

Partitioning abstractions

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

Xavier Rival

INRIA, ENS, CNRS

Jan, 9th. 2023

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: IO) -> (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**:

$$\forall x_0^\sharp \in \mathbb{D}_0^\sharp, x_1^\sharp \in \mathbb{D}_1^\sharp, \gamma_x(\perp_0, x_1^\sharp) = \gamma_x(x_0^\sharp, \perp_1) = \emptyset = \gamma_x(\perp_x)$$

Coalescent product

Given abstract domains $(\mathbb{D}_0^\sharp, \gamma_0, \perp_0)$ and $(\mathbb{D}_1^\sharp, \gamma_1, \perp_1)$, the **coalescent product abstraction** is $(\mathbb{D}_x^\sharp, \gamma_x, \perp_x)$ where:

- $\mathbb{D}_x^\sharp = \{\perp_x\} \uplus \{(x_0^\sharp, x_1^\sharp) \in \mathbb{D}_0^\sharp \times \mathbb{D}_1^\sharp \mid x_0^\sharp \neq \perp_0 \wedge x_1^\sharp \neq \perp_1\}$
- $\gamma_x(\perp_x) = \emptyset, \gamma_x(x_0^\sharp, x_1^\sharp) = \gamma_0(x_0^\sharp) \cap \gamma_1(x_1^\sharp)$

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
- $\gamma_x(\{x \in [3, 4]\}, \{x \equiv 0 \pmod{5}\}) = \emptyset$

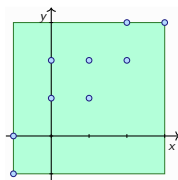
- how to define abstract domain combinators to **add disjunctions** ?

Outline

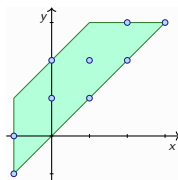
- 1 Introduction
- 2 Imprecisions in convex abstractions**
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning
- 6 Trace partitioning
- 7 Conclusion

Convex abstractions

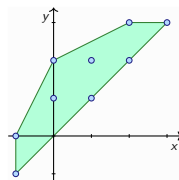
Many numerical abstractions describe **convex sets of points**



interval domain

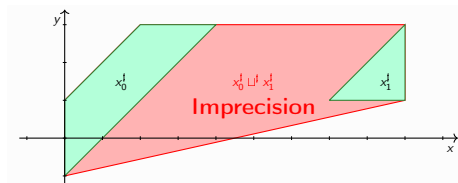


octagon domain



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

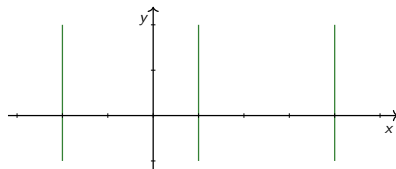
Non convex abstractions

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

Congruences:

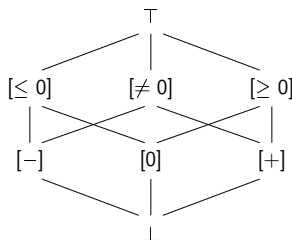
- $\mathbb{D}^\# = \mathbb{Z} \times \mathbb{N}$
- $\gamma(n, k) = \{n + k \cdot p \mid p \in \mathbb{Z}\}$
- $-1 \in \gamma(1, 2)$ and $1 \in \gamma(1, 2)$
but $0 \notin \gamma(1, 2)$

Non relational product two variables



Signs:

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



Example 1: verification problem

```

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

```

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

How to verify the division operation ?

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

$$\left\{ \begin{array}{l} b_0 \in \{\text{FALSE}, \text{TRUE}\} \wedge b_1 \in \{\text{FALSE}, \text{TRUE}\} \\ x : \mathbb{T} \end{array} \right.$$

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

Example 1: program annotated with local invariants

```

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

```

The obvious way to successfully 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

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion**
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning
- 6 Trace partitioning
- 7 Conclusion

Distributive abstract domain

Principle:

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

Definition: distributive abstract domain

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

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

Examples:

- the lattice $\{\perp, < 0, = 0, > 0, \leq 0, \neq 0, \geq 0, \top\}$ 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 $(\mathbb{D}^\#, \sqsubseteq^\#)$ with concretization function $\gamma : \mathbb{D}^\# \rightarrow \mathbb{D}$ is the **smallest abstract domain** $(\mathbb{D}_{\text{disj}}^\#, \sqsubseteq_{\text{disj}}^\#)$ with concretization function $\gamma_{\text{disj}} : \mathbb{D}_{\text{disj}}^\# \rightarrow \mathbb{D}$ such that:

- $\mathbb{D}^\# \subseteq \mathbb{D}_{\text{disj}}^\#$
- $\forall x^\# \in \mathbb{D}^\#, \gamma_{\text{disj}}(x^\#) = \gamma(x^\#)$
- $(\mathbb{D}_{\text{disj}}^\#, \sqsubseteq_{\text{disj}}^\#)$ with concretization γ_{disj} is distributive

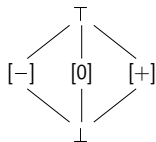
Building a disjunctive completion domain:

- 1 include in $\mathbb{D}_{\text{disj}}^\#$ all elements of $\mathbb{D}^\#$
- 2 for all set $\mathcal{E} \subseteq \mathbb{D}^\#$ such that there is no $x^\# \in \mathbb{D}^\#$, such that $\gamma(x^\#) = \bigsqcup_{y^\# \in \mathcal{E}} \gamma(y^\#)$, add $[\bigsqcup \mathcal{E}]$ to $\mathbb{D}_{\text{disj}}^\#$, and extend γ_{disj} by
$$\gamma_{\text{disj}}([\bigsqcup \mathcal{E}]) = \bigsqcup_{y^\# \in \mathcal{E}} \gamma(y^\#)$$

