Partitioning abstractions

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

Xavier Rival

INRIA, ENS, CNRS

Oct, 29th. 2014
Towards disjunctive abstractions

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

In this lecture, we will discuss:

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

General combination of abstract domains
- takes one or more abstract domains as inputs
- produces a new abstract domain

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

Advantages:
- general definition, formalized and proved once
- can be implemented in a separate way, e.g., in ML:
  - abstract domain: module
    module D = (struct ... end: Interface)
  - abstract domain combinator: functor
    module C = functor (D: Interface) ->
    (struct ... end: Interface)
Example: product abstraction

Notations

for sets:

• \( M \): stores
• \( V \): values
• \( X \): variables

Assumptions:

• concrete domain \((P(M), \subseteq)\) with \( M = X \rightarrow V \)
• we require an abstract domain \( D^{\#} \) to provide
  ▶ a concretization function \( \gamma : D^{\#} \rightarrow P(M) \)
  ▶ an element \( \perp \) with empty concretization \( \gamma(\perp) = \emptyset \)

Product combinator (implemented as a functor)

Given abstract domains \((D_0^{\#}, \gamma_0, \perp_0)\) and \((D_1^{\#}, \gamma_1, \perp_1)\), the product abstraction is \((D_x^{\#}, \gamma_x, \perp_x)\) where:

- \( D_x^{\#} = D_0^{\#} \times D_1^{\#} \)
- \( \gamma_x(x_0^{\#}, x_1^{\#}) = \gamma_0(x_0^{\#}) \cap \gamma_1(x_1^{\#}) \)
- \( \perp_x = (\perp_0, \perp_1) \)

This amounts to expressing conjunctions of elements of \( D_0^{\#} \) and \( D_1^{\#} \)
Example: product abstraction, coalescent product

The product abstraction **needs a reduction:***

\[ \forall x_0^\# \in D_0^\#, x_1^\# \in D_1^\#, \gamma_x(\bot_0, x_1^\#) = \gamma_x(x_0^\#, \bot_1) = \emptyset = \gamma_x(\bot_x) \]

**Coalescent product**

Given abstract domains \((D_0^\#, \gamma_0, \bot_0)\) and \((D_1^\#, \gamma_1, \bot_1)\), the **coalescent product abstraction** is \((D_x^\#, \gamma_x, \bot_x)\) where:

- \(D_x^\# = \{\bot_x\} \uplus \{(x_0^\#, x_1^\#) \in D_0^\# \times D_1^\# \mid x_0^\# \neq \bot_0 \land x_1^\# \neq \bot_1\} \)
- \(\gamma_x(\bot_x) = \emptyset, \gamma_x(x_0^\#, x_1^\#) = \gamma_0(x_0^\#) \cap \gamma_1(x_1^\#) \)

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

- let \(D_0^\#\) be the interval abstraction, \(D_1^\#\) be the congruences abstraction
- \(\gamma_x(\{x \in [3, 4]\}, \{x \equiv 0 \mod 5\}) = \emptyset \)

- how to define abstract domain combiners 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
Imprecisions in convex abstractions

Convex abstractions

Many numerical abstractions describe convex sets of points

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

Such imprecisions may impact analysis results
Non convex abstractions

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

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

**Signs:**
- $0 \notin \gamma([\neq 0])$ so $[\neq 0]$ describes a non convex set
- other abstract elements describe convex sets
Example 1: verification problem

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

- if ¬b0, then x < 0
- if ¬b1, then x > 0
- if either b0 or b1 is false, then x ≠ 0
- thus, if point ① is reached the division is safe
```

How to verify the division operation?

- Non relational abstraction (e.g., intervals), at point ①:
  \[
  \begin{cases}
  b_0 &= \text{FALSE} \\
  b_1 &= \text{FALSE} \\
  x &= \top
  \end{cases}
  \]

- Signs, congruences do not help:
in the concrete, x may take any value but 0
Example 1: program annotated with local invariants

```c
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)
}
```

We need to add symbolic disjunctions to our abstract domain
Example 2: verification problem

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

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

**How to verify the division operation?**

- In the concrete, $s$ is **always non null**: convex abstractions cannot establish this; congruences can
- Moreover, $s$ has always the **same sign** as $x$
- expressing this would require a fairly complex numerical abstraction
Example 2: program annotated with local invariants

```c
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)

1 y = x/s;
(x ≥ 0 ∧ s = 1 ∧ y ≥ 0) ∨ (x < 0 ∧ s = -1 ∧ y > 0)
2 assert(y ≥ 0);
```

We need to add 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 \((D, \sqsubseteq)\), with lower upper bound operator \(\sqcap\)
2. start with an abstract domain \((D^\#, \sqsubseteq^\#)\) with concretization \(\gamma : D^\# \to D\)
3. build a domain containing all the disjunctions of elements of \(D^\#\)

**Definition: distributive abstract domain**

Abstract domain \((D^\#, \sqsubseteq^\#)\) with concretization function \(\gamma : D^\# \to D\) is distributive (or complete for disjunction) if and only if:

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

**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: disjunctive completion**

The **disjunctive completion** of abstract domain \((\mathbb{D}^\# , \subseteq^\#)\) with concretization function \(\gamma : \mathbb{D}^\# \rightarrow \mathbb{D}\) is the **smallest abstract domain** \((\mathbb{D}^\#_{\text{disj}} , \subseteq^\#_{\text{disj}})\) with concretization function \(\gamma_{\text{disj}} : \mathbb{D}^\#_{\text{disj}} \rightarrow \mathbb{D}\) such that:

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

**Building a disjunctive completion domain:**

- start with \(\mathbb{D}^\#_{\text{disj}} = \mathbb{D}^\#\)
- for all set \(\mathcal{E} \subseteq \mathbb{D}^\#\) such that there is no \(x^\# \in \mathbb{D}^\#\), such that \(\gamma(x^\#) = \bigcup_{y^\# \in \mathcal{E}} \gamma(y^\#)\), add \([\sqcup \mathcal{E}]\) to \(\mathbb{D}^\#_{\text{disj}}\), and extend \(\gamma_{\text{disj}}\) by
  
  \[
  \gamma_{\text{disj}}([\sqcup \mathcal{E}]) = \bigcup_{y^\# \in \mathcal{E}} \gamma(y^\#)
  \]
Example 1: completion of signs

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

\[
\begin{align*}
\bot & \rightarrow \emptyset \\
[-0] & \rightarrow \{ k \in \mathbb{Z} \mid k < 0 \} \\
[=0] & \rightarrow \{ k \in \mathbb{Z} \mid k = 0 \} \\
[>0] & \rightarrow \{ k \in \mathbb{Z} \mid k > 0 \} \\
\top & \rightarrow \mathbb{Z}
\end{align*}
\]

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

- \{[<0], [=0]\}
- \{[<0], [>0]\}
- \{[=0], [>0]\}
Example 2: completion of constants

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

\[
\begin{align*}
\bot & \rightarrow \emptyset \\
\{k\} & \rightarrow \{k\} \\
\top & \rightarrow \mathbb{Z}
\end{align*}
\]

Then, the disjunctive completion is the power-set:

- \( \mathbb{D}_{\text{disj}}^\# \equiv \mathcal{P}(\mathbb{Z}) \)
- \( \gamma_{\text{disj}} \) is the **identity function**!
- this lattice contains **infinite sets which are not representable**
Example 3: completion of intervals

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

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

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

- $\mathbb{D}^\#_{\text{disj}}$ collects all the families of disjoint intervals
- this lattice contains infinite sets which are not representable

The disjunctive completion of $(\mathbb{D}^\#)^n$ is not equivalent to $(\mathbb{D}^\#_{\text{disj}})^n$

- which is more expressive?
- show it on an example!
Example 3: completion of intervals and verification

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

```plaintext
int x ∈ \mathbb{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 with disjunctive completion

Transfer functions:

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

  $$\tau \circ \gamma \subseteq \gamma \circ \tau^\#$$

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

  $$\tau_{\text{disj}}^\#([\sqcup \mathcal{E}]) = \sqcup \{\tau^\#(x^\#) \mid x^\# \in \mathcal{E}\}$$

Abstract join:

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

Inclusion check: exercise!
Disjunctive completion

Limitations of disjunctive completion

- **Combinatorial explosion:**
  - if $D^\#$ is infinite, $D^\#_{\text{disj}}$ may have elements that **cannot be represented**
  - even when $D^\#$ is finite, $D^\#_{\text{disj}}$ may be **huge**
    - in the worst case, if $D^\#$ has $n$ elements, $D^\#_{\text{disj}}$ may have $2^n$ elements

- **Many elements useless in practice:**
  - disjunctive completion of intervals: may express any set of integers...

