Formal Analysis and Verification
FAV 2018/2019

Ondřej Lengál, Tomáš Vojnar
{lengal, vojnar}@fit.vutbr.cz

Brno University of Technology
Faculty of Information Technology
Božetěchova 2, 612 66 Brno
Abstract Interpretation
**Introduction**

- Compared to model checking in which the stress is put on a systematic execution of a system being verified (or its model), the emphasis in static analysis is on minimization of the amount of execution of the code. It is either not executed at all (the case of looking for bug patterns) or just on some abstract level, typically with an in advance fixed abstraction (data flow analysis, abstract interpretation, . . .).

- However, the borderline between model checking and static analysis is not sharp (especially when considering abstract interpretation and model checking based on predicate abstraction).

- Many static analyses are such that they can be applied to parts of code without the need to describe their environment.

- **Static analysis approaches**: bug pattern analysis, type analysis, data flow analysis, . . ., abstract interpretation, (and sometimes even model checking).
Static Analyses

- Many different classes of programs:
  - control-intensive programs,
  - digital signal processing,
  - programs manipulating dynamic memory,
  - programs with integers, . . .

- To make an analysis efficient (effective), some form of abstraction is often needed.

- An analysis successful for one class may (and is very likely to) fail for a different one (too much imprecision, inefficiency, divergence).

- Analyses are tailored for specific classes of programs
  - the need to prove soundness (completeness) of each analysis.
Abstract Interpretation — A Motivating Example

Listing 1: SQL Input Sanitization

```c
void sanitize(char* query)
{
    for (size_t i = 0; i < strlen(query); ++i)
    {
        if ('\' == query[i])
            query[i] = '_';
    }

    if (!strcmp(query, '\ OR 1==1'))
        ERROR();
}
```

• The model checking approach — check for all values of `query` infeasible!
• The abstract interpretation approach — find a suitable abstract domain for `query`.
Abstract Interpretation — A Motivating Example

Listing 2: SQL Input Sanitization

```c
void sanitize(char* query)
{
    for (size_t i = 0; i < strlen(query); ++i)
    {
        if ('\' == query[i])
            query[i] = ' _';
    }

    if (!strcmp(query, "\' OR 1==1"))
        ERROR();
}
```

- The model checking approach — check for all values of query → infeasible!
- The abstract interpretation approach — find a suitable abstract domain for query.
Abstract Interpretation

- Introduced by Patrick and Radhia Cousot at POPL’77.
- A general framework for static analyses.
- Particular analyses are created by providing specific components (abstract domain, abstract transformers, ...) to the framework.
- Abstract interpretation assigns a program an abstract semantics over an abstract domain and analyses this abstract semantics (cf. predicate abstraction).
- When certain properties of the components are met (wrt the concrete semantics), the analysis is guaranteed to be sound.
Ingredients of Abstract Interpretation

- **Abstract domain**
  - sets of program states at program locations are represented using abstract contexts.

- **Abstract transformers**
  - for each program operation there is a corresponding transformer that represents the effect of the operation performed on an abstract context.

- **Join operator**
  - combines abstract contexts from several branches into a single one.

- **Widening**
  - performed on a sequence of abstract contexts appearing at a given location to accelerate obtaining a fixpoint.

- **Narrowing**
  - may be used to refine the result of widening.
Abstract Interpretation — formally

• A semilattice is a poset with a join (least upper bound, supremum) for every finite subset of its base set.
Abstract Interpretation — formally

- A **semilattice** is a poset with a **join** (least upper bound, supremum) for every finite subset of its base set.

- Abstract interpretation $I$ of a program $P$ with the instruction set $Instr$ is a tuple

$$I = (Q, \circ, \sqsubseteq, \top, \bot, \tau)$$

where

- $Q$ is the **abstract domain** (domain of abstract contexts),
- $\top \in Q$ is the supremum of $Q$,
- $\bot \in Q$ is the infimum of $Q$,
- $\circ : Q \times Q \to Q$ is the **join operator** for accumulation of abstract contexts,
  $(Q, \circ, \top)$ is a complete semilattice,
- $(\sqsubseteq) \subseteq Q \times Q$ is an ordering defined as $x \sqsubseteq y \iff x \circ y = y$ in $(Q, \circ, \top),$
- $\tau : Instr \times Q \to Q$ defines the **abstract transformers** for particular instructions.