Theorem: this process constructs a disjunctive abstraction

Example 1: completion of signs

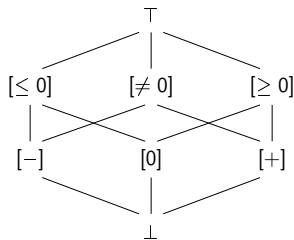
We consider **concrete lattice** $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\sqsubseteq = \subseteq$
and $(\mathbb{D}^\#, \sqsubseteq^\#)$ defined by:



$$\begin{aligned} \gamma: \perp &\mapsto \emptyset \\ [-] &\mapsto \{k \in \mathbb{Z} \mid k < 0\} \\ [= 0] &\mapsto \{k \in \mathbb{Z} \mid k = 0\} \\ [> 0] &\mapsto \{k \in \mathbb{Z} \mid k > 0\} \\ \top &\mapsto \mathbb{Z} \end{aligned}$$

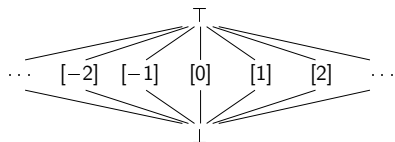
Then, the disjunctive completion is defined
by adding elements corresponding to:

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



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:



$$\begin{array}{lcl} \gamma : & \perp & \mapsto \emptyset \\ & [k] & \mapsto \{k\} \\ & \top & \mapsto \mathbb{Z} \end{array}$$

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

- $\mathbb{D}_{\text{disj}}^\sharp \equiv \mathcal{P}(\mathbb{Z})$
- **this abstraction loses no information**: γ_{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

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

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

- $\mathbb{D}_{\text{disj}}^\sharp$ 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 $(\mathbb{D}_{\text{disj}}^\sharp)^n$

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

Example 3: completion of intervals and verification

We use the disjunctive completion of $(\mathbb{D}^\#)^3$.

The invariants below (code example 2) can be expressed in the disjunctive completion:

```

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)
y = x/s;
    (x  $\geq$  0  $\wedge$  s = 1  $\wedge$  y  $\geq$  0)  $\vee$  (x < 0  $\wedge$  s = -1  $\wedge$  y > 0)
assert(y  $\geq$  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 $post : \mathbb{D} \rightarrow \mathbb{D}$, and an abstract $post^\sharp : \mathbb{D}^\sharp \rightarrow \mathbb{D}^\sharp$ such that $post \circ \gamma \sqsubseteq \gamma \circ post^\sharp$
- convention: if $\gamma(y^\sharp) = \sqcup\{\gamma(z^\sharp) \mid z^\sharp \in \mathcal{E}\}$, we note $y^\sharp = [\sqcup\mathcal{E}]$
- then, we can simply use, **for the disjunctive completion domain**:

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

(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, $\mathbb{D}_{\text{disj}}^\sharp$ may have elements that **cannot be represented**
e.g., completion of constants or intervals
- even when \mathbb{D}^\sharp is finite, $\mathbb{D}_{\text{disj}}^\sharp$ may be **huge**
in the worst case, if \mathbb{D}^\sharp has n elements, $\mathbb{D}_{\text{disj}}^\sharp$ 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

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions**
- 5 State partitioning
- 6 Trace partitioning
- 7 Conclusion

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

```

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

Cardinal power abstraction

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

$$\gamma_0 : \mathbb{D}_0^\# \longrightarrow \mathbb{D} \quad \gamma_1 : \mathbb{D}_1^\# \longrightarrow \mathbb{D}$$

Definition

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

- $\mathbb{D}_{\text{cp}}^\# = \mathbb{D}_0^\# \xrightarrow{\mathcal{M}} \mathbb{D}_1^\#$ be the set of monotone functions from $\mathbb{D}_0^\#$ into $\mathbb{D}_1^\#$
- $\sqsubseteq_{\text{cp}}^\#$ be the pointwise extension of $\sqsubseteq_1^\#$
- γ_{cp} is defined by:

$$\begin{aligned} \gamma_{\text{cp}} : \mathbb{D}_{\text{cp}}^\# &\longrightarrow \mathbb{D} \\ X^\# &\longmapsto \{y \in \mathcal{E} \mid \forall z^\# \in \mathbb{D}_0^\#, y \in \gamma_0(z^\#) \implies y \in \gamma_1(X^\#(z^\#))\} \end{aligned}$$

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

Use of cardinal power abstractions

Intuition: cardinal power expresses properties of the form

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

Two independent choices:

- 1 \mathbb{D}_0^\sharp : **set of partitions** (the “labels”), represents p_0, \dots, p_n
- 2 \mathbb{D}_1^\sharp : **abstraction of sets of states**, e.g., a numerical abstraction, represents p'_0, \dots, p'_n

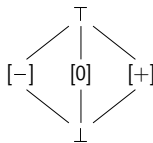
Application $(x \geq 0 \wedge s = 1 \wedge y \geq 0) \vee (x < 0 \wedge s = -1 \wedge y > 0)$

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

Another example, with a single variable

Assumptions:

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



Example abstract values:

- $[0, 8]$ is expressed by:

$$\left\{ \begin{array}{l} \perp \mapsto \perp_1 \\ [-] \mapsto \perp_1 \\ [0] \mapsto [0, 0] \\ [+] \mapsto [1, 8] \\ \top \mapsto [0, 8] \end{array} \right.$$

- $[-10, -3] \uplus [7, 10]$ is expressed by:

$$\left\{ \begin{array}{l} \perp \mapsto \perp_1 \\ [-] \mapsto [-10, -3] \\ [0] \mapsto \perp_1 \\ [+] \mapsto [7, 10] \\ \top \mapsto [-10, 10] \end{array} \right.$$

Cardinal power: why monotone functions ?