- **No general definition of a widening operator**
  - most common approach to achieve that: **$k$-limiting**
    - bound the numbers of disjuncts
    - i.e., the size of the sets added to the base domain
  - issue: the join operator should “select” which disjoints to merge
Outline

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

- disjuncts **that are needed for static analysis** can usually be **characterized** by some property for instance:
  - **sign** of a variable
  - **value** of a **boolean** variable
  - **execution path**, e.g., side of a condition that was visited

- **solution**: perform a kind of **indexing** of disjuncts
  - use an abstraction to **describe labels**
    - e.g., sign of a variable, value of a boolean, or trace property...
  - apply the abstraction that needs be completed on the images
Disjuncts indexing: example

```c
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

Definition

We assume \((\mathbb{D}, \subseteq) = (\mathcal{P}(E), \subseteq)\), and that two abstractions \((\mathbb{D}_0^\#(\subseteq_0^\#), (\mathbb{D}_1^\#(\subseteq_1^\#))\) given by their concretization functions:

\[
\gamma_0 : \mathbb{D}_0^\# \rightarrow \mathbb{D} \quad \gamma_1 : \mathbb{D}_1^\# \rightarrow \mathbb{D}
\]

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

- \(\mathbb{D}_{cp}^\# = \mathbb{D}_0^\# \xrightarrow{\mathcal{M}} \mathbb{D}_1^\#\) be the set of monotone functions from \(\mathbb{D}_0^\#\) into \(\mathbb{D}_1^\#\)
- \(\subseteq_{cp}^\#\) be the pointwise extension of \(\subseteq_1^\#\)
- \(\gamma_{cp}\) is defined by:

\[
\gamma_{cp} : \mathbb{D}_{cp}^\# \rightarrow \mathbb{D}
\]

\[
X^\# \mapsto \{y \in E \mid \forall z^\# \in \mathbb{D}_0^\#, y \in \gamma_0(z^\#) \Rightarrow y \in \gamma_1(X^#(z^#))\}
\]

We sometimes denote it by \(\mathbb{D}_0^\# \Rightarrow \mathbb{D}_1^\#, \gamma_{\mathbb{D}_0^\# \Rightarrow \mathbb{D}_1^\#}\).
Use of cardinal power abstractions

**Intuition:** we can express properties of the form

\[
\begin{cases}
  p_0 \implies p'_0 \\
  \land p_1 \implies p'_1 \\
  \vdots \\
  \land p_n \implies p'_n
\end{cases}
\]

Two independent choices:

1. \( D_0^\# \): set of partitions (the “labels”)
2. \( D_1^\# \): abstraction of sets of states, e.g., a numerical abstraction

**Application**

\((x \geq 0 \land s = 1 \land y \geq 0) \lor (x < 0 \land s = -1 \land y > 0)\)

- \( D_0^\# \): sign of \( s \)
- \( D_1^\# \): other constraints
Another example, with a single variable

We consider:

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\subseteq = \subseteq$
- $(\mathbb{D}_0^\#, \subseteq_0^\#)$ be the lattice of signs (strict values only)
- $(\mathbb{D}_1^\#, \subseteq_1^\#)$ be the lattice of intervals

A few example abstract values:

- $[0, 8]$ is expressed by:
  \[
  \begin{align*}
  \bot_0 & \rightarrow \bot_1 \\
  [< 0] & \rightarrow \bot_1 \\
  [= 0] & \rightarrow [0, 0] \\
  [> 0] & \rightarrow [1, 8] \\
  \top_0 & \rightarrow [0, 8]
  \end{align*}
  \]

- $[-10, -3] \uplus [7, 10]$ is expressed by:
  \[
  \begin{align*}
  \bot_0 & \rightarrow \bot_1 \\
  [< 0] & \rightarrow [-10, -3] \\
  [= 0] & \rightarrow \bot_1 \\
  [> 0] & \rightarrow [7, 10] \\
  \top_0 & \rightarrow [-10, 10]
  \end{align*}
  \]
Reduction (1): tightening disjunctions

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\subseteq\subseteq$
- $(\mathbb{D}_0^\#, \subseteq_0^\#)$ be the lattice of signs
- $(\mathbb{D}_1^\#, \subseteq_1^\#)$ be the lattice of intervals

We let:

$X^\# = \begin{cases}
\bot_0 & \mapsto \bot_1 \\
[< 0] & \mapsto [-5, -1] \\
[= 0] & \mapsto [0, 0] \\
[> 0] & \mapsto [1, 5] \\
\top_0 & \mapsto [-10, 10]
\end{cases}
$ and

$Y^\# = \begin{cases}
\bot_0 & \mapsto \bot_1 \\
[< 0] & \mapsto [-5, -1] \\
[= 0] & \mapsto [0, 0] \\
[> 0] & \mapsto [1, 5] \\
\top_0 & \mapsto [-5, 5]
\end{cases}$

Then, $\gamma_{cp}(X^\#) = \gamma_{cp}(Y^\#)$

$\gamma_0([< 0]) \cup \gamma_0([= 0]) \cup \gamma([> 0]) = \gamma(\top_0)$

but $\gamma_0(X^#([< 0])) \cup \gamma_0(X^#([= 0])) \cup \gamma(X^#([> 0])) \subset \gamma(X^#(\top_0))$

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

- $\sqcup \{\gamma_0(z^\#) \mid z^\# \in \mathcal{E}\} = \gamma_0(\sqcup \{z^\# \mid z^\# \in \mathcal{E}\})$

- $\exists y^\#, \quad \sqcup \{\gamma_1(X^#(z^\#)) \mid z^\# \in \mathcal{E}\} \subseteq \gamma_1(y^\#) \subset \gamma_1(X^#(\sqcup \{z^\# \mid z^\# \in \mathcal{E}\}))$
Reduction (2): relation between the two domains

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\subseteq \subseteq$
- $(\mathbb{D}_0^\#, \subseteq_0^\#)$ be the lattice of signs
- $(\mathbb{D}_1^\#, \subseteq_1^\#)$ be the lattice of intervals

We let:

\[
\begin{align*}
X^\# &= \begin{cases} 
\bot_0 & \mapsto \bot_1 \\
[< 0] & \mapsto [1, 8] \\
[= 0] & \mapsto [1, 8] \\
[> 0] & \mapsto \bot_1 \\
\top_0 & \mapsto [1, 8]
\end{cases} \\
Y^\# &= \begin{cases} 
\bot_0 & \mapsto \bot_1 \\
[< 0] & \mapsto [2, 45] \\
[= 0] & \mapsto [-5, -2] \\
[> 0] & \mapsto [-5, -2] \\
\top_0 & \mapsto \top_1
\end{cases} \\
Z^\# &= \begin{cases} 
\bot_0 & \mapsto \bot_1 \\
[< 0] & \mapsto \bot_1 \\
[= 0] & \mapsto \bot_1 \\
[> 0] & \mapsto \bot_1 \\
\top_0 & \mapsto \bot_1
\end{cases}
\end{align*}
\]

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

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

Binding $y_0^\# \mapsto y_1^\#$ can be improved if $\exists z_1^\# \neq y_1^\#$, $\gamma(y_1^\#) \cap \gamma(y_0^\#) \subseteq \gamma(z_1^\#)$
Representation of the cardinal power

**Basic ML representation:**

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

This is not a very efficient representation:

- if $D_0^\#$ has $N$ elements, then an abstract value in $D_{cp}^\#$ requires $N$ elements of $D_1^\#$
- if $D_0^\#$ is infinite, and $D_1^\#$ is non-trivial, then $D_{cp}^\#$ has elements that cannot be represented
- the 1st reduction shows it is unnecessary to represent bindings for all elements of $D_0^\#$
  
example: this is the case of $\bot_0$
