Partitioning abstractions

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

Xavier Rival

INRIA, ENS, CNRS

Dec, 11th. 2019

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 techniques** to **express disjunctive properties** using **abstract domain combination methods** (construction of abstract domains from other abstract domains):
  - **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,
- and abstract operations (post-conditions, widening...)

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: I)
    ```
  - abstract domain combinator: **functor**

    ```
    module C = functor (D: I0) -> (struct ... end: I1)
    ```

Example: product abstraction

Set notations:

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

Assumptions:

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

Product combinator (implemented as a functor)

Given abstract domains $(\mathbb{D}_0^\sharp, \gamma_0, \perp_0)$ and $(\mathbb{D}_1^\sharp, \gamma_1, \perp_1)$, the **product abstraction** is $(\mathbb{D}_\times^\sharp, \gamma_\times, \perp_\times)$ where:

- $\mathbb{D}_\times^\sharp = \mathbb{D}_0^\sharp \times \mathbb{D}_1^\sharp$
- $\gamma_\times(x_0^\sharp, x_1^\sharp) = \gamma_0(x_0^\sharp) \cap \gamma_1(x_1^\sharp)$
- $\perp_\times = (\perp_0, \perp_1)$

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 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**:

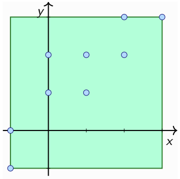
- let $\mathbb{D}_0^\sharp$ be the interval abstraction, $\mathbb{D}_1^\sharp$ be the congruences abstraction
- $\gamma_\times(\{\mathtt{x} \in [3, 4]\}, \{\mathtt{x} \equiv 0 \mod 5\}) = \emptyset$

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

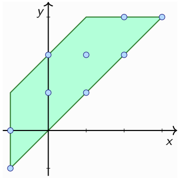
Outline

Convex abstractions

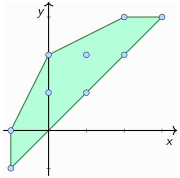
Many numerical abstractions describe **convex sets of points**



interval domain

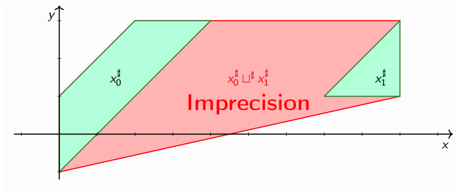


octagon domain



polyedra domain

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



Such imprecisions may make analyses fail

Similar issues also arise in non-numerical static analyses

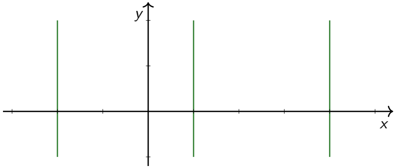
Non convex abstractions

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

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
{[\leq 0]} \arrow[d, dash] \arrow[dr, dash] & {[\neq 0]} \arrow[dl, dash] \arrow[dr, dash] & {[\geq 0]} \arrow[dl, dash] \arrow[d, dash] \\
{[-]} \arrow[dr, dash] & {[0]} \arrow[d, dash] & {[+]} \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

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:

$$\left\{ \begin{array}{c} b_0 \in \{\mathrm{FALSE}, \mathrm{TRUE}\} \wedge b_1 \in \{\mathrm{FALSE}, \mathrm{TRUE}\} \\ x : \top \end{array} \right.$$

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

Example 1: program annotated with local invariants

```
bool b_0, b_1;
int x, y;        (uninitialized)
b_0 = x ≥ 0;
            $(b_0 \wedge x \geq 0) \vee (\neg b_0 \wedge x < 0)$
b_1 = x ≤ 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;
            $(b_0 \wedge \neg b_1 \wedge x > 0) \vee (\neg b_0 \wedge b_1 \wedge x < 0)$
}
```

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

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

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 least 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 **disjunctive**, 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

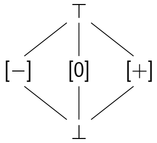
Building a disjunctive completion domain:

1. start with $\mathbb{D}^\sharp_{\mathbf{disj}} = \mathbb{D}^\sharp$
2. 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
$$\gamma_{\mathbf{disj}}([\sqcup\mathcal{E}]) = \bigsqcup_{y^\sharp \in \mathcal{E}} \gamma(y^\sharp)$$

Theorem: this process constructs a disjunctive abstraction

Example 1: completion of signs

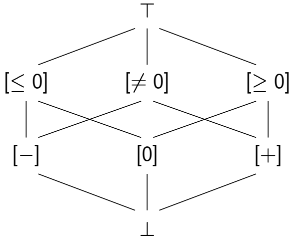
We consider **concrete lattice** $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
and $(\mathbb{D}^\sharp, \sqsubseteq^\sharp)$ defined by:



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

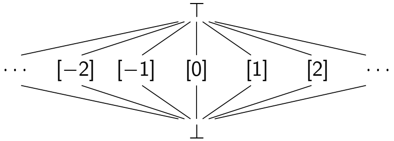
Then, the disjunctive completion is defined by adding elements corresponding to:

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



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 coincides with **the power-set**:

- $\mathbb{D}^\sharp_{\mathbf{disj}} \equiv \mathcal{P}(\mathbb{Z})$
- **this abstraction loses no information:** $\gamma_{\mathbf{disj}}$ is the **identity function !**
- obviously, this lattice contains **infinite sets which are not representable**

Middle ground solution: k-bounded disjunctive completion

- only add disjunctions of **at most k elements**
- *e.g.*, if $k = 2$, pairs are represented precisely, other sets abstracted to $\top$

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** :

- $\mathbb{D}^\sharp_{\mathbf{disj}}$ 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 $(\mathbb{D}^\sharp_{\mathbf{disj}})^n$

- 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:

```
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);
```

Static analysis

To carry out the analysis of a basic imperative language, we will define:

- **Operations for the computation of post-conditions:** sound over-approximation for basic program steps
  - concrete $post : \mathcal{P}(\mathbb{S}) \rightarrow \mathcal{P}(\mathbb{S})$ (where $\mathbb{S}$ is the set of states);
  - the **abstract** $post^\sharp : \mathbb{D}^\sharp \rightarrow \mathbb{D}^\sharp$ should be such that
  $$post \circ \gamma \sqsubseteq \gamma \circ post^\sharp$$
  - case where $post$ is an assignment: $post^\sharp = assign$ inputs a variable, an expression, an abstract pre-condition, outputs an abstract post-condition
  - case where $post$ is a condition test: $post^\sharp = test$ inputs a boolean expression, an abstract pre-condition, outputs an abstract post-condition
- An operator $join$ for **over-approximation of concrete unions**
- **A widening operator** ∇ for the analysis of loops
- **A conservative inclusion checking operator**

Static analysis with disjunctive completion

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

- we assume a monotone concrete post-condition operation $post : \mathbb{D} \to \mathbb{D}$, and an abstract $post^\sharp : \mathbb{D}^\sharp \to \mathbb{D}^\sharp$ such that $post \circ \gamma \sqsubseteq \gamma \circ post^\sharp$
- convention: if $\gamma(y^\sharp) = \bigsqcup\{\gamma(z^\sharp) \mid z^\sharp \in \mathcal{E}\}$, we note $y^\sharp = [\sqcup\mathcal{E}]$
- then, we can simply use, **for the disjunctive completion domain:**

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

  (note it may be an element of the initial domain)
- the proof is left as **exercise**
- this works for assignment, condition tests...

Abstract join:

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

Inclusion check: **exercise** !

Widening: **no general definition/solution to the disjunct explosion problem**

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 disjuncts to merge

Outline

Principle

Observation

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

Examples: each disjunct is **characterized** by

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

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

1. introduce a new abstraction to **describe labels**
   e.g., the sign of a variable, the value of a boolean, or another trace property...
2. apply the store abstraction (or another abstraction) to the set of states associated to each label

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:

- $\mathbb{D}_{\mathbf{cp}}^\sharp = \mathbb{D}_0^\sharp \xrightarrow{\mathcal{M}} \mathbb{D}_1^\sharp$ 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. $\mathbb{D}_0^\sharp$: **set of partitions** (the "labels"), represents $p_0, \ldots, p_n$
2. $\mathbb{D}_1^\sharp$: **abstraction of sets of states**, *e.g.*, a numerical abstraction, represents $p'_0, \ldots, p'_n$

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

- $\mathbb{D}_0^\sharp$: sign of $\mathtt{s}$
- $\mathbb{D}_1^\sharp$: other constraints
- we get: $\mathtt{s} > 0 \Longrightarrow (\mathtt{x} \geq 0 \wedge \mathtt{s} = 1 \wedge \mathtt{y} \geq 0) \wedge \mathtt{s} \leq 0 \Longrightarrow (\ldots)$

Another example, with a single variable

Assumptions:

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

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
{[-]} \arrow[dr, dash] & {[0]} \arrow[d, dash] & {[+]} \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

Example abstract values:

- $[0, 8]$ is expressed by: $\left\{ \begin{array}{rcl} \bot & \longmapsto & \bot_1 \\ {[-]} & \longmapsto & \bot_1 \\ {[0]} & \longmapsto & [0, 0] \\ {[+]} & \longmapsto & [1, 8] \\ \top & \longmapsto & [0, 8] \end{array} \right.$

- $[-10, -3] \uplus [7, 10]$ is expressed by: $\left\{ \begin{array}{rcl} \bot & \longmapsto & \bot_1 \\ {[-]} & \longmapsto & [-10, -3] \\ {[0]} & \longmapsto & \bot_1 \\ {[+]} & \longmapsto & [7, 10] \\ \top & \longmapsto & [-10, 10] \end{array} \right.$

Cardinal power: why monotone functions ?

We have seen the reduced cardinal power intuitively denotes a **conjunction of implications**, thus, assuming that $\mathbb{D}_0^\sharp$ has two comparable elements p_0, p_1 and:

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

Then:

- p_0, p_1 are comparable, so let us fix $p_0 \sqsubseteq_0^\sharp p_1$
- logically, this means $p_0 \Longrightarrow p_1$
- thus the abstract element represents states where $p_0 \Longrightarrow p_1 \Longrightarrow p'_1$
- as a conclusion, **if p'_0 is not as strong as p'_1, it is possible to reinforce it!**
- new abstract state:

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

This is a **reduction operation**.

Non monotone functions can be reduced into monotone functions

Example reduction (1): relation between the two domains

- 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**

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
{[-]} \arrow[dr, dash] & {[0]} \arrow[d, dash] & {[+]} \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

We let:

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

Then,

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

Note: monotone functions may also benefit from reduction

Example reduction (2): tightening relations

- 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**

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
{[-]} \arrow[dr, dash] & {[0]} \arrow[d, dash] & {[+]} \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

We let: $X^\sharp = \left\{ \begin{array}{lcl} \bot & \longmapsto & \bot_1 \\ {[-]} & \longmapsto & [-5,-1] \\ {[0]} & \longmapsto & [0,0] \\ {[+]} & \longmapsto & [1,5] \\ \top & \longmapsto & [-10,10] \end{array} \right.$ $\quad Y^\sharp = \left\{ \begin{array}{lcl} \bot & \longmapsto & \bot_1 \\ {[-]} & \longmapsto & [-5,-1] \\ {[0]} & \longmapsto & [0,0] \\ {[+]} & \longmapsto & [1,5] \\ \top & \longmapsto & [-5,5] \end{array} \right.$

- 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**

Hence, **reduced cardinal power** = **cardinal power** + **reduction**

Strengthening using both sides of $\Rightarrow$

Tightening of $y^\sharp_0 \mapsto y^\sharp_1$ when:

- $\exists z^\sharp_1 \neq y^\sharp_1,\ \gamma(y^\sharp_1) \cap \gamma(y^\sharp_0) \subseteq \gamma(z^\sharp_1)$

- in the example, $z^\sharp_1 = \perp_1$...

Strengthening of one relation using other relations

Tightening of relation $(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\}) \mapsto x^\sharp_1$ 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`
  $\Rightarrow$ usually 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...