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

$$\left\{ \begin{array}{l} p_0 \implies p'_0 \\ \wedge \\ p_1 \implies p'_1 \end{array} \right.$$

Then:

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

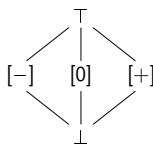
$$\left\{ \begin{array}{l} p_0 \implies p'_0 \wedge p'_1 \\ \wedge \\ p_1 \implies p'_1 \end{array} \right.$$

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



We let:

$$X^\sharp = \begin{cases} \perp & \mapsto \perp_1 \\ [-] & \mapsto [1, 8] \\ [0] & \mapsto [1, 8] \\ [+] & \mapsto \perp_1 \\ \top & \mapsto [1, 8] \end{cases} \quad Y^\sharp = \begin{cases} \perp & \mapsto \perp_1 \\ [-] & \mapsto [2, 45] \\ [0] & \mapsto [-5, -2] \\ [+] & \mapsto [-5, -2] \\ \top & \mapsto \top_1 \end{cases} \quad Z^\sharp = \begin{cases} \perp & \mapsto \perp_1 \\ [-] & \mapsto \perp_1 \\ [0] & \mapsto \perp_1 \\ [+] & \mapsto \perp_1 \\ \top & \mapsto \perp_1 \end{cases}$$

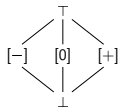
Then,

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

Note: monotone functions may also benefit from reduction

Example reduction (2): tightening relations

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



$$\text{We let: } X^\sharp = \begin{cases} \perp & \mapsto \perp_1 \\ [-] & \mapsto [-5, -1] \\ [0] & \mapsto [0, 0] \\ [+] & \mapsto [1, 5] \\ \top & \mapsto [-10, 10] \end{cases} \quad Y^\sharp = \begin{cases} \perp & \mapsto \perp_1 \\ [-] & \mapsto [-5, -1] \\ [0] & \mapsto [0, 0] \\ [+] & \mapsto [1, 5] \\ \top & \mapsto [-5, 5] \end{cases}$$

- Then, $\gamma_{\text{cp}}(X^\sharp) = \gamma_{\text{cp}}(Y^\sharp)$
- $\gamma_0([-]) \cup \gamma_0([0]) \cup \gamma_0([+]) = \gamma(\top)$
but

$$\gamma_0(X^\sharp([-])) \cup \gamma_0(X^\sharp([0])) \cup \gamma_0(X^\sharp([+])) \subset \gamma(X^\sharp(\top))$$

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

- $\exists z_1^\sharp \neq y_1^\sharp, \gamma_1(y_1^\sharp) \cap \gamma_0(y_0^\sharp) \subseteq \gamma(z_1^\sharp)$
- in the example, $z_1^\sharp = \perp_1 \dots$

Strengthening of one relation using other relations

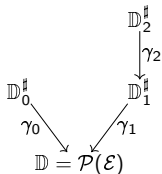
Tightening of relation $(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\}) \mapsto x_1^\sharp$ when:

- $\bigcup\{\gamma_0(z^\sharp) \mid z^\sharp \in \mathcal{E}\} = \gamma_0(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\})$
- $\exists y^\sharp, \bigcup\{\gamma_1(X^\sharp(z^\sharp)) \mid z^\sharp \in \mathcal{E}\} \subseteq \gamma_1(y^\sharp) \subset \gamma_1(X^\sharp(\sqcup\{z^\sharp \mid z^\sharp \in \mathcal{E}\}))$
- in the example, we use a set of elements that cover $\top \dots$

Composition with another abstraction

We assume three abstractions

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



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

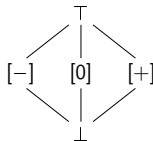
$$\begin{aligned} \gamma : (\mathbb{D}_0^\# \rightrightarrows \mathbb{D}_2^\#) &\longrightarrow (\mathbb{D}_0^\# \rightrightarrows \mathbb{D}_1^\#) \\ X^\# &\longmapsto \lambda(z^\# \in \mathbb{D}_0^\#). \gamma_2(X^\#(z^\#)) \end{aligned}$$

Applications:

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



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

- in $\mathbb{D}_0^\# \Rightarrow \mathbb{D}_1^\#$, $\begin{cases} [-] \mapsto \{-10, -9, -8, -7, -6, -5, -4, -3\} \\ [0] \mapsto \emptyset \\ [+] \mapsto \{7, 8, 9, 10\} \end{cases}$
 (note that, at this stage, the right hand sides are simply sets of values)
- in $\mathbb{D}_0^\# \Rightarrow \mathbb{D}_2^\#$, $\begin{cases} [-] \mapsto [-10, -3] \\ [0] \mapsto \perp_1 \\ [+] \mapsto [7, 10] \end{cases}$

Representation of the cardinal power

Basic ML representation:

- using **functions**, *i.e.* type $cp = d_0 \rightarrow d_1$
 \Rightarrow usually a bad choice, as it makes it hard to operate in the \mathbb{D}_0^\sharp side
- using **some kind of dictionaries** type $cp = (d_0, d_1)$ 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}_{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}_{cp}^\sharp **has elements that cannot be represented**
- the 2nd reduction shows it is **unnecessary to represent bindings for all elements of \mathbb{D}_0^\sharp**
example: this is the case of \perp_0

More compact representation of the cardinal power

Principle:

- use a **dictionary 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

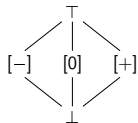
$$\forall x^\sharp \in \mathbb{D}_0^\sharp, \exists \mathcal{E} \subseteq \mathcal{C}, \gamma_0(x^\sharp) = \cup \{ \gamma_0(y^\sharp) \mid y^\sharp \in \mathcal{E} \}$$

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 $\sqsubseteq = \subseteq$
- $(\mathbb{D}_0^\sharp, \sqsubseteq_0^\sharp)$ be the **lattice of signs**
- $(\mathbb{D}_1^\sharp, \sqsubseteq_1^\sharp)$ 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:
$$\begin{cases} [-] \mapsto \perp_1 \\ [0] \mapsto [0, 0] \\ [+] \mapsto [1, 8] \end{cases}$$
- $[-10, -3] \uplus [7, 10]$ is expressed by:
$$\begin{cases} [-] \mapsto [-10, -3] \\ [0] \mapsto \perp_1 \\ [+] \mapsto [7, 10] \end{cases}$$

Lattice operations

Infimum:

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

Ordering test (sound, not necessarily optimal):

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

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

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

Join operation:

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

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

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

Abstract post-conditions

The general definition is quite involved so we first assume $\mathbb{D}_1^\sharp = \mathbb{D} = \mathcal{P}(\mathcal{E})$ and consider $f : \mathbb{D} \rightarrow \mathcal{P}(\mathbb{D})$.