More compact representation of the cardinal power

**Principle:**
- keep the **same data-type** (most likely functional arrays)
- avoid representing information attached to redundant elements

**Compact representation**

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

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

In particular:
- $C$ should be **minimal**
- in any case, $\bot_0 \notin C$
Example: compact cardinal power over signs

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

We remark that:

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

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

- $[0, 8]$ is expressed by: $\begin{cases} [< 0] &\mapsto \bot_1 \\ [= 0] &\mapsto [0, 0] \\ [> 0] &\mapsto [1, 8] \end{cases}$
- $[−10, −3] \cup [7, 10]$ is expressed by: $\begin{cases} [< 0] &\mapsto [−10, −3] \\ [= 0] &\mapsto \bot_1 \\ [> 0] &\mapsto [7, 10] \end{cases}$
Lattice operations

**Infimum:**
- we assume that \( \bot_1 \) is the infimum of \( D_1^\# \)
- then, \( \bot_{cp} = \lambda(z^\# \in D_0^\#) \cdot \bot_1 \) is the infimum of \( D_{cp}^\# \)

**Ordering:**
- we let \( \sqsubseteq_{cp} \) denote the pointwise ordering:
  \[
  X_0^\# \sqsubseteq_{cp} X_1^\# \iff \forall z^\# \in D_0^\#, X_0^\#(z^\#) \sqsubseteq_1 X_1^\#(z^\#)
  \]
  - then, \( X_0^\# \sqsubseteq_{cp} X_1^\# \implies \gamma_{cp}(X_0^\#) \subseteq \gamma_{cp}(X_1^\#) \)

**Join operation:**
- we assume that \( \sqcup_1 \) is a sound upper bound operator in \( D_1^\# \)
- then, \( \sqcup_{cp} \) defined below is a sound upper bound operator in \( D_{cp}^\# \):
  \[
  X_0^\# \sqcup_{cp} X_1^\# \overset{\text{def}}{=} \lambda(z^\# \in D_0^\#) \cdot (X_0^\#(z^\#) \sqcup_1 X_1^\#(z^\#))
  \]
  - the same construction applies to widening, if \( D_0^\# \) is finite
Composition with another abstraction

We assume three abstractions

- \((D^\#_0, \subseteq_0)\), with concretization \(\gamma_0 : D^\#_0 \rightarrow D\)
- \((D^\#_1, \subseteq_1)\), with concretization \(\gamma_1 : D^\#_1 \rightarrow D\)
- \((D^\#_2, \subseteq_2)\), with concretization \(\gamma_2 : D^\#_2 \rightarrow D^\#_1\)

Cardinal power abstract domains \(D^\#_0 \Rightarrow D^\#_1\) and \(D^\#_0 \Rightarrow D^\#_2\) can be bound by an abstraction relation defined by concretization function \(\gamma\):

\[
\gamma : (D^\#_0 \Rightarrow D^\#_2) \quad \rightarrow \quad (D^\#_0 \Rightarrow D^\#_1) \quad \rightarrow \quad \lambda(z^\# \in D^\#_0) \cdot \gamma(X^\#(z^\#))
\]

Applications:

- start with \(D^\#_1\) as the identity abstraction
- compose several cardinal power abstractions
  (or partitioning abstractions)
Composition with another abstraction

- concrete lattice $\mathbb{D} = \mathcal{P}(\mathbb{Z})$, with $\subseteq \subseteq$
- $(\mathbb{D}_0^\#, \subseteq_0^\#)$ be the lattice of signs
- $(\mathbb{D}_1^\#, \subseteq_1^\#)$ be the identity abstraction
  $\mathbb{D}_1^\# = \mathcal{P}(\mathbb{Z})$, $\gamma_1 = \text{Id}$
- $(\mathbb{D}_2^\#, \subseteq_2^\#)$ be the lattice of intervals

\[ [-] \quad [0] \quad [+] \]
\[ \downarrow \quad \quad \quad \quad \downarrow \]
\[ \downarrow \quad \quad \quad \quad \downarrow \]

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

- in $\mathbb{D}_0^\# \Rightarrow \mathbb{D}_1^\#$, 
  \[ \begin{align*} 
  [< 0] & \mapsto [-10, -3] \\
  [= 0] & \mapsto \emptyset \\
  [> 0] & \mapsto [7, 10]
  \end{align*} \]

- in $\mathbb{D}_0^\# \Rightarrow \mathbb{D}_2^\#$, 
  \[ \begin{align*} 
  [< 0] & \mapsto [-10, -3] \\
  [= 0] & \mapsto \bot_1 \\
  [> 0] & \mapsto [7, 10]
  \end{align*} \]
Outline

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

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

- \( \mathcal{S} = \mathcal{L} \times \mathcal{M} \) where \( \mathcal{L} \) denotes the set of control states
- \( \mathcal{M} = \mathcal{X} \rightarrow \mathcal{V} \)

**State partitioning**

A **state partitioning** abstraction is defined as the cardinal power of two abstractions \( (\mathcal{D}_0^\#, \sqsubseteq_0, \gamma_0) \) and \( (\mathcal{D}_1^\#, \sqsubseteq_1, \gamma_1) \) of sets of states:

- \( (\mathcal{D}_0^\#, \sqsubseteq_0, \gamma_0) \) defines the partitions
- \( (\mathcal{D}_1^\#, \sqsubseteq_1, \gamma_1) \) defines the abstraction of each element of partitions

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

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

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

We can apply **cardinal power construction**:

State partitioning abstraction

We let $\mathbb{D}_0^\# = \mathcal{E}$ and $\gamma_0 : e \mapsto e$. Thus, $\mathbb{D}_{cp}^\# = \mathcal{E} \to \mathbb{D}_1^\#$ and:

$$
\gamma_{cp} : \mathbb{D}_{cp}^\# \longrightarrow \mathbb{D} \\
X^\# \longmapsto \{ s \in \mathbb{S} \mid \forall e \in \mathcal{E}, \ s \in e \implies s \in \gamma_0(X^\#(e)) \}
$$

- each $e \in \mathcal{E}$ is attached to a piece of information in $\mathbb{D}_1^\$
- exercise: what happens if use only a **covering**, i.e., if we drop property 1?
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 interval abstract domain:

\[
\begin{align*}
l_0 & : \quad // \text{ assume } x \geq 0 \\
l_1 & : \quad \textbf{if}(x < 10)\{ \\
l_2 & : \quad y = x - 2; \\
l_3 & : \quad }\textbf{else}\{ \\
l_4 & : \quad y = 2 - x; \\
l_5 & : \quad } \\
l_6 & : \quad ...
\end{align*}
\]

\[
\begin{align*}
l_0 & \mapsto x : T \land y : T \\
l_1 & \mapsto x : [0, +\infty] \land y : T \\
l_2 & \mapsto x : [0, 9] \land y : T \\
l_3 & \mapsto x : [0, 9] \land y : [-2, 7] \\
l_4 & \mapsto x : [10, +\infty] \land y : T \\
l_5 & \mapsto x : [10, +\infty] \land y : ] - \infty, -2] \\
l_6 & \mapsto x : [0, +\infty] \land y : ] - \infty, 7] \\
\end{align*}
\]
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:

- $D_0^\# = \mathcal{L}$
- $\gamma_0 : \ell \mapsto \{\ell\} \times \mathbb{M}$

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

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