- The **soundness** of abstract interpretation may be guaranteed using **Galois connections**.
Galois Connections

- **Galois connection** is a quadruple $\pi = (\mathcal{P}, \alpha, \gamma, \mathcal{Q})$ such that:
  - $\mathcal{P} = \langle P, \leq \rangle$ and $\mathcal{Q} = \langle Q, \sqsubseteq \rangle$ are partially ordered sets (posets),
  - $\alpha : P \to Q$ and $\gamma : Q \to P$ are functions such that $\forall p \in P$ and $\forall q \in Q$:
    \[ p \leq \gamma(q) \iff \alpha(p) \sqsubseteq q \]
Galois Connections

- **Galois connection** is a quadruple $\pi = (P, \alpha, \gamma, Q)$ such that:
  - $P = \langle P, \leq \rangle$ and $Q = \langle Q, \sqsubseteq \rangle$ are **partially ordered sets** (posets),
  - $\alpha : P \to Q$ and $\gamma : Q \to P$ are functions such that $\forall p \in P$ and $\forall q \in Q$:
    \[ p \leq \gamma(q) \iff \alpha(p) \sqsubseteq q \]

- In abstract interpretation, $Q$ is the **abstract domain** and $P$ a (more) **concrete domain** – elements of both domains, called abstract/concrete **contexts**, represent sets of states:

![Diagram showing Galois connection with abstract and concrete domains, abstraction and concretization functions.](image-url)
Galois Connections

- **Galois connection** is a quadruple $\pi = (\mathcal{P}, \alpha, \gamma, \mathcal{Q})$ such that:
  - $\mathcal{P} = \langle P, \leq \rangle$ and $\mathcal{Q} = \langle Q, \sqsubseteq \rangle$ are partially ordered sets (posets),
  - $\alpha : P \rightarrow Q$ and $\gamma : Q \rightarrow P$ are functions such that $\forall p \in P$ and $\forall q \in Q$:
    $$ p \leq \gamma(q) \iff \alpha(p) \sqsubseteq q $$

- In abstract interpretation, $\mathcal{Q}$ is the abstract domain and $\mathcal{P}$ a (more) concrete domain – elements of both domains, called abstract/concrete contexts, represent sets of states:

- **Concrete contexts** such as:
  - $\{x \mapsto 1, y \mapsto 0\}$
  - $\{x \mapsto 2, y \mapsto 0\}$
  - $\{x \mapsto 3, y \mapsto 0\}$
  - $\{x \mapsto 0, y \mapsto 1\}$
  - $\{x \mapsto 0, y \mapsto 2\}$
  - $\{x \mapsto 0, y \mapsto 3\}$

- **Abstract contexts** such as:
  - $\{(x \geq 1 \land y = 0), (x = 0 \land y \geq 1)\}$
Galois Connections

- Implication: if the abstraction and concretization functions of an abstract interpretation form a Galois connection, $\forall p \in P. \ p \leq \gamma(\alpha(p))$.

Proof.

Take any $p \in P$, and let $q = \alpha(p)$.
As $q = \alpha(p) \Rightarrow \alpha(p) \sqsubseteq q$, the definition of Galois connections implies $p \leq \gamma(q)$.
By plugging in the fact that $q = \alpha(p)$, we get $p \leq \gamma(\alpha(p))$. 

\[ \]
Galois Connections

- **Implication**: if the abstraction and concretization functions of an abstract interpretation form a Galois connection, \( \forall p \in P. \ p \leq \gamma(\alpha(p)) \).

**Proof.**

Take any \( p \in P \), and let \( q = \alpha(p) \).
As \( q = \alpha(p) \Rightarrow \alpha(p) \sqsubseteq q \), the definition of Galois connections implies \( p \leq \gamma(q) \).
By plugging in the fact that \( q = \alpha(p) \), we get \( p \leq \gamma(\alpha(p)) \).

- If each instruction \( i \) from \( \text{Instr} \) and the corresponding abstract transformer \( \tau_i \) respect the Galois connection, i.e., for each \( p \in P \),

\[
\alpha(i(p)) \sqsubseteq \tau_i(\alpha(p)),
\]

the abstract interpretation may only over-approximate the concrete semantics. Hence, it is **sound**.
Fixpoint Approximation

- In some cases (e.g., loops), computation of the *most precise* abstract fixpoint is not generally guaranteed to terminate (consider \textit{id} as the abstraction function).

- To guarantee termination, the fixpoint can be approximated. This is done using the following two operations:
  - \textbf{widening}: performs an over-approximation of a fixpoint,
  - \textbf{narrowing}: refines an approximation of a fixpoint.

- Neither widening nor narrowing are necessary, but at least widening is often convenient. Narrowing may be sometimes missing (e.g., in polyhedral analysis).
Widening

- Let \( I = (Q, \circ, \sqsubseteq, \top, \bot, \tau) \) be an abstract interpretation of a program.