Definitions:

- for $x^\sharp, y^\sharp \in \mathbb{D}_0^\sharp$, we let $f_{x^\sharp, y^\sharp} : (\mathbb{D}_0^\sharp \rightarrow \mathbb{D}_1^\sharp) \rightarrow \mathbb{D}_1^\sharp$ be defined by
 $f_{x^\sharp, y^\sharp}(X^\sharp) = \gamma_0(y^\sharp) \cap f(X^\sharp(x^\sharp) \cap \gamma_0(x^\sharp))$
- for $y^\sharp \in \mathbb{D}_0^\sharp$, we note $P(y^\sharp)$ the set of “predecessor coverings” of y^\sharp :

$$\left\{ V \subseteq \mathbb{D}_0^\sharp \mid \forall c \in f^{-1}(\gamma_0(y^\sharp)), \exists x^\sharp \in V, c \in \gamma_0(x^\sharp) \right\}$$

Then the definition below provides a sound over-approximation of f :

$$f^\sharp : X^\sharp \longmapsto \lambda(y^\sharp \in \mathbb{D}_0^\sharp). \bigcap_{V \in P(y^\sharp)} \left(\bigcup_{x^\sharp \in V} f_{x^\sharp, y^\sharp}(X^\sharp) \right)$$

- this definition is **not practical**: using a direct abstraction of this formula will result in a prohibitive runtime cost!
- in the following, we set **specific instances**.

Outline

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning**
 - Definition and examples
 - Abstract interpretation with boolean partitioning
- 6 Trace partitioning
- 7 Conclusion

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

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

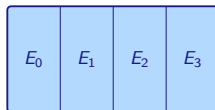
Typical instances:

- either $\mathbb{D}_1^\sharp = \mathcal{P}(\mathbb{S}) = \mathbb{D}$
- 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 $\forall E, E' \in \mathcal{U}, E \neq E' \implies E \cap E' = \emptyset$
- 2 $\mathbb{S} = \bigcup \mathcal{U}$



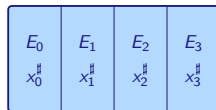
We can apply the **cardinal power construction**:

State partitioning abstraction

We let $\mathbb{D}_0^\sharp = \mathcal{U} \cup \{\perp, \top\}$ and $\gamma_0 : (E \in \mathcal{U}) \mapsto E$. Thus, $\mathbb{D}_{\text{cp}}^\sharp = \mathcal{U} \rightarrow \mathbb{D}_1^\sharp$ and:

$$\begin{aligned} \gamma_{\text{cp}} : \mathbb{D}_{\text{cp}}^\sharp &\longrightarrow \mathbb{D} \\ X^\sharp &\longmapsto \{s \in \mathbb{S} \mid \forall E \in \mathcal{U}, s \in E \implies s \in \gamma_1(X^\sharp(E))\} \end{aligned}$$

- each $E \in \mathcal{U}$ is attached to a piece of information in \mathbb{D}_1^\sharp
- 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 \perp, \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**:

l_0 : // assume $x \geq 0$	$l_0 \mapsto x : \top \wedge y : \top$
l_1 : if ($x < 10$) {	$l_1 \mapsto x : [0, +\infty[\wedge y : \top$
l_2 : $y = x - 2$;	$l_2 \mapsto x : [0, 9] \wedge y : \top$
l_3 : }else{	$l_3 \mapsto x : [0, 9] \wedge y : [-2, 7]$
l_4 : $y = 2 - x$;	$l_4 \mapsto x : [10, +\infty[\wedge y : \top$
l_5 : }	$l_5 \mapsto x : [10, +\infty[\wedge y :] - \infty, -8]$
l_6 : ...	$l_6 \mapsto x : [0, +\infty[\wedge y :] - \infty, 7]$

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,

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

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

Application 1: flow insensitive abstraction

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

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

Flow insensitive abstraction

We apply the cardinal power based partitioning abstraction with:

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

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

Application 1: flow insensitive abstraction

We compare with **flow sensitive abstraction**:

ℓ_0 : // assume $x \geq 0$	$\ell_0 \mapsto x : \top \wedge y : \top$
ℓ_1 : if ($x < 10$) {	$\ell_1 \mapsto x : [0, +\infty[\wedge y : \top$
ℓ_2 : $y = x - 2$;	$\ell_2 \mapsto x : [0, 9] \wedge y : \top$
ℓ_3 : } else {	$\ell_3 \mapsto x : [0, 9] \wedge y : [-2, 7]$
ℓ_4 : $y = 2 - x$;	$\ell_4 \mapsto x : [10, +\infty[\wedge y : \top$
ℓ_5 : }	$\ell_5 \mapsto x : [10, +\infty[\wedge y :] - \infty, -8]$
ℓ_6 : ...	$\ell_6 \mapsto x : [0, +\infty[\wedge y :] - \infty, 7]$

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

For a few specific applications flow insensitive is ok

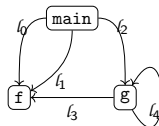
In **most cases** (e.g., numeric properties), flow sensitive is absolutely needed

Application 2: context sensitive abstraction

We consider programs **with procedures**

Example:

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



- 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
		$(f, l) \cdot \kappa$	call to f from stack κ at point l

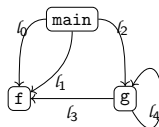
Application 2: context sensitive abstraction, ∞ -CFAFully context sensitive abstraction (∞ -CFA)