$$\gamma_{cp}(X^\#) = \{s \in \mathbb{S} \mid \forall x \in D_0^\#, s \in \gamma_0(x) \implies s \in \gamma_1(X^\#(x))\}$$

$$= \{(l, m) \in \mathbb{S} \mid m \in \gamma_1(X^\#(l))\}$$

- after this abstraction step, $D_1^\#$ only needs to represent sets of memory states (numeric abstractions...)
- this abstraction step is *very common* as part of the design of abstract interpreters
Application 1: flow insensitive abstraction

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

Flow sensitive abstraction
We apply the cardinal power based partitioning abstraction with:

- $D^0_0 = \{\cdot\}$
- $\gamma_0 : \cdot \mapsto S$
- $D^1_1 = \mathcal{P}(M)$
- $\gamma_1 : M \mapsto \{(l, m) \mid l \in \mathbb{L}, m \in M\}$

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

- used for some ultra-fast pointer analyses (very quick analyses used for, e.g., compiler optimization)
- otherwise, usually too coarse
Application 1: flow insensitive abstraction

We compare with **flow sensitive abstraction**:

\[
\begin{align*}
\ell_0 & : \text{ // assume } x \geq 0 & \ell_0 & \mapsto x : \top \land y : \top \\
\ell_1 & : \text{ if } (x < 10) \{ & \ell_1 & \mapsto x : [0, +\infty] \land y : \top \\
\ell_2 & : \quad y = x - 2; & \ell_2 & \mapsto x : [0, 9] \land y : \top \\
\ell_3 & : \quad } \text{else} \{ & \ell_3 & \mapsto x : [0, 9] \land y : [-2, 7] \\
\ell_4 & : \quad y = 2 - x; & \ell_4 & \mapsto x : [10, +\infty] \land y : \top \\
\ell_5 & : \quad } & \ell_5 & \mapsto x : [10, +\infty] \land y : ] - \infty, -2] \\
\ell_6 & : \quad ... & \ell_6 & \mapsto x : [0, +\infty] \land y : ] - \infty, 7]
\end{align*}
\]

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

For a few specific applications flow insensitive is ok  
In most cases (e.g., numeric programs), flow sensitive is absolutely needed
Application 2: context sensitive abstraction

We consider programs with procedures

Example:

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

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

**Call string**

Thus, \( S = K \times L \times M \), where \( K \) is the set of call strings

\[
\begin{align*}
\kappa & \in K & \text{calling contexts} \\
\kappa & ::= \epsilon & \text{empty call stack} \\
& | (f, l) \cdot \kappa & \text{call to } f \text{ from stack } \kappa \text{ at point } l
\end{align*}
\]
Application 2: context sensitive abstraction, $\infty$-CFA

Fully context sensitive abstraction ($\infty$-CFA)

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

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

Contexts in function $f$:

- $(l_0, f) \cdot \epsilon$, $(l_1, f) \cdot \epsilon$, $(l_4, f) \cdot (l_2, g) \cdot \epsilon$,
- $(l_4, f) \cdot (l_3, g) \cdot (l_2, g) \cdot \epsilon$, $(l_4, f) \cdot (l_3, g) \cdot (l_3, g) \cdot (l_2, g) \cdot \epsilon$, ...

- one invariant per calling context, very precise (used, e.g., in Astrée)
- infinite in presence of recursion (i.e., not practical in this case)
Application 2: context sensitive abstraction, 0-CFA

Non context sensitive abstraction (0-CFA)

- $D_0^\# = L$
- $\gamma_0 : \ell \mapsto \{(\kappa, \ell, m) \mid \kappa \in K, m \in M\}$

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

**Contexts in function f:**

$(?, f) \cdot \ldots,$

- merges all calling contexts to a same procedure, very coarse abstraction
- but usually quite efficient to compute
Application 2: context sensitive abstraction, \( k \)-CFA

Partially context sensitive abstraction (\( k \)-CFA)

\[
\mathbb{D}_0^\# = \{ \kappa \in \mathbb{K} \mid \text{length}(\kappa) \leq k \} \times \mathbb{L}
\]

\[
\gamma_0 : (\kappa, l) \mapsto \{(\kappa \cdot \kappa', l, m) \mid \kappa' \in \mathbb{K}, m \in \mathbb{M}\}
\]

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

**Contexts in function \( f \), in 2-CFA:**

\[
(l_0, f) \cdot \epsilon, (l_1, f) \cdot \epsilon, (l_4, f) \cdot (l_3, g) \cdot (?, g) \cdot ... , (l_4, f) \cdot (l_4, g) \cdot (?, g) \cdot ...
\]

- usually **intermediate** level of precision and efficiency
- can be applied to programs with **recursive procedures**
Application 3: partitioning by a boolean condition

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

Function guided memory states partitioning

We let:
- \( D_0^\# = \mathcal{P}(A) \) for some set \( A \), and \( \phi : \mathcal{M} \rightarrow A \)
- \( \gamma_0 \) be of the form \( (x^\# \in D_0^\#) \mapsto \{(l, m) \in \mathcal{S} \mid \phi(m) \in x^\#\} \)

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

Many choices for function \( \phi \) are possible:
- value of one or several variables (boolean or scalar)
- sign of a variable
- ...

Application 3: partitioning by a boolean condition

We assume:
- $\mathbb{X} = \mathbb{X}_{\text{bool}} \cup \mathbb{X}_{\text{int}}$, where $\mathbb{X}_{\text{bool}}$ (resp., $\mathbb{X}_{\text{int}}$) collects boolean (resp., integer) variables
- $\mathbb{X}_{\text{bool}} = \{b_0, \ldots, b_{k-1}\}$
- $\mathbb{X}_{\text{int}} = \{x_0, \ldots, x_{l-1}\}$

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

Boolean partitioning abstract domain

We apply the cardinal power abstraction, with a domain of partition defined by a function, with:
- $A = \mathbb{B}^k$
- $\phi(m) = (m(b_0), \ldots, m(b_{k-1}))$
- $(\mathbb{D}_1^\#, \sqsubseteq_1^\#, \gamma_1)$ an abstraction of $\mathcal{P}(\mathbb{V}_{\text{int}}^l)$
Application 3: example

With $X_{\text{bool}} = \{b_0, b_1\}$, $X_{\text{int}} = \{x, y\}$, we can express:

\[
\begin{align*}
\text{if } b_0 \land b_1 & \Rightarrow x_0 \in [-3, 0] \land y \in [0, 2] \\
\text{if } b_0 \land \neg b_1 & \Rightarrow x_0 \in [-3, 0] \land y \in [0, 2] \\
\text{if } \neg b_0 \land b_1 & \Rightarrow x_0 \in [0, 3] \land y \in [-2, 0] \\
\text{if } \neg b_0 \land \neg b_1 & \Rightarrow x_0 \in [0, 3] \land y \in [-2, 0]
\end{align*}
\]

- 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
- 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