Even 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**

A compact representation should be just sufficient to "represent" all elements of $\mathbb{D}_0^\sharp$:

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

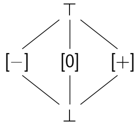
$$\forall x^\sharp \in \mathbb{D}_0^\sharp,\ \exists \mathcal{E} \subseteq \mathcal{C},\ \gamma_0(x^\sharp) = \cup\{\gamma_0(y^\sharp) \mid y^\sharp \in \mathcal{E}\}$$

In particular:

- if possible, $\mathcal{C}$ should be **minimal**
- in any case, $\perp_0 \notin \mathcal{C}$
- also, when $\top_0$ can be generated by a union of a set of elements, it can be removed

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

- $\perp$ does not need be considered (obvious right hand side: $\perp_1$)
- $\gamma_0([<0]) \cup \gamma_0([=0]) \cup \gamma([>0]) = \gamma(\top)$ thus $\top$ does not need be considered

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

- $[0, 8]$ is expressed by: $\left\{ \begin{array}{lcl} [-] & \longmapsto & \perp_1 \\ [0] & \longmapsto & [0, 0] \\ [+] & \longmapsto & [1, 8] \end{array} \right.$
- $[-10, -3] \uplus [7, 10]$ is expressed by: $\left\{ \begin{array}{lcl} [-] & \longmapsto & [-10, -3] \\ [0] & \longmapsto & \perp_1 \\ [+] & \longmapsto & [7, 10] \end{array} \right.$

Lattice operations

Infimum:

- if $\perp_1$ is the infimum of $\mathbb{D}_1^\sharp$, $\perp_{\mathbf{cp}} = \lambda(z^\sharp \in \mathbb{D}_0^\sharp) \cdot \perp_1$ is the **infimum** of $\mathbb{D}_{\mathbf{cp}}^\sharp$

Abstract post-conditions: no easy general definition, will be discussed later, based on specific instances of $\mathbb{D}_0^\sharp$

Ordering test (sound, not necessarily optimal):

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

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

- 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

- $(\mathbb{D}^\sharp_0, \sqsubseteq^\sharp_0)$, with concretization $\gamma_0 : \mathbb{D}^\sharp_0 \longrightarrow \mathbb{D}$
- $(\mathbb{D}^\sharp_1, \sqsubseteq^\sharp_1)$, with concretization $\gamma_1 : \mathbb{D}^\sharp_1 \longrightarrow \mathbb{D}$
- $(\mathbb{D}^\sharp_2, \sqsubseteq^\sharp_2)$, with concretization $\gamma_2 : \mathbb{D}^\sharp_2 \longrightarrow \mathbb{D}^\sharp_1$

$$\begin{tikzcd}
 & & \mathbb{D}^\sharp_2 \arrow[d, "\gamma_2"] \\
\mathbb{D}^\sharp_0 \arrow[dr, "\gamma_0"'] & & \mathbb{D}^\sharp_1 \arrow[dl, "\gamma_1"] \\
 & \mathbb{D} = \mathcal{P}(\mathcal{E}) &
\end{tikzcd}$$

Cardinal power abstract domains $\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_1$ and $\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_2$ can be bound by an **abstraction relation** defined by concretization function γ:

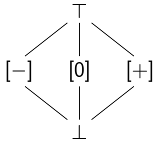
$$\begin{array}{rccl}
\gamma : & (\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_2) & \longrightarrow & (\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_1) \\
 & X^\sharp & \longmapsto & \lambda(z^\sharp \in \mathbb{D}^\sharp_0) \cdot \gamma_2(X^\sharp(z^\sharp))
\end{array}$$

Applications:

- start with $\mathbb{D}^\sharp_1, \gamma_1$ defined as the **identity abstraction**
- **compose an abstraction** for right hand side of relations
- **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 $\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_1$, $\left\{ \begin{array}{ccl} [-] & \longmapsto & \{-10, -9, -8, -7, -6, -5, -4, -3\} \\ [0] & \longmapsto & \emptyset \\ [+] & \longmapsto & \{7, 8, 9, 10\} \end{array} \right.$
  (note that, at this stage, the right hand sides are simply sets of values)
- in $\mathbb{D}^\sharp_0 \rightrightarrows \mathbb{D}^\sharp_2$, $\left\{ \begin{array}{ccl} [-] & \longmapsto & [-10, -3] \\ [0] & \longmapsto & \bot_1 \\ [+] & \longmapsto & [7, 10] \end{array} \right.$

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 the domain of sets of states $(\mathcal{P}(\mathbb{S}), \subseteq)$:

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

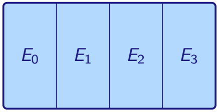
Typical instances:

- either $\mathbb{D}^\sharp_1 = \mathcal{P}(\mathbb{S}) = \mathbb{D}$
- 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. $\mathbb{S} = \bigcup \mathcal{U}$



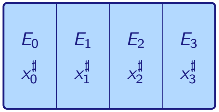
We can apply the **cardinal power construction**:

State partitioning abstraction

We let $\mathbb{D}^\sharp_0 = \mathcal{U} \cup \{\bot, \top\}$ and $\gamma_0 : E \longmapsto E$. Thus, $\mathbb{D}^\sharp_{\mathbf{cp}} = \mathcal{U} \to \mathbb{D}^\sharp_1$ and:

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

- each $E \in \mathcal{U}$ is attached to a piece of information in $\mathbb{D}^\sharp_1$
- exercise: what happens if we 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**:

```
ℓ0 :  // assume x ≥ 0
ℓ1 :  if(x < 10){
ℓ2 :     y = x − 2;
ℓ3 :  }else{
ℓ4 :     y = 2 − x;
ℓ5 :  }
ℓ6 :  ...
```

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

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:

- $\mathbb{D}^\sharp_0 = \{\cdot\}$
- $\gamma_0 : \cdot \mapsto \mathbb{S}$
- $\mathbb{D}^\sharp_1 = \mathcal{P}(\mathbb{M})$
- $\gamma_1 : M \mapsto \{(\ell, m) \mid \ell \in \mathbb{L}, m \in M\}$

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

Application 1: flow insensitive abstraction

We compare with **flow sensitive abstraction:**

$$\begin{array}{ll}
\ell_0: & \texttt{// assume } \mathtt{x} \geq 0 \\
\ell_1: & \textbf{if}(\mathtt{x} < 10)\{ \\
\ell_2: & \quad \mathtt{y} = \mathtt{x} - 2; \\
\ell_3: & \}\textbf{else}\{ \\
\ell_4: & \quad \mathtt{y} = 2 - \mathtt{x}; \\
\ell_5: & \} \\
\ell_6: & \ldots
\end{array}
\qquad
\begin{array}{lll}
\ell_0 & \mapsto & \mathtt{x} : \top \wedge \mathtt{y} : \top \\
\ell_1 & \mapsto & \mathtt{x} : [0, +\infty[\wedge \mathtt{y} : \top \\
\ell_2 & \mapsto & \mathtt{x} : [0, 9] \wedge \mathtt{y} : \top \\
\ell_3 & \mapsto & \mathtt{x} : [0, 9] \wedge \mathtt{y} : [-2, 7] \\
\ell_4 & \mapsto & \mathtt{x} : [10, +\infty[\wedge \mathtt{y} : \top \\
\ell_5 & \mapsto & \mathtt{x} : [10, +\infty[\wedge \mathtt{y} :]-\infty, -8] \\
\ell_6 & \mapsto & \mathtt{x} : [0, +\infty[\wedge \mathtt{y} :]-\infty, 7]
\end{array}$$

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

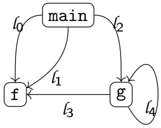
For a few specific applications flow insensitive is ok
In **most cases** (*e.g.*, numeric properties), flow sensitive is absolutely needed

Application 2: context sensitive abstraction

We consider programs **with procedures**

Example:

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


$$\begin{tikzcd}
 & \texttt{main} \arrow[dl, bend right, "\ell_0"'] \arrow[dl, "\ell_1"] \arrow[dr, bend left, "\ell_2"] & \\
\texttt{f} & & \texttt{g} \arrow[ll, "\ell_3"] \arrow[loop right, "\ell_4"]
\end{tikzcd}$$

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

Call stack (or, "call string")

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

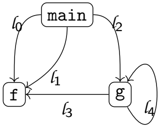
κ	$\in$	$\mathbb{K}$	call stacks
κ	$::=$	ϵ	empty call stack
	$\mid$	$(f, \ell) \cdot \kappa$	call to f from stack κ at point ℓ

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(){... ℓ0 : f(); ... ℓ1 : f(); ... ℓ2 : g() ...}
void f(){...}
void g(){if(...){ℓ3 : g()}else{ℓ4 : f()}}
```


$$\begin{tikzcd} & \text{main} \arrow[dl, bend right, "\ell_0"'] \arrow[dl, "\ell_1"] \arrow[dr, bend left, "\ell_2"] & \\ \text{f} & & \text{g} \arrow[ll, "\ell_3"] \arrow[loop right, "\ell_4"] \end{tikzcd}$$

Abstract 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**
- **infinite in presence of recursion** (*i.e.*, not practical in this case)

Application 2: context insensitive abstraction, 0-CFA

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

$$\begin{tikzcd}
 & \boxed{\mathtt{main}} \arrow[dl, bend right, "\ell_0"'] \arrow[dl, bend left, "\ell_1"] \arrow[dr, bend left, "\ell_2"] & \\
\boxed{\mathtt{f}} & & \boxed{\mathtt{g}} \arrow[ll, "\ell_3"] \arrow[loop right, "\ell_4"]
\end{tikzcd}$$

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

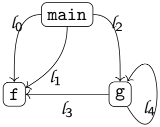
- 0-CFA merges **all** calling contexts to a same procedure, **very coarse** abstraction
- but is **usually quite efficient to compute**

Application 2: context sensitive abstraction, k-CFA

Partially context sensitive abstraction (k-CFA)

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

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


$$\begin{tikzcd} & \text{main} \arrow[dl, bend right, "\ell_0"'] \arrow[dl, "\ell_1"] \arrow[dr, bend left, "\ell_2"] & \\ \text{f} & & \text{g} \arrow[ll, "\ell_3"] \arrow[loop right, "\ell_4"] \end{tikzcd}$$

Abstract contexts in **function** `f`, in 2-**CFA:**

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

- 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 control states 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, $\mathbb{M} = \mathbb{X} \rightarrow \mathbb{V} \equiv (\mathbb{X}_{\text{bool}} \rightarrow \mathbb{V}_{\text{bool}}) \times (\mathbb{X}_{\text{int}} \rightarrow \mathbb{V}_{\text{int}}) \equiv \mathbb{V}_{\text{bool}}^{k} \times \mathbb{V}_{\text{int}}^{l}$

Boolean partitioning abstract domain

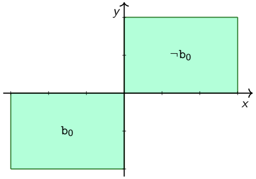
We apply the cardinal power abstraction, with a domain of partitions defined by a function, with:

- $A = \mathbb{B}^k$
- $\phi(m) = (m(\mathtt{b}_0), \ldots, m(\mathtt{b}_{k-1}))$
- 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 $\mathbb{X}_{\text{bool}} = \{\mathtt{b}_0, \mathtt{b}_1\}, \mathbb{X}_{\text{int}} = \{\mathtt{x}, \mathtt{y}\}$, we can express:

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



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

```
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 ∧ y ≥ 0) ∧ (¬b₀ ∧ b₁ ⟹ x < 0 ∧ y ≤ 0)
}
```

Application 3: partitioning by the sign of a variable

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

We assume:

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

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

Sign partitioning abstract domain

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

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

Application 3: example

- Sign abstraction fixing partitions shown in blue
- States 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){
            $(x < 0 \Rightarrow \bot) \wedge (x = 0 \Rightarrow \top) \wedge (x > 0 \Rightarrow \top)$
        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 ≥ 0);
```

Outline

Computation of abstract semantics and partitioning

We present abstract operations in the context of an analysis that **combines two forms of partitioning**:

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

Intuitively, the abstract values are of the form:

$$f^\sharp : (\mathbb{L} \times \mathbb{V}_{\mathrm{bool}}^k) \longrightarrow \mathbb{D}_1^\sharp$$

Yet, this is **not a very good representation**:

- **program transition from one control state to another are known before the analysis:**
  they correspond to the program transitions
- **program transition from one boolean configuration to another are not known before the analysis:** we need to know information about the values of the boolean variables, which the analysis is supposed to compute

A combination of two cardinal powers

Sequence of abstractions:

1. **concrete states**: $\mathcal{P}(\mathbb{L} \times \mathbb{M}) \equiv \mathcal{P}(\mathbb{L} \times (\mathbb{V}_{\text{bool}}^{k} \times \mathbb{V}_{\text{int}}^{k}))$
2. **partitioning of states by the control state:**
$$\mathbb{L} \longrightarrow \mathcal{P}(\mathbb{M}) \equiv \mathbb{L} \longrightarrow \mathcal{P}((\mathbb{V}_{\text{bool}}^{k} \times \mathbb{V}_{\text{int}}^{l}))$$
3. **partitioning by the boolean configuration:**
$$\mathbb{L} \longrightarrow (\mathbb{V}_{\text{bool}}^{k} \longrightarrow \mathcal{P}(\mathbb{V}_{\text{int}}^{l}))$$
4. **numerical abstraction of numerical stores:**
$$\mathbb{L} \longrightarrow (\mathbb{V}_{\text{bool}}^{k} \longrightarrow \mathbb{D}_{1}^{\sharp})$$

Computer representation:

```
type abs1 = ...   (* abstract elements of D♯1 *)
type abs_state = ...   (*
    boolean trees with elements of type abs1 at the leaves *)
type abs_cp = (labels, abs_state) Map.t
```

Abstract operations

Abstract post-conditions

- concrete $post : \mathcal{P}(\mathbb{S}) \rightarrow \mathcal{P}(\mathbb{S})$ (where $\mathbb{S}$ is the set of states);
- the **abstract** $post^\sharp : \mathbb{D}^\sharp \rightarrow \mathbb{D}^\sharp$ should be such that

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

In the next part, we seek for **abstract post-conditions** for the following operations, in the cardinal power domain, assuming similar functions are defined in the underlying domain (numeric abstract domain, cf previous course):

- **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.*, **if**$(\mathtt{x} \geq 8) \ldots$
- **boolean test**, *e.g.*, **if**$(\neg \mathtt{b}_1) \ldots$

Other lattice operations (**inclusion check**, **join**, **widening**) are left as exercise

Transfer functions: assignment to scalar (1/2)

Computation of an abstract post-condition

$$\mathtt{x}_k = \mathtt{e};$$

Example:

- **statement** $\mathtt{x} = 1 - \mathtt{x};$
- **abstract pre-condition**:

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

Intuition:

- the values of the boolean variables do not change
- the values of the numeric values can be updated separately for each partition

Transfer functions: assignment to scalar (2/2)

Definition of the abstract post-condition

$$assign_{\mathbf{cp}}(\mathrm{x}, \mathrm{e}, X^\sharp) = \lambda(z^\sharp \in \mathbb{V}_{\mathrm{bool}}^k) \cdot assign_1(\mathrm{x}, \mathrm{e}, X^\sharp(z^\sharp))$$

This post-condition is sound:

Soundness

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

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

- proof by case analysis over the value of the boolean variables

Example:

$$assign_{\mathbf{cp}}\left(\mathrm{x}, 1 - \mathrm{x}, \left\{ \begin{array}{rccc} & \mathrm{b} & \Rightarrow & \mathrm{x} \geq 0 \\ \wedge & \neg\mathrm{b} & \Rightarrow & \mathrm{x} \leq 0 \end{array} \right\}\right) = \left\{ \begin{array}{rccc} & \mathrm{b} & \Rightarrow & \mathrm{x} \leq 1 \\ \wedge & \neg\mathrm{b} & \Rightarrow & \mathrm{x} \geq 1 \end{array} \right\}$$

Transfer functions: scalar test (1/2)

Computation of an abstract post-condition

$$\textbf{if}(\mathrm{e})$$

where e only refers to numeric variables
(analysis of a condition test, of a loop test, of an assertion)

Example:

- **statement**: $\textbf{if}(\mathrm{x} \geq 8)\{\ldots$
- **abstract pre-condition**:

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

Intuition:

- the values of the variables do not change, no relations between boolean and numeric variables can be inferred
- new conditions on the numeric variables can be inferred, separately for each partition (possibly leading to empty abstract states)

Transfer functions: scalar test (2/2)

Definition of the abstract post-condition

$$\mathit{test}_{\mathbf{cp}}(\mathsf{c}, X^\sharp) = \lambda(z^\sharp \in \mathbb{V}_{\mathrm{bool}}^k) \cdot \mathit{test}_1(\mathsf{c}, X^\sharp(z^\sharp))$$

This post-condition is sound:

Soundness

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

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

- proof by case analysis over the value of the boolean variables

Example:

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

Transfer functions: boolean condition test (1/3)

Computation of an abstract post-condition

$$\mathbf{if}(\mathrm{e})$$

where e only refers to boolean variables
(analysis of a condition test, of a loop test, of an assertion)

Example:

- **statement**: $\mathbf{if}(\neg \mathrm{b}_1) \ldots$
- **abstract pre-condition**: $\left\{ \begin{array}{rccc} & \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 15 \leq \mathrm{x} \\ \wedge & \mathrm{b}_0 \wedge \neg \mathrm{b}_1 & \Rightarrow & 9 \leq \mathrm{x} \leq 14 \\ \wedge & \neg \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 6 \leq \mathrm{x} \leq 8 \\ \wedge & \neg \mathrm{b}_0 \wedge \neg \mathrm{b}_1 & \Rightarrow & \mathrm{x} \leq 5 \end{array} \right\}$

Intuition:

- the values of the variables do not change, no new relations between boolean and numeric variables can be inferred
- certain boolean configurations get discarded or refined

Transfer functions: boolean condition test (2/3)

Definition of the abstract post-condition

$$\mathit{test}_{\mathbf{cp}}(\mathrm{c}, X^\sharp) = \lambda(z^\sharp \in \mathbb{V}_{\mathrm{bool}}^k) \cdot \begin{cases} X^\sharp(z^\sharp) & \text{if } \mathit{test}_0(\mathrm{c}, X^\sharp(z^\sharp)) \neq \bot_0 \\ \bot_1 & \text{otherwise} \end{cases}$$

This post-condition is sound:

Soundness

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

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

Proof:

- case analysis over the boolean configurations
- in each situation, two cases depending on whether or not the condition test evaluates to `TRUE` or to `FALSE`

Transfer functions: boolean condition test (2/3)

Example abstract post-condition:

$$
\begin{array}{l}
\mathit{test}_{\mathbf{cp}}\left(\neg \mathrm{b}_1, \left\{ \begin{array}{cccc}
 & \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 15 \leq \mathrm{x} \\
\wedge & \mathrm{b}_0 \wedge \neg \mathrm{b}_1 & \Rightarrow & 9 \leq \mathrm{x} \leq 14 \\
\wedge & \neg \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 6 \leq \mathrm{x} \leq 8 \\
\wedge & \neg \mathrm{b}_0 \wedge \neg \mathrm{b}_1 & \Rightarrow & \mathrm{x} \leq 5
\end{array} \right\} \right) \\
\qquad = \left\{ \begin{array}{cccc}
 & \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & \perp_1 \\
\wedge & \mathrm{b}_0 \wedge \neg \mathrm{b}_1 & \Rightarrow & 9 \leq \mathrm{x} \leq 14 \\
\wedge & \neg \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & \perp_1 \\
\wedge & \neg \mathrm{b}_0 \wedge \neg \mathrm{b}_1 & \Rightarrow & \mathrm{x} \leq 5
\end{array} \right\}
\end{array}
$$

Transfer functions: assignment to boolean (1/3)

Computation of an abstract post-condition

$$\mathrm{b}_j = \mathrm{e};$$

where e only refers to numeric variables

Example:

- **statement**: $\mathtt{b}_0 = \mathtt{x} \leq 7$
- **abstract pre-condition**: $\left\{ \begin{array}{cccc} & \mathtt{b}_0 \wedge \mathtt{b}_1 & \Rightarrow & 15 \leq \mathtt{x} \\ \wedge & \mathtt{b}_0 \wedge \neg\mathtt{b}_1 & \Rightarrow & 9 \leq \mathtt{x} \leq 14 \\ \wedge & \neg\mathtt{b}_0 \wedge \mathtt{b}_1 & \Rightarrow & 6 \leq \mathtt{x} \leq 8 \\ \wedge & \neg\mathtt{b}_0 \wedge \neg\mathtt{b}_1 & \Rightarrow & \mathtt{x} \leq 5 \end{array} \right\}$

Intuition:

- the value of the boolean variable in the left hand side changes, thus partitions need to be recomputed
- new relations between boolean variables and numeric variables emerge (old relations get discarded)

Transfer functions: assignment to boolean (2/3)

Definition of the abstract post-condition

$$\begin{array}{rcl} assign_{\mathbf{cp}}(\mathrm{b}, \mathrm{e}, X^\sharp)(z^\sharp[\mathrm{b} \leftarrow \mathtt{TRUE}]) & = & \left\{ \begin{array}{ll} & test_1(\mathrm{e}, X^\sharp(z^\sharp[\mathrm{b} \leftarrow \mathtt{TRUE}])) \\ \sqcup_1 & test_1(\mathrm{e}, X^\sharp(z^\sharp[\mathrm{b} \leftarrow \mathtt{FALSE}])) \end{array} \right. \\ assign_{\mathbf{cp}}(\mathrm{b}, \mathrm{e}, X^\sharp)(z^\sharp[\mathrm{b} \leftarrow \mathtt{FALSE}]) & = & \left\{ \begin{array}{ll} & test_1(\neg\mathrm{e}, X^\sharp(z^\sharp[\mathrm{b} \leftarrow \mathtt{TRUE}])) \\ \sqcup_1 & test_1(\neg\mathrm{e}, X^\sharp(z^\sharp[\mathrm{b} \leftarrow \mathtt{FALSE}])) \end{array} \right. \end{array}$$

Soundness

$$\forall X^\sharp \in \mathbb{D}^\sharp_{\mathbf{cp}},\ \forall m \in \gamma_{\mathbf{cp}}(X^\sharp),\ m[\mathrm{b} \leftarrow [\![\mathrm{e}]\!](m)] \in \gamma_{\mathbf{cp}}(assign_{\mathbf{cp}}(\mathrm{b}, \mathrm{e}, X^\sharp))$$

Proof: if $z^\sharp \in \mathbb{D}^\sharp_0$ and $z^\sharp(\mathrm{b}) = \mathtt{TRUE}$, then, $assign_{\mathbf{cp}}(\mathrm{b}, \mathrm{e}[\mathrm{x}_0, \ldots, \mathrm{x}_i], X^\sharp)(z^\sharp)$ should account for all states where b becomes true, whatever the previous value, other boolean variables remaining unchanged; the case where $z^\sharp(\mathrm{b}) = \mathtt{FALSE}$ is symmetric.

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

Transfer functions: assignment to boolean (3/3)

Example abstract post-condition:

$$
\mathit{assign}_{\mathbf{cp}}\left(\mathrm{b}_0, \mathrm{x} \leq 7, \left\{ \begin{array}{ccccc} & \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 15 \leq \mathrm{x} \\ \wedge & \mathrm{b}_0 \wedge \neg\mathrm{b}_1 & \Rightarrow & 9 \leq \mathrm{x} \leq 14 \\ \wedge & \neg\mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 6 \leq \mathrm{x} \leq 8 \\ \wedge & \neg\mathrm{b}_0 \wedge \neg\mathrm{b}_1 & \Rightarrow & \mathrm{x} \leq 5 \end{array} \right\} \right)
$$

$$
= \left\{ \begin{array}{cccc} & \mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 6 \leq \mathrm{x} \leq 7 \\ \wedge & \mathrm{b}_0 \wedge \neg\mathrm{b}_1 & \Rightarrow & \mathrm{x} \leq 5 \\ \wedge & \neg\mathrm{b}_0 \wedge \mathrm{b}_1 & \Rightarrow & 8 \leq \mathrm{x} \\ \wedge & \neg\mathrm{b}_0 \wedge \neg\mathrm{b}_1 & \Rightarrow & 9 \leq \mathrm{x} \leq 14 \end{array} \right\}
$$

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)

In the last part of this course, we present another form of partitioning that can sometimes alleviate these issues

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:

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

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

Any state partitioning abstraction is also a trace partitioning abstraction:

- **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**:

```
ℓ0 :  if(c){
ℓ1 :       ...
ℓ2 :  }else{
ℓ3 :       ...
ℓ4 :  }
ℓ5 :  ...
```

Domain of partitions

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

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

Application:

discriminate the executions depending on the branch they visited

Application 2: partitioning guided by a condition

This partitioning **resolves the second example**:

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

Application 3: partitioning guided by a loop

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

```
ℓ0 :  while(c){
ℓ1 :        ...
ℓ2 :  }
ℓ3 :  ...
```

Domain of partitions

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

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

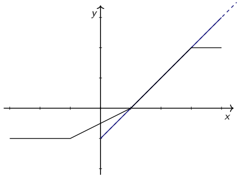
Application:

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
- here we assume the entrance is positive:

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

$$\begin{cases} \tau_{\text{loop:0}} \Rightarrow \bot & (\text{case } x \leq -1) \\ \tau_{\text{loop:1}} \Rightarrow 0 \leq x \leq 1 \wedge i = 1 & (\text{case } -1 \leq x \leq 1) \\ \tau_{\text{loop:2}} \Rightarrow 1 \leq x \leq 3 \wedge i = 2 & \\ \tau_{\text{loop:3}} \Rightarrow 3 \leq x \wedge i = 3 & \end{cases}$$

$$y = t_c[i] \times (x - t_x[i]) + t_y[i]$$

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:

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

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:

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

Application 4: partitioning guided by the value of a variable

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

```
        bool b₀, b₁;
        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 {
            $(x < 0@① \Rightarrow \neg b_0 \wedge b_1) \wedge (x = 0@① \Rightarrow \bot) \wedge (x > 0@① \Rightarrow b_0 \wedge \neg b_1)$
            $y = 100/x$;
            $(x < 0@① \Rightarrow \neg b_0 \wedge b_1 \wedge y \leq 0) \wedge (x = 0@① \Rightarrow \bot) \wedge (x > 0@① \Rightarrow b_0 \wedge \neg b_1 \wedge y \geq 0)$
        }
```

Outline

Trace partitioning induced by a refined transition system

We consider the **partitions for a condition, and formalize the analysis**:

- P_0: the analysis does merge them *right after the condition*, at ℓ_5 (this amounts to doing no partitioning at all)
- P_1: the analysis may merge them *at a further point* ℓ_6 (more precise, but more expensive)
- P_2: the analysis may *never* merge traces from both branches (very precise, but very expensive)

```
ℓ0   if(x < 0){
ℓ1       s = −1;
ℓ2   } else {
ℓ3       s = 1;
ℓ4   }
ℓ5   y = x/s;
ℓ6   ...
```

P_0

$$\begin{tikzcd}
& \ell_0 \arrow[dl] \arrow[dr] & \\
\ell_1 \arrow[d] & & \ell_3 \arrow[d] \\
\ell_2 \arrow[dr] & & \ell_4 \arrow[dl] \\
& \ell_5 \arrow[d] & \\
& \ell_6 &
\end{tikzcd}$$

P_1

$$\begin{tikzcd}
& (\ell_0, \top) \arrow[dl] \arrow[dr] & \\
(\ell_1, \tau_{\mathrm{if:t}}) \arrow[d] & & (\ell_3, \tau_{\mathrm{if:f}}) \arrow[d] \\
(\ell_2, \tau_{\mathrm{if:t}}) \arrow[d] & & (\ell_4, \tau_{\mathrm{if:f}}) \arrow[d] \\
(\ell_5, \tau_{\mathrm{if:t}}) \arrow[dr] & & (\ell_5, \tau_{\mathrm{if:f}}) \arrow[dl] \\
& (\ell_6, \top) &
\end{tikzcd}$$

P_2

$$\begin{tikzcd}
& (\ell_0, \top) \arrow[dl] \arrow[dr] & \\
(\ell_1, \tau_{\mathrm{if:t}}) \arrow[d] & & (\ell_3, \tau_{\mathrm{if:f}}) \arrow[d] \\
(\ell_2, \tau_{\mathrm{if:t}}) \arrow[d] & & (\ell_4, \tau_{\mathrm{if:f}}) \arrow[d] \\
(\ell_5, \tau_{\mathrm{if:t}}) \arrow[d] & & (\ell_5, \tau_{\mathrm{if:f}}) \arrow[d] \\
(\ell_6, \tau_{\mathrm{if:t}}) & & (\ell_6, \tau_{\mathrm{if:f}})
\end{tikzcd}$$

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**

Definition: partitioning transition system

We say that system $\mathcal{S}' = (\mathbb{S}', \rightarrow', \mathbb{S}'_{\mathcal{I}})$ is a **partition** of the transition system $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$ if and only if:

- (initial states) $\forall (\ell, m) \in \mathbb{S}_{\mathcal{I}},\ \exists \tau \in \mathbb{D}_0^\sharp,\ ((\ell, \tau), m) \in \mathbb{S}'_{\mathcal{I}}$
- (transitions) $\forall (\ell, m), (\ell', m') \in \mathbb{S},\ \forall \tau \in \mathbb{D}_0^\sharp$, if $((\ell, \tau), m) \in [\![\mathcal{S}]\!]_{\mathcal{R}}$ then,
  $$(\ell, m) \rightarrow (\ell', m') \Longrightarrow \exists \tau' \in \mathbb{D}_0^\sharp,\ ((\ell, \tau), m) \rightarrow ((\ell', \tau'), m')$$

In that case, we write:

$$\mathcal{S}' \prec \mathcal{S}$$

Meaning: system $\mathcal{S}'$ refines system $\mathcal{S}$ with additional execution history information

Partitionned transition system and semantics

The partitioned transition system over-approximates the behaviors of the initial system:

Partitioned system and semantic approximation

Let us assume that $\mathcal{S}' \prec \mathcal{S}$. We let $[\![\mathcal{S}]\!]_{\mathcal{T}}$ (*resp.*, $[\![\mathcal{S}']\!]_{\mathcal{T}}$) denote the trace semantics of $\mathcal{S}$ (*resp.*, $\mathcal{S}'$). Then:

$$\begin{array}{l}\forall \langle (\ell_0, m_0), \ldots, (\ell_n, m_n) \rangle \in [\![\mathcal{S}]\!]_{\mathcal{T}}, \\ \qquad \exists \tau_0, \ldots, \tau_n \in \mathbb{D}_0^\sharp,\ \langle ((\ell_0, \tau_0), m_0), \ldots, ((\ell_n, \tau_n), m_n) \rangle \in [\![\mathcal{S}']\!]_{\mathcal{T}},\end{array}$$

Proof: by induction over the length of executions (exercise).

Properties of $\mathcal{S}' \prec \mathcal{S}$

- all traces of $\mathcal{S}$ have a counterpart in $\mathcal{S}'$ (up to token addition)
- a trace in $\mathcal{S}'$ embeds more information than a trace in $\mathcal{S}$
- moreover, if we reason up to isomorphisms (*e.g.*, either $\ell \equiv (\ell, \bullet)$ or $((\ell, \tau), \tau') \equiv (\ell, (\tau, \tau'))$), $\prec$ **extends into a pre-order**

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:

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

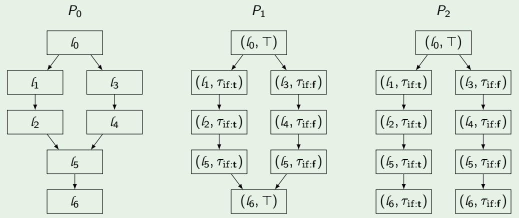
Not all instances of trace partitionings can be expressed that way but **many interesting instances can**:

- **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**:

```
ℓ0   if(x < 0){
ℓ1       s = −1;
ℓ2   } else {
ℓ3       s = 1;
ℓ4   }
ℓ5   y = x/s;
ℓ6   ...
```


P_0
ℓ_0
ℓ_1
ℓ_3
ℓ_2
ℓ_4
ℓ_5
ℓ_6
P_1
$(\ell_0, \top)$
$(\ell_1, \tau_{\mathrm{if:t}})$
$(\ell_3, \tau_{\mathrm{if:f}})$
$(\ell_2, \tau_{\mathrm{if:t}})$
$(\ell_4, \tau_{\mathrm{if:f}})$
$(\ell_5, \tau_{\mathrm{if:t}})$
$(\ell_5, \tau_{\mathrm{if:f}})$
$(\ell_6, \top)$
P_2
$(\ell_0, \top)$
$(\ell_1, \tau_{\mathrm{if:t}})$
$(\ell_3, \tau_{\mathrm{if:f}})$
$(\ell_2, \tau_{\mathrm{if:t}})$
$(\ell_4, \tau_{\mathrm{if:f}})$
$(\ell_5, \tau_{\mathrm{if:t}})$
$(\ell_5, \tau_{\mathrm{if:f}})$
$(\ell_6, \tau_{\mathrm{if:t}})$
$(\ell_6, \tau_{\mathrm{if:f}})$

- 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

$$P_2 \prec P_1 \prec P_0 \qquad \text{thus,} \qquad [\![P_0]\!]_{\mathcal{T}} \subseteq [\![P_1]\!]_{\mathcal{T}} \subseteq [\![P_2]\!]_{\mathcal{T}}$$

- 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

```
int x ∈ ℤ;
int s;
int y;
if(x ≥ 0){
        $\tau_{\text{if:t}} \Rightarrow (0 \leq x) \wedge \tau_{\text{if:f}} \Rightarrow \bot$                partition creation: $\tau_{\text{if:t}}$
    s = 1;
        $\tau_{\text{if:t}} \Rightarrow (0 \leq x \wedge s = 1) \wedge \tau_{\text{if:f}} \Rightarrow \bot$        no modification of partitions
} else {
        $\tau_{\text{if:f}} \Rightarrow (x < 0) \wedge \tau_{\text{if:t}} \Rightarrow \bot$                partition creation: $\tau_{\text{if:f}}$
    s = −1;
        $\tau_{\text{if:f}} \Rightarrow (x < 0 \wedge s = -1) \wedge \tau_{\text{if:t}} \Rightarrow \bot$       no modification of partitions
}
        $\left\{ \begin{array}{ccc} & \tau_{\text{if:t}} \Rightarrow & (0 \leq x \wedge s = 1) \\ \wedge & \tau_{\text{if:f}} \Rightarrow & (x < 0 \wedge s = -1) \end{array} \right.$        no modification of partitions
y = x/s;
        $\left\{ \begin{array}{ccc} & \tau_{\text{if:t}} \Rightarrow & (0 \leq x \wedge s = 1 \wedge 0 \leq y) \\ \wedge & \tau_{\text{if:f}} \Rightarrow & (x < 0 \wedge s = -1 \wedge 0 < y) \end{array} \right.$        no modification of partitions
...
        $\_ \Rightarrow s \in [-1, 1] \wedge 0 \leq y$                fusion of partitions
```

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

Transfer functions: partition creation

Analysis of an if statement, with partitioning

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

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

Observations:

- in the body of the condition: either $\tau_{\mathrm{if:t}}$ or $\tau_{\mathrm{if:f}}$
  i.e., **no partition modification there**
- effect at point ℓ_5: **both $\tau_{\mathrm{if:t}}$ and $\tau_{\mathrm{if:f}}$ 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 [always the case in this lecture]

- 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 choosing partitions beforehand 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

$$P_0 \vee \ldots \vee P_n$$

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

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

Two major cases:

- **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: proofs and paper reading

Proof 1:
prove the disjunctive completion algorithm (Slide 15)

Proof 2:
what happens in the case we use coverings instead of partitions (Slide 41)

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`