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

```

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

```

Abstract contexts in function f :
$$\begin{aligned}
 & (l_0, f) \cdot \epsilon, (l_1, f) \cdot \epsilon, (l_4, f) \cdot (l_2, g) \cdot \epsilon, \\
 & (l_4, f) \cdot (l_3, g) \cdot (l_2, g) \cdot \epsilon, (l_4, f) \cdot (l_3, g) \cdot (l_3, g) \cdot (l_2, g) \cdot \epsilon, \dots
 \end{aligned}$$

- 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

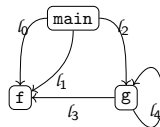
Context insensitive abstraction (0-CFA)

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

```

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

```



Abstract contexts in **function** f are of the form $(?, f) \cdot \dots$,

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

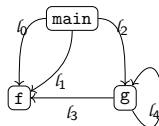
Application 2: context sensitive abstraction, k -CFA

Partially context sensitive abstraction (k -CFA)

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

```

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



Abstract contexts in function f , in 2-CFA:

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

- 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:

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

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

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

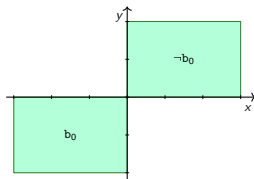
We apply the cardinal power abstraction, with a domain of partitions defined by a function, with:

- $A = \mathbb{B}^k$
- $\phi(m) = (m(\mathbf{b}_0), \dots, m(\mathbf{b}_{k-1}))$
- 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 $\mathbb{X}_{\text{bool}} = \{b_0, b_1\}$, $\mathbb{X}_{\text{int}} = \{x, y\}$, we can express:

$$\left\{ \begin{array}{ll} b_0 \wedge b_1 & \implies x \in [-3, 0] \wedge y \in [-2, 0] \\ b_0 \wedge \neg b_1 & \implies x \in [-3, 0] \wedge y \in [-2, 0] \\ \neg b_0 \wedge b_1 & \implies x \in [0, 3] \wedge y \in [0, 2] \\ \neg b_0 \wedge \neg b_1 & \implies x \in [0, 3] \wedge y \in [0, 2] \end{array} \right.$$



- this abstract value expresses a **relation** between b_0 and x, y (which induces a relation between x and y)
- **alternative**: partition with respect to only **some** variables e.g., here b_0 only since 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 b_0, b_1
- Right side abstraction **shown in green**: interval abstraction
- We omit the cases of the form $P \implies \perp \dots$

```

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:

- $\mathbb{X} = \mathbb{X}_{\text{int}}$, i.e., all variables have **integer** type
- $\mathbb{X}_{\text{int}} = \{x_0, \dots, x_{l-1}\}$

Thus, $\mathbb{M} = \mathbb{X} \rightarrow \mathbb{V} \equiv \mathbb{V}'_{\text{int}}$

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 $\mathcal{P}(\mathbb{V}'_{\text{int}})^{-1}$ (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 $P \implies \perp \dots$

```
int x ∈ ℤ;
```

```
int s;
```

```
int y;
```

```
if(x ≥ 0){
```

```
    (x < 0 ⇒ ⊥) ∧ (x = 0 ⇒ ⊤) ∧ (x > 0 ⇒ ⊤)
```

```
    s = 1;
```

```
    (x < 0 ⇒ ⊥) ∧ (x = 0 ⇒ s = 1) ∧ (x > 0 ⇒ s = 1)
```

```
} else {
```

```
    (x < 0 ⇒ ⊤) ∧ (x = 0 ⇒ ⊥) ∧ (x > 0 ⇒ ⊥)
```

```
    s = -1;
```

```
    (x < 0 ⇒ s = -1) ∧ (x = 0 ⇒ ⊥) ∧ (x > 0 ⇒ ⊥)
```

```
}
```

```
    (x < 0 ⇒ s = -1) ∧ (x = 0 ⇒ s = 1) ∧ (x > 0 ⇒ s = 1)
```

```
① y = x/s;
```

```
    (x < 0 ⇒ s = -1 ∧ y > 0) ∧ (x = 0 ⇒ s = 1 ∧ y = 0) ∧ (x > 0 ⇒ s = 1 ∧ y > 0)
```

```
② assert(y ≥ 0);
```


Outline

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning**
 - Definition and examples
 - Abstract interpretation with boolean partitioning
- 6 Trace partitioning
- 7 Conclusion

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:

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

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:** $\mathcal{P}(\mathbb{L} \times \mathbb{M}) \equiv \mathcal{P}(\mathbb{L} \times (\mathbb{V}_{\text{bool}}^k \times \mathbb{V}_{\text{int}}^l))$

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  $\mathbb{D}_1^\sharp$  *)
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., $x = 1 - x$;
- **assignment to boolean**, e.g., $b_0 = x \leq 7$
- **scalar test**, e.g., $\text{if}(x \geq 8) \dots$
- **boolean test**, e.g., $\text{if}(\neg b_1) \dots$

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

Transfer functions: assignment to scalar (1/2)

Computation of an abstract post-condition

$$x_k = e;$$

Example:

- **statement** $x = 1 - x$;
- **abstract pre-condition:**

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

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

$$\text{assign}_{\text{cp}}(x, e, X^\sharp) = \lambda(z^\sharp \in \mathbb{V}_{\text{bool}}^k) \cdot \text{assign}_1(x, e, X^\sharp(z^\sharp))$$

This post-condition is sound:

Soundness

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

$$\forall X^\sharp \in \mathbb{D}_{\text{cp}}^\sharp, \forall m \in \gamma_{\text{cp}}(X^\sharp), m[x \leftarrow \llbracket e \rrbracket(m)] \in \gamma_{\text{cp}}(\text{assign}_{\text{cp}}(x, e, X^\sharp))$$