```c
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 assume:

- $X = X_{\text{int}}$, i.e., all variables have \textit{integer} type
- $X_{\text{int}} = \{x_0, \ldots, x_{l-1}\}$

Thus, $M = X \rightarrow V \equiv V^l_{\text{int}}$

### Sign partitioning abstract domain

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

- $A = \{[< 0], [= 0], [> 0]\}$
- $\phi(m) = \begin{cases} 
  [< 0] & \text{if } x_0 < 0 \\
  [= 0] & \text{if } x_0 = 0 \\
  [> 0] & \text{if } x_0 > 0 
\end{cases}$
- $(D_1^\#, \sqsubseteq_1^\#, \gamma_1)$ an abstraction of $P(V^l_{\text{int}})$ (no need to abstract $x_0$ twice)
Application 3: example

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

```c
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)
y = x/s;
(1) (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
   - Control states partitioning and iteration techniques
   - Abstract interpretation with boolean partitioning
6. Trace partitioning
7. Conclusion
Computation of abstract semantics and partitioning

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

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

\[
(\mathcal{P}(\mathcal{L} \times \mathcal{M}), \subseteq) \xleftarrow{\alpha_{\text{part}}} \xrightarrow{\gamma_{\text{part}}} (\mathcal{D}_{\text{part}}^\#, \subseteq)
\]

where \( \mathcal{D}_{\text{part}}^\# = \mathcal{L} \rightarrow \mathcal{P}(\mathcal{M}) \) and:

\[
\begin{align*}
\alpha_{\text{part}} : \mathcal{P}(\mathcal{L} \times \mathcal{M}) & \rightarrow \mathcal{D}_{\text{part}}^\# \\
\mathcal{E} & \mapsto \lambda(l \in \mathcal{L}) \cdot \{ m \in \mathcal{M} \mid (l, m) \in \mathcal{E} \}
\end{align*}
\]

\[
\begin{align*}
\gamma_{\text{part}} : \mathcal{D}_{\text{part}}^\# & \rightarrow \mathcal{P}(\mathcal{L} \times \mathcal{M}) \\
\mathcal{X}^\# & \mapsto \{ (l, m) \in \mathcal{S} \mid m \in \mathcal{X}^\#(l) \}
\end{align*}
\]
Fixpoint form of a partitioned semantics

- We consider a transition system $S = (\mathcal{S}, \rightarrow, S_I)$
- The reachable states are computed as $[S]_R = \text{lfp}_{S_I} F$ where

$$F : \mathcal{P}(\mathcal{S}) \longrightarrow \mathcal{P}(\mathcal{S})$$

$$X \longmapsto \{s \in \mathcal{S} \mid \exists s' \in X, s' \rightarrow s\}$$

Semantic function over the partitioned system

We let $F_{\text{part}}$ be defined over $D_{\text{part}}^\#$ by:

$$F_{\text{part}} : \mathcal{D}_{\text{part}}^\# \longrightarrow \mathcal{D}_{\text{part}}^\#$$

$$X^\# \longmapsto \lambda (l \in L) \cdot \{m \in M \mid \exists l' \in L, \exists m' \in X^\#(l'), (l', m') \rightarrow (l, m)\}$$

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

$$\alpha_{\text{part}}([S]_R) = \text{lfp}_{\alpha_{\text{part}}(S_I)} F_{\text{part}}$$
Abstract equations form of a partitioned semantics

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

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

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

$$\alpha\text{part}(\mathcal{S}\mathcal{R})$$ is the least solution of the system

$$
\begin{align*}
\mathcal{E}_1 &= F_1(\mathcal{E}_1, \ldots, \mathcal{E}_s) \\
\vdots \\
\mathcal{E}_s &= F_s(\mathcal{E}_1, \ldots, \mathcal{E}_s)
\end{align*}
$$
How to compute an abstract invariant for a partitioned system described by a set of abstract equations?

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

- In practice $F_i$ depends **only on a few of its arguments**
  i.e., $E_k$ depends only on the predecessors of $l_k$ in the control flow graph of the program under consideration

- **Example** of a simple system of abstract equations:

  $\begin{cases}
  E_0 &= I \cup F_0(E_3) \\
  E_1 &= F_1(E_0) \\
  E_2 &= F_2(E_0) \\
  E_3 &= F_3(E_1, E_2)
  \end{cases}$

  where $\alpha_{\text{part}}(S_I) = (S_I, \bot, \bot, \bot)$ (i.e., init states are at point $l_0$)
Partitioned systems and fixpoint computation

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

\[
\begin{align*}
\mathcal{E}_0^\# &= (\emptyset, \bot, \bot, \bot) \\
\mathcal{E}_1^\# &= (\emptyset, F_1^\#(\emptyset), F_2^\#(\emptyset), \bot) \\
\mathcal{E}_2^\# &= (\emptyset, F_1^\#(\emptyset), F_2^\#(\emptyset), F_3^\#(F_1^\#(\emptyset), F_2^\#(\emptyset))) \\
\mathcal{E}_3^\# &= (\emptyset \sqcup F_0^\#(F_3^\#(F_1^\#(\emptyset), F_2^\#(\emptyset))), F_1^\#(\emptyset), F_2^\#(\emptyset), F_3^\#(F_1^\#(\emptyset), F_2^\#(\emptyset)))
\end{align*}
\]

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

**Fairness**

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

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

**Chaotic iterations**

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

$$X^\#_{n+1} = \lambda(l_i \in \mathbb{L}) \cdot \begin{cases} X^\#_n(l_i) & \text{if } i \neq k_n \\ X^\#_n(l_i) \sqcup F^\#(X^\#_n(l_1), \ldots, X^\#_n(l_s)) & \text{if } i = k_n \end{cases}$$
Chaotic iterations: soundness

**Soundness**

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

$$\lim_{n \in \mathbb{N}} (X^\#_n) = \alpha_{\text{part}}([\mathcal{S}]_{\mathcal{R}})$$

**Proof**: exercise

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

\[
\begin{align*}
\mathcal{E}_0^\# &= (\bot, \bot, \bot) \\
\mathcal{E}_1^\# &= (\bot, F_1^\#(\bot), \bot) \\
\mathcal{E}_2^\# &= (\bot, F_1^\#(\bot), F_2^\#(\bot), \bot) \\
\mathcal{E}_3^\# &= (\bot, F_1^\#(\bot), F_2^\#(\bot), F_3^\#(F_1^\#(\bot), F_2^\#(\bot))) \\
\mathcal{E}_4^\# &= (\bot \sqcup F_0^\#(F_3^\#(F_1^\#(\bot), F_2^\#(\bot))), F_1^\#(\bot), F_2^\#(\bot), F_3^\#(F_1^\#(\bot), F_2^\#(\bot)))
\end{align*}
\]
Chaotic iterations: work-list algorithm

Work-list algorithms

Principle:

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

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

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

**Pseudo code implementation**, with $\delta_{\ell,\ell'}^\#$ denoting the transfer function from $\ell$ to $\ell'$:

```plaintext
to_propagate ← \{initial states\}
E_{\text{initial}}^\# ← \top
while (to_propagate ≠ ∅) {
    pick $\ell ∈ to\_propagate$
    to_propagate = to_propagate \ {\ell}
    for ($\ell'$ successor of $\ell$ in the CFG) {
        $y^\# ← \delta_{\ell,\ell'}^\#(E_\ell^\#)$
        if ($\neg(y^\# ⊑ E_{\ell'}^\#)$) {
            $E_{\ell'}^\# = E_{\ell'}^\# ∪ y^\#
            to_propagate = to_propagate ∪ \{\ell'\}
        }
    }
}
```

Xavier Rival (INRIA, ENS, CNRS)
Selection of a set of widening points for a partitioned system

- We compose an abstraction $D^\#_{\text{num}}$, with concretization $\gamma_{\text{num}}: D^\#_{\text{num}} \rightarrow \mathcal{P}(M)$, that may not satisfy ascending chain condition.
- We assume $D^\#_{\text{num}}$ provides widening operator $\nabla$.

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

**Enforcing termination of chaotic iterates**

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

\[
X^\#_{n+1} = \lambda(l_i \in \mathbb{L}) \cdot \begin{cases} 
X^\#_n(l_i) & \text{if } i \neq k_n \\
X^\#_n(l_i) \cup F^\#(X^\#_n(l_1), \ldots, X^\#_n(l_s)) & \text{if } i = k_n \land l_i \notin K_\nabla \\
X^\#_n(l_i) \nabla F^\#(X^\#_n(l_1), \ldots, X^\#_n(l_s)) & \text{if } i = k_n \land l_i \in K_\nabla
\end{cases}
\]
State partitioning  |  Control states partitioning and iteration techniques

Selection of a set of widening points for a partitioned system

**Soundness and termination**

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

\[
\exists n_0 \in \mathbb{N}, \left\{ \forall n \geq n_0, \ X^n_\# = X^n_\# \right\} \subseteq \gamma_{\text{part}}(X_{n_0})
\]

**Proof**: exercise

**Algorithm for iteration with widening**: exercise
Outline

1 Introduction

2 Imprecisions in convex abstractions

3 Disjunctive completion

4 Cardinal power and partitioning abstractions

5 State partitioning
   - Definition and examples
   - Control states partitioning and iteration techniques
   - Abstract interpretation with boolean partitioning

6 Trace partitioning

7 Conclusion
Computation of abstract semantics and partitioning

We now compose two forms of partitioning

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

Thus, the abstract domain is of the form

\[ \mathcal{L} \rightarrow (\forall^k_{\text{bool}} \rightarrow D^\#_0) \]

- we could do a partitioning by \( \mathcal{L} \times \forall^k_{\text{bool}} \)
- yet, it is not practical, as transitions from “boolean states” are not know before the analysis
- data types skeleton:

  ```plaintext
type abs0 = ... (* abstract elements of \( D^\#_0 \) *)
type abs_state = ... (* boolean trees with elements of type abs0 at the leaves *)
type abs_cp = (labels, abs_state) Map.t
```
Abstract operations

Transfer functions:

- we seek, for all pair \( \ell, \ell' \in \mathbb{L} \) for an approximation \( \delta_{\ell, \ell}' \) of

\[
\delta_{\ell, \ell} : \mathbb{M} \longrightarrow \mathcal{P}(\mathbb{M}) \\
\quad m \longmapsto \{ m' \in \mathbb{M} \mid (\ell, m) \rightarrow (\ell', m') \}
\]

- that includes
  - scalar assignment, e.g., \( x = 1 - x \);
  - scalar test, e.g., \( \text{if}(x \geq 8) \ldots \)
  - boolean test, e.g., \( \text{if}(\neg b_1) \ldots \)
  - mixed assignment, e.g., \( b_0 = x \leq 7 \)

Lattice operations: inclusion check, join, widening
Transfer functions: scalar assignment

**Assignment** \( \ell_0 : x = e; \ell_1 \) affecting **only integer variables** (i.e., \( e \) depends only on \( x_0, \ldots, x_l \)):

- **example**: \( x = 1 - x \);
- **concrete transition** \( \delta_{\ell_0, \ell_1} \) defined by
  \[
  \delta_{\ell_0, \ell_1}(m) = \{ m[x \leftarrow \llbracket e \rrbracket(m)] \}
  \]
- the values of the boolean variables are unchanged
  thus the partitions are preserved (**pointwise** transfer function):

\[
\text{assign}_\rightarrow(x, e, X^\#) = \lambda(z^\# \in \mathbb{D}_0^\#) \cdot \text{assign}_1(x, e, X^\#(z^\#))
\]

**Soundness**

If \( \text{assign}_1 \) is sound, so is \( \text{assign}_\rightarrow \), in the sense that:

\[
\forall X^\# \in \mathbb{D}_{cp}^\#, \forall m \in \gamma_{cp}(X^\#), \ m[x \leftarrow \llbracket e \rrbracket(m)] \in \gamma_{cp}(\text{assign}_\rightarrow(x, e, X^\#))
\]
Transfer functions: scalar assignment, example

- **abstract precondition:**

\[
\begin{align*}
&\{ \quad b \Rightarrow x \geq 0 \\
&\quad \land \quad \neg b \Rightarrow x \leq 0 \quad \}
\end{align*}
\]

- **statement:**

\[
x = 1 - x;
\]

- **abstract post-condition:**

\[
\text{assign} \rightarrow (x, 1 - x, \{ \quad b \Rightarrow x \geq 0 \\
\quad \land \quad \neg b \Rightarrow x \leq 0 \quad \})
\]

\[
= \{ \quad b \Rightarrow x \geq 8 \\
\quad \land \quad \neg b \Rightarrow \top \quad \}
\]
Transfer functions: scalar test

**Condition test** $l_0 : \text{if}(c) \{ l_1 : \ldots \}$ affecting *only* scalar variables (i.e., $c$ depends only on $x_0, \ldots, x_l$):

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

\[
\delta_{l_0, l_1}(m) = \begin{cases} 
\{m\} & \text{if } \llbracket c \rrbracket(m) = \text{TRUE} \\
\emptyset & \text{if } \llbracket c \rrbracket(m) = \text{FALSE} 
\end{cases}
\]

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

\[
\text{test}_\rightarrow(c, X^\#) = \lambda(z^\# \in D^\#_0) \cdot \text{test}_1(c, X^\#(z^\#))
\]

**Soundness**

If $\text{test}_1$ is sound, so is $\text{test}_\rightarrow$, in the sense that:

\[
\forall X^\# \in D^\#_{cp}, \forall m \in \gamma_{cp}(X^\#), \llbracket c \rrbracket(m) = \text{TRUE} \implies m \in \gamma_{cp}(\text{test}_\rightarrow(x, e, X^\#))
\]
Transfer functions: scalar test, example

- **abstract pre-condition:**

\[
\begin{align*}
\{ & b \Rightarrow x \geq 0 \\
\wedge & \neg b \Rightarrow x \leq 0 \}
\end{align*}
\]

- **statement:**

\[
\text{if}(x \geq 8) \ldots
\]

- **abstract post-condition:**

\[
\text{test} \rightarrow \left( x \geq 8, \left\{ \begin{array}{c} b \Rightarrow x \geq 0 \\ \wedge \neg b \Rightarrow x \leq 0 \end{array} \right\} \right) = \left\{ \begin{array}{c} b \Rightarrow x \geq 8 \\ \wedge \neg b \Rightarrow \perp \end{array} \right\}
\]
Transfer functions: boolean condition test

**Condition test** $\ell_0 : \text{if}(c)\{\ell_1 : \ldots\}$ affecting **only boolean variables** (i.e., $c$ depends only on $b_0, \ldots, b_k$):

- **example**: $\text{if}(\neg b_1)\ldots$
- then, we simply need to filter the boolean partitions **satisfying** $c$:

$$\text{test}_\rightarrow(c, X^\#) = \lambda(z^\# \in \mathbb{D}^\#_0). \begin{cases} X^\#(z^\#) & \text{if } \text{test}_0(c, X^\#(z^\#)) \neq \bot_0 \\ \bot_1 & \text{otherwise} \end{cases}$$

### Soundness
If $\text{test}_0$ is sound, so is $\text{test}_\rightarrow$, in the sense that:

$$\forall X^\# \in \mathbb{D}^\#_{cp}, \forall m \in \gamma_{cp}(X^\#), \llbracket c \rrbracket(m) = \text{TRUE} \implies m \in \gamma_{cp}(\text{test}_\rightarrow(x, e, X^\#))$$
Transfer functions: boolean condition test, example

- abstract pre-condition:
  \[
  \begin{cases}
  b_0 \land b_1 \Rightarrow 15 \leq x \\
  \land b_0 \land \neg b_1 \Rightarrow 9 \leq x \leq 14 \\
  \land \neg b_0 \land b_1 \Rightarrow 6 \leq x \leq 8 \\
  \land \neg b_0 \land \neg b_1 \Rightarrow x \leq 5
  \end{cases}
  \]

- statement: \( \text{if}(\neg b_1) \ldots \)

- abstract post-condition:
  \[
  \text{test} \rightarrow \left( \neg b_1, \begin{cases}
  b_0 \land b_1 \Rightarrow \bot_1 \\
  \land b_0 \land \neg b_1 \Rightarrow 9 \leq x \leq 14 \\
  \land \neg b_0 \land b_1 \Rightarrow \bot_1 \\
  \land \neg b_0 \land \neg b_1 \Rightarrow x \leq 5
  \end{cases} \right)
  \]
Transfer functions: mixed assignment

**Assignment** \( l_0 : b = e; l_1 \) to a **boolean variable**, where the right hand side contains **only integer variables** (i.e., \( e \) depends only on \( x_0, \ldots, x_l \)):

- **example**: \( b_0 = x \leq 7 \)
- let \( z^\# \in \mathcal{D}_0^\# \), such that \( z^\#(b) = \text{TRUE} \)

\[ \text{assign} \rightarrow (b, e[x_0, \ldots, x_i], X^\#)(z^\#) \] should account for all states where \( b \) becomes true, other boolean variables remaining unchanged:

\[ \text{assign} \rightarrow (b, e, X^\#)(z^\#) = \left\{ \begin{array}{ll} \text{test}_1(e, X^\#(z^\#)) \\ \sqcup_1 \text{test}_1(e, X^\#(z^\#[b \leftarrow \text{FALSE}]))) \end{array} \right. \]

- same computation for cases where \( z^\#(b) = \text{FALSE} \)

**The partitions get modified** (this is a **costly step**, involving join)
Transfer functions: mixed assignment, example

- **abstract pre-condition:**
  \[
  \begin{aligned}
  &b_0 \land b_1 \quad \Rightarrow \quad 15 \leq x \\
  &\land b_0 \land \neg b_1 \quad \Rightarrow \quad 9 \leq x \leq 14 \\
  &\land \neg b_0 \land b_1 \quad \Rightarrow \quad 6 \leq x \leq 8 \\
  &\land \neg b_0 \land \neg b_1 \quad \Rightarrow \quad x \leq 5
  \end{aligned}
  \]

- **statement:** \( b_0 = x \leq 7 \)

- **abstract post-condition:**
  \[
  \begin{aligned}
  &b_0 \land b_1 \quad \Rightarrow \quad 15 \leq x \\
  &\land b_0 \land \neg b_1 \quad \Rightarrow \quad 9 \leq x \leq 14 \\
  &\land \neg b_0 \land b_1 \quad \Rightarrow \quad 6 \leq x \leq 8 \\
  &\land \neg b_0 \land \neg b_1 \quad \Rightarrow \quad x \leq 5
  \end{aligned}
  \]

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
- These relations are expensive:
  1. Partitioning with respect to $N$ boolean variables translates into a $2^N$ space cost factor
  2. After assignments, partitions need be recomputed
- Packing addresses the first issue:
  - select groups of variables for which relations would be useful
  - can be based on syntactic or semantic criteria
  Whatever the packs, the transfer functions will produce a sound result (but possibly not the most precise one)
- How to alleviate the second issue?
Outline

1. Introduction
2. Imprecisions in convex abstractions
3. Disjunctive completion
4. Cardinal power and partitioning abstractions
5. State partitioning
6. Trace partitioning
7. Conclusion
**Assumptions:** we start from a trace semantics and use an abstraction of execution history for partitioning

- **concrete domain:** \( D = \mathcal{P}(S^*) \)
- **left side abstraction** \( \gamma_0 : D^\#_0 \rightarrow D \): a trace abstraction
- **right side abstraction**, as a composition of two abstractions:
  - the **final state abstraction** defined by \( (D^\#_1, \sqsubseteq^\#_1) = (\mathcal{P}(S), \subseteq) \) and:
    \[
    \gamma_1 : D^\#_1 \rightarrow \mathcal{P}(S^*) \quad M \mapsto \{ (s_0, \ldots, s_k, (l, m)) | m \in M, l \in L, s_0, \ldots, s_k \in S \}
    \]
  - a **store abstraction** applied to the traces final memory state
    \[
    \gamma_2 : D^\#_2 \rightarrow D^\#_1
    \]

**Cardinal power abstraction** defined by the above, and by an abstraction of sets of traces \( \gamma_0 : D^\#_0 \rightarrow \mathcal{P}(S^*) \)
Application 1: partitioning by control states

Flow sensitive abstraction

- We let $D_0^{\#} = \mathbb{L}$
- Concretization is defined by:

\[
\gamma_0 : D_0^{\#} \longrightarrow \mathcal{P}(S^*) \\
\ell \longmapsto S^* \cdot (\{\ell\} \times M)
\]

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

Trace partitioning is more general than state partitioning

It can also express

- context-sensitivity, partial context sensitivity
- partitioning guided by a boolean condition...
**Application 2: partitioning guided by a condition**

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

\[
\begin{align*}
\ell_0 & : \text{if}(c) \\
\ell_1 & : \ldots \\
\ell_2 & : \text{else} \\
\ell_3 & : \ldots \\
\ell_4 & : \\
\ell_5 & : \ldots
\end{align*}
\]

**Domain of partitions**

The partitions are defined by \( \mathbb{D}_0^\# = \{ \text{if}_t, \text{if}_f, \top \} \) and:

\[
\begin{align*}
\gamma_0 : \ \text{if}_t & \mapsto \{ \langle (\ell_0, m), (\ell_1, m'), \ldots \rangle \mid m \in M, m' \in M \} \\
\gamma_0 : \ \text{if}_f & \mapsto \{ \langle (\ell_0, m), (\ell_3, m'), \ldots \rangle \mid m \in M, m' \in M \} \\
\gamma_0 : \ \top & \mapsto S^*
\end{align*}
\]

**Application**: discriminate the executions depending on the branch they visited
Application 2: partitioning guided by a condition

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

