The structure of an abstract interpreter
Static Analysis by Abstract Interpretation

Xavier Rival

East China Normal University

November, 29th. 2013
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
  - abstract domain combinator: functor
Example: product abstraction

For this example,

- we assume the concrete domain is \((P(M), \subseteq)\)
  where \(M = X \rightarrow V\)
- we require an abstract domain \(D^\#\) to provide
  - a concretization function \(\gamma : D^\# \rightarrow P(M)\)
  - an element \(\bot\) with empty concretization \(\gamma(\bot) = \emptyset\)

Product combinator

Given abstract domains \((D^\#_0, \gamma_0, \bot_0)\) and \((D^\#_1, \gamma_1, \bot_1)\), the product abstraction is \((D^\#_\times, \gamma_\times, \bot_\times)\) where:

- \(D^\#_\times = D^\#_0 \times D^\#_1\)
- \(\gamma_\times(x^\#_0, x^\#_1) = \gamma_0(x^\#_0) \cap \gamma_1(x^\#_1)\)
- \(\bot_\times = (\bot_0, \bot_1)\)
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:

Such imprecisions may impact analysis results
Non convex abstractions

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

**Congruences:**

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

**Signs:**

- $0 \notin \gamma([\neq 0])$
Example 1: verification problem

```c
bool b0, b1;
int x, y; // (uninitialized)
b0 = x >= 0;
b1 = x <= 0;
if(b0 && b1){
    y = 0;
} else {
    y = 100/x;
}
```

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

How to verify the division operation?

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

- Signs, congruences do not help:
in the concrete, \( x \) may take any value but 0
Imprecisions in convex abstractions

Example 1: Hoare style program proof

```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 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: Hoare style program proof

```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
Disjunctive completion

Distributive abstract domain

Principle:
1. consider concrete domain \((D, \subseteq)\), with lower upper bound operator \(\sqcup\)
2. start with an abstract domain \((D^\#, \sqsubseteq^\#)\) with concretization \(\gamma : D^\# \rightarrow D\)
3. build a domain containing all the disjunctions of elements of \(D^\#\)

Definition: distributive abstract domain

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

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

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 \(\gamma([0, 10]) \cup \gamma([12, 20])\)
Disjunctive completion

Definition

**Definition: disjunctive completion**

The disjunctive completion of abstract domain $(\mathcal{D}^\#, \sqsubseteq^\#)$ with concretization function $\gamma : \mathcal{D}^\# \rightarrow \mathcal{D}$ is the smallest abstract domain $(\mathcal{D}_\lor^\#, \sqsubseteq_\lor^\#)$ with concretization function $\gamma_\lor : \mathcal{D}_\lor^\# \rightarrow \mathcal{D}$ such that:

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

Building a disjunctive completion domain:

- start with $\mathcal{D}_\lor^\# = \mathcal{D}^\#$
- for all set $\mathcal{E} \subseteq \mathcal{D}^\#$ such that there is no $x^\# \in \mathcal{D}^\#$, such that $\gamma(x^\#) = \bigsqcup_{y^\# \in \mathcal{E}} \gamma(y^\#)$, add $\mathcal{E}$ to $\mathcal{D}_\lor^\#$, and extend $\gamma_\lor$ by

$$\gamma_\lor(\mathcal{E}) = \bigsqcup_{y^\# \in \mathcal{E}} \gamma(y^\#)$$
Example 1: completion of signs

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

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 $(\mathbb{D}^\#, \subseteq)$ be defined by:

- $\perp$  
- $\{ -2 \}$  
- $\{ -1 \}$  
- $\{ 0 \}$  
- $\{ 1 \}$  
- $\{ 2 \}$  
- $\top$

Then, the disjunctive completion is the powerset:

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

\[
\begin{align*}
\gamma: & \quad \perp \quad \mapsto \quad \emptyset \\
& \{k\} \quad \mapsto \quad \{k\} \\
& \top \quad \mapsto \quad \mathbb{Z}
\end{align*}
\]
Example 3: completion of intervals