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

Example:

$$\text{assign}_{\text{cp}} \left(x, 1 - x, \left\{ \begin{array}{l} b \Rightarrow x \geq 0 \\ \wedge \neg b \Rightarrow x \leq 0 \end{array} \right\} \right) = \left\{ \begin{array}{l} b \Rightarrow x \leq 1 \\ \wedge \neg b \Rightarrow x \geq 1 \end{array} \right\}$$

Transfer functions: scalar test (1/2)

Computation of an abstract post-condition

if(e){...}

where e only refers to numeric variables

(analysis of a condition test, of a loop test, of an assertion)

Example:

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

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

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

$$test_{cp}(c, X^\sharp) = \lambda(z^\sharp \in \mathbb{V}_{bool}^k) \cdot test_1(c, X^\sharp(z^\sharp))$$

This post-condition is sound:

Soundness

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

$$\forall X^\sharp \in \mathbb{D}_{cp}^\sharp, \forall m \in \gamma_{cp}(X^\sharp), \llbracket c \rrbracket(m) = \text{TRUE} \implies m \in \gamma_{cp}(test_{cp}(x, e, X^\sharp))$$

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

Example:

$$test_{cp} \left(x \geq 8, \left\{ \begin{array}{l} b \Rightarrow x \geq 0 \\ \wedge \neg b \Rightarrow x \leq 0 \end{array} \right\} \right) = \left\{ \begin{array}{l} b \Rightarrow x \geq 8 \\ \wedge \neg b \Rightarrow \perp \end{array} \right\}$$

Transfer functions: boolean condition test (1/3)

Computation of an abstract post-condition

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

where e only refers to boolean variables

(analysis of a condition test, of a loop test, of an assertion)

Example:

- **statement:** $\mathbf{if}(\neg b_1)\dots$

- **abstract pre-condition:**

$$\left\{ \begin{array}{l} b_0 \wedge b_1 \Rightarrow 15 \leq x \\ \wedge \quad b_0 \wedge \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \wedge \quad \neg b_0 \wedge b_1 \Rightarrow 6 \leq x \leq 8 \\ \wedge \quad \neg b_0 \wedge \neg b_1 \Rightarrow x \leq 5 \end{array} \right\}$$

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

$$test_{cp}(c, X^\sharp) = \lambda(z^\sharp \in \mathbb{V}_{bool}^k) \cdot \begin{cases} X^\sharp(z^\sharp) & \text{if } test_0(c, X^\sharp(z^\sharp)) \neq \perp_0 \\ \perp_1 & \text{otherwise} \end{cases}$$

This post-condition is sound:

Soundness

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

$$\forall X^\sharp \in \mathbb{D}_{cp}^\sharp, \forall m \in \gamma_{cp}(X^\sharp), \llbracket c \rrbracket(m) = \text{TRUE} \implies m \in \gamma_{cp}(test_{cp}(x, e, X^\sharp))$$

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 (3/3)

Example abstract post-condition:

$$\begin{aligned}
 test_{cp} \left(\neg b_1, \left\{ \begin{array}{l} b_0 \wedge b_1 \Rightarrow 15 \leq x \\ \wedge \quad b_0 \wedge \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \wedge \quad \neg b_0 \wedge b_1 \Rightarrow 6 \leq x \leq 8 \\ \wedge \quad \neg b_0 \wedge \neg b_1 \Rightarrow x \leq 5 \end{array} \right\} \right) \\
 = \left\{ \begin{array}{l} b_0 \wedge b_1 \Rightarrow \perp_1 \\ \wedge \quad b_0 \wedge \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \wedge \quad \neg b_0 \wedge b_1 \Rightarrow \perp_1 \\ \wedge \quad \neg b_0 \wedge \neg b_1 \Rightarrow x \leq 5 \end{array} \right\}
 \end{aligned}$$

Transfer functions: assignment to boolean (1/3)

Computation of an abstract post-condition

$$b_j = e;$$

where e only refers to numeric variables

Example:

• **statement:** $b_0 = x \leq 7$

• **abstract pre-condition:**
$$\left\{ \begin{array}{l} b_0 \wedge b_1 \Rightarrow 15 \leq x \\ \wedge b_0 \wedge \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \wedge \neg b_0 \wedge b_1 \Rightarrow 6 \leq x \leq 8 \\ \wedge \neg b_0 \wedge \neg b_1 \Rightarrow x \leq 5 \end{array} \right\}$$

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

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

Soundness

$$\forall X^\sharp \in \mathbb{D}_{\text{cp}}^\sharp, \forall m \in \gamma_{\text{cp}}(X^\sharp), m[\mathbf{b} \leftarrow \llbracket e \rrbracket(m)] \in \gamma_{\text{cp}}(\text{assign}_{\text{cp}}(\mathbf{b}, e, X^\sharp))$$

Proof: if $z^\sharp \in \mathbb{D}_0^\sharp$ and $z^\sharp(\mathbf{b}) = \text{TRUE}$, then, $\text{assign}_{\text{cp}}(\mathbf{b}, e[x_0, \dots, x_i], X^\sharp)(z^\sharp)$ should account for all states where \mathbf{b} becomes true, whatever the previous value, other boolean variables remaining unchanged; the case where $z^\sharp(\mathbf{b}) = \text{FALSE}$ is symmetric.