```plaintext
int x ∈ \( \mathbb{Z} \);
int s;
int y;
if(x ≥ 0) {
    if \( t \Rightarrow (0 ≤ x) \land if \Rightarrow \bot \)
    s = 1;
    if \( t \Rightarrow (0 ≤ x \land s = 1) \land if \Rightarrow \bot \)
} else {
    if \( f \Rightarrow (x < 0) \land if \Rightarrow \bot \)
    s = -1;
    if \( f \Rightarrow (x < 0 \land s = -1) \land if \Rightarrow \bot \)
}
\{
    if \( t \Rightarrow (0 ≤ x \land s = 1) \land if \Rightarrow (x < 0 \land s = -1) \)
\}
y = x/s;
```
Application 3: partitioning guided by a loop

We consider a program with a conditional statement:

\[
\begin{align*}
\ell_0 & : \text{while}(c) \{ \\
\ell_1 & : \ldots \\
\ell_2 & : \} \\
\ell_3 & : \ldots
\end{align*}
\]

Domain of partitions

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

\[
D_{\#}^0 = \{ \text{loop}_0, \text{loop}_1, \ldots, \text{loop}_k, \top \}
\]

and:

\[
\begin{align*}
\gamma_0 : \text{loop}_i \mapsto & \text{traces that visit } \ell_1 \text{ } i \text{ times} \\
\top \mapsto & \mathcal{S}^*
\end{align*}
\]

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 \( x \)

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

\[
y = t_c[i] \times (x - t_x[i]) + t_y[i]
\]
Application 4: partitioning guided by the value of a variable

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

```plaintext
int x; ...; l: ...
```

### Domain of partitions: partitioning by the value of a variable

For a given $E \subseteq \mathbb{V}_{\text{int}}$ finite set of integer values, the partitions are defined by $D_0^\# = \{\text{val}_i \mid i \in E\} \cup \{\top\}$ and:

$\gamma_0 : \text{val}_k \mapsto \{\langle \ldots, (l, m), \ldots \rangle \mid m(x) = k\}$

$\top \mapsto \mathbb{S}^*$

### 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 $D_0^\# = \{\text{var}_{\nu^\#} \mid \nu^\# \in V^\#\}$ and:

$\gamma_0 : \text{val}_{\nu^\#} \mapsto \{\langle \ldots, (l, m), \ldots \rangle \mid m(x) \in \text{var}_{\nu^\#}\}$
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

