Partitioning abstractions

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

Xavier Rival

INRIA, ENS, CNRS

Oct, 29th. 2014

Towards disjunctive abstractions

- **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

Notations

for sets:

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

Assumptions:

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

Product combinator (implemented as a functor)

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

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

This amounts to expressing conjunctions of elements of $\mathbb{D}_0^\sharp$ and $\mathbb{D}_1^\sharp$

Example: product abstraction, coalescent product

The product abstraction **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**:

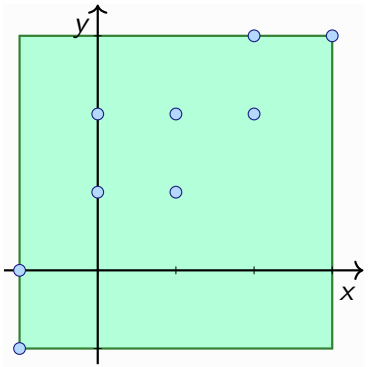
- let $\mathbb{D}_0^\sharp$ be the interval abstraction, $\mathbb{D}_1^\sharp$ be the congruences abstraction
- $\gamma_\times(\{\mathtt{x} \in [3,4]\}, \{\mathtt{x} \equiv 0 \mod 5\}) = \emptyset$

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

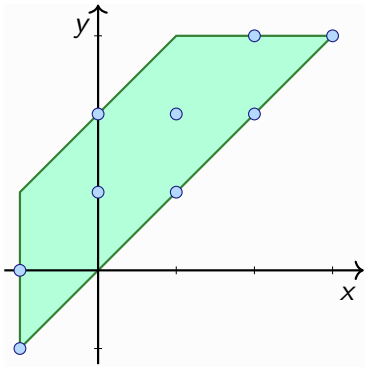
Outline

Convex abstractions

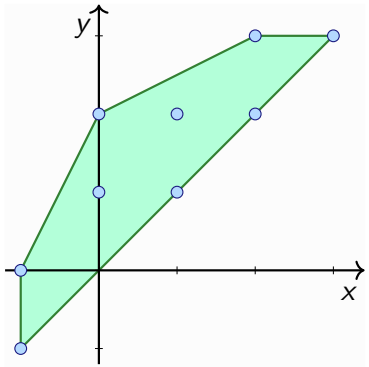
Many numerical abstractions describe **convex sets of points**



interval domain

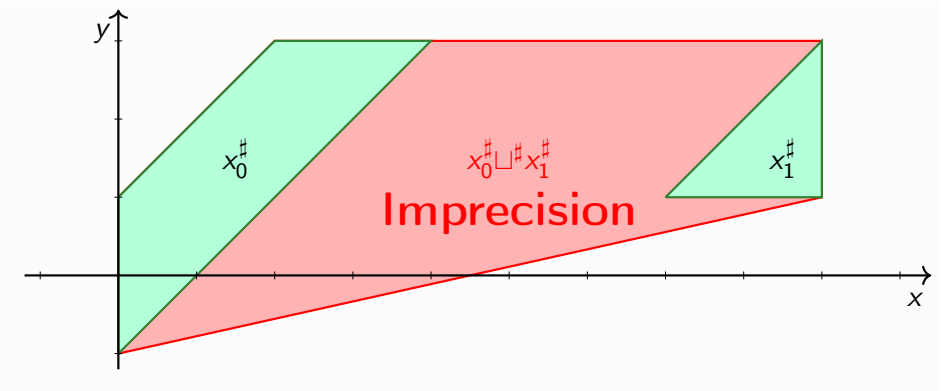


octagon domain



polyedra domain

Imprecisions inherent in the **convexity**, and when computing **abstract join:**



Such imprecisions may impact analysis results

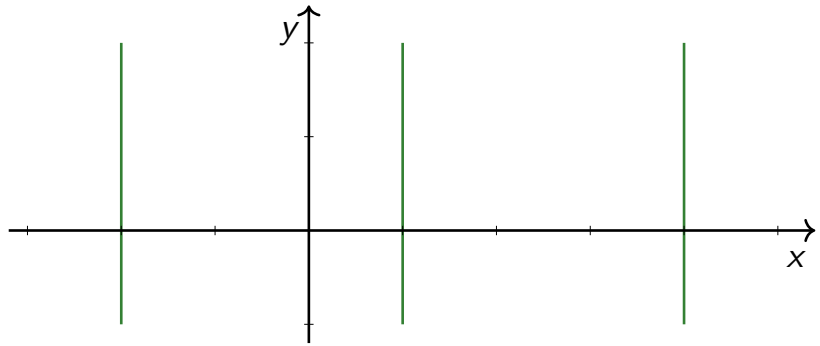
Non convex abstractions

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

Congruences:

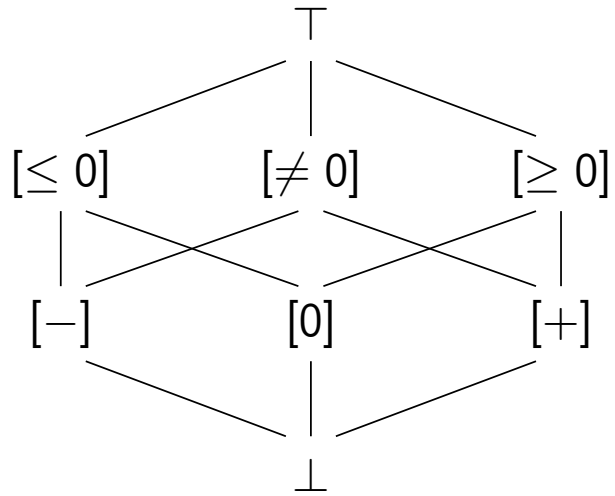
- $\mathbb{D}^\sharp = \mathbb{Z} \times \mathbb{N}$
- $\gamma(n, k) = \{n + k \cdot p \mid p \in \mathbb{Z}\}$
- $-2, 1 \in \gamma(1, 2)$
  but $0 \notin \gamma(1, 2)$

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:

$$\begin{cases} b_0 & = & \texttt{FALSE} \\ b_1 & = & \texttt{FALSE} \\ x & : & \top \end{cases}$$

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

Example 1: program annotated with local invariants

bool b_0, b_1;
int x, y; (uninitialized)
$b_0 = x \geq 0$;
$(b_0 \wedge x \geq 0) \vee (\neg b_0 \wedge x < 0)$
$b_1 = x \leq 0$;
$(b_0 \wedge b_1 \wedge x = 0) \vee (b_0 \wedge \neg b_1 \wedge x > 0) \vee (\neg b_0 \wedge b_1 \wedge x < 0)$
if$(b_0$ && $b_1)\{$
$(b_0 \wedge b_1 \wedge x = 0)$
$y = 0$;
$(b_0 \wedge b_1 \wedge x = 0 \wedge y = 0)$
$\}$ **else** $\{$
$(b_0 \wedge \neg b_1 \wedge x > 0) \vee (\neg b_0 \wedge b_1 \wedge x < 0)$
$y = 100/x$;
$(b_0 \wedge \neg b_1 \wedge x > 0) \vee (\neg b_0 \wedge b_1 \wedge x < 0)$
$\}$

We need to **add 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 fairly complex numerical abstraction

Example 2: program annotated with local invariants

$\mathbf{int}\ \mathtt{x} \in \mathbb{Z};$
$\mathbf{int}\ \mathtt{s};$
$\mathbf{int}\ \mathtt{y};$
$\mathbf{if}(\mathtt{x} \geq 0)\{$
$\qquad (\mathtt{x} \geq 0)$
$\quad \mathtt{s} = 1;$
$\qquad (\mathtt{x} \geq 0 \wedge \mathtt{s} = 1)$
$\}\,\mathbf{else}\,\{$
$\qquad (\mathtt{x} < 0)$
$\quad \mathtt{s} = -1;$
$\qquad (\mathtt{x} < 0 \wedge \mathtt{s} = -1)$
$\}$
$\qquad (\mathtt{x} \geq 0 \wedge \mathtt{s} = 1) \vee (\mathtt{x} < 0 \wedge \mathtt{s} = -1)$
① $\mathtt{y} = \mathtt{x}/\mathtt{s};$
$\qquad (\mathtt{x} \geq 0 \wedge \mathtt{s} = 1 \wedge \mathtt{y} \geq 0) \vee (\mathtt{x} < 0 \wedge \mathtt{s} = -1 \wedge \mathtt{y} > 0)$
② $\mathbf{assert}(\mathtt{y} \geq 0);$