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

Transfer functions: assignment to boolean (3/3)

Example abstract post-condition:

$$\begin{aligned}
 \text{assign}_{\text{cp}} \left(b_0, x \leq 7, \left\{ \begin{array}{l} b_0 \wedge b_1 \Rightarrow 15 \leq x \\ \wedge \quad b_0 \wedge \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \wedge \quad \neg b_0 \wedge b_1 \Rightarrow 6 \leq x \leq 8 \\ \wedge \quad \neg b_0 \wedge \neg b_1 \Rightarrow x \leq 5 \end{array} \right\} \right) \\
 = \left\{ \begin{array}{l} b_0 \wedge b_1 \Rightarrow 6 \leq x \leq 7 \\ \wedge \quad b_0 \wedge \neg b_1 \Rightarrow x \leq 5 \\ \wedge \quad \neg b_0 \wedge b_1 \Rightarrow 8 \leq x \\ \wedge \quad \neg b_0 \wedge \neg b_1 \Rightarrow 9 \leq x \leq 14 \end{array} \right\}
 \end{aligned}$$

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

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning
- 6 Trace partitioning**
 - Principles and examples
 - Abstract interpretation with trace partitioning
- 7 Conclusion

Definition of trace partitioning

Principle

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

- **concrete domain**: $\mathbb{D} = \mathcal{P}(\mathbb{S}^*)$
- **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:

$$\gamma_1 : M \mapsto \{ \langle s_0, \dots, s_k, (\ell, m) \rangle \mid m \in M, \ell \in \mathbb{L}, s_0, \dots, s_k \in \mathbb{S} \}$$
 - ▶ 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:

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

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 $\mathbb{D}_0^\sharp = \{\tau_{\text{if:t}}, \tau_{\text{if:f}}, \top\}$ and:

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

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 ∈ ℤ;
int s;
int y;
if(x ≥ 0){
    τif:t ⇒ (0 ≤ x) ∧ τif:f ⇒ ⊥
    s = 1;
    τif:t ⇒ (0 ≤ x ∧ s = 1) ∧ τif:f ⇒ ⊥
} else {
    τif:f ⇒ (x < 0) ∧ τif:t ⇒ ⊥
    s = -1;
    τif:f ⇒ (x < 0 ∧ s = -1) ∧ τif:t ⇒ ⊥
}
    {
      τif:t ⇒ (0 ≤ x ∧ s = 1)
    ∧ τif:f ⇒ (x < 0 ∧ s = -1)
    }
y = x/s;
    {
      τif:t ⇒ (0 ≤ x ∧ s = 1 ∧ 0 ≤ y)
    ∧ τif:f ⇒ (x < 0 ∧ s = -1 ∧ 0 < y)
    }

```

Application 3: partitioning guided by a loop

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

```

 $l_0$  : while( $c$ ){
 $l_1$  :   ...
 $l_2$  : }
 $l_3$  : ...

```

Domain of partitions

For a given $k \in \mathbb{N}$, the partitions are defined by

$\mathbb{D}_0^\# = \{\tau_{\text{loop}:0}, \tau_{\text{loop}:1}, \dots, \tau_{\text{loop}:k}, \top\}$ and:

$$\begin{array}{ll} \gamma_0 : & \tau_{\text{loop}:i} \mapsto \text{traces that visit } l_1 \text{ } i \text{ times} \\ & \top \mapsto \mathbb{S}^* \end{array}$$

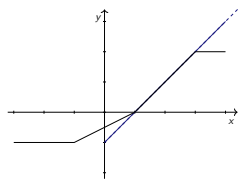
Application:

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++;
}

```

$$\left\{ \begin{array}{ll} \tau_{\text{loop}:0} \Rightarrow & \perp & (\text{case } x \leq -1) \\ \tau_{\text{loop}:1} \Rightarrow & 0 \leq x \leq 1 \wedge i = 1 & (\text{case } -1 \leq x \leq 1) \\ \tau_{\text{loop}:2} \Rightarrow & 1 \leq x \leq 3 \wedge i = 2 & \\ \tau_{\text{loop}:3} \Rightarrow & 3 \leq x \wedge i = 3 & \end{array} \right.$$

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

Application 4: partitioning guided by the value of a variable

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

```
int x; ...;  $\ell$  : ...
```

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

$\mathbb{D}_0^\# = \{\tau_{\text{val}:i} \mid i \in \mathcal{E}\} \uplus \{\top\}$ and:

$$\begin{aligned} \gamma_0 : \tau_{\text{val}:k} &\longmapsto \{\langle \dots, (\ell, m), \dots \rangle \mid m(x) = k\} \\ \top &\longmapsto \mathbb{S}^* \end{aligned}$$

Domain of partitions: partitioning by the property of a variable

For a given abstraction $\gamma : (V^\#, \sqsubseteq^\#) \rightarrow (\mathcal{P}(\mathbb{V}_{\text{int}}), \subseteq)$, the partitions are defined by

$\mathbb{D}_0^\# = \{\tau_{\text{var}:v^\#} \mid v^\# \in V^\#\}$ and:

$$\gamma_0 : \tau_{\text{val}:v^\#} \longmapsto \{\langle \dots, (\ell, m), \dots \rangle \mid m(x) \in \tau_{\text{var}:v^\#}\}$$