```c
bool b0, b1;
int x, y; (uninitialized)

if (b0 && b1) {
    (x < 0@1 ⇒ ⊤) ∧ (x = 0@1 ⇒ ⊤) ∧ (x > 0@1 ⇒ ⊤)
    y = 0;
    (x < 0@1 ⇒ ⊥) ∧ (x = 0@1 ⇒ b0 ∧ b1) ∧ (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)
}
```
Trace partitioning induced by a refined transition system

Let us consider the partitions induced by a condition:

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

Thus, we can view this form of trace partitioning as the use of a refined control flow graph.
We now **formalize this intuition:**

- we **augment** control states with **partitioning tokens**: \( L' = L \times D^\#_0 \)
- and let \( S' = L' \times M \)
- let \( \rightarrow' \subseteq S' \times S' \) be an **extended transition relation**

**Partition of a transition system**

System \( S' = (S', \rightarrow', S'_I) \) is a **partition** of transition system \( S = (S, \rightarrow, S_I) \) (and note \( S' \prec S \)) if and only if

- \( \forall (l, m) \in S_I, \exists \text{tok} \in D^\#_0, ((l, \text{tok}), m) \in S'_I \)
- \( \forall (l, m), (l', m') \in S, \forall \text{tok} \in D^\#_0, (l, m) \rightarrow (l', m') \Rightarrow \exists \text{tok}' \in D^\#_0, ((l, \text{tok}), m) \rightarrow ((l', \text{tok}'), m') \)

Then:

\[ \forall \langle (l_0, m_0), \ldots, (l_n, m_n) \rangle \in [S]_R, \exists \text{tok}_0, \ldots, \text{tok}_n \in D^\#_0, \langle ((l_0, \text{tok}_0), m_0), \ldots, ((l_n, \text{tok}_n), m_n) \rangle \in [S']_R, \]
Trace partitioning induced by a refined transition system

- we assume \((S', \to', S'_I) \prec (S, \to, S_I)\)
- erasure function: \(\Psi: (S')^* \to S^*\) removes the tokens

**Definition of a trace partitioning**

The abstraction defining partitions is defined by:

\[
\gamma_0 : D_0^\# \longrightarrow P(S^*)
\]

\[
tok \longrightarrow \{ \sigma \in S^* \mid \exists\sigma' = \langle \ldots, ((l, tok), m) \rangle \in (S')^*, \ \Psi(\sigma') = \sigma \}\]

- not all instances of trace partitionings can be expressed that way
- ... but many interesting instances can
Trace partitioning induced by a refined transition system

Example of the partitioning guided by a condition:

\[
\begin{align*}
\ell_0 & \text{ if}(x < 0) \{ \\
\ell_1 & \text{ } s = -1; \\
\ell_2 & \text{ } } \text{ else } \{ \\
\ell_3 & \text{ } s = 1; \\
\ell_4 & \} \\
\ell_5 & \text{ } y = x/s; \\
\ell_6 & \text{ } \ldots
\end{align*}
\]

- each system induces a partitioning, with different merging points:
  \[ P_1 \prec P_0 \quad P_2 \prec P_1 \]
- these systems induce hierarchy of refining control structures
  \[ P_2 \prec P_1 \]
- this approach also applies to:
  - partitioning induced by a loop
  - partitioning induced by the value of a variable at a given point...
Abstract interpretation of a partitioned transition system

- let $S = (S, \rightarrow, S_I)$, and a refining system $S' = (S', \rightarrow', S'_I)$, with $S = L \times M$, $S' = (L \times D^0) \times M$

- **transfer functions of $S'$**: 
  $\delta_{\ell, \ell'} : (D^0 \rightarrow D^1) \rightarrow (D^0 \rightarrow D^1)$ over-approximating $\rightarrow'$

**Partition irrelevant transfer function**

$\ell, \ell'$ induces a **partition irrelevant transfer function** if and only if:

$$\forall tok, tok' \in D^0_0, \forall m, m' \in M, 
((\ell, tok), m) \rightarrow' ((\ell', tok'), m') \implies tok = tok'$$

- partition irrelevant transfer functions: **pointwise operators of $D^1_1$** for our examples of partitioning: this is the **most common case**

- **other transfer functions**: usually for partition **creation** or **fusion** or **simple composition** of a creation / fusion + partition irrelevant t.f.
Transfer functions: example

\begin{verbatim}
int x ∈ ℤ;
int s;
int y;
if(x ≥ 0){
    \begin{align*}
        & \text{if}_t \Rightarrow (0 ≤ x) \land \text{if}_f \Rightarrow ⊥ \\
        & s = 1;
    \end{align*}
    \begin{align*}
        & \text{if}_t \Rightarrow (0 ≤ x \land s = 1) \land \text{if}_f \Rightarrow ⊥ \\
        & s = 1;
    \end{align*}
} else {
    \begin{align*}
        & \text{if}_f \Rightarrow (x < 0) \land \text{if}_t \Rightarrow ⊥ \\
        & s = -1;
    \end{align*}
    \begin{align*}
        & \text{if}_f \Rightarrow (x < 0 \land s = -1) \land \text{if}_t \Rightarrow ⊥ \\
        & s = -1;
    \end{align*}
}\end{verbatim}

\begin{align*}
    & y = x / s;
    \begin{align*}
        & \text{if}_t \Rightarrow (0 ≤ x \land s = 1 \land 0 ≤ y) \\
        & \land \text{if}_f \Rightarrow (x < 0 \land s = -1 \land 0 < y)
    \end{align*}
\end{align*}

... \Rightarrow s ∈ [-1, 1] \land 0 ≤ y

In general, partitions are rarely modified (only some branching points)
Analysis of an if statement, with partitioning

\[
\begin{align*}
\ell_0 : & \quad \text{if}(c) \{ \\
\ell_1 : & \quad \ldots \\
\ell_2 : & \quad } \text{else} \{ \\
\ell_3 : & \quad \ldots \\
\ell_4 : & \quad } \\
\ell_5 : & \quad \ldots
\end{align*}
\]

\[
\begin{align*}
\delta^\sharp_{\ell_0,\ell_1}(X^\#) &= \text{[if}_t \mapsto test(c, \sqcup X^\#(\ell_0)(t)), \top \mapsto \bot] \\
\delta^\sharp_{\ell_0,\ell_3}(X^\#) &= \text{[if}_t \mapsto test(\neg c, \sqcup_t X^\#(\ell_0)(t)), \top \mapsto \bot] \\
\delta^\sharp_{\ell_2,\ell_5}(X^\#) &= X^\# \\
\delta^\sharp_{\ell_4,\ell_5}(X^\#) &= X^\#
\end{align*}
\]

- in the body of the condition: either if\(_t\) or if\(_f\)
- effect at point \(\ell_5\): \textbf{both if}_t \textbf{ and if}_f \textbf{ exist}
Transfer functions: partition fusion

When partitions are not useful anymore, they can be merged

\[ \delta_{\lambda_0, \lambda_1}^*(X^\#) = [\_ \mapsto \sqcup_t X^\#(\lambda_0)(t)] \]

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

How are the partitions chosen?

Static partitioning

- a fixed partitioning abstraction $D^#_0, \gamma_0$ is fixed before the analysis
- usually $D^#_0, \gamma_0$ are chosen by a pre-analysis

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

Dynamic partitioning

- the partitioning abstraction $D^#_0, \gamma_0$ 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 $D^#$ and computes $(D^#_0, \gamma_0)$ 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
Conclusion

Adding disjunctions in static analyses

- **Disjunctive completion** is too expensive in practice
- The **cardinal power abstraction** expresses collections of implications between abstract facts in two abstract domains
- **State partitioning** and **trace partitioning** are particular cases of cardinal power abstraction
- State partitioning is **easier** to use when the criteria for partitioning can be easily expressed at the state level
- Trace partitioning is **more expressive** in general; it can also allow the use of **simpler partitioning criteria**, with less “re-partitioning”
Assignment: paper reading

Abstract interpretation by dynamic partitioning,
François Bourdoncle,
Extended report available at: