product two variables



Signs:

- $0 \notin \gamma([\neq 0])$ so $[\neq 0]$ describes a non convex set
- other abstract elements describe convex sets



Example 1: verification problem

```
bool b0, b1;
int x, y;        (uninitialized)
b0 = x ≥ 0;
b1 = x ≤ 0;
if(b0 && b1){
    y = 0;
} else {
①   y = 100/x;
}
```

- if $\neg b_0$, then $x < 0$
- if $\neg b_1$, then $x > 0$
- if either b_0 or b_1 is false, then $x \neq 0$
- thus, if point ① is reached the division is safe

How to verify the division operation ?

- Non relational abstraction (e.g., intervals), at point ①:

$$\left\{ \begin{array}{c} b_0 = \mathtt{FALSE} \vee b_1 = \mathtt{FALSE} \\ x : \top \end{array} \right.$$

- Signs, congruences do not help:
  in the concrete, x may take any value but 0

Example 1: program annotated with local invariants

```
bool b0, b1;
int x, y;        (uninitialized)
b0 = x ≥ 0;
            (b0 ∧ x ≥ 0) ∨ (¬b0 ∧ x < 0)
b1 = x ≤ 0;
            (b0 ∧ b1 ∧ x = 0) ∨ (b0 ∧ ¬b1 ∧ x > 0) ∨ (¬b0 ∧ b1 ∧ x < 0)
if(b0 && b1){
            (b0 ∧ b1 ∧ x = 0)
    y = 0;
            (b0 ∧ b1 ∧ x = 0 ∧ y = 0)
} else {
            (b0 ∧ ¬b1 ∧ x > 0) ∨ (¬b0 ∧ b1 ∧ x < 0)
    y = 100/x;
            (b0 ∧ ¬b1 ∧ x > 0) ∨ (¬b0 ∧ b1 ∧ x < 0)
}
```

The obvious way to sucessfully analyzing this program consists in **adding symbolic disjunctions** to our abstract domain

Example 2: verification problem

```
    int x ∈ ℤ;
    int s;
    int y;
    if(x ≥ 0){
        s = 1;
    } else {
        s = −1;
    }
①   y = x/s;
②   assert(y ≥ 0);
```

- s is either 1 or -1
- thus, the division at ① should not fail
- moreover s has the same sign as x
- thus, the value stored in y should always be positive at ②

- **How to verify the division operation ?**
- In the concrete, s is **always non null**:
  convex abstractions **cannot** establish this; **congruences** can
- Moreover, s has always the **same sign** as x
  expressing this would require a non trivial numerical abstraction

Example 2: program annotated with local invariants

$$
\begin{array}{lll}
 & \textbf{int}\ \mathrm{x} \in \mathbb{Z}; & \\
 & \textbf{int}\ \mathrm{s}; & \\
 & \textbf{int}\ \mathrm{y}; & \\
 & \textbf{if}(\mathrm{x} \geq 0)\{ & \\
 & & (\mathrm{x} \geq 0) \\
 & \quad \mathrm{s} = 1; & \\
 & & (\mathrm{x} \geq 0 \wedge \mathrm{s} = 1) \\
 & \}\ \textbf{else}\ \{ & \\
 & & (\mathrm{x} < 0) \\
 & \quad \mathrm{s} = -1; & \\
 & & (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \\
 & \} & \\
 & & (\mathrm{x} \geq 0 \wedge \mathrm{s} = 1) \vee (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \\
① & \mathrm{y} = \mathrm{x}/\mathrm{s}; & \\
 & & (\mathrm{x} \geq 0 \wedge \mathrm{s} = 1 \wedge \mathrm{y} \geq 0) \vee (\mathrm{x} < 0 \wedge \mathrm{s} = -1 \wedge \mathrm{y} > 0) \\
② & \textbf{assert}(\mathrm{y} \geq 0); &
\end{array}
$$

Again, the obvious solution consists in
adding disjunctions to our abstract domain

Outline

Distributive abstract domain

Principle:

1. consider concrete domain $(\mathbb{D}, \sqsubseteq)$, with lower upper bound operator $\sqcup$
2. assume an abstract domain $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ with concretization $\gamma : \mathbb{D}^\sharp \rightarrow \mathbb{D}$
3. build a domain containing **all the disjunctions** of elements of $\mathbb{D}^\sharp$

Definition: distributive abstract domain

Abstract domain $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ with concretization function $\gamma : \mathbb{D}^\sharp \rightarrow \mathbb{D}$ is **distributive** (or **complete for disjunction**) if and only if:

$$\forall \mathcal{E} \subseteq \mathbb{D}^\sharp,\ \exists x^\sharp \in \mathbb{D}^\sharp,\ \gamma(x^\sharp) = \bigsqcup_{y^\sharp \in \mathcal{E}} \gamma(y^\sharp)$$

Examples:

- the lattice $\{\bot, < 0, = 0, > 0, \leq 0, \neq 0, \geq 0, \top\}$ is distributive
- the lattice of intervals is not distributive:
  there is no interval with concretization $\gamma([0, 10]) \cup \gamma([12, 20])$

Definition

Definition: disjunctive completion

The **disjunctive completion** of abstract domain $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ with concretization function $\gamma : \mathbb{D}^\sharp \to \mathbb{D}$ is the **smallest abstract domain** $(\mathbb{D}^\sharp_{\mathbf{disj}}, \sqsubseteq^\sharp_{\mathbf{disj}})$ with concretization function $\gamma_{\mathbf{disj}} : \mathbb{D}^\sharp_{\mathbf{disj}} \to \mathbb{D}$ such that:

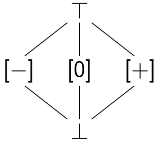
- $\mathbb{D}^\sharp \subseteq \mathbb{D}^\sharp_{\mathbf{disj}}$
- $\forall x^\sharp \in \mathbb{D}^\sharp,\ \gamma_{\mathbf{disj}}(x^\sharp) = \gamma(x^\sharp)$
- $(\mathbb{D}^\sharp_{\mathbf{disj}}, \sqsubseteq^\sharp_{\mathbf{disj}})$ with concretization $\gamma_{\mathbf{disj}}$ is distributive

Building a disjunctive completion domain:

- start with $\mathbb{D}^\sharp_{\mathbf{disj}} = \mathbb{D}^\sharp$
- for all set $\mathcal{E} \subseteq \mathbb{D}^\sharp$ such that there is no $x^\sharp \in \mathbb{D}^\sharp$, such that $\gamma(x^\sharp) = \bigsqcup_{y^\sharp \in \mathcal{E}} \gamma(y^\sharp)$, add $[\sqcup\mathcal{E}]$ to $\mathbb{D}^\sharp_{\mathbf{disj}}$, and extend $\gamma_{\mathbf{disj}}$ by
$$\gamma_{\mathbf{disj}}([\sqcup\mathcal{E}]) = \bigsqcup_{y^\sharp \in \mathcal{E}} \gamma(y^\sharp)$$

Example 1: completion of signs

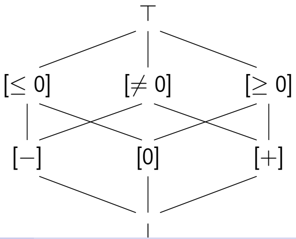
We consider **concrete lattice** $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$ and $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ defined by:



$$\gamma : \begin{array}{lcl} \bot & \longmapsto & \emptyset \\ [< 0] & \longmapsto & \{k \in \mathbb{Z} \mid k < 0\} \\ [= 0] & \longmapsto & \{k \in \mathbb{Z} \mid k = 0\} \\ [> 0] & \longmapsto & \{k \in \mathbb{Z} \mid k > 0\} \\ \top & \longmapsto & \mathbb{Z} \end{array}$$

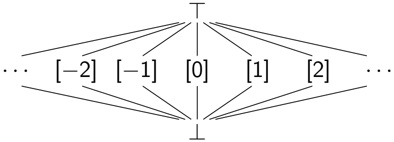
Then, the disjunctive completion is defined by adding elements corresponding to:

- $\{[-], [0]\}$
- $\{[-], [+]\}$
- $\{[0], [+]\}$



Example 2: completion of constants

We consider **concrete lattice** $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
and $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ defined by:



$$\gamma : \begin{array}{ccc} \bot & \longmapsto & \emptyset \\ \{k\} & \longmapsto & \{k\} \\ \top & \longmapsto & \mathbb{Z} \end{array}$$

Then, the disjunctive completion coincides with **the power-set**:

- $\mathbb{D}^\sharp_{\mathbf{disj}} \equiv \mathcal{P}(\mathbb{Z})$
- $\gamma_{\mathbf{disj}}$ is the **identity function !**
- this lattice contains **infinite sets which are not representable**

Example 3: completion of intervals

We consider concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
and let $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ the domain of intervals

- $\mathbb{D}^\sharp = \{\bot, \top\} \uplus \{[a, b] \mid a \leq b\}$
- $\gamma([a, b]) = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$

Then, the disjunctive completion is the set of **unions of intervals** :

- $\mathbb{D}^\sharp_{\textbf{disj}}$ collects all the families of disjoint intervals
- this lattice contains **infinite sets which are not representable**
- as expressive as the completion of constants, but more efficient representation

The disjunctive completion of $(\mathbb{D}^\sharp)^n$ is **not equivalent** to $(\mathbb{D}^\sharp_{\textbf{disj}})^n$

- which is more expressive ?
- show it on an example !

Example 3: completion of intervals and verification

We use the disjunctive completion of $(\mathbb{D}^\sharp)^3$.
The invariants below can be expressed in the disjunctive completion:

$$
\begin{array}{l}
\textbf{int } \mathrm{x} \in \mathbb{Z}; \\
\textbf{int } \mathrm{s}; \\
\textbf{int } \mathrm{y}; \\
\textbf{if}(\mathrm{x} \geq 0)\{ \\
\qquad\qquad (\mathrm{x} \geq 0) \\
\qquad \mathrm{s} = 1; \\
\qquad\qquad (\mathrm{x} \geq 0 \wedge \mathrm{s} = 1) \\
\}\,\textbf{else}\,\{ \\
\qquad\qquad (\mathrm{x} < 0) \\
\qquad \mathrm{s} = -1; \\
\qquad\qquad (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \\
\} \\
\qquad\qquad (\mathrm{x} \geq 0 \wedge \mathrm{s} = 1) \vee (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \\
\mathrm{y} = \mathrm{x}/\mathrm{s}; \\
\qquad\qquad (\mathrm{x} \geq 0 \wedge \mathrm{s} = 1 \wedge \mathrm{y} \geq 0) \vee (\mathrm{x} < 0 \wedge \mathrm{s} = -1 \wedge \mathrm{y} > 0) \\
\textbf{assert}(\mathrm{y} \geq 0);
\end{array}
$$

Static analysis with disjunctive completion

Transfer functions for the computation of **abstract post-conditions**:

- we assume a concrete post-condition operation (assingment, guard...) $post : \mathbb{D} \rightarrow \mathbb{D}$, and an abstract $post^\sharp : \mathbb{D}^\sharp \rightarrow \mathbb{D}^\sharp$ such that:

$$post \circ \gamma \sqsubseteq \gamma \circ post^\sharp$$

- then, we can simply use, **for the disjunctive completion domain:**

$$post^\sharp_{\mathbf{disj}}([\sqcup \mathcal{E}]) = \begin{cases} y^\sharp \text{ if } \gamma(y^\sharp) = \sqcup\{post^\sharp(x^\sharp) \mid x^\sharp \in \mathcal{E}\} \\ [\sqcup\{post^\sharp(x^\sharp) \mid x^\sharp \in \mathcal{E}\}] \text{ otherwise} \end{cases}$$

Abstract join:

- disjunctive completion provides **an exact join** (exercise !)

Inclusion check: exercise !

Limitations of disjunctive completion

- **Combinatorial explosion**:
  - if $\mathbb{D}^\sharp$ is infinite, $\mathbb{D}^\sharp_{\mathbf{disj}}$ may have elements that **cannot be represented**
    e.g., completion of constants or intervals
  - even when $\mathbb{D}^\sharp$ is finite, $\mathbb{D}^\sharp_{\mathbf{disj}}$ may be **huge**
    in the worst case, if $\mathbb{D}^\sharp$ has n elements, $\mathbb{D}^\sharp_{\mathbf{disj}}$ may have 2^n elements
- **Many elements useless in practice**:
  disjunctive completion of intervals: may express any set of integers...
- **No general definition of a widening operator**
  - most common approach to achieve that: ***k*-limiting**
    bound the numbers of disjuncts
    i.e., the size of the sets added to the base domain
  - **remaining issue:** the join operator should "select" which disjoints to merge

Outline

Principle

Observation

Disjuncts **that are required for static analysis**
can usually be **characterized** by some **semantic property**

Examples:

- **sign** of a variable
- **value** of a **boolean** variable
- **execution path**, e.g., side of a condition that was visited

Solution: perform a kind of **indexing** of disjuncts

- use an abstraction to **describe labels**
  e.g., sign of a variable, value of a boolean, or trace property...
- apply the abstraction that needs be completed on the images

Disjuncts indexing: example

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

- natural "indexing": **sign of** x
- but we could also rely on the **sign of** s

Cardinal power abstraction

We assume $(\mathbb{D}, \sqsubseteq) = (\mathcal{P}(\mathcal{E}), \subseteq)$, and two abstractions $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp), (\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$ given by their concretization functions:

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

Definition

We let the **cardinal power abstract domain** be defined by:

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

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

We sometimes denote it by $\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp$, $\gamma_{\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp}$ to make it more explicit.

Use of cardinal power abstractions

Intuition: cardinal power expresses properties of the form

$$\left\{\begin{array}{cccc} & p_0 & \Longrightarrow & p'_0 \\ \wedge & p_1 & \Longrightarrow & p'_1 \\ \vdots & \vdots & \vdots & \vdots \\ \wedge & p_n & \Longrightarrow & p'_n \end{array}\right.$$

Two independent choices:

1. $\mathbb{D}^\sharp_0$: **set of partitions** (the "labels")
2. $\mathbb{D}^\sharp_1$: **abstraction of sets of states**, e.g., a numerical abstraction

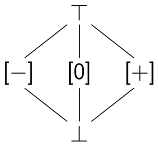
Application $(\mathtt{x} \geq 0 \wedge \mathtt{s} = 1 \wedge \mathtt{y} \geq 0) \vee (\mathtt{x} < 0 \wedge \mathtt{s} = -1 \wedge \mathtt{y} > 0)$

- $\mathbb{D}^\sharp_0$: sign of $\mathtt{s}$
- $\mathbb{D}^\sharp_1$: other constraints

Another example, with a single variable

Assumptions:

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
- $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp)$ be the lattice of signs (strict values only)
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$ be the lattice of intervals



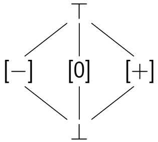
Example abstract values:

- $[0, 8]$ is expressed by: $\begin{cases} \bot \longmapsto \bot_1 \\ [-] \longmapsto \bot_1 \\ [0] \longmapsto [0, 0] \\ [+] \longmapsto [1, 8] \\ \top \longmapsto [0, 8] \end{cases}$

- $[-10, -3] \uplus [7, 10]$ is expressed by: $\begin{cases} \bot \longmapsto \bot_1 \\ [-] \longmapsto [-10, -3] \\ [0] \longmapsto \bot_1 \\ [+] \longmapsto [7, 10] \\ \top \longmapsto [-10, 10] \end{cases}$

Example reduction (1): relation between the two domains

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
- $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp)$ be the **lattice of signs**
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$ be the **lattice of intervals**



We let:

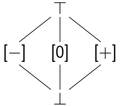
$$X^\sharp = \left\{ \begin{array}{lcl} \bot & \longmapsto & \bot_1 \\ [-] & \longmapsto & [1,8] \\ [0] & \longmapsto & [1,8] \\ [+] & \longmapsto & \bot_1 \\ \top & \longmapsto & [1,8] \end{array} \right. \qquad Y^\sharp = \left\{ \begin{array}{lcl} \bot & \longmapsto & \bot_1 \\ [-] & \longmapsto & [2,45] \\ [0] & \longmapsto & [-5,-2] \\ [+] & \longmapsto & [-5,-2] \\ \top & \longmapsto & \top_1 \end{array} \right. \qquad Z^\sharp = \left\{ \begin{array}{lcl} \bot & \longmapsto & \bot_1 \\ [-] & \longmapsto & \bot_1 \\ [0] & \longmapsto & \bot_1 \\ [+] & \longmapsto & \bot_1 \\ \top & \longmapsto & \bot_1 \end{array} \right.$$

Then,

$$\gamma_{\mathbf{cp}}(X^\sharp) = \gamma_{\mathbf{cp}}(Y^\sharp) = \gamma_{\mathbf{cp}}(Z^\sharp) = \emptyset$$

Example reduction (2): tightening disjunctions

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
- $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$ be the **lattice of signs**
- $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1)$ be the **lattice of intervals**



We let:
$$X^\sharp = \begin{cases} \bot & \longmapsto & \bot_1 \\ [-] & \longmapsto & [-5,-1] \\ [0] & \longmapsto & [0,0] \\ [+] & \longmapsto & [1,5] \\ \top & \longmapsto & [-10,10] \end{cases} \qquad Y^\sharp = \begin{cases} \bot & \longmapsto & \bot_1 \\ [-] & \longmapsto & [-5,-1] \\ [0] & \longmapsto & [0,0] \\ [+] & \longmapsto & [1,5] \\ \top & \longmapsto & [-5,5] \end{cases}$$

- Then, $\gamma_{\mathbf{cp}}(X^\sharp) = \gamma_{\mathbf{cp}}(Y^\sharp)$
- $\gamma_0([-]) \cup \gamma_0([0]) \cup \gamma([+]) = \gamma(\top)$
  but
  $$\gamma_0(X^\sharp([-])) \cup \gamma_0(X^\sharp([0])) \cup \gamma(X^\sharp([+])) \subset \gamma(X^\sharp(\top))$$
  In fact, **we can improve the image of $\top$ into $[-5,5]$**

Reduction, and improving precision in the cardinal power

In general, **the cardinal power construction requires reduction**

Strengthening using both sides of $\Rightarrow$

Tightening of $y_0^\sharp \mapsto y_1^\sharp$ when:

- $\exists z_1^\sharp \neq y_1^\sharp,\ \gamma(y_1^\sharp) \cap \gamma(y_0^\sharp) \subseteq \gamma(z_1^\sharp)$

- in the example, $z_1^\sharp = \bot_1$...

Strengthening of one mapping using other mappings

Tightening of mapping $(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\}) \mapsto x_1^\sharp$ when:

- $\bigcup\{\gamma_0(z^\sharp) \mid z^\sharp \in \mathcal{E}\} = \gamma_0(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\})$
- $\exists y^\sharp,\ \bigcup\{\gamma_1(X^\sharp(z^\sharp)) \mid z^\sharp \in \mathcal{E}\} \subseteq \gamma_1(y^\sharp) \subset \gamma_1(X^\sharp(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\}))$

- in the example, we use a set of elements that cover $\top$...

Representation of the cardinal power

Basic ML representation:

- using **functions**, i.e. `type cp = d0 -> d1`
  $\Rightarrow$ obviously a bad choice, as it makes it hard to operate in the $\mathbb{D}_0^\sharp$ side
- using **some kind of dictionnaries** `type cp = (d0,d1) map` $\Rightarrow$ better, but not straightforward...

The latter is not a very efficient representation:

- if $\mathbb{D}_0^\sharp$ has N elements, then an abstract value in $\mathbb{D}_{\mathbf{cp}}^\sharp$ requires **N elements of $\mathbb{D}_1^\sharp$**
- if $\mathbb{D}_0^\sharp$ is infinite, and $\mathbb{D}_1^\sharp$ is non trivial, then $\mathbb{D}_{\mathbf{cp}}^\sharp$ **has elements that cannot be represented**
- the 1st reduction shows it is **unnecessary to represent bindings for all elements of $\mathbb{D}_0^\sharp$**
  example: this is the case of $\bot_0$

More compact representation of the cardinal power

Principle:

- use a **dictionnary data-type** (most likely functional arrays)
- **avoid representing information attached to redundant elements**

Compact representation

Reduced cardinal power of $\mathbb{D}_0^\sharp$ and $\mathbb{D}_1^\sharp$ can be represented by considering only a subset $\mathcal{C} \subseteq \mathbb{D}_0^\sharp$ where

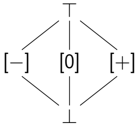
$$\forall x^\sharp \in \mathbb{D}_0^\sharp,\ \exists \mathcal{E} \subseteq \mathcal{C},\ \gamma_0(x^\sharp) = \cup\{\gamma_0(y^\sharp) \mid y^\sharp \in \mathcal{E}\}$$

In particular:

- if possible, $\mathcal{C}$ should be **minimal**
- in any case, $\perp_0 \notin \mathcal{C}$

Example: compact cardinal power over signs

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
- $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$ be the **lattice of signs**
- $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1)$ be the **lattice of intervals**



Observations

- $\bot$ does not need be considered (obvious right hand side: $\bot_1$)
- $\gamma_0([<0]) \cup \gamma_0([=0]) \cup \gamma([>0]) = \gamma(\top)$ thus $\top$ does not need be considered

Thus, we let $\mathcal{C} = \{[-], [0], [+]\}$

- $[0, 8]$ is expressed by: $\begin{cases} [-] & \longmapsto & \bot_1 \\ [0] & \longmapsto & [0, 0] \\ [+] & \longmapsto & [1, 8] \end{cases}$
- $[-10, -3] \uplus [7, 10]$ is expressed by: $\begin{cases} [-] & \longmapsto & [-10, -3] \\ [0] & \longmapsto & \bot_1 \\ [+] & \longmapsto & [7, 10] \end{cases}$

Lattice operations

Infimum:

- we assume that $\bot_1$ is the infimum of $\mathbb{D}_1^\sharp$
- then, $\bot_{\mathbf{cp}} = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \bot_1$ is the **infimum** of $\mathbb{D}_{\mathbf{cp}}^\sharp$

Ordering:

- we let $\sqsubseteq_{\mathbf{cp}}^\sharp$ denote the **pointwise ordering**:

$$X_0^\sharp \sqsubseteq_{\mathbf{cp}}^\sharp X_1^\sharp \quad \overset{def}{\Longleftrightarrow} \quad \forall z^\sharp \in \mathbb{D}_0^\sharp,\ X_0^\sharp(z^\sharp) \sqsubseteq_1^\sharp X_1^\sharp(z^\sharp)$$

- then, $X_0^\sharp \sqsubseteq_{\mathbf{cp}}^\sharp X_1^\sharp \Longrightarrow \gamma_{\mathbf{cp}}(X_0^\sharp) \subseteq \gamma_{\mathbf{cp}}(X_1^\sharp)$

Join operation:

- we assume that $\sqcup_1$ is a sound upper bound operator in $\mathbb{D}_1^\sharp$
- then, $\sqcup_{\mathbf{cp}}$ defined below is a **sound upper bound operator** in $\mathbb{D}_{\mathbf{cp}}^\sharp$:

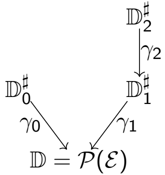
$$X_0^\sharp \sqcup_{\mathbf{cp}} X_1^\sharp \quad \overset{def}{::=} \quad \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot (X_0^\sharp(z^\sharp) \sqcup_1 X_1^\sharp(z^\sharp))$$

- the same construction applies to widening, if $\mathbb{D}_0^\sharp$ is finite

Composition with another abstraction

We assume three abstractions

- $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp)$, with concretization $\gamma_0 : \mathbb{D}_0^\sharp \longrightarrow \mathbb{D}$
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$, with concretization $\gamma_1 : \mathbb{D}_1^\sharp \longrightarrow \mathbb{D}$
- $(\mathbb{D}_2^\sharp, \sqsubseteq_2^\sharp)$, with concretization $\gamma_2 : \mathbb{D}_2^\sharp \longrightarrow \mathbb{D}_1^\sharp$



Cardinal power abstract domains $\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp$ and $\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_2^\sharp$ can be bound by an **abstraction relation** defined by concretization function γ:

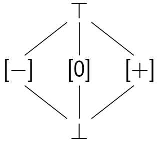
$$\begin{array}{rccc} \gamma : & (\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_2^\sharp) & \longrightarrow & (\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp) \\ & X^\sharp & \longmapsto & \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \gamma(X^\sharp(z^\sharp)) \end{array}$$

Applications:

- start with $\mathbb{D}_1^\sharp$ as the **identity abstraction**
- **compose several** cardinal power abstractions (or partitioning abstractions)

Composition with another abstraction

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
- $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$ be the **lattice of signs**
- $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1)$ be the **identity abstraction**
  $\mathbb{D}^\sharp_1 = \mathcal{P}(\mathbb{Z})$, $\gamma_1 = \mathbf{Id}$
- $(\mathbb{D}^\sharp_2, \sqsubseteq^\sharp_2)$ be the **lattice of intervals**



Then, $[-10, -3] \uplus [7, 10]$ is **abstracted in two steps**:

- in $\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_1$, $\left\{ \begin{array}{rcl} [-] & \longmapsto & [-10, -3] \\ [0] & \longmapsto & \emptyset \\ [+] & \longmapsto & [7, 10] \end{array} \right.$
- in $\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_2$, $\left\{ \begin{array}{rcl} [-] & \longmapsto & [-10, -3] \\ [0] & \longmapsto & \bot_1 \\ [+] & \longmapsto & [7, 10] \end{array} \right.$

Outline

Definition

We consider **concrete domain** $\mathbb{D} = \mathcal{P}(\mathbb{S})$ where

- $\mathbb{S} = \mathbb{L} \times \mathbb{M}$ where $\mathbb{L}$ denotes the set of control states
- $\mathbb{M} = \mathbb{X} \longrightarrow \mathbb{V}$

State partitioning

A **state partitioning** abstraction is defined as the cardinal power of two abstractions $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0, \gamma_0)$ and $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1, \gamma_1)$ of sets of states:

- $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0, \gamma_0)$ defines the **partitions**
- $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1, \gamma_1)$ defines the **abstraction of each element of partitions**

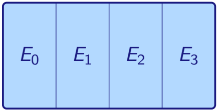
Typical instances:

- either $\mathbb{D}^\sharp_1 = \mathcal{P}(\mathbb{S})$, ordered with the inclusion
- or an abstraction of sets of memory states: numerical abstraction can be obtained by composing another abstraction on top of $(\mathcal{P}(\mathbb{S}), \subseteq)$

Use of a partition: intuition

We fix a partition $\mathcal{U}$ of $\mathcal{P}(\mathbb{S})$:

1. $\forall E, E' \in \mathcal{U},\ E \neq E' \Longrightarrow E \cap E' = \emptyset$
2. $\mathbb{S} = \bigcup \mathcal{U}$



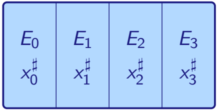
We can apply the **cardinal power construction**:

State partitioning abstraction

We let $\mathbb{D}_0^\sharp = \mathcal{U} \cup \{\bot, \top\}$ and $\gamma_0 : E \longmapsto E$. Thus, $\mathbb{D}_{\mathbf{cp}}^\sharp = \mathcal{U} \to \mathbb{D}_1^\sharp$ and:

$$\begin{array}{rccl} \gamma_{\mathbf{cp}} : & \mathbb{D}_{\mathbf{cp}}^\sharp & \longrightarrow & \mathbb{D} \\ & X^\sharp & \longmapsto & \{s \in \mathbb{S} \mid \forall E \in \mathcal{U},\, s \in E \Longrightarrow s \in \gamma_0(X^\sharp(E))\} \end{array}$$

- each $E \in \mathcal{U}$ is attached to a piece of information in $\mathbb{D}_1^\sharp$
- exercise: what happens if use only a **covering**, i.e., if we drop property 1 ?
- we will often focus on $\mathcal{U}$ and drop $\bot, \top$



Application 1: flow sensitive abstraction

Principle: abstract separately the states at distinct control states

This is **what we have been often doing already**, without formalizing it for instance, using the **the interval abstract domain**:

$$
\begin{array}{ll}
\ell_0: & \texttt{// assume } \mathtt{x} \geq 0 \\
\ell_1: & \textbf{if}(\mathtt{x} < 10)\{ \\
\ell_2: & \quad \mathtt{y} = \mathtt{x} - 2; \\
\ell_3: & \}\textbf{else}\{ \\
\ell_4: & \quad \mathtt{y} = 2 - \mathtt{x}; \\
\ell_5: & \} \\
\ell_6: & \ldots
\end{array}
$$

$$
\begin{array}{lcl}
\ell_0 & \mapsto & \mathtt{x} : \top \wedge \mathtt{y} : \top \\
\ell_1 & \mapsto & \mathtt{x} : [0, +\infty[\wedge \mathtt{y} : \top \\
\ell_2 & \mapsto & \mathtt{x} : [0, 9] \wedge \mathtt{y} : \top \\
\ell_3 & \mapsto & \mathtt{x} : [0, 9] \wedge \mathtt{y} : [-2, 7] \\
\ell_4 & \mapsto & \mathtt{x} : [10, +\infty[\wedge \mathtt{y} : \top \\
\ell_5 & \mapsto & \mathtt{x} : [10, +\infty[\wedge \mathtt{y} :]-\infty, -8] \\
\ell_6 & \mapsto & \mathtt{x} : [0, +\infty[\wedge \mathtt{y} :]-\infty, 7]
\end{array}
$$

Application 1: flow sensitive abstraction

Principle: abstract separately the states at distinct control states

Flow sensitive abstraction

We apply the cardinal power based partitioning abstraction with:

- $\mathcal{U} = \mathbb{L}$
- $\gamma_0 : \ell \mapsto \{\ell\} \times \mathbb{M}$

It is induced by partition $\{\{\ell\} \times \mathbb{M} \mid \ell \in \mathbb{L}\}$

Then, if $X^\sharp$ is an element of the reduced cardinal power,

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

- after this abstraction step, $\mathbb{D}_1^\sharp$ only needs to represent sets of memory states (numeric abstractions...)
- this abstraction step is *very common* as part of the design of abstract interpreters

Application 1: flow insensitive abstraction

Flow sensitive abstraction is **sometimes too costly**:

- e.g., **ultra fast pointer analyses** (a few seconds for 1 MLOC) for compilation and program transformation
- **context insensitive** abstraction simply **collapses all control states**

Flow insensitive abstraction

We apply the cardinal power based partitioning abstraction with:

- $\mathbb{D}_0^\sharp = \{\cdot\}$
- $\gamma_0 : \cdot \mapsto \mathbb{S}$
- $\mathbb{D}_1^\sharp = \mathcal{P}(\mathbb{M})$
- $\gamma_1 : M \mapsto \{(\ell, m) \mid \ell \in \mathbb{L}, m \in M\}$

It is induced by a trivial partition of $\mathcal{P}(\mathbb{S})$

Application 1: flow insensitive abstraction

We compare with **flow sensitive abstraction:**

$$\begin{array}{ll} \ell_0: & \texttt{// assume } x \geq 0 \\ \ell_1: & \textbf{if}(x < 10)\{ \\ \ell_2: & \quad y = x - 2; \\ \ell_3: & \}\textbf{else}\{ \\ \ell_4: & \quad y = 2 - x; \\ \ell_5: & \} \\ \ell_6: & \ldots \end{array}$$

$$\begin{array}{lcl} \ell_0 & \mapsto & x : \top \wedge y : \top \\ \ell_1 & \mapsto & x : [0, +\infty[\wedge y : \top \\ \ell_2 & \mapsto & x : [0, 9] \wedge y : \top \\ \ell_3 & \mapsto & x : [0, 9] \wedge y : [-2, 7] \\ \ell_4 & \mapsto & x : [10, +\infty[\wedge y : \top \\ \ell_5 & \mapsto & x : [10, +\infty[\wedge y :] - \infty, -8] \\ \ell_6 & \mapsto & x : [0, +\infty[\wedge y :] - \infty, 7] \end{array}$$

- the **best global information** is $x : \top \wedge y : \top$ (**very imprecise**)
- even if we exclude the point before the assume, we get $x : [0, +\infty[\wedge y : \top$ (still **very imprecise**)

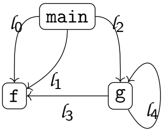
For a few specific applications flow insensitive is ok
In **most cases** (e.g., numeric properties), flow sensitive is absolutely needed

Application 2: context sensitive abstraction

We consider programs **with procedures**

Example:

void main(){... ℓ_0 : f(); ... ℓ_1 : f(); ... ℓ_2 : g() ...}
void f(){...}
void g(){if(...){ℓ_3 : f()}else{ℓ_4 : g()}}



- assumption: **flow sensitive abstraction** used inside each function
- we need to also describe the **call stack state**

Call string

Thus, $\mathbb{S} = \mathbb{K} \times \mathbb{L} \times \mathbb{M}$, where $\mathbb{K}$ is the set of **call strings**

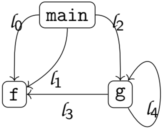
$$\begin{array}{rcll} \kappa & \in & \mathbb{K} & \text{calling contexts} \\ \kappa & ::= & \epsilon & \text{empty call stack} \\ & | & (f, \ell) \cdot \kappa & \text{call to } f \text{ from stack } \kappa \text{ at point } \ell \end{array}$$

Application 2: context sensitive abstraction, ∞-CFA

Fully context sensitive abstraction (∞-CFA)

- $\mathbb{D}_0^\sharp = \mathbb{K} \times \mathbb{L}$
- $\gamma_0 : (\kappa, \ell) \mapsto \{(\kappa, \ell, m) \mid m \in \mathbb{M}\}$

$$\begin{array}{l}\textbf{void}\ \texttt{main}()\{\ldots \ell_0 : \texttt{f}(); \ldots \ell_1 : \texttt{f}(); \ldots \ell_2 : \texttt{g}() \ldots\}\\ \textbf{void}\ \texttt{f}()\{\ldots\}\\ \textbf{void}\ \texttt{g}()\{\textbf{if}(\ldots)\{\ell_3 : \texttt{f}()\}\textbf{else}\{\ell_4 : \texttt{g}()\}\}\end{array}$$



Abstract contexts in **function** `f`:

$$(\ell_0, \texttt{f}) \cdot \epsilon,\ (\ell_1, \texttt{f}) \cdot \epsilon,\ (\ell_4, \texttt{f}) \cdot (\ell_2, \texttt{g}) \cdot \epsilon,$$
$$(\ell_4, \texttt{f}) \cdot (\ell_3, \texttt{g}) \cdot (\ell_2, \texttt{g}) \cdot \epsilon,\ (\ell_4, \texttt{f}) \cdot (\ell_3, \texttt{g}) \cdot (\ell_3, \texttt{g}) \cdot (\ell_2, \texttt{g}) \cdot \epsilon,\ \ldots$$

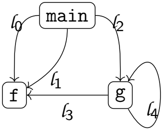
- one invariant per calling context, **very precise** (used, e.g., in Astrée)
- **infinite in presence of recursion** (i.e., not practical in this case)

Application 2: context sensitive abstraction, 0-CFA

Non context sensitive abstraction (0-CFA)

- $\mathbb{D}_0^\sharp = \mathbb{L}$
- $\gamma_0 : \ell \mapsto \{(\kappa, \ell, m) \mid \kappa \in \mathbb{K}, m \in \mathbb{M}\}$

$$\begin{array}{l}\textbf{void}\ \texttt{main}()\{\ldots \ell_0 : \texttt{f}(); \ldots \ell_1 : \texttt{f}(); \ldots \ell_2 : \texttt{g}() \ldots\}\\ \textbf{void}\ \texttt{f}()\{\ldots\}\\ \textbf{void}\ \texttt{g}()\{\textbf{if}(\ldots)\{\ell_3 : \texttt{f}()\}\textbf{else}\{\ell_4 : \texttt{g}()\}\}\end{array}$$



Abstract contexts in **function** $\texttt{f}$ are of the form $(?, \texttt{f}) \cdot \ldots,$

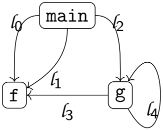
- 0-CFA merges **all** calling contexts to a same procedure, **very coarse** abstraction
- but is **usually quite efficient to compute**

Application 2: context sensitive abstraction, k-CFA

Partially context sensitive abstraction (k-CFA)

- $\mathbb{D}_0^\sharp = \{\kappa \in \mathbb{K} \mid \textbf{length}(\kappa) \leq k\} \times \mathbb{L}$
- $\gamma_0 : (\kappa, \ell) \mapsto \{(\kappa \cdot \kappa', \ell, m) \mid \kappa' \in \mathbb{K}, m \in \mathbb{M}\}$

$$\begin{array}{l}\textbf{void}\ \mathtt{main}()\{\ldots \ell_0 : \mathtt{f}(); \ldots \ell_1 : \mathtt{f}(); \ldots \ell_2 : \mathtt{g}() \ldots\} \\ \textbf{void}\ \mathtt{f}()\{\ldots\} \\ \textbf{void}\ \mathtt{g}()\{\textbf{if}(\ldots)\{\ell_3 : \mathtt{f}()\}\textbf{else}\{\ell_4 : \mathtt{g}()\}\}\end{array}$$



Abstract contexts in **function** $\mathtt{f}$, in 2-**CFA:**

$$(\ell_0, \mathtt{f}) \cdot \epsilon,\ (\ell_1, \mathtt{f}) \cdot \epsilon,\ (\ell_4, \mathtt{f}) \cdot (\ell_3, \mathtt{g}) \cdot (?, \mathtt{g}) \cdot \ldots, (\ell_4, \mathtt{f}) \cdot (\ell_4, \mathtt{g}) \cdot (?, \mathtt{g}) \cdot \ldots$$

- usually **intermediate** level of precision and efficiency
- can be applied to programs with **recursive procedures**

Application 3: partitioning by a boolean condition

- so far, we only used abstractions of the context to partition
- we now consider abstractions of memory states properties

Function guided memory states partitioning

We let:

- $\mathbb{D}_0^\sharp = A$ where A finite set is a finite set of values / properties
- $\phi : \mathbb{M} \to A$ maps each store to its property
- γ_0 is of the form $(a \in A) \mapsto \{(\ell, m) \in \mathbb{S} \mid \phi(m) = a\}$

Common choice for A: **the set of boolean values $\mathbb{B}$**
(or another finite set of values —convenient for enum types!)

Many choices for function ϕ are possible:

- **value** of one or several variables (boolean or scalar)
- **sign** of a variable
- ...

Application 3: partitioning by a boolean condition

We assume:

- $\mathbb{X} = \mathbb{X}_{\text{bool}} \uplus \mathbb{X}_{\text{int}}$, where $\mathbb{X}_{\text{bool}}$ (resp., $\mathbb{X}_{\text{int}}$) collects **boolean** (resp., **integer**) variables
- $\mathbb{X}_{\text{bool}} = \{\mathtt{b}_0, \ldots, \mathtt{b}_{k-1}\}$
- $\mathbb{X}_{\text{int}} = \{\mathtt{x}_0, \ldots, \mathtt{x}_{l-1}\}$

Thus, $\mathbb{M} = \mathbb{X} \to \mathbb{V} \equiv (\mathbb{X}_{\text{bool}} \to \mathbb{V}_{\text{bool}}) \times (\mathbb{X}_{\text{int}} \to \mathbb{V}_{\text{int}}) \equiv \mathbb{V}_{\text{bool}}^k \times \mathbb{V}_{\text{int}}^l$

Boolean partitioning abstract domain

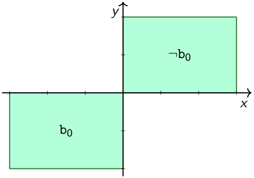
We apply the cardinal power abstraction, with a domain of partitions defined by a function, with:

- $A = \mathbb{B}^k$
- $\phi(m) = (m(\mathtt{b}_0), \ldots, m(\mathtt{b}_{k-1}))$
- we let $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp, \gamma_1)$ be any **numerical abstract domain** for $\mathcal{P}(\mathbb{V}_{\text{int}}^l)$

Application 3: example

With $\mathbb{X}_{\mathrm{bool}} = \{\mathtt{b}_0, \mathtt{b}_1\}, \mathbb{X}_{\mathrm{int}} = \{\mathtt{x}, \mathtt{y}\}$, we can express:

$$\begin{cases} \mathtt{b}_0 \wedge \mathtt{b}_1 & \Longrightarrow & \mathtt{x}_0 \in [-3, 0] \wedge \mathtt{y} \in [-2, 0] \\ \mathtt{b}_0 \wedge \neg\mathtt{b}_1 & \Longrightarrow & \mathtt{x}_0 \in [-3, 0] \wedge \mathtt{y} \in [-2, 0] \\ \neg\mathtt{b}_0 \wedge \mathtt{b}_1 & \Longrightarrow & \mathtt{x}_0 \in [0, 3] \wedge \mathtt{y} \in [0, 3] \\ \neg\mathtt{b}_0 \wedge \neg\mathtt{b}_1 & \Longrightarrow & \mathtt{x}_0 \in [0, 3] \wedge \mathtt{y} \in [0, 3] \end{cases}$$



- this abstract value expresses a **relation** between $\mathtt{b}_0$ and $\mathtt{x}, \mathtt{y}$ (which induces a relation between $\mathtt{x}$ and $\mathtt{y}$)
- **alternative**: partition with respect to only **some** variables e.g., here $\mathtt{b}_0$ only as $\mathtt{b}_1$ is irrelevant
- **typical representation** of abstract values: based on some kind of decision trees (variants of BDDs)

Application 3: example

- Left side abstraction shown in blue: boolean partitioning for $\mathtt{b}_0, \mathtt{b}_1$
- Right side abstraction shown in green: interval abstraction
- We omit the cases of the form $P \Longrightarrow \bot$...

bool $\mathtt{b}_0$, $\mathtt{b}_1$;
int x, y; (uninitialized)
$\mathtt{b}_0 = \mathtt{x} \geq 0$;
$$(\mathtt{b}_0 \Longrightarrow \mathtt{x} \geq 0) \wedge (\neg \mathtt{b}_0 \Longrightarrow \mathtt{x} < 0)$$
$\mathtt{b}_1 = \mathtt{x} \leq 0$;
$$(\mathtt{b}_0 \wedge \mathtt{b}_1 \Longrightarrow \mathtt{x} = 0) \wedge (\mathtt{b}_0 \wedge \neg \mathtt{b}_1 \Longrightarrow \mathtt{x} > 0) \wedge (\neg \mathtt{b}_0 \wedge \mathtt{b}_1 \Longrightarrow \mathtt{x} < 0)$$
if($\mathtt{b}_0$ && $\mathtt{b}_1$){
$$(\mathtt{b}_0 \wedge \mathtt{b}_1 \Longrightarrow \mathtt{x} = 0)$$
  $\mathtt{y} = 0$;
$$(\mathtt{b}_0 \wedge \mathtt{b}_1 \Longrightarrow \mathtt{x} = 0 \wedge \mathtt{y} = 0)$$
}**else**{
$$(\mathtt{b}_0 \wedge \neg \mathtt{b}_1 \Longrightarrow \mathtt{x} > 0) \wedge (\neg \mathtt{b}_0 \wedge \mathtt{b}_1 \Longrightarrow \mathtt{x} < 0)$$
  $\mathtt{y} = 100/\mathtt{x}$;
$$(\mathtt{b}_0 \wedge \neg \mathtt{b}_1 \Longrightarrow \mathtt{x} > 0 \wedge \mathtt{y} \geq 0) \wedge (\neg \mathtt{b}_0 \wedge \mathtt{b}_1 \Longrightarrow \mathtt{x} < 0 \wedge \mathtt{y} \leq 0)$$
}

Application 3: partitioning by the sign of a variable

We now consider a **semantic property**: the **sign of a variable**

We assume:

- $\mathbb{X} = \mathbb{X}_{\mathrm{int}}$, i.e., all variables have **integer** type
- $\mathbb{X}_{\mathrm{int}} = \{\mathtt{x}_0, \ldots, \mathtt{x}_{l-1}\}$

Thus, $\mathbb{M} = \mathbb{X} \to \mathbb{V} \equiv \mathbb{V}_{\mathrm{int}}^{l}$

Sign partitioning abstract domain

We apply the cardinal power abstraction, with a domain of partition defined by a function, with:

- $A = \{[<0], [=0], [>0]\}$
- $\phi(m) = \begin{cases} [<0] & \text{if } x_0 < 0 \\ [=0] & \text{if } x_0 = 0 \\ [>0] & \text{if } x_0 > 0 \end{cases}$
- $(\mathbb{D}_1^{\sharp}, \sqsubseteq_1^{\sharp}, \gamma_1)$ an abstraction of $\mathcal{P}(\mathbb{V}_{\mathrm{int}}^{l-1})$ (no need to abstract $\mathtt{x}_0$ twice)

Application 3: example

- Sign abstraction fixing partitions shown in blue
- Right side abstraction shown in green: interval abstraction
- We omit the cases of the form $P \Longrightarrow \bot$...

```
int x ∈ ℤ;
int s;
int y;
if(x ≥ 0){
            (x < 0 ⇒ ⊥) ∧ (x = 0 ⇒ ⊤) ∧ (x > 0 ⇒ ⊤)
      s = 1;
            (x < 0 ⇒ ⊥) ∧ (x = 0 ⇒ s = 1) ∧ (x > 0 ⇒ s = 1)
} else {
            (x < 0 ⇒ ⊤) ∧ (x = 0 ⇒ ⊥) ∧ (x > 0 ⇒ ⊥)
      s = −1;
            (x < 0 ⇒ s = −1) ∧ (x = 0 ⇒ ⊥) ∧ (x > 0 ⇒ ⊥)
}
            (x < 0 ⇒ s = −1) ∧ (x = 0 ⇒ s = 1) ∧ (x > 0 ⇒ s = 1)
① y = x/s;
            (x < 0 ⇒ s = −1 ∧ y > 0) ∧ (x = 0 ⇒ s = 1 ∧ y = 0) ∧ (x > 0 ⇒ s = 1 ∧ y > 0)
② assert(y ≥ 0);
```

Outline

Computation of abstract semantics and partitioning

- we first consider **partitioning by control states** and let $\mathbb{L} = \{\ell_0, \ldots, \ell_s\}$
- we rely on the two steps partitioning abstraction i.e., to be **composed** with an abstraction of $\mathcal{P}(\mathbb{M})$
- the techniques shown below **extend to other forms of partitioning**

The first abstraction defines to a **Galois connection**:

$$(\mathcal{P}(\mathbb{L} \times \mathbb{M}), \subseteq) \underset{\alpha_{\mathrm{part}}}{\overset{\gamma_{\mathrm{part}}}{\leftrightarrows}} (\mathbb{D}^{\sharp}_{\mathrm{part}}, \dot{\subseteq})$$

where $\mathbb{D}^{\sharp}_{\mathrm{part}} = \mathbb{L} \rightarrow \mathcal{P}(\mathbb{M})$ and:

$$\begin{array}{rlcl}
\alpha_{\mathrm{part}}: & \mathcal{P}(\mathbb{L} \times \mathbb{M}) & \longrightarrow & \mathbb{D}^{\sharp}_{\mathrm{part}} \\
 & \mathcal{E} & \longmapsto & \lambda(\ell \in \mathbb{L}) \cdot \{m \in \mathbb{M} \mid (\ell, m) \in \mathcal{E}\} \\
\gamma_{\mathrm{part}}: & \mathbb{D}^{\sharp}_{\mathrm{part}} & \longrightarrow & \mathcal{P}(\mathbb{L} \times \mathbb{M}) \\
 & X^{\sharp} & \longmapsto & \{(\ell, m) \in \mathbb{S} \mid m \in X^{\sharp}(\ell)\}
\end{array}$$

We first study the **"computational form"** of this semantics (fixpoint)

Fixpoint form of a partitioned semantics

- we consider a **transition system** $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$ where $\mathbb{S}_{\mathcal{I}} = \{\ell_0\} \times \mathbb{M}$
- the **reachable states** are computed as $[\![\mathcal{S}]\!]_{\mathcal{R}} = \mathbf{lfp}_{\mathbb{S}_{\mathcal{I}}} F_{\mathcal{R}}$ where

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

Semantic function over the partitioned system

We let F_{part} be defined over $\mathbb{D}^{\sharp}_{\mathrm{part}}$ by:

$$\begin{array}{rccl} F_{\mathrm{part}}: & \mathbb{D}^{\sharp}_{\mathrm{part}} & \longrightarrow & \mathbb{D}^{\sharp}_{\mathrm{part}} \\ & X^{\sharp} & \longmapsto & \lambda(\ell \in \mathbb{L}) \cdot \{m \in \delta_{\ell,\ell'}(m') \mid \ell' \in \mathbb{L}, m' \in X^{\sharp}(\ell')\} \\ & & & \{m \in \mathbb{M} \mid \exists \ell' \in \mathbb{L}, \exists m' \in X^{\sharp}(\ell'), \end{array}$$

where $\delta_{\ell,\ell'}(m') = \{m' \in \mathbb{M} \mid (\ell, m) \rightarrow (\ell', m')\}$.

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

Concrete equations form of a partitioned semantics

We look for **an equivalent set of abstract equations** following the intuition:

- at ℓ_0, we observe any memory state (start **from any memory state**)
- at $\ell \neq \ell_0$, we observe states reached from a predecessor of ℓ, **in a single step**

Set of concrete semantic equations

We define the **set of concrete semantic equations** by:

$$\begin{cases} \mathcal{M}_0 & = & \mathbb{M} \\ \mathcal{M}_1 & = & \bigcup_i \{m \in \mathbb{M} \mid \exists m' \in \mathcal{M}_i,\ (l_i, m') \rightarrow (l_1, m)\} \\ & \vdots & \\ \mathcal{M}_s & = & \bigcup_i \{m \in \mathbb{M} \mid \exists m' \in \mathcal{M}_i,\ (l_i, m') \rightarrow (l_s, m)\} \end{cases}$$

where variables $\mathcal{M}_0, \ldots, \mathcal{M}_s$ range over set of memory states, i.e., we look for solutions where $\mathcal{M}_i \subseteq \mathbb{M}$

Concrete equations form of a partitioned semantics

In the following, we note:

$$F_j : (\mathcal{M}_1, \ldots, \mathcal{M}_s) \mapsto \bigcup_i \{m \in \mathbb{M} \mid \exists m' \in \mathcal{E}_i,\ (l_i, m') \rightarrow (l_j, m)\}$$

Computational form of the concrete semantics

$\alpha_{\mathrm{part}}([\![\mathcal{S}]\!]_{\mathcal{R}})$ is the least solution of the system

$$\left\{ \begin{array}{lcl} \mathcal{M}_0 & = & \mathbb{M} \\ \mathcal{M}_1 & = & F_1(\mathcal{M}_1, \ldots, \mathcal{M}_s) \\ & \vdots & \\ \mathcal{M}_s & = & F_s(\mathcal{M}_1, \ldots, \mathcal{M}_s) \end{array} \right.$$

- **proof:** the system defines exactly the fixpoints of F_{part}
- iterating F_{part} boils down to running the system from $(\mathbb{M}, \emptyset, \ldots, \emptyset)$
- we cannot "implement" this (convergence issue !), but we can do the same in the abstract

Abstract equations form of a partitioned semantics

We can now move on **to the abstract level**:

- $\mathcal{M}_i^\sharp$ denotes an element of $\mathbb{D}_1^\sharp$
- **abstract functions** $F_i^\sharp : (\mathbb{D}_1^\sharp)^s \rightarrow \mathbb{D}_1^\sharp$ over-approximate the concrete functions $F_i^\sharp$

Abstract equations

A solution of the system

$$\begin{cases} \mathcal{M}_0^\sharp & \sqsupseteq & \top \\ \mathcal{M}_1^\sharp & \sqsupseteq & F_1^\sharp(\mathcal{M}_1^\sharp, \ldots, \mathcal{M}_s^\sharp) \\ & \vdots & \\ \mathcal{M}_s^\sharp & \sqsupseteq & F_s^\sharp(\mathcal{M}_1^\sharp, \ldots, \mathcal{M}_s^\sharp) \end{cases}$$

over-approximates $\alpha_{\mathrm{part}}(\llbracket \mathcal{S} \rrbracket_{\mathcal{R}})$

Partitioned systems and fixpoint computation

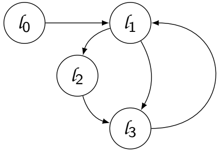
For now, we overlook the convergence issue, and focus on **the computation of the iterates**

Typical properties of real transition systems:

- 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 under consideration
- also, F_i is **$\emptyset$-strict**:
  if there is no predecessor, there is no transition...

Example of a simple system of abstract equations:

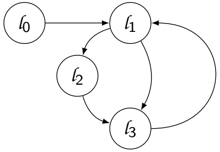
$$\begin{cases} \mathcal{M}_0 & = & \mathbb{M} \\ \mathcal{M}_1 & = & F_1(\mathcal{M}_0) \cup G_1(\mathcal{M}_3) \\ \mathcal{M}_2 & = & F_2(\mathcal{M}_1) \\ \mathcal{M}_3 & = & F_3(\mathcal{M}_1) \cup G_3(\mathcal{M}_2) \end{cases}$$



Example concrete iteration

System:

$$\begin{cases} \mathcal{M}_0 & = & \mathbb{M} \\ \mathcal{M}_1 & = & F_1(\mathcal{M}_0) \cup G_1(\mathcal{M}_3) \\ \mathcal{M}_2 & = & F_2(\mathcal{M}_1) \\ \mathcal{M}_3 & = & F_3(\mathcal{M}_1) \cup G_3(\mathcal{M}_2) \end{cases}$$



Iterates for $(\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3)$:

rank	iterate
0	$(\mathbb{M}, \emptyset, \emptyset, \emptyset)$
1	$(\mathbb{M}, F_1(\mathbb{M}), \emptyset, \emptyset)$
2	$(\mathbb{M}, F_1(\mathbb{M}), F_2 \circ F_1(\mathbb{M}), F_3 \circ F_1(\mathbb{M}))$
3	$(\mathbb{M}, F_1(\mathbb{M}) \cup G_1 \circ F_3 \circ F_1(\mathbb{M}), F_2 \circ F_1(\mathbb{M}), F_3 \circ F_1(\mathbb{M}) \cup G_3 \circ F_2 \circ F_1(\mathbb{M}))$
4	...

- we highlight in red the "computations" that are done
- there is a lot of **un-necessary recomputation**

Example computation of abstract iterates

Using an abstraction of sets of memory states:

$$\begin{cases} \mathcal{M}_0 & = & \mathbb{M} \\ \mathcal{M}_1 & = & F_1(\mathcal{M}_0) \cup G_1(\mathcal{M}_3) \\ \mathcal{M}_2 & = & F_2(\mathcal{M}_1) \\ \mathcal{M}_3 & = & F_3(\mathcal{M}_1) \cup G_3(\mathcal{M}_2) \end{cases}$$

- we assume **abstract domain** $\mathbb{M}^\sharp$, over-approximating $\mathcal{P}(\mathbb{M})$
- we assume **abstract transfer functions** $F_i^\sharp : \mathbb{M}^\sharp \to \mathbb{M}^\sharp$ over-approximates F_i

Abstract iterates:

rank	iterate
0	$(\mathbb{M}, \emptyset, \emptyset, \emptyset)$
1	$(\mathbb{M}, F_1^\sharp(\mathbb{M}), \emptyset, \emptyset)$
2	$(\mathbb{M}, F_1^\sharp(\mathbb{M}), F_2^\sharp \circ F_1^\sharp(\mathbb{M}), F_3^\sharp \circ F_1^\sharp(\mathbb{M}))$
3	$(\mathbb{M}, F_1^\sharp(\mathbb{M}) \sqcup G_1^\sharp \circ F_3^\sharp \circ F_1^\sharp(\mathbb{M}), F_2^\sharp \circ F_1^\sharp(\mathbb{M}), F_3^\sharp \circ F_1^\sharp(\mathbb{M}) \sqcup G_3^\sharp \circ F_2^\sharp \circ F_1^\sharp(\mathbb{M}))$
4	...

The **same issue** occurs: **recomputation of abstract iterates**

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.

- 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 infinitely 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 $\{1, \ldots s\}$ (we disregard component 0, which is $\top$) such that:

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

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
- **benefit**: **no more useless recomputation**
- **back to the example**, where **recomputed components** are in red:

rank	iterate
0	$(\mathbb{M}, \emptyset, \emptyset, \emptyset)$
1	$(\mathbb{M}, F^\sharp_1(\mathbb{M}), \emptyset, \emptyset)$
2	$(\mathbb{M}, F^\sharp_1(\mathbb{M}), F^\sharp_2 \circ F^\sharp_1(\mathbb{M}), \emptyset)$
3	$(\mathbb{M}, F^\sharp_1(\mathbb{M}), F^\sharp_2 \circ F^\sharp_1(\mathbb{M}), F^\sharp_3 \circ F^\sharp_1(\mathbb{M}))$
4	$(\mathbb{M}, F^\sharp_1(\mathbb{M}), F^\sharp_2 \circ F^\sharp_1(\mathbb{M}), F^\sharp_3 \circ F^\sharp_1(\mathbb{M}) \sqcup G^\sharp_3 \circ F^\sharp_2 \circ F^\sharp_1(\mathbb{M}))$
5	...

Chaotic iterations: work-list algorithm

Work-list algorithms

Principle:

1. maintain a **queue of partitions to update**
2. initialize the queue with the **entry label** of the program and the local invariant at that point at $\top$
3. 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
4. **terminate** when the queue is **empty**

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

- **benefit**: **no more useless recomputation**
- implemented in **many static analyzers**

Work-list algorithm

Pseudo code implementation, where $\delta^{\sharp}_{\ell,\ell'}$ denotes the transfer function from ℓ to ℓ', that over-approximates $\delta_{\ell,\ell'}$:

$\text{to\_propagate} \leftarrow \{\text{initial states}\}$
$\mathcal{E}^{\sharp}_{\text{initial}} \leftarrow \top$
while$(\text{to\_propagate} \neq \emptyset)\{$
    pick $\ell \in \text{to\_propagate}$
    $\text{to\_propagate} = \text{to\_propagate} \setminus \{\ell\}$
    for(ℓ' successor of ℓ in the CFG)$\{$
        $y^{\sharp} \leftarrow \delta^{\sharp}_{\ell,\ell'}(\mathcal{E}^{\sharp}_{\ell})$
        if$(\neg(y^{\sharp} \sqsubseteq^{\sharp} \mathcal{E}^{\sharp}_{\ell'}))\{$
            $\mathcal{E}^{\sharp}_{\ell'} = \mathcal{E}^{\sharp}_{\ell'} \sqcup^{\sharp} y^{\sharp}$
            $\text{to\_propagate} = \text{to\_propagate} \cup \{\ell'\}$
        $\}$
    $\}$
$\}$

Selection of a set of widening points for a partitioned system

Assumptions:

- abstract domain $\mathbb{D}^{\sharp}_{\mathrm{num}}$, with concretization $\gamma_{\mathrm{num}} : \mathbb{D}^{\sharp}_{\mathrm{num}} \to \mathcal{P}(\mathbb{M})$, **does not satisfy ascending chain condition**
- $\mathbb{D}^{\sharp}_{\mathrm{num}}$ provides **widening** operator $\triangledown$

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

Enforcing termination of chaotic iterates

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

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

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

Algorithm for iteration with widening: exercise

Outline

Computation of abstract semantics and partitioning

We now **compose two forms of partitioning**

- by **control states** (as previously), using a chaotic iteration strategy
- by **the values of the boolean variables**

Thus, **the abstract domain is of the form**

$$\mathbb{L} \longrightarrow (\mathbb{V}_{\text{bool}}^{k} \longrightarrow \mathbb{D}_{1}^{\sharp})$$

- we could do a partitioning by $\mathbb{L} \times \mathbb{V}_{\text{bool}}^{k}$
- yet, it is not practical, as transitions from "boolean states" are not know before the analysis
- **data types** skeleton:

```
type abs1 = ...  (* abstract elements of D1# *)
type abs_state = ...  (*
    boolean trees with elements of type abs1 at the leaves *)
type abs_cp = (labels, abs_state) Map.t
```

Abstract operations

Transfer functions:
we seek, for all pair $\ell, \ell' \in \mathbb{L}$ for an approximation $\delta^{\sharp}_{\ell,\ell'}$ of

$$\begin{array}{rccl} \delta_{\ell,\ell'}: & \mathbb{M} & \longrightarrow & \mathcal{P}(\mathbb{M}) \\ & m & \longmapsto & \{m' \in \mathbb{M} \mid (\ell, m) \rightarrow (\ell', m')\} \end{array}$$

This includes:

- **assignment to scalar**, e.g., $\mathtt{x} = 1 - \mathtt{x}$;
- **assignment to boolean**, e.g., $\mathtt{b}_0 = \mathtt{x} \leq 7$
- **scalar test**, e.g., $\textbf{if}(\mathtt{x} \geq 8) \ldots$
- **boolean test**, e.g., $\textbf{if}(\neg \mathtt{b}_1) \ldots$

Lattice operations: **inclusion check**, **join**, **widening**

Transfer functions: assignment to scalar

Assignment $\ell_0 : \mathrm{x} = \mathrm{e}; \ell_1$ affecting **only integer variables** (i.e., e depends only on $\mathrm{x}_0, \ldots, \mathrm{x}_l$):

- **example**: $\mathrm{x} = 1 - \mathrm{x};$
- **concrete transition** δ_{ℓ_0,ℓ_1} defined by
$$\delta_{\ell_0,\ell_1}(m) = \{m[\mathrm{x} \leftarrow [\![\mathrm{e}]\!](m)]\}$$
- the values of the boolean variables are unchanged
thus the partitions are preserved (**pointwise** transfer function):
$$\mathit{assign}_{\mathbf{cp}}(\mathrm{x}, \mathrm{e}, X^\sharp) = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \mathit{assign}_1(\mathrm{x}, \mathrm{e}, X^\sharp(z^\sharp))$$

Soundness

If assign_1 is sound, so is $\mathit{assign}_{\mathbf{cp}}$, in the sense that:

$$\forall X^\sharp \in \mathbb{D}_{\mathbf{cp}}^\sharp,\ \forall m \in \gamma_{\mathbf{cp}}(X^\sharp),\ m[\mathrm{x} \leftarrow [\![\mathrm{e}]\!](m)] \in \gamma_{\mathbf{cp}}(\mathit{assign}_{\mathbf{cp}}(\mathrm{x}, \mathrm{e}, X^\sharp))$$

Transfer functions: assignment to scalar, example

- **abstract precondition**:

$$\left\{ \begin{array}{ccc} & \mathtt{b} & \Rightarrow & \mathtt{x} \geq 0 \\ \wedge & \neg \mathtt{b} & \Rightarrow & \mathtt{x} \leq 0 \end{array} \right\}$$

- **statement**:

$$\mathtt{x} = 1 - \mathtt{x};$$

- **abstract post-condition**:

$$\begin{array}{l} \mathit{assign}_{\mathbf{cp}} \left(\mathtt{x}, 1 - \mathtt{x}, \left\{ \begin{array}{cccc} & \mathtt{b} & \Rightarrow & \mathtt{x} \geq 0 \\ \wedge & \neg \mathtt{b} & \Rightarrow & \mathtt{x} \leq 0 \end{array} \right\} \right) \\ \qquad = \left\{ \begin{array}{cccc} & \mathtt{b} & \Rightarrow & \mathtt{x} \leq 1 \\ \wedge & \neg \mathtt{b} & \Rightarrow & \mathtt{x} \geq 1 \end{array} \right\} \end{array}$$

Transfer functions: scalar test

Condition test $\ell_0 : \mathbf{if}(\mathrm{c})\{\ell_1 : \ldots\}$ affecting **only scalar variables** (i.e., c depends only on $\mathtt{x}_0, \ldots, \mathtt{x}_l$):

- **example**: $\mathbf{if}(\mathtt{x} \geq 8) \ldots$
- **concrete transition** δ_{ℓ_0,ℓ_1} defined by

$$\delta_{\ell_0,\ell_1}(m) = \begin{cases} \{m\} & \text{if } [\![\mathrm{c}]\!](m) = \mathtt{TRUE} \\ \emptyset & \text{if } [\![\mathrm{c}]\!](m) = \mathtt{FALSE} \end{cases}$$

- the partitions are preserved, thus we get a **pointwise** transfer function:

$$test_{\mathbf{cp}}(\mathrm{c}, X^\sharp) = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot test_1(\mathrm{c}, X^\sharp(z^\sharp))$$

Soundness

If $test_1$ is sound, so is $test_{\mathbf{cp}}$, in the sense that:

$$\forall X^\sharp \in \mathbb{D}^\sharp_{\mathbf{cp}},\ \forall m \in \gamma_{\mathbf{cp}}(X^\sharp),\ [\![\mathrm{c}]\!](m) = \mathtt{TRUE} \Longrightarrow m \in \gamma_{\mathbf{cp}}(test_{\mathbf{cp}}(\mathtt{x}, \mathrm{e}, X^\sharp))$$

Transfer functions: scalar test, example

- **abstract pre-condition**:

$$\left\{ \begin{array}{ccccc} & \mathtt{b} & \Rightarrow & \mathtt{x} \geq 0 \\ \wedge & \neg\mathtt{b} & \Rightarrow & \mathtt{x} \leq 0 \end{array} \right\}$$

- **statement**:

$$\mathbf{if}(\mathtt{x} \geq 8) \ldots$$

- **abstract post-condition**:

$$\mathit{test}_{\mathbf{cp}}\left(\mathtt{x} \geq 8, \left\{ \begin{array}{ccccc} & \mathtt{b} & \Rightarrow & \mathtt{x} \geq 0 \\ \wedge & \neg\mathtt{b} & \Rightarrow & \mathtt{x} \leq 0 \end{array} \right\}\right) = \left\{ \begin{array}{ccccc} & \mathtt{b} & \Rightarrow & \mathtt{x} \geq 8 \\ \wedge & \neg\mathtt{b} & \Rightarrow & \bot \end{array} \right\}$$

Transfer functions: boolean condition test

Condition test $\ell_0 : \mathbf{if}(\mathtt{c})\{\ell_1 : \ldots\}$ affecting **only boolean variables** (i.e., $\mathtt{c}$ depends only on $\mathtt{b}_0, \ldots, \mathtt{b}_k$):

- **example**: $\mathbf{if}(\neg \mathtt{b}_1) \ldots$
- then, we simply need to filter the boolean partitions **satisfying** $\mathtt{c}$:
$$test_{\mathbf{cp}}(\mathtt{c}, X^\sharp) = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \begin{cases} X^\sharp(z^\sharp) & \text{if } test_0(\mathtt{c}, X^\sharp(z^\sharp)) \neq \bot_0 \\ \bot_1 & \text{otherwise} \end{cases}$$

Soundness

If $test_0$ is sound, so is $test_{\mathbf{cp}}$, in the sense that:

$$\forall X^\sharp \in \mathbb{D}_{\mathbf{cp}}^\sharp,\ \forall m \in \gamma_{\mathbf{cp}}(X^\sharp),\ [\![\mathtt{c}]\!](m) = \mathtt{TRUE} \Longrightarrow m \in \gamma_{\mathbf{cp}}(test_{\mathbf{cp}}(\mathtt{x}, \mathtt{e}, X^\sharp))$$

Transfer functions: boolean condition test, example

- **abstract pre-condition**: $$\left\{ \begin{array}{cccc} & b_0 \wedge b_1 & \Rightarrow & 15 \leq x \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & 6 \leq x \leq 8 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array} \right\}$$
- **statement**: $\mathbf{if}(\neg b_1) \ldots$
- **abstract post-condition**:

$$test_{\mathbf{cp}}\left(\neg b_1, \left\{ \begin{array}{cccc} & b_0 \wedge b_1 & \Rightarrow & 15 \leq x \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & 6 \leq x \leq 8 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array} \right\}\right)$$
$$= \left\{ \begin{array}{cccc} & b_0 \wedge b_1 & \Rightarrow & \bot_1 \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & \bot_1 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array} \right\}$$

Transfer functions: assignment to boolean

Assignment $\ell_0 : \mathtt{b} = \mathtt{e}; \ell_1$ to a **boolean variable**, where the right hand side contains **only integer variables** (i.e., e depends only on $\mathtt{x}_0, \ldots, \mathtt{x}_l$):

- **example**: $\mathtt{b}_0 = \mathtt{x} \leq 7$

Algorithm:

- let $z^\sharp \in \mathbb{D}_0^\sharp$, and let us assume that $z^\sharp(\mathtt{b}) = \mathtt{TRUE}$:
  then, $\mathit{assign}_{\mathbf{cp}}(\mathtt{b}, \mathtt{e}[\mathtt{x}_0, \ldots, \mathtt{x}_i], X^\sharp)(z^\sharp)$ should account for all states where b becomes true, other boolean variables remaining unchanged:

$$\mathit{assign}_{\mathbf{cp}}(\mathtt{b}, \mathtt{e}, X^\sharp)(z^\sharp) = \begin{cases} & \mathit{test}_1(\mathtt{e}, X^\sharp(z^\sharp)) \\ \sqcup_1 & \mathit{test}_1(\mathtt{e}, X^\sharp(z^\sharp[\mathtt{b} \leftarrow \mathtt{FALSE}])) \end{cases}$$

- when $z^\sharp \in \mathbb{D}_0^\sharp$, and $z^\sharp(\mathtt{b}) = \mathtt{FALSE}$: similar computation

The partitions get modified (this is a **costly step**, involving join)

Transfer functions: assignment to boolean, example

- **abstract pre-condition**: $\left\{ \begin{array}{cccc} & b_0 \wedge b_1 & \Rightarrow & 15 \leq x \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & 6 \leq x \leq 8 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array} \right\}$
- **statement**: $b_0 = x \leq 7$
- **abstract post-condition**:

$$\mathit{assign}_{\mathbf{cp}} \left(b_0, x \leq 7, \left\{ \begin{array}{cccc} & b_0 \wedge b_1 & \Rightarrow & 15 \leq x \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & 6 \leq x \leq 8 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array} \right\} \right)$$

$$= \left\{ \begin{array}{cccc} & b_0 \wedge b_1 & \Rightarrow & 6 \leq x \leq 7 \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & 8 \leq x \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \end{array} \right\}$$

The partitions get modified (this is a **costly step**, involving join)

Choice of boolean partitions

Boolean partitioning allows to express relations between boolean and scalar variables

But these relations are expensive to maintain:

1. partitioning with respect to N boolean variables translates into a 2^N **space cost factor**
2. after assignments, partitions need be recomputed (**use of join**)

Packing addresses the first issue

- select groups of variables for which relations would be **useful**
- can be based on **syntactic** or **semantic** criteria

Whatever the packs, the transfer functions will produce a sound result (but possibly not the most precise one)

How to alleviate the second issue ?

Outline

Definition of trace partitioning

Principle

We start from a **trace semantics** and rely on **an abstraction of execution history for partitioning**

- **concrete domain**: $\mathbb{D} = \mathcal{P}(\mathbb{S}^\star)$
- **left side abstraction** $\gamma_0 : \mathbb{D}_0^\sharp \rightarrow \mathbb{D}$: a **trace abstraction** **to be defined precisely later**
- **right side abstraction**, as a **composition** of two abstractions:
  - the **final state abstraction** defined by $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp) = (\mathcal{P}(\mathbb{S}), \subseteq)$ and:
    $$\gamma_1 : M \longmapsto \{\langle s_0, \ldots, s_k, (\ell, m)\rangle \mid m \in M, \ell \in \mathbb{L}, s_0, \ldots, s_k \in \mathbb{S}\}$$
  - a **store abstraction** applied to the traces final memory state $\gamma_2 : \mathbb{D}_2^\sharp \rightarrow \mathbb{D}_1^\sharp$

Trace partitioning

Cardinal power abstraction defined by abstractions γ_0 and $\gamma_1 \circ \gamma_2$

Application 1: partitioning by control states

Flow sensitive abstraction

- We let $\mathbb{D}_0^\sharp = \mathbb{L} \cup \{\top\}$
- Concretization is defined by:

$$\begin{array}{rccl} \gamma_0 : & \mathbb{D}_0^\sharp & \longrightarrow & \mathcal{P}(\mathbb{S}^\star) \\ & \ell & \longmapsto & \mathbb{S}^\star \cdot (\{\ell\} \times \mathbb{M}) \end{array}$$

This produces the same flow sensitive abstraction as with state partitioning; in the following we always compose context sensitive abstraction with other abstractions...

Trace partitioning is more general than state partitioning

It can also express

- **context-sensitivity**, **partial context sensitivity**
- partitioning guided by a **boolean condition**...

Application 2: partitioning guided by a condition

We consider a program with a **conditional statement**:

$$
\begin{array}{ll}
\ell_0: & \mathbf{if}(\mathrm{c})\{ \\
\ell_1: & \qquad \ldots \\
\ell_2: & \}\mathbf{else}\{ \\
\ell_3: & \qquad \ldots \\
\ell_4: & \} \\
\ell_5: & \ldots
\end{array}
$$

Domain of partitions

The partitions are defined by $\mathbb{D}_0^\sharp = \{\tau_{\mathrm{if}:\mathbf{t}}, \tau_{\mathrm{if}:\mathbf{f}}, \top\}$ and:

$$
\gamma_0 : \begin{array}{lll}
\tau_{\mathrm{if}:\mathbf{t}} & \longmapsto & \{\langle(\ell_0, m), (\ell_1, m'), \ldots\rangle \mid m \in \mathbb{M}, m' \in \mathbb{M}\} \\
\tau_{\mathrm{if}:\mathbf{f}} & \longmapsto & \{\langle(\ell_0, m), (\ell_3, m'), \ldots\rangle \mid m \in \mathbb{M}, m' \in \mathbb{M}\} \\
\top & \longmapsto & \mathbb{S}^\star
\end{array}
$$

Application: **discriminate the executions depending on the branch they visited**

Application 2: partitioning guided by a condition

This partitioning **resolves the second example**:

$$
\begin{array}{l}
\textbf{int}\ \mathrm{x} \in \mathbb{Z};\\
\textbf{int}\ \mathrm{s};\\
\textbf{int}\ \mathrm{y};\\
\textbf{if}(\mathrm{x} \geq 0)\{\\
\qquad\qquad \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow (0 \leq \mathrm{x}) \wedge \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow \bot\\
\qquad \mathrm{s} = 1;\\
\qquad\qquad \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow (0 \leq \mathrm{x} \wedge \mathrm{s} = 1) \wedge \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow \bot\\
\}\ \textbf{else}\ \{\\
\qquad\qquad \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow (\mathrm{x} < 0) \wedge \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow \bot\\
\qquad \mathrm{s} = -1;\\
\qquad\qquad \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \wedge \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow \bot\\
\}\\
\qquad\qquad \left\{ \begin{array}{ccc} & \tau_{\mathrm{if:}\mathbf{t}} & \Rightarrow & (0 \leq \mathrm{x} \wedge \mathrm{s} = 1)\\ \wedge & \tau_{\mathrm{if:}\mathbf{f}} & \Rightarrow & (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \end{array} \right.\\
\mathrm{y} = \mathrm{x}/\mathrm{s};\\
\qquad\qquad \left\{ \begin{array}{ccc} & \tau_{\mathrm{if:}\mathbf{t}} & \Rightarrow & (0 \leq \mathrm{x} \wedge \mathrm{s} = 1 \wedge 0 \leq \mathrm{y})\\ \wedge & \tau_{\mathrm{if:}\mathbf{f}} & \Rightarrow & (\mathrm{x} < 0 \wedge \mathrm{s} = -1 \wedge 0 < \mathrm{y}) \end{array} \right.
\end{array}
$$

Application 3: partitioning guided by a loop

We consider a program with a **conditional statement**:

$$
\begin{array}{ll}
\ell_0: & \textbf{while}(\mathtt{c})\{ \\
\ell_1: & \quad\quad \ldots \\
\ell_2: & \} \\
\ell_3: & \ldots
\end{array}
$$

Domain of partitions

For a given $k \in \mathbb{N}$, the partitions are defined by $\mathbb{D}_0^\sharp = \{\tau_{\mathrm{loop}:0}, \tau_{\mathrm{loop}:1}, \ldots, \tau_{\mathrm{loop}:k}, \top\}$ and:

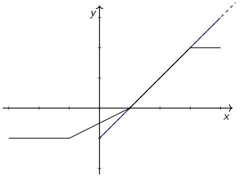
$$
\begin{array}{rlcl}
\gamma_0: & \tau_{\mathrm{loop}:i} & \longmapsto & \text{traces that visit } \ell_1 \ i \text{ times} \\
 & \top & \longmapsto & \mathbb{S}^\star
\end{array}
$$

Application: **discriminate executions depending on the number of iterations in a loop**

Application 3: partitioning guided by a loop

An interpolation function:

$$y = \begin{cases} -1 & \text{if } x \leq -1 \\ -\frac{1}{2} + \frac{x}{2} & \text{if } x \in [-1, 1] \\ -1 + x & \text{if } x \in [1, 3] \\ 2 & \text{if } 3 \leq x \end{cases}$$



Typical implementation:

- use tables of coefficients and loops to search for the range of x

$$\begin{array}{l} \textbf{int } \mathtt{i} = 0; \\ \textbf{while}(\mathtt{i} < 4 \ \&\& \ \mathtt{x} > \mathtt{t}_x[\mathtt{i}+1])\{ \\ \quad \mathtt{i}++; \\ \} \\ \qquad \left\{ \begin{array}{lcr} \tau_{\text{loop:0}} & \Rightarrow & \mathtt{x} \leq -1 \\ \tau_{\text{loop:1}} & \Rightarrow & -1 \leq \mathtt{x} \leq 1 \\ \tau_{\text{loop:2}} & \Rightarrow & 1 \leq \mathtt{x} \leq 3 \\ \tau_{\text{loop:3}} & \Rightarrow & 3 \leq \mathtt{x} \end{array} \right. \\ \mathtt{y} = \mathtt{t}_c[\mathtt{i}] \times (\mathtt{x} - \mathtt{t}_x[\mathtt{i}]) + \mathtt{t}_y[\mathtt{i}] \end{array}$$

Application 4: partitioning guided by the value of a variable

We consider a program with an integer **variable** x, and a **program point** ℓ:

$$\textbf{int } \mathrm{x}; \ldots; \ell : \ldots$$

Domain of partitions: partitioning by the value of a variable

For a given $\mathcal{E} \subseteq \mathbb{V}_{\mathrm{int}}$ finite set of integer values, the partitions are defined by $\mathbb{D}_0^\sharp = \{\tau_{\mathrm{val}:i} \mid i \in \mathcal{E}\} \uplus \{\top\}$ and:

$$\gamma_0 : \begin{array}{lcl} \tau_{\mathrm{val}:k} & \longmapsto & \{\langle \ldots, (\ell, m), \ldots \rangle \mid m(\mathrm{x}) = k\} \\ \top & \longmapsto & \mathbb{S}^\star \end{array}$$

Domain of partitions: partitioning by the property of a variable

For a given abstraction $\gamma : (V^\sharp, \sqsubseteq^\sharp) \rightarrow (\mathcal{P}(\mathbb{V}_{\mathrm{int}}), \subseteq)$, the partitions are defined by $\mathbb{D}_0^\sharp = \{\tau_{\mathrm{var}:v^\sharp} \mid v^\sharp \in V^\sharp\}$ and:

$$\gamma_0 : \tau_{\mathrm{val}:v^\sharp} \longmapsto \{\langle \ldots, (\ell, m), \ldots \rangle \mid m(\mathrm{x}) \in \tau_{\mathrm{var}:v^\sharp}\}$$

Application 4: partitioning guided by the value of a variable

- Left side abstraction shown in blue: **sign of x at entry**
- Right side abstraction shown in green:
  non relational abstraction (we omit the information about x)
- **Same precision** and **similar results** as boolean partitioning,
  but **very different abstraction**, fewer partitions, no re-partitioning

bool b_0, b_1;
int x, y; (uninitialized)

① $(x < 0@① \Rightarrow \top) \wedge (x = 0@① \Rightarrow \top) \wedge (x > 0@① \Rightarrow \top)$

$b_0 = x \geq 0;$

$(x < 0@① \Rightarrow \neg b_0) \wedge (x = 0@① \Rightarrow b_0) \wedge (x > 0@① \Rightarrow b_0)$

$b_1 = x \leq 0;$

$(x < 0@① \Rightarrow \neg b_0 \wedge b_1) \wedge (x = 0@① \Rightarrow b_0 \wedge b_1) \wedge (x > 0@① \Rightarrow b_0 \wedge \neg b_1)$

if$(b_0$ && $b_1)\{$

$(x < 0@① \Rightarrow \bot) \wedge (x = 0@① \Rightarrow b_0 \wedge b_1) \wedge (x > 0@① \Rightarrow \bot)$

$y = 0;$

$(x < 0@① \Rightarrow \bot) \wedge (x = 0@① \Rightarrow b_0 \wedge b_1 \wedge y = 0) \wedge (x > 0@① \Rightarrow \bot)$

$\}$ **else** $\{$

$(x < 0@① \Rightarrow \neg b_0 \wedge b_1) \wedge (x = 0@① \Rightarrow \bot) \wedge (x > 0@① \Rightarrow b_0 \wedge \neg b_1)$

$y = 100/x;$

$(x < 0@① \Rightarrow \neg b_0 \wedge b_1 \wedge y \leq 0) \wedge (x = 0@① \Rightarrow \bot) \wedge (x > 0@① \Rightarrow b_0 \wedge \neg b_1 \wedge y \geq 0)$

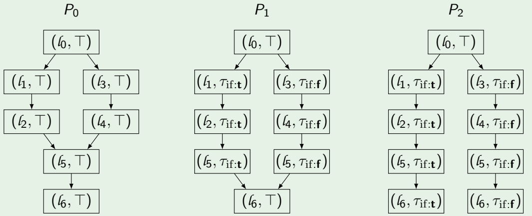
$\}$

Trace partitioning induced by a refined transition system

We consider the **partitions induced by a condition**:

- P_0: the analysis may *never* merge traces from both branches
- P_1: the analysis may merge them *right after the condition* (this amounts to doing no partitioning at all)
- P_2: the analysis may merge them *at some point*

ℓ_0 **if**$(x < 0)\{$
ℓ_1 $\quad s = -1;$
ℓ_2 $\}$ **else** $\{$
ℓ_3 $\quad s = 1;$
ℓ_4 $\}$
ℓ_5 $y = x/s;$
ℓ_6 $\ldots$



Intuition: we can view this form of trace partitioning as **the use of a refined control flow graph**

Trace partitioning induced by a refined transition system

We now **formalize this intuition**:

- we **augment** control states **with partitioning tokens**: $\mathbb{L}' = \mathbb{L} \times \mathbb{D}_0^\sharp$ and let $\mathbb{S}' = \mathbb{L}' \times \mathbb{M}$
- let $\rightarrow' \subseteq \mathbb{S}' \times \mathbb{S}'$ be an **extended transition relation**

Partition of a transition system

System $\mathcal{S}' = (\mathbb{S}', \rightarrow', \mathbb{S}'_{\mathcal{I}})$ is a **partition** of transition system $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$ (and note $\mathcal{S}' \prec \mathcal{S}$) if and only if

- $\forall (\ell, m) \in \mathbb{S}_{\mathcal{I}},\ \exists \tau \in \mathbb{D}_0^\sharp,\ ((\ell, \tau), m) \in \mathbb{S}'_{\mathcal{I}}$
- $\forall (\ell, m), (\ell', m') \in \mathbb{S},\ \forall \tau \in \mathbb{D}_0^\sharp,$
  $$(\ell, m) \rightarrow (\ell', m') \Longrightarrow \exists \tau' \in \mathbb{D}_0^\sharp,\ ((\ell, \tau), m) \rightarrow ((\ell', \tau'), m')$$

Then:

$$\forall \langle (\ell_0, m_0), \ldots, (\ell_n, m_n) \rangle \in [\![\mathcal{S}]\!]_{\mathcal{R}},$$
$$\exists \tau_0, \ldots, \tau_n \in \mathbb{D}_0^\sharp,\ \langle ((\ell_0, \tau_0), m_0), \ldots, ((\ell_n, \tau_n), m_n) \rangle \in [\![\mathcal{S}']\!]_{\mathcal{R}},$$

Trace partitioning induced by a refined transition system

Assumptions:

- **refined control system** $(\mathbb{S}', \rightarrow', \mathbb{S}'_{\mathcal{I}}) \prec (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$
- **erasure function**: $\Psi : (\mathbb{S}')^\star \rightarrow \mathbb{S}^\star$ removes the tokens

Definition of a trace partitioning

The abstraction defining partitions is defined by:

$$\begin{array}{rrcl} \gamma_0 : & \mathbb{D}_0^\sharp & \longrightarrow & \mathcal{P}(\mathbb{S}^\star) \\ & \tau & \longmapsto & \{\sigma \in \mathbb{S}^\star \mid \exists \sigma' = \langle \ldots, ((\ell, \tau), m) \rangle \in (\mathbb{S}')^\star,\ \Psi(\sigma') = \sigma\} \end{array}$$

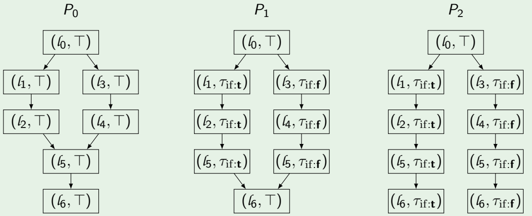
Not all instances of trace partitionings can be expressed that way but **many interesting instances can**:

- **control states** and **call stack** partitioning
- partitioning guided by **conditions** and **loops**
- partitioning **guided by the value of a variable**

Trace partitioning induced by a refined transition system

Example of the **partitioning guided by a condition**:

ℓ_0 if$(\mathrm{x} < 0)\{$
ℓ_1 $\quad \mathrm{s} = -1;$
ℓ_2 $\}$ else $\{$
ℓ_3 $\quad \mathrm{s} = 1;$
ℓ_4 $\}$
ℓ_5 $\mathrm{y} = \mathrm{x}/\mathrm{s};$
ℓ_6 $\ldots$



- each system induces a partitioning, with different merging points:

$$P_1 \prec P_0 \qquad\qquad P_2 \prec P_1$$

- these systems induce **hierarchy** of refining control structures

$$P_2 \prec P_1 \prec P_0$$

- this approach **also applies to**:
  - partitioning **induced by a loop**
  - partitioning **induced by the value of a variable at a given point**...

Transfer functions: example

$\mathbf{int}\ \mathrm{x} \in \mathbb{Z};$
$\mathbf{int}\ \mathrm{s};$
$\mathbf{int}\ \mathrm{y};$
$\mathbf{if}(\mathrm{x} \geq 0)\{$

$\tau_{\mathrm{if:}\mathbf{t}} \Rightarrow (0 \leq \mathrm{x}) \wedge \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow \bot$ — partition creation: $\tau_{\mathrm{if:}\mathbf{t}}$

$\quad \mathrm{s} = 1;$

$\tau_{\mathrm{if:}\mathbf{t}} \Rightarrow (0 \leq \mathrm{x} \wedge \mathrm{s} = 1) \wedge \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow \bot$ — no modification of partitions

$\}\ \mathbf{else}\ \{$

$\tau_{\mathrm{if:}\mathbf{f}} \Rightarrow (\mathrm{x} < 0) \wedge \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow \bot$ — partition creation: $\tau_{\mathrm{if:}\mathbf{f}}$

$\quad \mathrm{s} = -1;$

$\tau_{\mathrm{if:}\mathbf{f}} \Rightarrow (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \wedge \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow \bot$ — no modification of partitions

$\}$

$$\left\{ \begin{array}{lll} & \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow & (0 \leq \mathrm{x} \wedge \mathrm{s} = 1) \\ \wedge & \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow & (\mathrm{x} < 0 \wedge \mathrm{s} = -1) \end{array} \right.$$
no modification of partitions

$\mathrm{y} = \mathrm{x}/\mathrm{s};$

$$\left\{ \begin{array}{lll} & \tau_{\mathrm{if:}\mathbf{t}} \Rightarrow & (0 \leq \mathrm{x} \wedge \mathrm{s} = 1 \wedge 0 \leq \mathrm{y}) \\ \wedge & \tau_{\mathrm{if:}\mathbf{f}} \Rightarrow & (\mathrm{x} < 0 \wedge \mathrm{s} = -1 \wedge 0 < \mathrm{y}) \end{array} \right.$$
no modification of partitions

$\ldots$

$\_ \Rightarrow \mathrm{s} \in [-1, 1] \wedge 0 \leq \mathrm{y}$ — fusion of partitions

Partitions are rarely modified, and only *some* (branching) points

Transfer functions: partition creation

Analysis of an if statement, with partitioning

$$
\begin{array}{ll}
\ell_0: & \textbf{if}(\mathtt{c})\{ \\
\ell_1: & \quad\quad \ldots \\
\ell_2: & \}\textbf{else}\{ \\
\ell_3: & \quad\quad \ldots \\
\ell_4: & \} \\
\ell_5: & \ldots
\end{array}
$$

$$
\begin{array}{rcl}
\delta^\sharp_{\ell_0,\ell_1}(X^\sharp) & = & [\tau_{\mathrm{if:}\mathbf{t}} \mapsto \mathit{test}(\mathtt{c}, \sqcup X^\sharp(\ell_0)(\tau)), \top \mapsto \bot] \\
\delta^\sharp_{\ell_0,\ell_3}(X^\sharp) & = & [\tau_{\mathrm{if:}\mathbf{t}} \mapsto \mathit{test}(\neg\mathtt{c}, \sqcup_\tau X^\sharp(\ell_0)(\tau)), \top \mapsto \bot] \\
\delta^\sharp_{\ell_2,\ell_5}(X^\sharp) & = & X^\sharp \\
\delta^\sharp_{\ell_4,\ell_5}(X^\sharp) & = & X^\sharp
\end{array}
$$

Observations:

- in the body of the condition: either $\tau_{\mathrm{if:}\mathbf{t}}$ or $\tau_{\mathrm{if:}\mathbf{f}}$
  i.e., **no partition modification there**
- effect at point ℓ_5: **both $\tau_{\mathrm{if:}\mathbf{t}}$ and $\tau_{\mathrm{if:}\mathbf{f}}$ exist**
- **partitions are modified only at the condition point**, that is only by $\delta^\sharp_{\ell_0,\ell_1}(X^\sharp)$ and $\delta^\sharp_{\ell_0,\ell_2}(X^\sharp)$

Transfer functions: partition fusion

When **partitions are not useful anymore, they can be merged**

$$\delta^{\sharp}_{\ell_0,\ell_1}(X^{\sharp}) \quad = \quad [\_ \mapsto \sqcup_{\tau} X^{\sharp}(\ell_0)(\tau)]$$

Remarks:

- at this point, all partitions are **effectively collapsed** into just one set
- **example**: fusion of the partition of a condition when not useful
- **choice of fusion point**:
  - **precision**: merge point should not occur as long as partitions are useful
  - **efficiency**: merge point should occur as early as partitions are not needed anymore

Choice of partitions

How are the partitions chosen ?

Static partitioning

- a fixed partitioning abstraction $\mathbb{D}_0^\sharp, \gamma_0$ is **fixed before the analysis**
- usually $\mathbb{D}_0^\sharp, \gamma_0$ are chosen by a pre-analysis

- static partitioning is rather easy to formalize and implement
- but it might be limiting, when the choice of partitions is hard

Dynamic partitioning

- the partitioning abstraction $\mathbb{D}_0^\sharp, \gamma_0$ is **not fixed before the analysis**
- instead, it is **computed as part of the analysis**
- i.e., the analysis uses on a lattice of partitioning abstractions $\mathcal{D}^\sharp$ and computes $(\mathbb{D}_0^\sharp, \gamma_0)$ as an element of this lattice

Outline

Adding disjunctions in static analyses

- **Disjunctive completion**: **brutally adds disjunctions**
  too expensive in practice

- **Cardinal power abstraction** expresses collections of implications between abstract facts in **two abstract domains**

  $$(P_0 \Longrightarrow Q_0) \wedge \ldots \wedge (P_n \Longrightarrow Q_n)$$

  State partitioning and **trace partitioning** are particular cases of cardinal power abstraction

- **State partitioning** is **easier** to use when the criteria for partitioning can be easily expressed at the state level

- **Trace partitioning** is **more expressive** in general
  it can also allow the use of **simpler partitioning criteria**, with less "re-partitioning"

Assignment: paper reading

Refining static analyses by trace-partitioning using control flow
Maria Handjieva and Stanislas Tzolovski,
Static Analysis Symposium, 1998,
http://link.springer.com/chapter/10.1007/3-540-49727-7_12

Abstract interpretation by dynamic partitioning,
François Bourdoncle,
Journal of Functional Programming, 2(4) 407–423, 1992.
Extended report available at:
http://www.hpl.hp.com/techreports/Compaq-DEC/PRL-RR-18.pdf