- The binary widening operation \( \triangledown \) is defined as:
  - \( \triangledown : Q \times Q \rightarrow Q \),
  - \( \forall C, D \in Q : (C \circ D) \sqsubseteq (C \triangledown D) \),
  - for all increasing infinite sequences \( C_0 \sqsubseteq C_1 \sqsubseteq \ldots \sqsubseteq C_n \sqsubseteq \ldots \), it holds that the infinite sequence \( s_0, s_1, \ldots, s_n, \ldots \) defined recursively as

  \[
  \begin{align*}
  s_0 &= C_0, \\
  s_n &= s_{n-1} \triangledown C_n
  \end{align*}
  \]

  is not strictly increasing (and because the result of \( \triangledown \) is an upper bound, the sequence eventually stabilizes).

- Widening can be applied later in the computation, the later it is applied the more precise is the result (but the computation takes longer time).
Narrowing

- Let $I = (Q, \circ, \sqsubseteq, \top, \bot, \tau)$ be an abstract interpretation of a program.

- The binary narrowing operation $\triangle$ is defined as:
  - $\triangle: Q \times Q \rightarrow Q$,
  - $\forall C, D \in Q : C \sqsupseteq D \Rightarrow (C \sqsupseteq (C \triangle D) \sqsupseteq D)$,
  - for all decreasing infinite sequences $C_0 \sqsupseteq C_1 \sqsupseteq \ldots \sqsupseteq C_n \sqsupseteq \ldots$, it holds that the infinite sequence $s_0, s_1, \ldots, s_n, \ldots$ defined recursively as
    $$
    \begin{align*}
    s_0 &= C_0, \\
    s_n &= s_{n-1} \triangle C_n
    \end{align*}
    $$

    is not strictly decreasing (and because the result of $C \triangle D$ is a lower bound of $C$, the sequence eventually stabilizes).

- Narrowing is performed only once a fixpoint is reached via widening.
We choose (deterministic) finite flowcharts as a language independent representation of programs.

A finite flowchart is a directed graph with 5 types of nodes:
- entries,
- assignments,
- tests,
- junctions,
- exits.

Abstract interpretation iteratively computes abstract contexts for each edge of the flowchart.

An equation is associated with each edge of the flowchart according to the type of the tail node of the edge.
Representation of a Program

- **Entry**: denotes the entry point of a program. $C_O = T$. 
  \[
  \downarrow_{C_O}
  \]
Representation of a Program

- **Entry**: denotes the entry point of a program. $C_O = \top$.

  $$\downarrow C_O$$

- **Assignment**: denotes the assignment $A$ of expression $\langle \text{Expr} \rangle$ to the variable $\langle \text{Ident} \rangle$. $C_O = \tau(A, C_I)$.

  $$\downarrow C_I$$
  $$\langle \text{Ident} \rangle := \langle \text{Expr} \rangle$$
  $$\downarrow C_O$$
Representation of a Program

- **Entry:** denotes the entry point of a program. \( C_O = \top. \)

\[ \downarrow C_O \]

- **Assignment:** denotes the assignment \( A \) of expression \(<\text{Expr}>\) to the variable \(<\text{Ident}>\). \( C_O = \tau(A, C_I). \)

\[ \downarrow C_I \]
\[ \begin{align*}
<\text{Ident}> & := <\text{Expr}> \\
\downarrow C_O 
\end{align*} \]

- **Test:** denotes splitting of the flow to branches \( B_{true} \) and \( B_{false} \) according to the Boolean condition \(<\text{Cond}>\). Two contexts are computed: \( C_{true} = \tau(B_{true}, C_I) \) and \( C_{false} = \tau(B_{false}, C_I). \)

\[ \begin{align*}
<\text{Cond}> & \quad \tau \quad C_{false} \\
\downarrow C_I \\
\downarrow C_{true} 
\end{align*} \]
Representation of a Program

- **Junction**: denotes join $J$ of several branches of code execution (e.g., after \texttt{...then ... and ...else ...} branches of an if statement or for a loop join).

  \[ C_O = \tau(J, C_1 \circ \cdots \circ C_n). \]

  ![Diagram of a junction](image)

  It often holds for junctions that:
  
  - \( \tau(J) = \lambda x . x \) for simple junctions (if branches),
  - \( \tau(J) = \lambda x . C_p \triangledown x \) for loop junctions,
  - \( \tau(J) = \lambda x . C_p \triangleright x \) for loop junctions (only after widening)

  where $C_p$ is the abstract context computed for the node in the previous iteration.
Representation of a Program

- **Junction**: denotes join $J$ of several branches of code execution (e.g., after \( \ldots \) then \( \ldots \) and \( \ldots \) else \( \ldots \) branches of an if statement or for a loop join).
  \[ C_O = \tau(J, C_1 \circ \cdots \circ C_n). \]

It often holds for junctions that:
- \( \tau(J) = \lambda x . x \) — for simple junctions (if branches),
- \( \tau(J) = \lambda x . C_p \triangleleft x \) — for loop junctions,
- \( \tau(J) = \lambda x . C_p \triangleright x \) — for loop junctions (only after widening)
  where $C_p$ is the abstract context computed for the node in the previous iteration.