We need to **add disjunctions** to our abstract domain

Outline

Distributive abstract domain

Principle:

1. consider concrete domain $(\mathbb{D}, \sqsubseteq)$, with lower upper bound operator $\sqcup$
2. start with an abstract domain $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ with concretization $\gamma : \mathbb{D}^\sharp \to \mathbb{D}$
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 $\gamma : \mathbb{D}^\sharp \to \mathbb{D}$ 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** $(\mathbb{D}^\sharp_{\textbf{disj}}, \sqsubseteq^\sharp_{\textbf{disj}})$ with concretization function $\gamma_{\textbf{disj}} : \mathbb{D}^\sharp_{\textbf{disj}} \to \mathbb{D}$ such that:

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

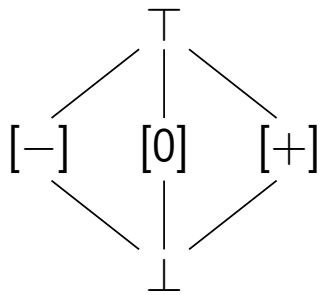
Building a disjunctive completion domain:

- start with $\mathbb{D}^\sharp_{\textbf{disj}} = \mathbb{D}^\sharp$
- 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 $\mathbb{D}^\sharp_{\textbf{disj}}$, and extend $\gamma_{\textbf{disj}}$ by

$$\gamma_{\textbf{disj}}([\sqcup\mathcal{E}]) = \bigsqcup_{y^\sharp \in \mathcal{E}} \gamma(y^\sharp)$$

Example 1: completion of signs

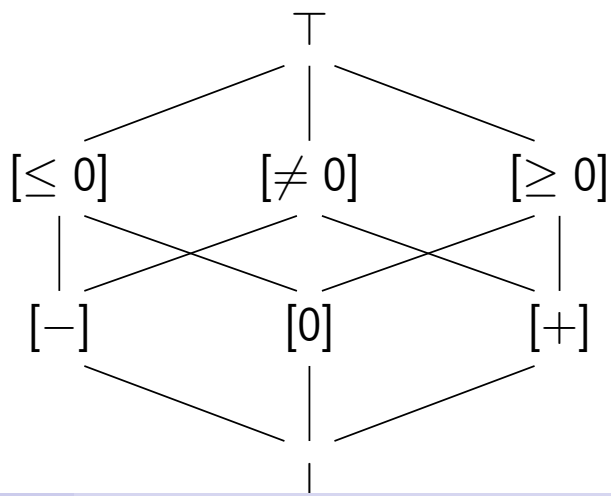
We consider **concrete lattice** $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$ and $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ defined by:



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

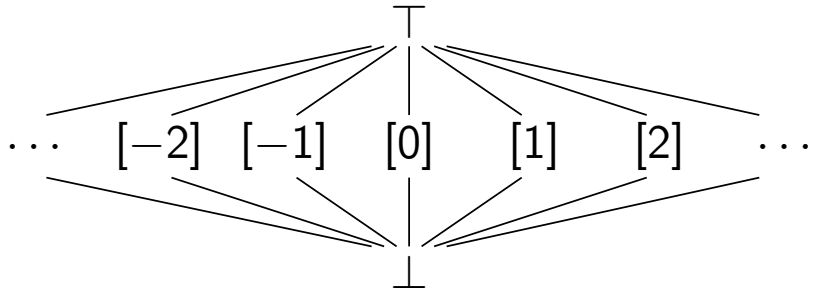
Then, the disjunctive completion is defined by adding elements corresponding to:

- $\{[< 0], [= 0]\}$
- $\{[< 0], [> 0]\}$
- $\{[= 0], [> 0]\}$



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:



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

Then, the disjunctive completion is the power-set:

- $\mathbb{D}^\sharp_{\textbf{disj}} \equiv \mathcal{P}(\mathbb{Z})$
- $\gamma_{\textbf{disj}}$ 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**

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:

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

Static analysis with disjunctive completion

Transfer functions:

- e.g. to compute **abstract post-conditions** (assingment, guard...): given concrete $\tau : \mathbb{D} \to \mathbb{D}$, we assume $\tau^\sharp : \mathbb{D}^\sharp \to \mathbb{D}^\sharp$ such that:

$$\tau \circ \gamma \sqsubseteq \gamma \circ \tau^\sharp$$

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

$$\tau^\sharp_{\textbf{disj}}([\sqcup \mathcal{E}]) = \sqcup[\{\tau^\sharp(x^\sharp) \mid x^\sharp \in \mathcal{E}\}]$$

Abstract join:

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

Inclusion check: exercise !

Limitations of disjunctive completion

- **Combinatorial explosion**:
  - if $\mathbb{D}^\sharp$ is infinite, $\mathbb{D}^\sharp_{\textbf{disj}}$ may have elements that **cannot be represented**
  - even when $\mathbb{D}^\sharp$ is finite, $\mathbb{D}^\sharp_{\textbf{disj}}$ may be **huge**

    in the worst case, if $\mathbb{D}^\sharp$ has n elements, $\mathbb{D}^\sharp_{\textbf{disj}}$ 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

  issue: the join operator should "select" which disjoints to merge

Outline

Principle

- disjuncts **that are needed for static analysis** can usually be **characterized** by some property

  for instance:
  - **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

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

- natural "indexing": **sign of** $\mathtt{x}$
- but we could also rely on the **sign of** $\mathtt{s}$

Cardinal power abstraction

Definition

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

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

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

- $\mathbb{D}_{\mathbf{cp}}^\sharp = \mathbb{D}_0^\sharp \overset{\mathcal{M}}{\to} \mathbb{D}_1^\sharp$ 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 $\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp$, $\gamma_{\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp}$.

Use of cardinal power abstractions

Intuition: we can express properties of the form

$$\begin{cases} & p_0 & \Longrightarrow & p'_0 \\ \wedge & p_1 & \Longrightarrow & p'_1 \\ \vdots & \vdots & \vdots & \vdots \\ \wedge & p_n & \Longrightarrow & p'_n \end{cases}$$

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

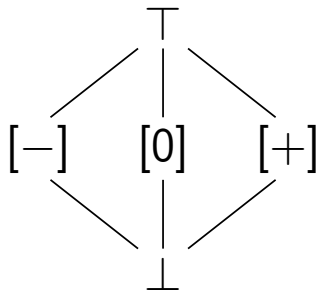
Application $(\mathtt{x} \geq 0 \wedge \mathtt{s} = 1 \wedge \mathtt{y} \geq 0) \vee (\mathtt{x} < 0 \wedge \mathtt{s} = -1 \wedge \mathtt{y} > 0)$

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

Another example, with a single variable

We consider:

- 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**

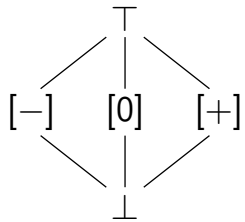


A few example abstract values:

- $[0, 8]$ is expressed by: $\begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [< 0] & \longmapsto & \bot_1 \\ [= 0] & \longmapsto & [0, 0] \\ [> 0] & \longmapsto & [1, 8] \\ \top_0 & \longmapsto & [0, 8] \end{cases}$
- $[-10, -3] \uplus [7, 10]$ is expressed by: $\begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [< 0] & \longmapsto & [-10, -3] \\ [= 0] & \longmapsto & \bot_1 \\ [> 0] & \longmapsto & [7, 10] \\ \top_0 & \longmapsto & [-10, 10] \end{cases}$