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

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

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

- $\mathbb{D}_\vee^\#$ 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 $(\mathbb{D}_\vee^\#)^n$
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);
```
Limitations of disjunctive completion

- **Combinatorial explosion:**
  - if $D'$ is infinite, $D'_V$ may have elements that **cannot be represented**
  - even when $D'$ is finite, $D'_V$ may be **huge**
    - in the worst case, if $D'$ has $n$ elements, $D'_V$ 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: bound the numbers of disjuncts
    - i.e., the size of the sets added to the base 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
disjuncts 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
Cardinal power abstraction

Definition

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

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

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

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

\[
\gamma_\# : \mathbb{D}_\# \longrightarrow \mathbb{D} \\
X^\# \longmapsto \{ y \in \mathcal{E} \mid \forall z^\# \in \mathbb{D}_0^\#, \ y \in \gamma_0(z^\#) \implies y \in \gamma_1(X^\#(z^\#))\}
\]

We sometimes denote it by \(\mathbb{D}_0^\# \Rightarrow \mathbb{D}_1^\#, \ \gamma_{\mathbb{D}_0^\# \Rightarrow \mathbb{D}_1^\#}\).
Intuition: we can express properties of the form

\[
\begin{align*}
& p_0 \Rightarrow p'_0 \\
\land & p_1 \Rightarrow p'_1 \\
& \vdots \\
\land & p_n \Rightarrow p'_n
\end{align*}
\]

Two independent choices:

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

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

A few example abstract values:
- $[0, 8]$ is expressed by:

\[
\begin{array}{c}
\bot_0 & \mapsto & \bot_1 \\
\leq 0 & \mapsto & \bot_1 \\
\equiv 0 & \mapsto & [0, 0] \\
\geq 0 & \mapsto & [1, 8] \\
\top_0 & \mapsto & [0, 8]
\end{array}
\]

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

\[
\begin{array}{c}
\bot_0 & \mapsto & \bot_1 \\
\leq 0 & \mapsto & [-10, -3] \\
\equiv 0 & \mapsto & \bot_1 \\
\geq 0 & \mapsto & [7, 10] \\
\top_0 & \mapsto & [-10, 10]
\end{array}
\]
Reduction (1): tightening disjunctions

- 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 let:

$$X^\# = \begin{cases} \perp_0 & \mapsto -\perp_0 \\ [< 0] & \mapsto [-5, -1] \\ [= 0] & \mapsto [0, 0] \\ [> 0] & \mapsto [1, 5] \\ T_0 & \mapsto [-10, 10] \end{cases}$$

$$Y^\# = \begin{cases} \perp_0 & \mapsto -\perp_1 \\ [< 0] & \mapsto [-5, -1] \\ [= 0] & \mapsto [0, 0] \\ [> 0] & \mapsto [1, 5] \\ T_0 & \mapsto [-5, 5] \end{cases}$$

Then, $\gamma_\rightarrow(X^\#) = \gamma_\rightarrow(Y^\#)$

$\gamma_0([< 0]) \cup \gamma_0([= 0]) \cup \gamma([> 0]) = \gamma(T_0)$

but $\gamma_0(X^\#([< 0])) \cup \gamma_0(X^\#([= 0])) \cup \gamma(X^\#([> 0])) \subset \gamma(X^\#(T_0))$

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

- $\sqcup \{\gamma_0(z^\#) \mid z^\# \in \mathcal{E}\} = \gamma_0(\sqcup \{z^\# \mid z^\# \in \mathcal{E}\})$
- $\exists y^\#, \sqcup \{\gamma_1(X^\#(z^\#)) \mid z^\# \in \mathcal{E}\} \subseteq \gamma_1(y^\#) \subset \gamma_1(X^\#(\sqcup \{z^\# \mid z^\# \in \mathcal{E}\}))$
Reduction (2): relation between the two domains

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

We let:

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

Then, $\gamma(X^\#) = \gamma(Y^\#) = \gamma(Z^\#) = \emptyset$

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

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

- If \( D^\#_0 \) has \( N \) elements, then an abstract value in \( D^\#_1 \) requires \( N \) elements of \( D^\#_1 \).
- If \( D^\#_0 \) is infinite, and \( D^\#_1 \) is non-trivial, then \( D^\#_1 \) has elements that cannot be represented.
- The 1st reduction shows it is unnecessary to represent bindings for all elements of \( D^\#_0 \).
- The 2nd reduction shows it is unnecessary to represent a binding for \( \bot_0 \).

Compact representation

Reduced cardinal power of \( D^\#_0 \) and \( D^\#_1 \) can be represented by considering only a subset \( C \subseteq D^\#_0 \) where

\[
\forall x^\# \in D^\#_0, \ \exists E \subseteq C, \ \gamma_0(x^\#) = \bigcup \{ \gamma_0(y^\#) | y^\# \in E \}
\]

- In particular, we should let \( \bot_0 \not\in C \).
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 $C = \{[< 0], [= 0], [> 0]\}$; then:

- $[0, 8]$ is expressed by: $\begin{cases} [< 0] \mapsto \bot_1 \\ [= 0] \mapsto [0, 0] \\ [> 0] \mapsto [1, 8] \end{cases}$

- $[−10, −3] \uplus [7, 10]$ 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, $\bot_{\downarrow} = \lambda (z^\# \in D_0) \cdot \bot_1$ is the infimum of $D_{\downarrow}$