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 b0, b1;
int x, y;      (uninitialized)
①      (x < 0@① ⇒ ⊤) ∧ (x = 0@① ⇒ ⊤) ∧ (x > 0@① ⇒ ⊤)
b0 = x ≥ 0;
      (x < 0@① ⇒ ¬b0) ∧ (x = 0@① ⇒ b0) ∧ (x > 0@① ⇒ b0)
b1 = x ≤ 0;
      (x < 0@① ⇒ ¬b1 ∧ b1) ∧ (x = 0@① ⇒ b0 ∧ b1) ∧ (x > 0@① ⇒ b0 ∧ ¬b1)
if(b0 && b1){
      (x < 0@① ⇒ ⊥) ∧ (x = 0@① ⇒ b0 ∧ b1) ∧ (x > 0@① ⇒ ⊥)
      y = 0;
      (x < 0@① ⇒ ⊥) ∧ (x = 0@① ⇒ b0 ∧ b1 ∧ y = 0) ∧ (x > 0@① ⇒ ⊥)
} else {
      (x < 0@① ⇒ ¬b0 ∧ b1) ∧ (x = 0@① ⇒ ⊥) ∧ (x > 0@① ⇒ b0 ∧ ¬b1)
      y = 100/x;
      (x < 0@① ⇒ ¬b0 ∧ b1 ∧ y ≤ 0) ∧ (x = 0@① ⇒ ⊥) ∧ (x > 0@① ⇒ b0 ∧ ¬b1 ∧ y ≥ 0)
}

```

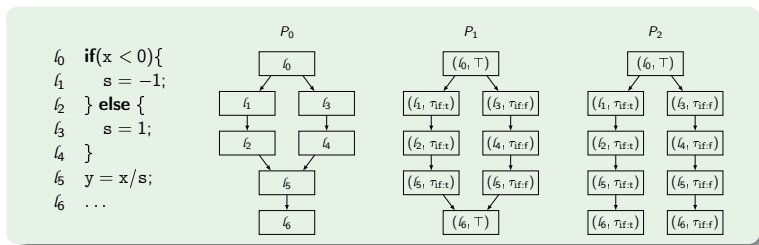

Outline

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning
- 6 Trace partitioning**
 - Principles and examples
 - Abstract interpretation with trace partitioning
- 7 Conclusion

Trace partitioning induced by a refined transition system

We consider possible **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)



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 $((\ell, \tau), m) \in \llbracket \mathcal{S} \rrbracket_{\mathcal{R}}$ then,
 $(\ell, m) \rightarrow (\ell', m') \implies \exists \tau' \in \mathbb{D}_0^\sharp, ((\ell, \tau), m) \rightarrow ((\ell', \tau'), m')$

In that case, we write:

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

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

Partitioned 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 $S' \prec S$. We let $\llbracket S \rrbracket_{\mathcal{T}^{*\omega}}$ (*resp.*, $\llbracket S' \rrbracket_{\mathcal{T}^{*\omega}}$) denote the trace semantics of S (*resp.*, S'). Then:

$$\begin{aligned} \forall \langle (\ell_0, m_0), \dots, (\ell_n, m_n) \rangle \in \llbracket S \rrbracket_{\mathcal{T}^{*\omega}}, \\ \exists \tau_0, \dots, \tau_n \in \mathbb{D}_0^\sharp, \langle ((\ell_0, \tau_0), m_0), \dots, ((\ell_n, \tau_n), m_n) \rangle \in \llbracket S' \rrbracket_{\mathcal{T}^{*\omega}}, \end{aligned}$$

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

Properties of $S' \prec S$

- all traces of S have a counterpart in S' (up to token addition)
- a trace in S' embeds more information than a trace in 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** $(S', \rightarrow', S'_I) \prec (S, \rightarrow, S_I)$
- **erasure function**: $\Psi : (S')^* \rightarrow S^*$ removes the tokens

Definition of a trace partitioning

The abstraction defining partitions is defined by:

$$\begin{aligned} \gamma_0 : \mathbb{D}_0^\sharp &\longrightarrow \mathcal{P}(S^*) \\ \tau &\longmapsto \{\sigma \in S^* \mid \exists \sigma' = \langle \dots, ((l, \tau), m) \rangle \in (S')^*, \Psi(\sigma') = \sigma\} \end{aligned}$$

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

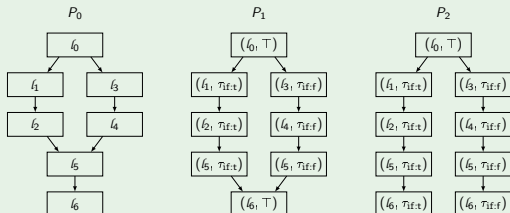
- **control states** and **call stack** partitioning
- partitioning guided by **conditions** and **loops**
- partitioning **guided by the value of a variable**

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  ...
  
```



- 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

$$P_2 \prec P_1 \prec P_0 \quad \text{thus,} \quad \llbracket P_0 \rrbracket_{\mathcal{T}^* \omega} \subseteq \llbracket P_1 \rrbracket_{\mathcal{T}^* \omega} \subseteq \llbracket P_2 \rrbracket_{\mathcal{T}^* \omega}$$

- 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

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

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

Transfer functions: partition creation

Analysis of an if statement, with partitioning

$$\begin{array}{ll}
 \ell_0 : & \mathbf{if}(c)\{ \\
 \ell_1 : & \dots \\
 \ell_2 : & \} \mathbf{else}\{ \\
 \ell_3 : & \dots \\
 \ell_4 : & \} \\
 \ell_5 : & \dots
 \end{array}
 \quad
 \begin{array}{l}
 \delta_{\ell_0, \ell_1}^{\#}(X^{\#}) = [\tau_{\text{if:t}} \mapsto \text{test}(c, \sqcup X^{\#}(\tau)), \tau_{\text{if:f}} \mapsto \perp] \\
 \delta_{\ell_0, \ell_3}^{\#}(X^{\#}) = [\tau_{\text{if:t}} \mapsto \perp, \tau_{\text{if:f}} \mapsto \text{test}(\neg c, \sqcup X^{\#}(\tau))] \\
 \delta_{\ell_2, \ell_5}^{\#}(X^{\#}) = X^{\#} \\
 \delta_{\ell_4, \ell_5}^{\#}(X^{\#}) = X^{\#}
 \end{array}$$

Observations:

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

Transfer functions: partition fusion

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

$$\delta_{\ell_0, \ell_1}^\#(X^\#) = [_ \mapsto \sqcup_\tau X^\#(\ell_0)(\tau)]$$

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

- 1 Introduction
- 2 Imprecisions in convex abstractions
- 3 Disjunctive completion
- 4 Cardinal power and partitioning abstractions
- 5 State partitioning
- 6 Trace partitioning
- 7 Conclusion**

Adding disjunctions in static analyses

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

$$P_0 \vee \dots \vee P_n$$

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

$$(P_0 \implies Q_0) \wedge \dots \wedge (P_n \implies Q_n)$$

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 (simple):

prove the disjunctive completion algorithm (Slide 15)

Proof 2 (harder):

justify the general cardinal power post-condition (Slide 37)

Proof 3:

what happens in the case we use coverings instead of partitions (Slide 42)

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