Reduction (1): tightening disjunctions

- 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**



$$\text{We let: } X^\sharp = \begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [<0] & \longmapsto & [-5,-1] \\ [=0] & \longmapsto & [0,0] \\ [>0] & \longmapsto & [1,5] \\ \top_0 & \longmapsto & [-10,10] \end{cases} \qquad Y^\sharp = \begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [<0] & \longmapsto & [-5,-1] \\ [=0] & \longmapsto & [0,0] \\ [>0] & \longmapsto & [1,5] \\ \top_0 & \longmapsto & [-5,5] \end{cases}$$

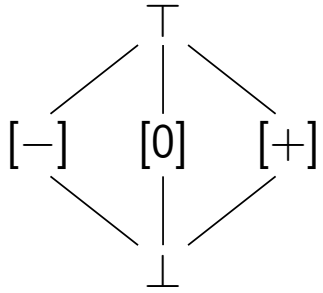
- Then, $\gamma_{\mathbf{cp}}(X^\sharp) = \gamma_{\mathbf{cp}}(Y^\sharp)$
- $\gamma_0([<0]) \cup \gamma_0([=0]) \cup \gamma([>0]) = \gamma(\top_0)$
  but $\gamma_0(X^\sharp([<0])) \cup \gamma_0(X^\sharp([=0])) \cup \gamma(X^\sharp([>0])) \subset \gamma(X^\sharp(\top_0))$

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

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

Reduction (2): 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:

$$X^\sharp = \begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [< 0] & \longmapsto & [1, 8] \\ [= 0] & \longmapsto & [1, 8] \\ [> 0] & \longmapsto & \bot_1 \\ \top_0 & \longmapsto & [1, 8] \end{cases} \quad Y^\sharp = \begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [< 0] & \longmapsto & [2, 45] \\ [= 0] & \longmapsto & [-5, -2] \\ [> 0] & \longmapsto & [-5, -2] \\ \top_0 & \longmapsto & \top_1 \end{cases} \quad Z^\sharp = \begin{cases} \bot_0 & \longmapsto & \bot_1 \\ [< 0] & \longmapsto & \bot_1 \\ [= 0] & \longmapsto & \bot_1 \\ [> 0] & \longmapsto & \bot_1 \\ \top_0 & \longmapsto & \bot_1 \end{cases}$$

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

Relation between $\mathbb{D}_0^\sharp$ elements and $\mathbb{D}_1^\sharp$ elements

Binding $y_0^\sharp \mapsto y_1^\sharp$ can be improved if $\exists z_1^\sharp \neq y_1^\sharp,\ \gamma(y_1^\sharp) \cap \gamma(y_0^\sharp) \subseteq \gamma(z_1^\sharp)$

Representation of the cardinal power

Basic ML representation:

`type cp = d0 -> d1`	not convenient to operate on d0
`type cp = (d0,d1) map`	maps or functional arrays

This 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:

- keep the **same 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

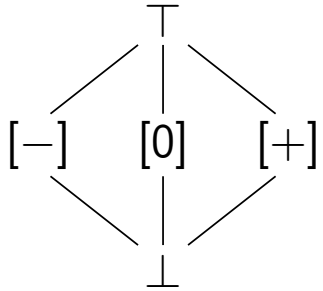
$$\forall x^\sharp \in \mathbb{D}_0^\sharp,\ \exists \mathcal{E} \subseteq \mathcal{C},\ \gamma_0(x^\sharp) = \cup\{\gamma_0(y^\sharp) \mid y^\sharp \in \mathcal{E}\}$$

In particular:

- $\mathcal{C}$ should be **minimal**
- in any case, $\bot_0 \notin \mathcal{C}$

Example: compact cardinal power over signs

- 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 remark that:

- $\bot_0$ does not need be considered
- $\gamma_0([< 0]) \cup \gamma_0([= 0]) \cup \gamma([> 0]) = \gamma(\top_0)$ thus $\top_0$ does not need be considered

Thus, we let $\mathcal{C} = \{[< 0], [= 0], [> 0]\}$; then:

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

Lattice operations

Infimum:

- we assume that $\bot_1$ is the infimum of $\mathbb{D}^\sharp_1$
- then, $\bot_{\mathbf{cp}} = \lambda(z^\sharp \in \mathbb{D}^\sharp_0) \cdot \bot_1$ is the **infimum** of $\mathbb{D}^\sharp_{\mathbf{cp}}$

Ordering:

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

$$X^\sharp_0 \sqsubseteq^\sharp_{\mathbf{cp}} X^\sharp_1 \quad \overset{def}{\iff} \quad \forall z^\sharp \in \mathbb{D}^\sharp_0,\ X^\sharp_0(z^\sharp) \sqsubseteq^\sharp_1 X^\sharp_1(z^\sharp)$$

- then, $X^\sharp_0 \sqsubseteq^\sharp_{\mathbf{cp}} X^\sharp_1 \Longrightarrow \gamma_{\mathbf{cp}}(X^\sharp_0) \subseteq \gamma_{\mathbf{cp}}(X^\sharp_1)$

Join operation:

- we assume that $\sqcup_1$ is a sound upper bound operator in $\mathbb{D}^\sharp_1$
- then, $\sqcup_{\mathbf{cp}}$ defined below is a **sound upper bound operator** in $\mathbb{D}^\sharp_{\mathbf{cp}}$:

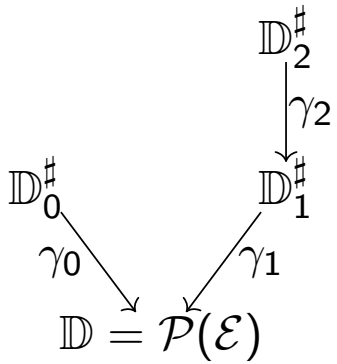
$$X^\sharp_0 \sqcup_{\mathbf{cp}} X^\sharp_1 \quad \overset{def}{::=} \quad \lambda(z^\sharp \in \mathbb{D}^\sharp_0) \cdot (X^\sharp_0(z^\sharp) \sqcup_1 X^\sharp_1(z^\sharp))$$

- the same construction applies to widening, if $\mathbb{D}^\sharp_0$ 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:

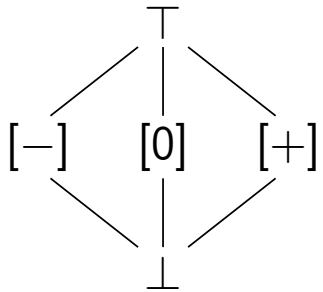
$$\begin{array}{rccc} \gamma : & (\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_2^\sharp) & \longrightarrow & (\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp) \\ & X^\sharp & \longmapsto & \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \gamma(X^\sharp(z^\sharp)) \end{array}$$

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 $\sqsubseteq = \subseteq$
- $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp)$ be the **lattice of signs**
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$ be the **identity abstraction**
  $\mathbb{D}_1^\sharp = \mathcal{P}(\mathbb{Z})$, $\gamma_1 = \mathbf{Id}$
- $(\mathbb{D}_2^\sharp, \sqsubseteq_2^\sharp)$ be the **lattice of intervals**



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

- in $\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_1^\sharp$, $\begin{cases} [<0] & \longmapsto & [-10,-3] \\ [=0] & \longmapsto & \emptyset \\ [>0] & \longmapsto & [7,10] \end{cases}$
- in $\mathbb{D}_0^\sharp \rightrightarrows \mathbb{D}_2^\sharp$, $\begin{cases} [<0] & \longmapsto & [-10,-3] \\ [=0] & \longmapsto & \bot_1 \\ [>0] & \longmapsto & [7,10] \end{cases}$

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 $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp, \gamma_0)$ and $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp, \gamma_1)$ of sets of states:

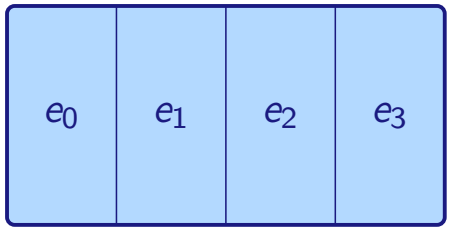
- $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp, \gamma_0)$ defines the **partitions**
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp, \gamma_1)$ defines the **abstraction of each element of partitions**