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

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

We assume three abstractions

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

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

\[
\gamma : (\mathbb{D}_0^# \Rightarrow \mathbb{D}_2^#) \quad \rightarrow \quad (\mathbb{D}_0^# \Rightarrow \mathbb{D}_1^#)
\]

\[
X^# \quad \mapsto \quad \lambda(z^# \in \mathbb{D}_0^#) \cdot \gamma(X^#(z^#))
\]

Applications:

- start with \(\mathbb{D}_1^#\) 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 \( \subseteq \subseteq \)
- \((\mathbb{D}_0^\#, \sqsubseteq_0^\#)\) 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, \([-10, -3] \cup [7, 10]\) 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 \(\mathbb{D}_0^\# \Rightarrow \mathbb{D}_2^\#\), \[
\begin{align*}
[< 0] & \mapsto [-10, -3] \\
[= 0] & \mapsto \bot \\
[> 0] & \mapsto [7, 10]
\end{align*}
\]
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
Definition

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

- $S = L \times M$
- $M = X \rightarrow V$

State partitioning

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

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

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

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

$$\forall e, e' \in \mathcal{E}, \ e \neq e' \implies e \cap e' = \emptyset$$

$$S = \bigcup \mathcal{E}$$

It induces the partitioning abstraction

$$\mathbb{D}^\#_0 = \mathcal{E}$$

$$\gamma_0 : e \mapsto e$$
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:

- $D^0_0 = L$
- $\gamma_0 : l \mapsto \{l\} \times M$

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

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

$$\gamma(X^\#) = \{ s \in S \mid \forall x \in D^\#_0, s \in \gamma_0(x) \implies s \in \gamma_1(X^\#(x)) \}$$

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

- after this abstraction step, $D^\#_1$ may simply 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:

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

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)
We consider the language with procedures

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

\[
\begin{align*}
\kappa & \in K & \text{calling contexts} \\
\kappa & := \epsilon & \text{empty call stack} \\
& \ | \ (f, \ell) \cdot \kappa & \text{call to } f \text{ from stack } \kappa \text{ at point } \ell
\end{align*}
\]

We assume that inside each function, we use the flow sensitive abstraction
Application 2: context sensitive abstraction

Various level of sensitivity can be defined by partitioning:

**Fully context sensitive abstraction (∞-CFA)**
- \( D_0^\# = K \times L \)
- \( \gamma_0 : (\kappa, l) \mapsto \{ (\kappa, l, m) \mid m \in M \} \)

**Partially context sensitive abstraction (k-CFA)**
- \( D_0^\# = \{ \kappa \in K \mid \text{length} (\kappa) \leq k \} \times L \)
- \( \gamma_0 : (\kappa, l) \mapsto \{ (\kappa \cdot \kappa', l, m) \mid \kappa' \in K, m \in M \} \)

**Non context sensitive abstraction (0-CFA)**
- \( D_0^\# = L \)
- \( \gamma_0 : l \mapsto \{ (\kappa, l, m) \mid \kappa \in K, m \in M \} \)
Application 2: context sensitive abstraction

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

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

k-CFA:
- 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 $\phi: M \rightarrow A$
- $\gamma_0$ be of the form $(x^\# \in D_0^\#) \mapsto \{(l, m) \in S \mid \phi(m) \in x^\#\}$

Many choices of functions are possible:
- value of one or several variables (boolean or scalar)
- sign of a variable
- ...
Application 3: partitioning by a boolean condition

We assume:

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

Thus, \( \mathcal{M} = X \to V \equiv (X_{\text{bool}} \to V_{\text{bool}}) \times (X_{\text{int}} \to V_{\text{int}}) \equiv V_{\text{bool}}^k \times V_{\text{int}}^l \)

Boolean partitioning abstract domain

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

- \( A = \mathbb{B}^k \)
- \( \phi(m) = (m(b_0), \ldots, m(b_{k-1})) \)
- \( (D_1^\#, \sqsubseteq_1^\#, \gamma_1) \) an abstraction of \( \mathcal{P}(V_{\text{int}}^l) \)
Application 3: example

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

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

- this abstract value expresses a \textbf{relation} between $b_0$ and $x, y$ (which induces a relation between $x$ and $y$)
- \textbf{alternative}: partition with respect to only \textbf{some} variables
Application 3: example

- Left side abstraction shown in blue: boolean partitioning for $b_0, b_1$
- Right side abstraction shown in green: interval abstraction