- either $\mathbb{D}_1^\sharp = \mathcal{P}(\mathbb{S})$, ordered with the inclusion
- or an abstraction of sets of memory states: numerical abstraction can be obtained by composing another abstraction on top of $(\mathcal{P}(\mathbb{S}), \subseteq)$

Instantiation with a partition

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

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



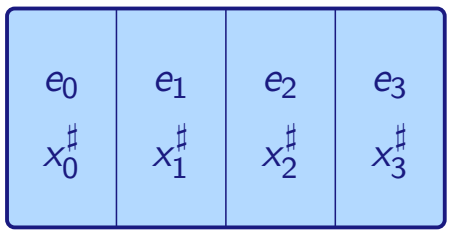
We can apply **cardinal power construction**:

State partitioning abstraction

We let $\mathbb{D}^\sharp_0 = \mathcal{E}$ and $\gamma_0 : e \mapsto e$. Thus, $\mathbb{D}^\sharp_{\mathbf{cp}} = \mathcal{E} \to \mathbb{D}^\sharp_1$ and:

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

- each $e \in \mathcal{E}$ is attached to a piece of information in $\mathbb{D}^\sharp_1$
- exercise: what happens if use only a **covering**, i.e., if we drop property 1 ?



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**:

$$
\begin{array}{ll}
\ell_0: & \texttt{// assume } \mathtt{x} \geq 0 \\
\ell_1: & \textbf{if}(\mathtt{x} < 10)\{ \\
\ell_2: & \quad \mathtt{y} = \mathtt{x} - 2; \\
\ell_3: & \}\textbf{else}\{ \\
\ell_4: & \quad \mathtt{y} = 2 - \mathtt{x}; \\
\ell_5: & \} \\
\ell_6: & \ldots
\end{array}
\qquad
\begin{array}{lcl}
\ell_0 & \mapsto & \mathtt{x} : \top \wedge \mathtt{y} : \top \\
\ell_1 & \mapsto & \mathtt{x} : [0, +\infty[\wedge \mathtt{y} : \top \\
\ell_2 & \mapsto & \mathtt{x} : [0, 9] \wedge \mathtt{y} : \top \\
\ell_3 & \mapsto & \mathtt{x} : [0, 9] \wedge \mathtt{y} : [-2, 7] \\
\ell_4 & \mapsto & \mathtt{x} : [10, +\infty[\wedge \mathtt{y} : \top \\
\ell_5 & \mapsto & \mathtt{x} : [10, +\infty[\wedge \mathtt{y} :] -\infty, -2] \\
\ell_6 & \mapsto & \mathtt{x} : [0, +\infty[\wedge \mathtt{y} :] -\infty, 7]
\end{array}
$$

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:

- $\mathbb{D}_0^\sharp = \mathbb{L}$
- $\gamma_0 : l \mapsto \{l\} \times \mathbb{M}$

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

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

- representing one set of memory states per program point may be costly for some applications (e.g., compilation)
- **context insensitive** abstraction simply **forgets about control states**

Flow sensitive 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 $\mathcal{P}(\mathbb{S})$

- used for some ultra-fast pointer analyses (very quick analyses used for, e.g., compiler optimization)
- otherwise, usually **too coarse**

Application 1: flow insensitive abstraction

We compare with **flow sensitive abstraction:**

$$
\begin{array}{ll}
\ell_0: & \texttt{// assume } x \geq 0 \\
\ell_1: & \textbf{if}(x < 10)\{ \\
\ell_2: & \quad y = x - 2; \\
\ell_3: & \}\textbf{else}\{ \\
\ell_4: & \quad y = 2 - x; \\
\ell_5: & \} \\
\ell_6: & \ldots
\end{array}
\qquad
\begin{array}{lll}
\ell_0 & \mapsto & x : \top \wedge y : \top \\
\ell_1 & \mapsto & x : [0, +\infty[\wedge y : \top \\
\ell_2 & \mapsto & x : [0, 9] \wedge y : \top \\
\ell_3 & \mapsto & x : [0, 9] \wedge y : [-2, 7] \\
\ell_4 & \mapsto & x : [10, +\infty[\wedge y : \top \\
\ell_5 & \mapsto & x : [10, +\infty[\wedge y :]-\infty, -2] \\
\ell_6 & \mapsto & x : [0, +\infty[\wedge y :]-\infty, 7]
\end{array}
$$

- 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**)

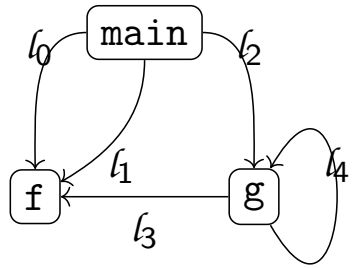
For a few specific applications flow insensitive is ok
In most cases (e.g., numeric programs), flow sensitive is absolutely needed

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**

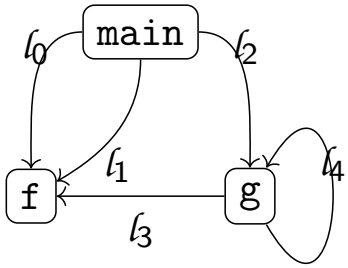
κ	$\in$	$\mathbb{K}$	calling contexts
κ	$::=$	ϵ	empty call stack
	$\mid$	$(f, \ell) \cdot \kappa$	call to f from stack κ at point ℓ

Application 2: context sensitive abstraction, ∞-CFA

Fully context sensitive abstraction (∞-CFA)

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

void main(){$\ldots \ell_0 :$ f(); $\ldots \ell_1 :$ f(); $\ldots \ell_2 :$ g()$\ldots$}
void f(){$\ldots$}
void g(){**if**$(\ldots)$**{**$\ell_3 :$ f()**}else{**$\ell_4 :$ g()**}}**



Contexts in **function** f:

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

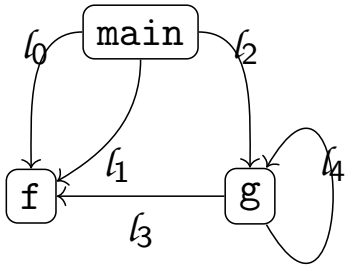
- one invariant per calling context, **very precise** (used, e.g., in Astrée)
- **infinite in presence of recursion** (i.e., not practical in this case)

Application 2: context sensitive abstraction, 0-CFA

Non context sensitive abstraction (0-CFA)

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

void `main`(){$\ldots \ell_0 :$ `f`(); $\ldots \ell_1 :$ `f`(); $\ldots \ell_2 :$ g() $\ldots$}
void `f`(){$\ldots$}
void g(){**if**($\ldots$){$\ell_3 :$ `f`()}**else**{$\ell_4 :$ g()}}



Contexts in **function** `f`:

$$(?, \texttt{f}) \cdot \ldots,$$

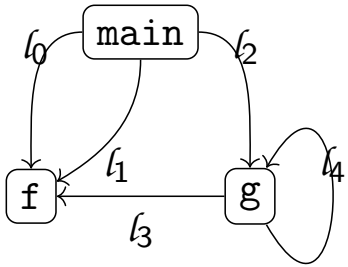
- merges **all** calling contexts to a same procedure, **very coarse** abstraction
- but **usually quite efficient to compute**

Application 2: context sensitive abstraction, k-CFA

Partially context sensitive abstraction (k-CFA)

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

void main(){$\ldots \ell_0 :$ f(); $\ldots \ell_1 :$ f(); $\ldots \ell_2 :$ g()$\ldots$}
void f(){$\ldots$}
void g(){**if**($\ldots$){$\ell_3 :$ f()}**else**{$\ell_4 :$ g()}}



Contexts in **function** 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:

- $\mathbb{D}_0^\sharp = \mathcal{P}(A)$ for some set A, and $\phi : \mathbb{M} \to A$
- γ_0 be of the form $(x^\sharp \in \mathbb{D}_0^\sharp) \mapsto \{(\ell, m) \in \mathbb{S} \mid \phi(m) \in x^\sharp\}$

Common choice for A: **the set of boolean values** $\mathbb{B}$
(or a variation of this)

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

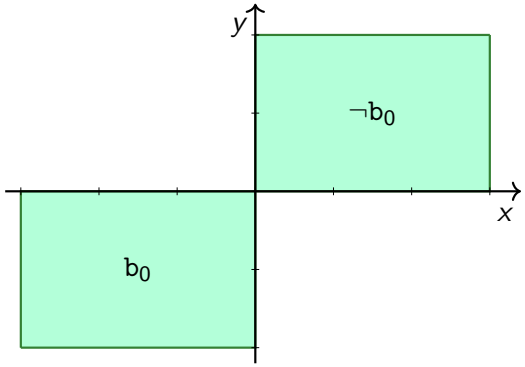
We apply the cardinal power abstraction, with a domain of partition defined by a function, with:

- $A = \mathbb{B}^k$
- $\phi(m) = (m(\mathtt{b}_0), \ldots, m(\mathtt{b}_{k-1}))$
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp, \gamma_1)$ an abstraction of $\mathcal{P}(\mathbb{V}_{\text{int}}^l)$

Application 3: example

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

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



- 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
- **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 b_0, b_1
- Right side abstraction shown in green: interval abstraction

bool b_0, b_1;
int x, y; (uninitialized)
$b_0 = x \geq 0$;
$(b_0 \implies x \geq 0) \wedge (\neg b_0 \implies x < 0)$
$b_1 = x \leq 0$;
$(b_0 \wedge b_1 \implies x = 0) \wedge (b_0 \wedge \neg b_1 \implies x > 0) \wedge (\neg b_0 \wedge b_1 \implies x < 0)$
if(b_0 && b_1){
$(b_0 \wedge b_1 \implies x = 0)$
y = 0;
$(b_0 \wedge b_1 \implies x = 0 \wedge y = 0)$
}**else**{
$(b_0 \wedge \neg b_1 \implies x > 0) \wedge (\neg b_0 \wedge b_1 \implies x < 0)$
y = 100/x;
$(b_0 \wedge \neg b_1 \implies x > 0 \wedge y \geq 0) \wedge (\neg b_0 \wedge b_1 \implies x < 0 \wedge y \leq 0)$
}

Application 3: partitioning by the sign of a variable

We assume:

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

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

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 $\mathcal{P}(\mathbb{V}_{\text{int}}^{l-1})$ (no need to abstract $\mathtt{x}_0$ twice)

Application 3: example

- Abstraction fixing partitions shown in blue
- Right side abstraction shown in green: interval abstraction

int $x \in \mathbb{Z}$;
int s;
int y;
if$(x \geq 0)\{$
$(x < 0 \Rightarrow \bot) \wedge (x = 0 \Rightarrow \top) \wedge (x > 0 \Rightarrow \top)$
$s = 1;$
$(x < 0 \Rightarrow \bot) \wedge (x = 0 \Rightarrow s = 1) \wedge (x > 0 \Rightarrow s = 1)$
$\}$ **else** $\{$
$(x < 0 \Rightarrow \top) \wedge (x = 0 \Rightarrow \bot) \wedge (x > 0 \Rightarrow \bot)$
$s = -1;$
$(x < 0 \Rightarrow s = -1) \wedge (x = 0 \Rightarrow \bot) \wedge (x > 0 \Rightarrow \bot)$
$\}$
$(x < 0 \Rightarrow s = -1) \wedge (x = 0 \Rightarrow s = 1) \wedge (x > 0 \Rightarrow s = 1)$
① $y = x/s;$
$(x < 0 \Rightarrow s = -1 \wedge y > 0) \wedge (x = 0 \Rightarrow s = 1 \wedge y = 0) \wedge (x > 0 \Rightarrow s = 1 \wedge y > 0)$
② **assert**$(y \geq 0)$;

Outline

Computation of abstract semantics and partitioning

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

This abstraction corresponds to a **Galois connection**:

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

where $\mathbb{D}^\sharp_{\text{part}} = \mathbb{L} \to \mathcal{P}(\mathbb{M})$ and:

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

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 $\mathbb{D}^{\sharp}_{\text{part}}$ by:

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

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

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

Abstract equations form of a partitioned semantics

- we look for a set of equivalent abstract equations
- let us consider the system of semantic equations over sets of states $\mathcal{E}_1, \ldots, \mathcal{E}_s \in \mathcal{P}(\mathbb{M})$:

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

If we let $F_i : (\mathcal{E}_1, \ldots, \mathcal{E}_s) \mapsto \bigcup_i \{m \in \mathbb{M} \mid \exists m' \in \mathcal{E}_i,\ (l_i, m') \to (l_i, m)\}$, then, we can prove that:

$\alpha_{\text{part}}([\![\mathcal{S}]\!]_{\mathcal{R}})$ is the least solution of the system $\begin{cases} \mathcal{E}_1 &= F_1(\mathcal{E}_1, \ldots, \mathcal{E}_s) \\ & \vdots \\ \mathcal{E}_s &= F_s(\mathcal{E}_1, \ldots, \mathcal{E}_s) \end{cases}$

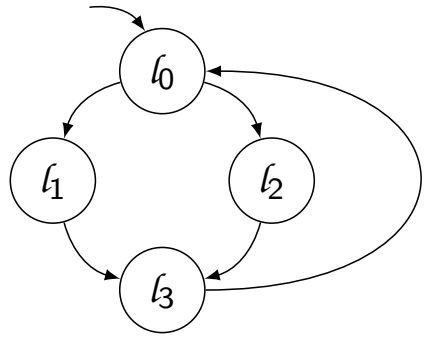
Partitioned systems and fixpoint computation

How to compute an abstract invariant for a partitioned system 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 under consideration
- **Example** of a simple system of abstract equations:

$$\begin{cases} \mathcal{E}_0 &= \mathcal{I} \cup F_0(\mathcal{E}_3) \\ \mathcal{E}_1 &= F_1(\mathcal{E}_0) \\ \mathcal{E}_2 &= F_2(\mathcal{E}_0) \\ \mathcal{E}_3 &= F_3(\mathcal{E}_1, \mathcal{E}_2) \end{cases}$$



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

Partitioned systems and fixpoint computation

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

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

- 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 infinitely often

Chaotic iterations

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

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

Chaotic iterations: soundness

Soundness

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

$$\lim (X_n^\sharp)_{n\in\mathbb{N}} = \alpha_{\text{part}}([\![\mathcal{S}]\!]_{\mathcal{R}})$$

Proof: exercise

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

$$\begin{array}{lcllll}
\mathcal{E}_0^\sharp & = & (\mathbb{I}, & \bot, & \bot, & \bot) \\
\mathcal{E}_1^\sharp & = & (\mathbb{I}, & F_1^\sharp(\mathbb{I}), & \bot, & \bot) \\
\mathcal{E}_2^\sharp & = & (\mathbb{I}, & F_1^\sharp(\mathbb{I}), & F_2^\sharp(\mathbb{I}), & \bot) \\
\mathcal{E}_3^\sharp & = & (\mathbb{I}, & F_1^\sharp(\mathbb{I}), & F_2^\sharp(\mathbb{I}), & F_3^\sharp(F_1^\sharp(\mathbb{I}), F_2^\sharp(\mathbb{I}))) \\
\mathcal{E}_4^\sharp & = & (\mathbb{I} \sqcup F_0^\sharp(F_3^\sharp(F_1^\sharp(\mathbb{I}), F_2^\sharp(\mathbb{I}))), & F_1^\sharp(\mathbb{I}), & F_2^\sharp(\mathbb{I}), & F_3^\sharp(F_1^\sharp(\mathbb{I}), F_2^\sharp(\mathbb{I})))
\end{array}$$

Chaotic iterations: work-list algorithm

Work-list 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 $\alpha_{\text{num}}(\mathbb{S}_{\mathcal{I}})$
- 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**

Work-list algorithm

Pseudo code implementation, with $\delta^\sharp_{\ell,\ell'}$ denoting the transfer function from ℓ to ℓ':

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

Selection of a set of widening points for a partitioned system

- We compose an abstraction $\mathbb{D}^\sharp_{\text{num}}$, with concretization $\gamma_{\text{num}} : \mathbb{D}^\sharp_{\text{num}} \to \mathcal{P}(\mathbb{M})$, that may not satisfy ascending chain condition
- We assume $\mathbb{D}^\sharp_{\text{num}}$ 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:

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

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

$$\mathbb{L} \longrightarrow (\mathbb{V}^k_{\text{bool}} \longrightarrow \mathbb{D}^\sharp_0)$$

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

```
type abs0 = ...   (* abstract elements of D#_0 *)
type abs_state = ...   (*
    boolean trees with elements of type abs0 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

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

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

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

Transfer functions: scalar assignment

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

- **example**: $\mathtt{x} = 1 - \mathtt{x};$
- **concrete transition** δ_{ℓ_0,ℓ_1} defined by

$$\delta_{\ell_0,\ell_1}(m) = \{m[\mathtt{x} \leftarrow [\![\mathtt{e}]\!](m)]\}$$

- the values of the boolean variables are unchanged
  thus the partitions are preserved (**pointwise** transfer function):

$$\mathit{assign}_{\rightarrow}(\mathtt{x}, \mathtt{e}, X^\sharp) = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \mathit{assign}_1(\mathtt{x}, \mathtt{e}, X^\sharp(z^\sharp))$$

Soundness

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

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

Transfer functions: scalar assignment, example

- **abstract precondition**:

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

- **statement**:

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

- **abstract post-condition**:

$$\begin{array}{l} \mathit{assign}_{\rightarrow} \left(\mathtt{x}, 1-\mathtt{x}, \left\{ \begin{array}{lcl} & \mathtt{b} & \Rightarrow \quad \mathtt{x} \geq 0 \\ \wedge & \neg\mathtt{b} & \Rightarrow \quad \mathtt{x} \leq 0 \end{array} \right\} \right) \\ = \left\{ \begin{array}{lcl} & \mathtt{b} & \Rightarrow \quad \mathtt{x} \geq 8 \\ \wedge & \neg\mathtt{b} & \Rightarrow \quad \top \end{array} \right\} \end{array}$$

Transfer functions: scalar test

Condition test $\ell_0 : \mathbf{if}(\mathtt{c})\{\ell_1 : \ldots\}$ 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

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

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

$$\mathit{test}_{\rightarrow}(\mathtt{c}, X^\sharp) = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \mathit{test}_1(\mathtt{c}, X^\sharp(z^\sharp))$$

Soundness

If test_1 is sound, so is $\mathit{test}_{\rightarrow}$, in the sense that:

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

Transfer functions: scalar test, example

- **abstract pre-condition**:

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

- **statement**:

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

- **abstract post-condition**:

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

Transfer functions: boolean condition test

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

- **example**: $\textbf{if}(\neg \mathsf{b}_1) \ldots$
- then, we simply need to filter the boolean partitions **satisfying** c:

$$test_{\rightarrow}(\mathsf{c}, X^\sharp) = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \begin{cases} X^\sharp(z^\sharp) & \text{if } test_0(\mathsf{c}, X^\sharp(z^\sharp)) \neq \bot_0 \\ \bot_1 & \text{otherwise} \end{cases}$$

Soundness

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

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

Transfer functions: boolean condition test, example

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

$$\mathit{test}_{\rightarrow}\left(\neg b_1, \left\{\begin{array}{lccc} & b_0 \wedge b_1 & \Rightarrow & 15 \leq x \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & 6 \leq x \leq 8 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array}\right\}\right)$$
$$= \left\{\begin{array}{lccc} & b_0 \wedge b_1 & \Rightarrow & \bot_1 \\ \wedge & b_0 \wedge \neg b_1 & \Rightarrow & 9 \leq x \leq 14 \\ \wedge & \neg b_0 \wedge b_1 & \Rightarrow & \bot_1 \\ \wedge & \neg b_0 \wedge \neg b_1 & \Rightarrow & x \leq 5 \end{array}\right\}$$

Transfer functions: mixed assignment

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**: $\mathtt{b}_0 = \mathtt{x} \leq 7$
- let $z^\sharp \in \mathbb{D}^\sharp_0$, such that $z^\sharp(\mathtt{b}) = \mathtt{TRUE}$
  $\mathit{assign}_{\rightarrow}(\mathtt{b}, \mathtt{e}[\mathtt{x}_0, \ldots, \mathtt{x}_i], X^\sharp)(z^\sharp)$ should account for all states where b becomes true, other boolean variables remaining unchanged:

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

- same computation for cases where $z^\sharp(\mathtt{b}) = \mathtt{FALSE}$

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

Transfer functions: mixed assignment, example

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

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

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

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**
- These relations are **expensive**:
  1. Partitioning with respect to N boolean variables translates into a 2^N space cost factor
  2. After assignments, partitions need be recomputed
- **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

Assumptions: we start from a **trace semantics**
and use **an abstraction of execution history for partitioning**

- **concrete domain**: $\mathbb{D} = \mathcal{P}(\mathbb{S}^\star)$
- **left side abstraction** $\gamma_0 : \mathbb{D}_0^\sharp \to \mathbb{D}$: a **trace abstraction**
- **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:

$$\begin{array}{rccl} \gamma_1 : & \mathbb{D}_1^\sharp & \longrightarrow & \mathcal{P}(\mathbb{S}^\star) \\ & M & \longmapsto & \{\langle s_0, \ldots, s_k, (\ell, m)\rangle \mid m \in M, \ell \in \mathbb{L}, s_0, \ldots, s_k \in \mathbb{S}\} \end{array}$$

  - a **store abstraction** applied to the traces final memory state $\gamma_2 : \mathbb{D}_2^\sharp \to \mathbb{D}_1^\sharp$

Trace partitioning

Cardinal power abstraction defined by the above, and by an abstraction of sets of traces $\gamma_0 : \mathbb{D}_0^\sharp \longrightarrow \mathcal{P}(\mathbb{S}^\star)$

Application 1: partitioning by control states

Flow sensitive abstraction

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

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

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**:

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

Domain of partitions

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

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

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

Application 2: partitioning guided by a condition

This partitioning **resolves the second example** (we do not represent $\top$ when it gives no information):

int $x \in \mathbb{Z}$;
int s;
int y;
if$(x \geq 0)\{$
$\quad \mathrm{if_t} \Rightarrow (0 \leq x) \wedge \mathrm{if_f} \Rightarrow \bot$
$\quad s = 1;$
$\quad \mathrm{if_t} \Rightarrow (0 \leq x \wedge s = 1) \wedge \mathrm{if_f} \Rightarrow \bot$
$\}$ **else** $\{$
$\quad \mathrm{if_f} \Rightarrow (x < 0) \wedge \mathrm{if_t} \Rightarrow \bot$
$\quad s = -1;$
$\quad \mathrm{if_f} \Rightarrow (x < 0 \wedge s = -1) \wedge \mathrm{if_t} \Rightarrow \bot$
$\}$

$$\begin{cases} \mathrm{if_t} \Rightarrow (0 \leq x \wedge s = 1) \\ \wedge \ \mathrm{if_f} \Rightarrow (x < 0 \wedge s = -1) \end{cases}$$

$y = x/s;$

$$\begin{cases} \mathrm{if_t} \Rightarrow (0 \leq x \wedge s = 1 \wedge 0 \leq y) \\ \wedge \ \mathrm{if_f} \Rightarrow (x < 0 \wedge s = -1 \wedge 0 < y) \end{cases}$$

Application 3: partitioning guided by a loop

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

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

Domain of partitions

For a given $k \in \mathbb{N}$, the partitions are defined by $\mathbb{D}^\sharp_0 = \{\mathrm{loop}_0, \mathrm{loop}_1, \ldots, \mathrm{loop}_k, \top\}$ and:

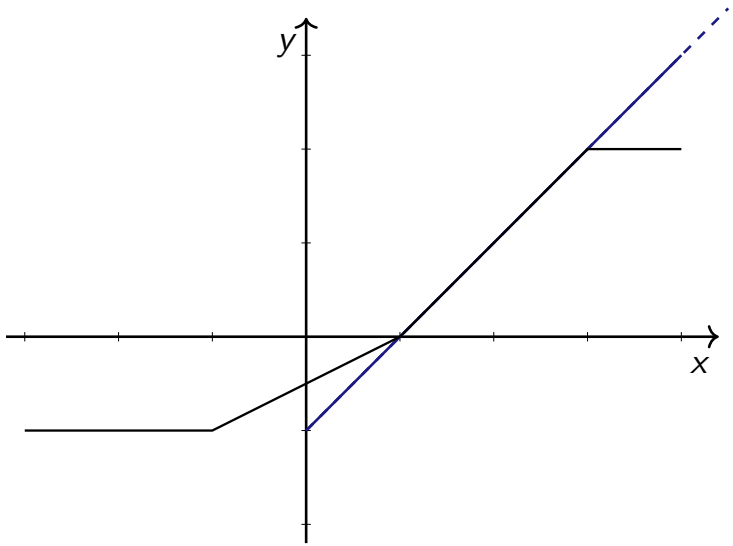
$$\begin{array}{lccl} \gamma_0: & \mathrm{loop}_i & \longmapsto & \text{traces that visit } \ell_1 \ i \text{ times} \\ & \top & \longmapsto & \mathbb{S}^\star \end{array}$$

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

Application 3: partitioning guided by a loop

An interpolation function:

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



Typical implementation:

- use tables of coefficients and loops to search for the range of $\mathtt{x}$

```
int i = 0;
while(i < 4 && x > t_x[i + 1]){
    i++;
}
```

$$\begin{cases} \text{loop}_0 & \Rightarrow & \mathtt{x} \leq -1 \\ \text{loop}_1 & \Rightarrow & -1 \leq \mathtt{x} \leq 1 \\ \text{loop}_2 & \Rightarrow & 1 \leq \mathtt{x} \leq 3 \\ \text{loop}_3 & \Rightarrow & 3 \leq \mathtt{x} \end{cases}$$

$$\mathtt{y} = \mathtt{t}_c[\mathtt{i}] \times (\mathtt{x} - \mathtt{t}_x[\mathtt{i}]) + \mathtt{t}_y[\mathtt{i}]$$

Application 4: partitioning guided by the value of a variable

We consider a program with an integer **variable** $\mathtt{x}$, and a **program point** ℓ:

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

Domain of partitions: partitioning by the value of a variable

For a given $\mathcal{E} \subseteq \mathbb{V}_{\text{int}}$ finite set of integer values, the partitions are defined by $\mathbb{D}^\sharp_0 = \{\text{val}_i \mid i \in \mathcal{E}\} \uplus \{\top\}$ and:

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

Domain of partitions: partitioning by the property of a variable

For a given abstraction $\gamma : (V^\sharp, \sqsubseteq^\sharp) \to (\mathcal{P}(\mathbb{V}_{\text{int}}), \subseteq)$, the partitions are defined by $\mathbb{D}^\sharp_0 = \{\text{var}_{v^\sharp} \mid v^\sharp \in V^\sharp\}$ and:

$$\gamma_0 : \quad \text{val}_{v^\sharp} \quad \longmapsto \quad \{\langle \ldots, (\ell, m), \ldots \rangle \mid m(\mathtt{x}) \in \text{var}_{v^\sharp}\}$$

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

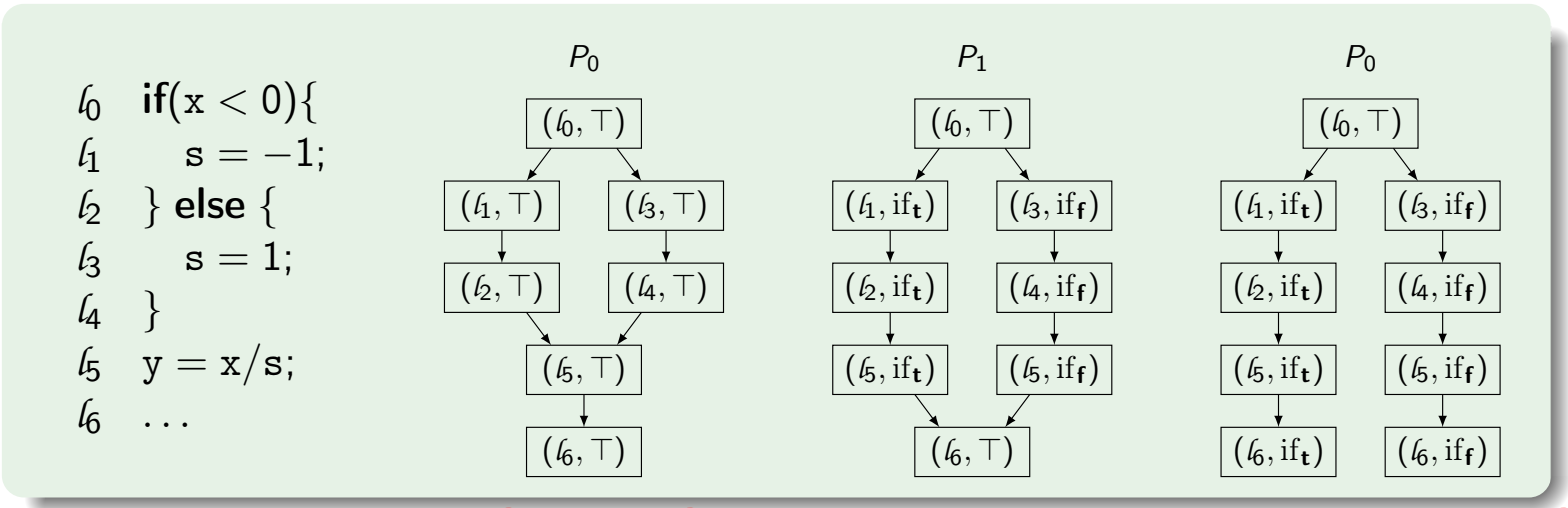
$y = 100/x$;
$(x < 0@① \Rightarrow \neg b_0 \wedge b_1 \wedge y \leq 0) \wedge (x = 0@① \Rightarrow \bot) \wedge (x > 0@① \Rightarrow b_0 \wedge \neg b_1 \wedge y \geq 0)$

}

Trace partitioning induced by a refined transition system

Let us consider the **partitions induced by a condition**:

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



Thus, 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**: $\mathbb{L}' = \mathbb{L} \times \mathbb{D}^\sharp_0$ 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

- $\forall (\ell, m) \in \mathbb{S}_\mathcal{I},\ \exists \mathrm{tok} \in \mathbb{D}^\sharp_0,\ ((\ell, \mathrm{tok}), m) \in \mathbb{S}'_\mathcal{I}$
- $\forall (\ell, m), (\ell', m') \in \mathbb{S},\ \forall \mathrm{tok} \in \mathbb{D}^\sharp_0,$
  $(\ell, m) \rightarrow (\ell', m') \Longrightarrow \exists \mathrm{tok}' \in \mathbb{D}^\sharp_0,\ ((\ell, \mathrm{tok}), m) \rightarrow ((\ell', \mathrm{tok}'), m')$

Then:

$$\forall \langle (\ell_0, m_0), \ldots, (\ell_n, m_n) \rangle \in [\![\mathcal{S}]\!]_\mathcal{R},$$
$$\exists \mathrm{tok}_0, \ldots, \mathrm{tok}_n \in \mathbb{D}^\sharp_0,\ \langle ((\ell_0, \mathrm{tok}_0), m_0), \ldots, ((\ell_n, \mathrm{tok}_n), m_n) \rangle \in [\![\mathcal{S}']\!]_\mathcal{R},$$

Trace partitioning induced by a refined transition system

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

Definition of a trace partitioning

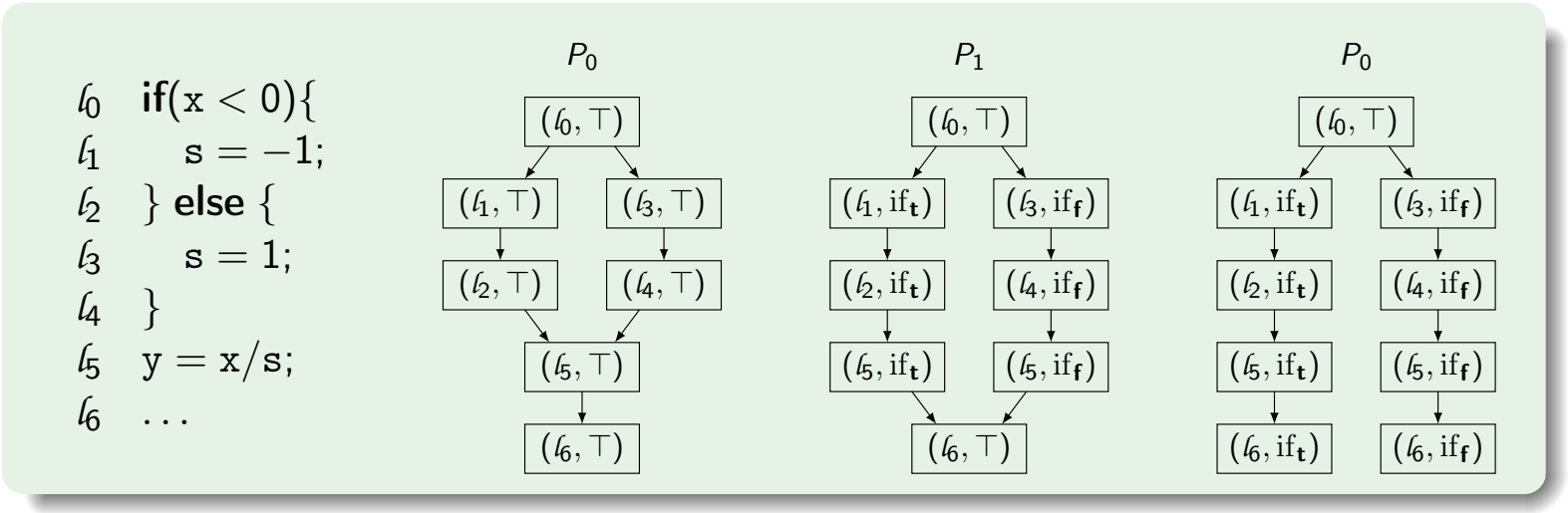
The abstraction defining partitions is defined by:

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

- not all instances of trace partitionings can be expressed that way
- ... but many interesting instances can

Trace partitioning induced by a refined transition system

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



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

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

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

$$P_2 \prec P_1$$

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

Abstract interpretation of a partitioned transition system

- let $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$, and a refining system $\mathcal{S}' = (\mathbb{S}', \rightarrow', \mathbb{S}'_{\mathcal{I}})$, with $\mathbb{S} = \mathbb{L} \times \mathbb{M}$, $\mathbb{S}' = (\mathbb{L} \times \mathbb{D}_0^\sharp) \times \mathbb{M}$
- **transfer functions of $\mathcal{S}'$**:
  $\delta_{\ell,\ell'} : (\mathbb{D}_0^\sharp \to \mathbb{D}_1^\sharp) \longrightarrow (\mathbb{D}_0^\sharp \to \mathbb{D}_1^\sharp)$ over-approximating $\rightarrow'$

Partition irrelevant transfer function

ℓ, ℓ' induces a **partition irrelevant transfer function** if and only if:

$$\forall \mathrm{tok}, \mathrm{tok}' \in \mathbb{D}_0^\sharp,\ \forall m, m' \in \mathbb{M},$$
$$((\ell, \mathrm{tok}), m) \rightarrow' ((\ell', \mathrm{tok}'), m') \Longrightarrow \mathrm{tok} = \mathrm{tok}'$$

- partition irrelevant transfer functions: **pointwise operators of $\mathbb{D}_1^\sharp$** for our examples of partitioning: this is the **most common case**
- **other transfer functions**: usually for partition **creation** or **fusion** or **simple composition** of a creation / fusion + partition irrelevant t.f.

Transfer functions: example

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

$\mathrm{if}_{\mathbf{t}} \Rightarrow (0 \leq \mathtt{x}) \wedge \mathrm{if}_{\mathbf{f}} \Rightarrow \bot$	partition creation: $\mathrm{if}_{\mathbf{t}}$
$\mathtt{s} = 1;$	
$\mathrm{if}_{\mathbf{t}} \Rightarrow (0 \leq \mathtt{x} \wedge \mathtt{s} = 1) \wedge \mathrm{if}_{\mathbf{f}} \Rightarrow \bot$	no modification of partitions

$\}\,\textbf{else}\,\{$

$\mathrm{if}_{\mathbf{f}} \Rightarrow (\mathtt{x} < 0) \wedge \mathrm{if}_{\mathbf{t}} \Rightarrow \bot$	partition creation: $\mathrm{if}_{\mathbf{f}}$
$\mathtt{s} = -1;$	
$\mathrm{if}_{\mathbf{f}} \Rightarrow (\mathtt{x} < 0 \wedge \mathtt{s} = -1) \wedge \mathrm{if}_{\mathbf{t}} \Rightarrow \bot$	no modification of partitions

$\}$

$\begin{cases} \mathrm{if}_{\mathbf{t}} \Rightarrow (0 \leq \mathtt{x} \wedge \mathtt{s} = 1) \\ \wedge\ \mathrm{if}_{\mathbf{f}} \Rightarrow (\mathtt{x} < 0 \wedge \mathtt{s} = -1) \end{cases}$	no modification of partitions
$\mathtt{y} = \mathtt{x}/\mathtt{s};$	
$\begin{cases} \mathrm{if}_{\mathbf{t}} \Rightarrow (0 \leq \mathtt{x} \wedge \mathtt{s} = 1 \wedge 0 \leq \mathtt{y}) \\ \wedge\ \mathrm{if}_{\mathbf{f}} \Rightarrow (\mathtt{x} < 0 \wedge \mathtt{s} = -1 \wedge 0 < \mathtt{y}) \end{cases}$	no modification of partitions
$\ldots$	
$\_ \Rightarrow \mathtt{s} \in [-1, 1] \wedge 0 \leq \mathtt{y}$	fusion of partitions

In general, partitions are rarely modified (only *some* branching points)

Transfer functions: partition creation

Analysis of an if statement, with partitioning

$$
\begin{array}{ll}
\ell_0: & \textbf{if}(\mathrm{c})\{ \\
\ell_1: & \quad \ldots \\
\ell_2: & \}\textbf{else}\{ \\
\ell_3: & \quad \ldots \\
\ell_4: & \} \\
\ell_5: & \ldots
\end{array}
\qquad
\begin{array}{rcl}
\delta^{\sharp}_{\ell_0,\ell_1}(X^{\sharp}) & = & [\mathrm{if}_{\mathbf{t}} \mapsto \mathit{test}(\mathrm{c}, \sqcup X^{\sharp}(\ell_0)(\mathrm{t})), \top \mapsto \bot] \\
\delta^{\sharp}_{\ell_0,\ell_3}(X^{\sharp}) & = & [\mathrm{if}_{\mathbf{t}} \mapsto \mathit{test}(\neg\mathrm{c}, \sqcup_{\mathrm{t}} X^{\sharp}(\ell_0)(\mathrm{t})), \top \mapsto \bot] \\
\delta^{\sharp}_{\ell_2,\ell_5}(X^{\sharp}) & = & X^{\sharp} \\
\delta^{\sharp}_{\ell_4,\ell_5}(X^{\sharp}) & = & X^{\sharp}
\end{array}
$$

- in the body of the condition: either $\mathrm{if}_{\mathbf{t}}$ or $\mathrm{if}_{\mathbf{f}}$
- effect at point ℓ_5: **both $\mathrm{if}_{\mathbf{t}}$ and $\mathrm{if}_{\mathbf{f}}$ exist**

Transfer functions: partition fusion

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

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

- 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** is **too expensive** in practice
- The **cardinal power abstraction** expresses collections of implications between abstract facts in **two abstract domains**
- **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

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`