```plaintext
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 integer type
- $X_{\text{int}} = \{x_0, \ldots, x_{l-1}\}$

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

Sign partitioning abstract domain

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

- $A = \{[< 0], [= 0], [> 0]\}$
- $\phi(m) = \begin{cases} [< 0] & \text{if } x_0 < 0 \\ [= 0] & \text{if } x_0 = 0 \\ [> 0] & \text{if } x_0 > 0 \end{cases}$
- $(D_{\text{int}}, \sqsubseteq_{\text{int}}, \gamma_1)$ an abstraction of $\mathcal{P}(V_{\text{int}}^{l-1})$ (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 ⇒ ⊥)
}

y = x/s;
(① x < 0 ⇒ s = −1 ∧ y > 0) ∧ (x = 0 ⇒ s = 1 ∧ y = 0) ∧ (x > 0 ⇒ s = 1 ∧ y > 0)

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**:

$$(\mathcal{P}(\mathcal{L} \times \mathcal{M}), \subseteq) \xleftarrow{\gamma_{\text{part}}} \overset{\alpha_{\text{part}}}{\longrightarrow} (\mathbb{D}_{\text{part}}^\#, \subseteq)$$

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

$$\begin{align*}
\alpha_{\text{part}} : \ & \mathcal{P}(\mathcal{L} \times \mathcal{M}) \quad \longrightarrow \quad \mathbb{D}_{\text{part}}^\
\mathcal{E} \quad \longmapsto \quad \lambda(l \in \mathcal{L}) \cdot \{ m \in \mathcal{M} \mid (l, m) \in \mathcal{E} \} \\
\gamma_{\text{part}} : \ & \mathbb{D}_{\text{part}}^\# \quad \longmapsto \quad \mathcal{P}(\mathcal{L} \times \mathcal{M}) \\
X^\# \quad \longmapsto \quad \{(l, m) \in \mathcal{S} \mid m \in X^\#(l)\}
\end{align*}$$
Fixpoint form of a partitioned semantics

- We consider a transition system $S = (\mathcal{S}, \rightarrow, \mathcal{S}_I)$
- 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 $\mathcal{E}_1, \ldots, \mathcal{E}_s \in \mathcal{P}(M)$:

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

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

$$
\alpha_{\text{part}}([S]_R) \text{ is the least solution of the system } \\
\left\{ \begin{array}{c}
\mathcal{E}_1 = F_1(\mathcal{E}_1, \ldots, \mathcal{E}_s) \\
\vdots \\
\mathcal{E}_s = F_s(\mathcal{E}_1, \ldots, \mathcal{E}_s)
\end{array} \right\}
$$
Partitioned systems and fixpoint computation

How to compute an abstract invariant for a partitioned system described by a set of abstract equations?

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

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

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

\[
\begin{align*}
\mathcal{E}_0 & = \mathcal{I} \cup F_0(\mathcal{E}_3) \\
\mathcal{E}_1 & = F_1(\mathcal{E}_0) \\
\mathcal{E}_2 & = F_2(\mathcal{E}_0) \\
\mathcal{E}_3 & = F_3(\mathcal{E}_1, \mathcal{E}_2)
\end{align*}
\]

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

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

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

- 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 $D^\#_{\text{part}}$ such that there exists a sequence $(k_n)_{n \in \mathbb{N}}$ of elements of $\{1, \ldots, s\}$ such that:

$$X_{n+1}^\# = \lambda(l_i \in \mathbb{L}) \cdot \left\{ \begin{array}{ll}
X_n^\#(l_i) & \text{if } i \neq k_n \\
X_n^\#(l_i) \sqcup F^\#(X_n^\#(l_1), \ldots, X_n^\#(l_s)) & \text{if } i = k_n
\end{array} \right.$$
Chaotic iterations: soundness

Soundness

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

$$\lim_{n \in \mathbb{N}} (X_n^\#) = \alpha_{\text{part}}(\llbracket S \rrbracket_R)$$

Proof: exercise

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

\[
\begin{align*}
\mathcal{E}_0^\# &= (\bot, \bot, \bot) \\
\mathcal{E}_1^\# &= (\bot, F_1^\#(\bot), \bot)
\end{align*}
\]

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

**Worklist 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 $\alpha_{num}(\mathcal{S}_I)$
- 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**
Selection of a set of widening points for a partitioned system

- We compose an abstraction $D^\#_{\text{num}}$, with concretization $\gamma_{\text{num}} : D^\#_{\text{num}} \to \mathcal{P}(M)$, 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:

\[
X^\#_{n+1} = \lambda(l_i \in \mathbb{I}) \cdot \begin{cases} 
X^\#_n(l_i) & \text{if } i \neq k_n \\
X^\#_n(l_i) \sqcup F^\#(X^\#_n(l_1), \ldots, X^\#_n(l_s)) & \text{if } i = k_n \land l_i \notin K_\nabla \\
X^\#_n(l_i) \nabla F^\#(X^\#_n(l_1), \ldots, X^\#_n(l_s)) & \text{if } i = k_n \land l_i \in K_\nabla 
\end{cases}
\]
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:

\[
\exists n_0 \in \mathbb{N}, \quad \left\{ \begin{array}{l}
\forall n \geq n_0, \quad X^\#_{n_0} = X^\#_n \\
[[S]]_R \subseteq \gamma_{\text{part}}(X_{n_0})
\end{array} \right.
\]

**Proof**: 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

\[ L \longrightarrow (V_{\text{bool}}^k \longrightarrow D_0^\#) \]

- we could do a partitioning by \( L \times V_{\text{bool}}^k \)
- yet, it is not practical, as transitions from “boolean states” are not known before the analysis
- thus, we seek for an approximation, for all pair \( \ell, \ell' \in L \) of

\[ \delta_{\ell, \ell'} : \mathbb{M} \longrightarrow \mathcal{P}(\mathbb{M}) \]

\[ m \mapsto \{ m' \in \mathbb{M} \mid (\ell, m) \rightarrow (\ell', m') \} \]
Transfer functions: scalar test and assignment

**Assignment** \( \ell_0 : x = e; \ell_1 \) affecting **only integer variables** (i.e., \( e \) depends only on \( x_0, \ldots, x_1 \)):

- **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):
  \[
  assign_{\rightarrow}(x, e, X^\#) = \lambda(z^\# \in D^\#_0) \cdot assign_1(x, e, X^\#(z^\#))
  \]

**Soundness**

If \( assign_1 \) is sound, so is \( assign_{\rightarrow} \), in the sense that:

\[
\forall X^\# \in D^\#_{\rightarrow}, \forall m \in \gamma_{\rightarrow}(X^\#), m[x \leftarrow \llbracket e \rrbracket(m)] \in \gamma_{\rightarrow}(assign_{\rightarrow}(x, e, X^\#))
\]
Transfer functions: scalar test and assignment

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

- 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:
  
  $$
  test \rightarrow (c, X^\#) = \lambda(z^\# \in D^\#_0) \cdot test_1(c, X^\#(z^\#))
  $$

- example:
  
  $$
  test \rightarrow \left( x \geq 8, \left\{ \begin{array}{l} b \Rightarrow x \geq 0 \\
  \neg b \Rightarrow x \leq 0 \end{array} \right\} \right) = \left\{ \begin{array}{l} b \Rightarrow x \geq 8 \\
  \neg b \Rightarrow \top \end{array} \right\}
  $$

**Soundness**

If $test_1$ is sound, so is $test \rightarrow$, in the sense that:

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

Condition test \( l_0 : \text{if}(c)\{l_1 : \ldots \} \) affecting only boolean variables (i.e., \( c \) depends only on \( b_0, \ldots, b_k \)):

- then, we simply need to filter the boolean partitions satisfying \( c \):

\[
\text{test}\rightarrow(c, X^\#) = \lambda(z^\# \in D^\#_0). \begin{cases} X^\#(z^\#) & \text{if } \text{test}_0(c, X^\#(z^\#)) \neq \perp_0 \\ \perp_1 & \text{otherwise} \end{cases}
\]

- for instance:

\[
\text{test}\rightarrow \begin{pmatrix} \neg b_1, \\ \land b_0 \land b_1 \Rightarrow 15 \leq x \\ \land b_0 \land \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \land \neg b_0 \land b_1 \Rightarrow 6 \leq x \leq 8 \\ \land \neg b_0 \land \neg b_1 \Rightarrow x \leq 5 \end{pmatrix} = \begin{pmatrix} b_0 \land b_1 \Rightarrow \perp_1 \\ \land b_0 \land \neg b_1 \Rightarrow 9 \leq x \leq 14 \\ \land \neg b_0 \land b_1 \Rightarrow \perp_1 \\ \land \neg b_0 \land \neg b_1 \Rightarrow x \leq 5 \end{pmatrix}
\]

Soundness

If \( \text{test}_0 \) is sound, so is \( \text{test}\rightarrow \), in the sense that:

\[
\forall X^\# \in D^\#, \forall m \in \gamma\rightarrow(X^\#), \llbracket c \rrbracket(m) = \text{TRUE} \implies m \in \gamma\rightarrow(\text{test}\rightarrow(x, e, X^\#))
\]
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 $x_0, \ldots, x_l$):

- let $z^\# \in D_0^\#$, 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:
  \[
  \text{assign} \rightarrow (b, e, X^\#)(z^\#) = \begin{cases} 
  \text{test}_1(e, X^\#(z^\#)) \\
  \bigcup_1 \text{test}_1(e, X^\#(z^\#[b \leftarrow \text{FALSE}]))
  \end{cases}
  \]
- same computation for cases where $z^\#(b) = \text{FALSE}$
- **for instance:**

\[
\text{assign} \rightarrow \left( b_0, x \leq 7, \begin{cases} 
  b_0 \land b_1 \Rightarrow 15 \leq x \\
  b_0 \land \neg b_1 \Rightarrow 9 \leq x \leq 14 \\
  \neg b_0 \land b_1 \Rightarrow 6 \leq x \leq 8 \\
  \neg b_0 \land \neg b_1 \Rightarrow x \leq 5
\end{cases} \right) = \left( b_0 \land b_1 \Rightarrow 6 \leq x \leq 7 \\
\land b_0 \land \neg b_1 \Rightarrow x \leq 5 \\
\land \neg b_0 \land b_1 \Rightarrow 8 \leq x \\
\land \neg b_0 \land \neg b_1 \Rightarrow 9 \leq x \leq 14 \right)
\]

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

- Boolean partitioning allows to express relations between boolean and scalar variables
- These relations are expensive:
  - Partitioning with respect to $N$ boolean variables translates into a $2^N$ space cost factor
  - 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
Definition of trace partitioning

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

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

**Trace partitioning**

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

Flow sensitive abstraction

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

$$\gamma_0 : \mathbb{D}_0^\# \rightarrow \mathcal{P}(\mathbb{S}^*)$$
$$l \rightarrow \mathbb{S}^* \cdot (\{l\} \times \mathcal{M})$$

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**:

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

**Domain of partitions**

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

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

**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):

\[
\begin{align*}
\text{int } x & \in \mathbb{Z}; \\
\text{int } s; \\
\text{int } y; \\
\text{if}(x \geq 0) \{ \\
& \quad \text{if}_t \Rightarrow (0 \leq x) \land \text{if}_f \Rightarrow \bot \\
& \quad s = 1; \\
& \quad \text{if}_t \Rightarrow (0 \leq x \land s = 1) \land \text{if}_f \Rightarrow \bot \\
\} \text{ else } \{ \\
& \quad \text{if}_f \Rightarrow (x < 0) \land \text{if}_t \Rightarrow \bot \\
& \quad s = -1; \\
& \quad \text{if}_f \Rightarrow (x < 0 \land s = -1) \land \text{if}_t \Rightarrow \bot \\
\} \\
\quad \{ \\
& \quad \text{if}_t \Rightarrow (0 \leq x \land s = 1) \\
& \quad \land \text{if}_f \Rightarrow (x < 0 \land s = -1) \\
\} \\
\end{align*}
\]

\( y = x/s; \)

\[
\begin{align*}
\text{if}_t \Rightarrow (0 \leq x \land s = 1 \land 0 \leq y) \\
\land \text{if}_f \Rightarrow (x < 0 \land s = -1 \land 0 < y)
\end{align*}
\]
Application 3: partitioning guided by a loop

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

\[
\begin{align*}
l_0 &: \text{ while}(c)\{ \\
l_1 &: \ldots \\
l_2 &: \\
l_3 &: \ldots
\end{align*}
\]

**Domain of partitions**

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

\[
\mathcal{D}^\#_0 = \{ \text{loop}_0, \text{loop}_1, \ldots, \text{loop}_k, \top \} \quad \text{and:}
\]

\[
\begin{align*}
\gamma_0 &: \text{loop}_i \quad \mapsto \quad \text{traces that visit} \ l_1 \ i \ \text{times} \\
\top &\quad \mapsto \quad \mathbb{S}^*
\end{align*}
\]

**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++;
}
```

\[
\begin{align*}
\text{loop}_0 & \Rightarrow x \leq -1 \\
\text{loop}_1 & \Rightarrow -1 \leq x \leq 1 \\
\text{loop}_2 & \Rightarrow 1 \leq x \leq 3 \\
\text{loop}_3 & \Rightarrow 3 \leq x
\end{align*}
\]

\[
y = tc[i] \times (x - tx[i]) + ty[i]
\]
Application 4: partitioning guided by the value of a variable

We consider a program with an integer variable \( x \), and a program point \( \ell \):

\[
\text{int } x; \ldots; \ell : \ldots
\]

**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:

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

\[
\top \mapsto S^*
\]

**Domain of partitions: partitioning by the property of a variable**

For a given abstraction \( \gamma : (V^\#, \sqsubseteq^\#) \to (\mathcal{P}(\mathbb{V}_{\text{int}}), \subseteq) \), the partitions are defined by \( D_0^\# = \{ \text{var}_{v^\#} \mid v^\# \in V^\# \} \) and:

\[
\gamma_0 : \text{val}_{v^\#} \mapsto \{ \langle \ldots, (\ell, m), \ldots \rangle \mid m(x) \in \text{var}_{v^\#} \} \]
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

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

1 (x < 0@1 ⇒ T) ∧ (x = 0@1 ⇒ T) ∧ (x > 0@1 ⇒ T)

b0 = x ≥ 0;
   (x < 0@1 ⇒ ¬b0) ∧ (x = 0@1 ⇒ b0) ∧ (x > 0@1 ⇒ b0)

b1 = x ≤ 0;
   (x < 0@1 ⇒ ¬b0 ∧ 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)
}
```
Trace partitioning

We search for **general** way to **generate** and **compute** partitions.
- 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 \( S' \preceq S \)) if and only if

- \( \forall (\ell, m) \in S_I, \exists tok \in D_0^\#, ((\ell, tok), m) \in S_I' \)
- \( \forall (\ell, m), (\ell', m') \in S, \forall tok \in D_0^\#, (\ell, m) \rightarrow (\ell', m') \implies \exists tok' \in D_0^\#, ((\ell, tok), m) \rightarrow ((\ell', tok'), m') \)

Then:

\[
\forall \langle (\ell_0, m_0), \ldots, (\ell_n, m_n) \rangle \in \llbracket S \rrbracket_R,
\exists tok_0, \ldots, tok_n \in D_0^\#, \langle ((\ell_0, tok_0), m_0), \ldots, ((\ell_n, tok_n), m_n) \rangle \in \llbracket S' \rrbracket_R,
\]
Trace partitioning induced by a refined transition system

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

**Definition of a trace partitioning**

The abstraction defining partitions is defined by:

\[
γ_0 : \mathbb{D}^\#_0 \rightarrow \mathcal{P}(S^*)
\]

\[
tok \mapsto \{σ ∈ S^* \mid ∃σ' = ⟨..., ((l, tok), m)⟩ ∈ (S')^*, Ψ(σ') = σ\}
\]

- 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:

```plaintext
l_0  if(x < 0) {
    l_1   s = -1;
  } else {
    l_2   s = 1;
  }  

l_3  y = x/s;

l_4 ...

P_0  
    (l_0, T) 
    /  
   /   
(4, T)  (5, T) 

P_1  
    (l_0, T) 
    /  
   /   
(4, T)  (5, if_t) 

P_2  
    (l_0, T) 
    /  
   /   
(4, T)  (5, if_t) 
```

- 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 $S = L \times M$, $S' = (L \times D^0_0) \times M$
- transfer functions of $S'$:
  $\delta_{l,l'} : (D^\#_0 \rightarrow D^\#_1) \rightarrow (D^\#_0 \rightarrow D^\#_1)$ over-approximating $\rightarrow'$

Partition irrelevant transfer function

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

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

- partition irrelevant transfer functions: pointwise operators of $D^\#_1$
  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

```c
int x ∈ Z;
int s;
int y;
if(x ≥ 0){
  if_t ⇒ (0 ≤ x) ∧ if_f ⇒ ⊥
  s = 1;
  if_t ⇒ (0 ≤ x ∧ s = 1) ∧ if_f ⇒ ⊥
} else {
  if_f ⇒ (x < 0) ∧ if_t ⇒ ⊥
  s = −1;
  if_f ⇒ (x < 0 ∧ s = −1) ∧ if_t ⇒ ⊥
}

\begin{align*}
  \left\{ & \begin{align*}
    & \text{if}_t \: ⇒ \: (0 ≤ x ∧ s = 1) \quad & \text{partition creation: if}_t \\
    & \quad \land \quad \text{if}_f \: ⇒ \: (x < 0 ∧ s = −1) \quad & \text{no modification of partitions}
  \end{align*} \right.
  \quad \text{no modification of partitions}
\end{align*}

y = x/s;

\begin{align*}
  \left\{ & \begin{align*}
    & \text{if}_t \: ⇒ \: (0 ≤ x ∧ s = 1 ∧ 0 ≤ y) \quad & \text{no modification of partitions} \\
    & \quad \land \quad \text{if}_f \: ⇒ \: (x < 0 ∧ s = −1 ∧ 0 < y) \quad & \text{no modification of partitions}
  \end{align*} \right.
  \quad \text{fusion of partitions}
\end{align*}

... 

⇒ s ∈ [-1, 1] ∧ 0 ≤ y
```

In general, partitions are rarely modified (only some branching points)
Transfer functions: partition creation

Analysis of an if statement, with partitioning

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

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

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

When partitions are not useful anymore, they can be merged

\[ \delta_{l_0, l_1}^\#(X^\#) = [\_ \mapsto \sqcup_t X^\#(l_0)(t)] \]

- 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
Trace partitioning

Choice of partitions

How are the partitions chosen?

Static partitioning

- a fixed partitioning abstraction $D^\#, \gamma_0$ is fixed before the analysis
- usually $D^\#, \gamma_0$ 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 $D^\#, \gamma_0$ 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 $(D^\#, \gamma_0)$ 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 “repartitioning”.