- **Exit**: denotes the exit point of a program.
Program Example

- Consider the following flowchart program and interval analysis:

```
C₀
x := 1
C₁
C₂
x ≤ 100
false
C₃
true
C₄
x := x + 1
C₅
```
Program Example

- Consider the following flowchart program and interval analysis:

- We will use notation $[a, b]$ for the predicate $a \leq x \leq b$.
- Assignments: interval arithmetic (e.g., $[i, j] + [k, l] = [i + k, j + l]$).
- Join: $[a, b] \circ [c, d] = [\min(a, c), \max(b, d)]$.
- Tests: intersections of intervals.
- We define the widening $\nabla$ of intervals as:
  - $[\ , \ ]$ is the null element of $\nabla$ and $\circ$,
  - $[i, j] \nabla [k, l] = \begin{cases} -\infty & \text{if } k < i \text{ then } \text{ and } l > j \text{ then } +\infty \text{ else } i, \\ j & \text{if } l > j \text{ then } +\infty \text{ else } j. \end{cases}$
Program Example

- Consider the following flowchart program and interval analysis:

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).
- **Assignments:** interval arithmetic (e.g., \([i, j] + [k, l] = [i + k, j + l]\)).
- **Join:** \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).
- **Tests:** intersections of intervals.
- We define the widening \(\triangledown\) of intervals as:
  - \([, ]\) is the null element of \(\triangledown\) and \(\circ\),
  - \([i, j] \triangledown [k, l] = [\text{if } k < i \text{ then } -\infty \text{ else } i, \\
    \text{if } l > j \text{ then } +\infty \text{ else } j]\).

- \(C_0 = [-\infty, +\infty]\)
- \(C_1 = [1, 1]\)
- \(C_2 = C_2 \triangledown (C_1 \circ C_4)\)
- \(C_3 = C_2 \cap [-\infty, 100]\)
- \(C_4 = C_3 + [1, 1]\)
- \(C_5 = C_2 \cap [101, +\infty]\)
Program Example

- Consider the following flowchart program and interval analysis:

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).
- **Assignments:** interval arithmetic (e.g., \([i, j] + [k, l] = [i+k, j+l]\)).
- **Join:** \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).
- **Tests:** intersections of intervals.
- We define the **widening** \(\triangledown\) of intervals as:
  - \([a, b]\) is the null element of \(\triangledown\) and \(\circ\),
  - \([i, j] \triangledown [k, l] = \text{if } k < i \text{ then } -\infty \text{ else } i, \text{ if } l > j \text{ then } +\infty \text{ else } j\).

- \(C_0^0 = [−\infty, +\infty]\)
- \(C_0^1 = [1, 1]\)
- \(C_0^2 = [1, 1]\)
- \(C_0^3 = [100, +\infty]\)
- \(C_0^4 = [1, 1]\)
- \(C_0^5 = [101, +\infty]\)
Program Example

- Consider the following flowchart program and interval analysis:

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).
- Assignments: interval arithmetic (e.g., \([i, j] + [k, l] = [i + k, j + l]\)).
- Join: \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).
- Tests: intersections of intervals.
- We define the widening \(\triangledown\) of intervals as:
  - \([, \ ]\) is the null element of \(\triangledown\) and \(\circ\),
  - \([i, j]\triangledown[k, l] = \text{if } k < i \text{ then } -\infty \text{ else } i,
    \text{ if } l > j \text{ then } +\infty \text{ else } j\).

- \(C_0^0 = [\infty, +\infty]\)
- \(C_1^0 = [1, 1]\)
- \(C_2^0 = [1, +\infty]\)
- \(C_3^0 = [1, 100]\)
- \(C_4^0 = [1, +\infty]\)
- \(C_5^0 = [\infty, +\infty]\)
- \(C_0 = [\infty, +\infty]\)
- \(C_1 = [1, 1]\)
- \(C_2 = C_2 \triangledown (C_1 \circ C_4)\)
- \(C_3 = C_2 \cap [\infty, 100]\)
- \(C_4 = C_3 + [1, 1]\)
- \(C_5 = C_2 \cap [101, +\infty]\)
Program Example

- Consider the following flowchart program and interval analysis:

  ![Flowchart Diagram]

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).

- **Assignments:** interval arithmetic (e.g., \([i, j] + [k, l] = [i+k, j+l]\)).

- **Join:** \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).

- **Tests:** intersections of intervals.

- We define the widening \(\nabla\) of intervals as:
  - \([, ,]\) is the null element of \(\nabla\) and \(\circ\),
  - \([i, j] \nabla [k, l] = \begin{cases} \infty & \text{if } k < i \text{ then } -\infty \text{ else } i, \\ i & \text{if } l > j \text{ then } +\infty \text{ else } j \end{cases}\).

- \(C_0^0 = [-\infty, +\infty]\)
- \(C_1^0 = [1, 1]\)
- \(C_2^0 = [1, 1]\)
- \(C_3^0 = [1, 1]\)
- \(C_4^0 = [1, 1]\)
- \(C_5^0 = [1, 1]\)
- \(C_0 = [-\infty, +\infty]\)
- \(C_1 = [1, 1]\)
- \(C_2 = C_2 \nabla (C_1 \circ C_4)\)
- \(C_3 = C_2 \cap [-\infty, 100]\)
- \(C_4 = C_3 + [1, 1]\)
- \(C_5 = C_2 \cap [101, +\infty]\)
Program Example

- Consider the following flowchart program and interval analysis:

```
C0
x := 1
C1
C2
x ≤ 100
false
C4
true
C3
x := x + 1
```

- We will use notation $[a, b]$ for the predicate $a \leq x \leq b$.

- **Assignments**: interval arithmetic (e.g., $[i, j] + [k, l] = [i + k, j + l]$).

- **Join**: $[a, b] \circ [c, d] = [\min(a, c), \max(b, d)]$.

- **Tests**: intersections of intervals.

- We define the widening $\nabla$ of intervals as:
  - $[\ , \ ]$ is the null element of $\nabla$ and $\circ$,
  - $[i, j] \nabla [k, l] = [\begin{cases} k < i \text{ then } -\infty \\ i \text{ else } i \\ l > j \text{ then } +\infty \\ j \text{ else } j \end{cases}]$.

- $C_0^0 = [\ -\infty, +\infty]$  
- $C_1^0 = [\ , \ ] C_1^1 = [1, 1]$  
- $C_2^0 = [\ , \ ] C_2^1 = [1, 1]$  
- $C_3^0 = [\ , \ ] C_3^1 = [1, 1]$  
- $C_4^0 = [\ , \ ]$  
- $C_5^0 = [\ , \ ]$

- $C_0 = [\ -\infty, +\infty]$  
- $C_1 = [1, 1]$  
- $C_2 = C_2 \nabla (C_1 \circ C_4)$  
- $C_3 = C_2 \cap [-\infty, 100]$  
- $C_4 = C_3 + [1, 1]$  
- $C_5 = C_2 \cap [101, +\infty]$
Program Example

- Consider the following flowchart program and interval analysis:

  \[\begin{align*}
  &C_0 \quad x := 1 \\
  &C_1 \\
  &C_2 \quad x \leq 100 \\
  &C_4 \quad x := x + 1 \\
  &C_5 \quad \text{false}
  \end{align*}\]

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).

- **Assignments**: interval arithmetic (e.g., \([i, j] + [k, l] = [i + k, j + l]\)).

- **Join**: \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).

- **Tests**: intersections of intervals.

- We define the **widening** \(\nabla\) of intervals as:
  - \([,]\) is the null element of \(\nabla\) and \(\circ\),
  - \([i, j] \nabla [k, l] = \begin{cases} i & \text{if } k < i \text{ then } -\infty \text{ else } i, \\ j & \text{if } l > j \text{ then } +\infty \text{ else } j \end{cases}\).

- \[\begin{align*}
  &C_0^0 = [\infty, +\infty] \\
  &C_1^0 = [1, 1] \\
  &C_2^0 = [1, 1] \\
  &C_3^0 = [1, 1] \\
  &C_4^0 = [2, 2] \\
  &C_5^0 = [,] \\
  &C_0 = [\infty, +\infty] \\
  &C_1 = [1, 1] \\
  &C_2 = C_2 \nabla (C_1 \circ C_4) \\
  &C_3 = C_2 \cap [-\infty, 100] \\
  &C_4 = C_3 + [1, 1] \\
  &C_5 = C_2 \cap [101, +\infty]
  \end{align*}\]
Program Example

- Consider the following flowchart program and interval analysis:

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).
- Assignments: interval arithmetic (e.g., \([i, j] + [k, l] = [i + k, j + l]\)).
- Join: \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).
- Tests: intersections of intervals.
- We define the widening \(\nabla\) of intervals as:
  - \([a, b]\) is the null element of \(\nabla\) and \(\circ\),
  - \([i, j] \nabla [k, l] = [\text{if } k < i \text{ then } -\infty \text{ else } i, \text{ if } l > j \text{ then } +\infty \text{ else } j]\).

- \(C_0^0 = [-\infty, +\infty]\)
- \(C_0^0 = [\ , ] \ C_1^1 = [1, 1]\)
- \(C_0^2 = [\ , ] \ C_2^1 = [1, 1] \ C_2^2 = [1, +\infty]\)
- \(C_0^0 = [\ , ] \ C_3^1 = [1, 1]\)
- \(C_0^0 = [\ , ] \ C_4^1 = [2, 2]\)
- \(C_0^0 = [\ , ] \ C_5^1 = [2, 2]\)

- \(C_0 = [-\infty, +\infty]\)
- \(C_1 = [1, 1]\)
- \(C_2 = C_2 \nabla (C_1 \circ C_4)\)
- \(C_3 = C_2 \cap [-\infty, 100]\)
- \(C_4 = C_3 + [1, 1]\)
- \(C_5 = C_2 \cap [101, +\infty]\)
Program Example

Consider the following flowchart program and interval analysis:

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).
- Assignments: interval arithmetic (e.g., \([i, j] + [k, l] = [i + k, j + l]\)).
- Join: \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).
- Tests: intersections of intervals.
- We define the widening \(\nabla\) of intervals as:
  - \([ , ]\) is the null element of \(\nabla\) and \(\circ\),
  - \([i, j] \nabla [k, l] = [\text{if } k < i \text{ then } -\infty \text{ else } i, \text{if } l > j \text{ then } +\infty \text{ else } j]\).

- \(C_0^0 = [-\infty, +\infty]\)
- \(C_1^0 = [ , ] \ C_1^1 = [1, 1]\)
- \(C_2^0 = [ , ] \ C_2^1 = [1, 1] \ C_2^2 = [1, +\infty]\)
- \(C_3^0 = [ , ] \ C_3^1 = [1, 1] \ C_3^2 = [1, 100]\)
- \(C_4^0 = [ , ] \ C_4^1 = [2, 2]\)
- \(C_5^0 = [ , ]\)
- \(C_0 = [-\infty, +\infty]\)
- \(C_1 = [1, 1]\)
- \(C_2 = C_2 \nabla (C_1 \circ C_4)\)
- \(C_3 = C_2 \cap [-\infty, 100]\)
- \(C_4 = C_3 + [1, 1]\)
- \(C_5 = C_2 \cap [101, +\infty]\)
Program Example

- Consider the following flowchart program and *interval* analysis:

```
  C0
  x := 1
  C1
  C2
  x ≤ 100
  false
  C4
  C3
  x := x + 1
  true
  C5
```

- We will use notation $[a, b]$ for the predicate $a \leq x \leq b$.

- **Assignments**: *interval arithmetic* (e.g., $[i, j] + [k, l] = [i + k, j + l]$).

- **Join**: $[a, b] \circ [c, d] = [\min(a, c), \max(b, d)]$.

- **Tests**: *intersections* of intervals.

- We define the *widening* $\nabla$ of intervals as:
  - $[\ , \ ]$ is the null element of $\nabla$ and $\circ$,
  - $[i, j] \nabla [k, l] = [\text{if } k < i \text{ then } -\infty \text{ else } i,$
    if $l > j \text{ then } +\infty \text{ else } j]$.

- $C_0^0 = [-\infty, +\infty]$
- $C_1^0 = [\ ,\ ] C_1^1 = [1, 1]$
- $C_2^0 = [\ ,\ ] C_2^1 = [1, 1] C_2^2 = [1, +\infty]$
- $C_3^0 = [\ ,\ ] C_3^1 = [1, 1] C_3^2 = [1, 100]$
- $C_4^0 = [\ ,\ ] C_4^1 = [2, 2] C_4^2 = [2, 101]$
- $C_5^0 = [\ ,\ ]$
- $C_0^1 = [1, 1]$
- $C_1^1 = [1, 1]$
- $C_2^1 = C_2 \nabla (C_1 \circ C_4)$
- $C_3^1 = C_2 \cap [101, +\infty]$
- $C_4^1 = C_3 + [1, 1]$
- $C_5^1 = C_2 \cap [101, +\infty]$
Program Example

- Consider the following flowchart program and interval analysis:

\[
\begin{align*}
\text{C}_0 & \quad x := 1 \\
\text{C}_1 & \\
\text{C}_2 &\
\text{C}_3 & \quad x \leq 100 \\
\text{C}_4 & \\
\text{C}_5 & \\
\end{align*}
\]

- We will use notation \([a, b]\) for the predicate \(a \leq x \leq b\).
- Assignments: interval arithmetic (e.g., \([i, j] + [k, l] = [i + k, j + l]\)).
- Join: \([a, b] \circ [c, d] = [\min(a, c), \max(b, d)]\).
- Tests: intersections of intervals.
- We define the widening \(\nabla\) of intervals as:
  - \([, ,]\) is the null element of \(\nabla\) and \(\circ\),
  - \([i, j] \nabla [k, l] = [\text{if } k < i \text{ then } -\infty \text{ else } i, \text{if } l > j \text{ then } +\infty \text{ else } j]\).

\[
\begin{align*}
\text{C}_0^0 & = [-\infty, +\infty] \\
\text{C}_1^0 & = [1, 1] \\
\text{C}_1^1 & = [1, 1] \\
\text{C}_2^0 & = [1, 1] \\
\text{C}_2^1 & = [1, 1] \\
\text{C}_2^2 & = [1, +\infty] \\
\text{C}_3^0 & = [1, 1] \\
\text{C}_3^1 & = [1, 1] \\
\text{C}_3^2 & = [1, 100] \\
\text{C}_4^0 & = [2, 2] \\
\text{C}_4^1 & = [2, 2] \\
\text{C}_4^2 & = [2, 101] \\
\text{C}_5^0 & = [101, +\infty] \\
\end{align*}
\]

\[
\begin{align*}
\text{C}_0 & = [-\infty, +\infty] \\
\text{C}_1 & = [1, 1] \\
\text{C}_2 & = \text{C}_2 \nabla (\text{C}_1 \circ \text{C}_4) \\
\text{C}_3 & = \text{C}_2 \cap [-\infty, 100] \\
\text{C}_4 & = \text{C}_3 + [1, 1] \\
\text{C}_5 & = \text{C}_2 \cap [101, +\infty] \\
\end{align*}
\]
Program Example

- Consider the following flowchart program and *interval* analysis:

  ![Flowchart](image)

  - Let us now define the *narrowing* operation $\triangle$ over intervals as
    - $[i, j]$ is the null element of $\triangle$,
    - $[i, j] \triangle [k, l] =$
      - if $i = -\infty$ then $k$ else $\min(i, k)$,
      - if $j = +\infty$ then $l$ else $\max(j, l)$.

  - We now substitute the equation for $C_2$ with a new one that uses narrowing.

- $C_0^0 = [\infty, +\infty]$
- $C_1^0 = [\infty, 1]$
- $C_1^1 = [1, 1]$
- $C_2^0 = [\infty, 100]$
- $C_2^1 = [1, 1]$
- $C_2^2 = [1, +\infty]$
- $C_3^0 = [\infty, 100]$
- $C_3^1 = [1, 100]$
- $C_3^2 = [1, 100]$
- $C_4^0 = [2, 101]$
- $C_4^1 = [2, 2]$
- $C_4^2 = [2, 101]$
- $C_5^0 = [101, +\infty]$
- $C_5^1 = [101, +\infty]$

- $C_0 = [-\infty, +\infty]$
- $C_1 = [1, 1]$
- $C_2 = C_2 \triangle (C_1 \circ C_4)$
- $C_3 = C_2 \cap [-\infty, 100]$
- $C_4 = C_3 \circ [1, 1]$
- $C_5 = C_2 \cap [101, +\infty]$
Program Example

- Consider the following flowchart program and interval analysis:

  ![Flowchart Diagram]

- Let us now define the narrowing operation $\triangle$ over intervals as
  - $[\ , \ ]$ is the null element of $\triangle$
  - $[i, j] \triangle [k, l] = [
    \begin{align*}
    &\text{if } i = -\infty \text{ then } k \text{ else } \min(i, k), \\
    &\text{if } j = +\infty \text{ then } l \text{ else } \max(j, l)
    \end{align*}
  ]$

- We now substitute the equation for $C_2$ with a new one that uses narrowing.

  - $C_0 = [\infty, +\infty]$
  - $C_1 = [1, 1]$
  - $C_2 = C_2 \triangle (C_1 \circ C_4)$
  - $C_3 = C_2 \cap [1, 100]$
  - $C_4 = C_3 + [1, 1]$
  - $C_5 = C_2 \cap [101, +\infty]$
Program Example

• Consider the following flowchart program and interval analysis:

```
 x := 1
```

```
x ≤ 100
```

```
x := x + 1
```

```
false
```

```
true
```

Let us now define the **narrowing** operation $\triangle$ over intervals as

• $[i, j]$ is the null element of $\triangle$,

• $[i, j] \triangle [k, l] = [
  \begin{array}{l}
    \text{if } i = -\infty \text{ then } k \\
    \text{else } \min(i, k)
  \end{array},
  \begin{array}{l}
    \text{if } j = +\infty \text{ then } l \\
    \text{else } \max(j, l)
  \end{array}].$

We now substitute the equation for $C_2$ with a new one that uses narrowing.

• $C_0 = [-\infty, +\infty]$

• $C_1 = [1, 1]$

• $C_2 = \mathcal{C}_2 \triangledown (C_1 \circ C_4)$

• $C_3 = C_2 \cap [101, +\infty]$

• $C_4 = C_3 + [1, 1]$

• $C_5 = C_2 \cap [101, +\infty]$
Applications of Abstract Interpretation

- There exist libraries of abstract domains: e.g., APRON.

- **Value analysis**: represents values of variables by intervals/DBMs/octagons/polyhedra/... of possible values. Discovers invariants, can be used for checking various safety properties (overflows, buffer overflows, ...)
  - e.g., FramaC.

- **Heap analysis**: overapproximates graphs representing the memory heap. This can be used, e.g., for memory leak detection.
  - CINV, Facebook Infer SL, Forester, MemCAD, Predator, Space Invader, TVLA, ...

- **Worst-case execution time (WCET) analysis**: may involve the analysis of the cache behaviour, the pipelines, etc.
  - AbsInt, ...

- and many more ...
  - Astrée, ECLAIR, Fluctuat, Polyspace, Stanford Checker, ...
Polyhedral Analysis
Polyhedral Analysis

- An abstract interpretation-based approach of automatic discovery of relations among program variables expressible as linear inequations,
  - this can be seen as a generalization of the interval analysis.
- Let \( \vec{x} = x_1, \ldots, x_n \in \mathbb{R}^n \) be the variables of a program. We can use a convex polyhedron \( P \subseteq \mathbb{R}^n \) to represent a set of assignments to \( \vec{x} \).
- We use convex polyhedra because operations on them are reasonably efficient (a set \( C \subseteq \mathbb{R}^n \) is convex iff \( \forall x_1, x_2 \in C, \forall 0 \leq \lambda \leq 1 : \lambda x_1 + (1 - \lambda)x_2 \in C \)).

- **Use:** compile-time determination of bounds of variables, discovery of constants, ...
• We use two dual ways to represent a convex polyhedron:
  • by a system of linear inequations, and
  • by the frame of the polyhedron.

• We can alter between these representations (with some overhead).

• Efficient execution of different operations require different representation.
System of Linear Inequations

- Let \( \vec{x} = x_1, \ldots, x_n \in \mathbb{R}^n \) be the variables of a program. Given a finite set of \( m \) linear inequations over \( \vec{x} \) of the form

\[
\left\{ \sum_{i=1}^{n} a_{ji} x_i \leq b_j \mid 1 \leq j \leq m \right\}
\]

or equivalently using vectors and matrices as

\[
\vec{x} \cdot \vec{A} \leq \vec{b},
\]

we can geometrically interpret the solutions of the inequations as a convex polyhedron in \( \mathbb{R}^n \) defined by the intersection of halfspaces corresponding to each inequality.
The Frame of a Convex Polyhedron

• A convex polyhedron $P$ can also be characterized by its frame $F = (V, R, L)$:
  
  • **Vertices** $V$: points $\vec{v}$ of a polyhedron $P$ which are not convex combinations of other points $\{\vec{w_1}, \ldots, \vec{w_m}\}$ of $P$,

$$
\left( \vec{v} = \sum_{i=1}^{m} \lambda_i \vec{w_i} \right) \land (\forall 1 \leq i \leq m : (\vec{w_i} \in P \land \lambda_i \geq 0)) \land \left( \sum_{i=1}^{m} \lambda_i = 1 \right) \Rightarrow
$$

$$
\Rightarrow (\forall 1 \leq i \leq m : (\lambda_i = 0 \lor \vec{w_i} = \vec{v})).
$$

• **Convex hull**: the set of all convex combinations of $V$.
The Frame of a Convex Polyhedron

- **Extreme rays** $R$: rays $\vec{r}$ of $P$ (i.e. vectors such that there exists a half-line parallel to $\vec{r}$ and entirely included in $P$) which are not positive combinations of other rays $\vec{s}_1, \ldots, \vec{s}_p$ of $P$:

$$\left(\vec{r} = \sum_{i=1}^{p} \mu_i \vec{s}_i \land (\forall 1 \leq i \leq p : \mu_i \in \mathbb{R}^+)\right) \Rightarrow (\forall 1 \leq i \leq p : (\mu_i = 0 \lor \vec{s}_i = \vec{r})).$$
The Frame of a Convex Polyhedron

- **Extreme rays** $R$: rays $\vec{r}$ of $P$ (i.e. vectors such that there exists a half-line parallel to $\vec{r}$ and entirely included in $P$) which are not positive combinations of other rays $\vec{s}_1, \ldots, \vec{s}_p$ of $P$:

$$\vec{r} = \sum_{i=1}^{p} \mu_i \vec{s}_i \wedge (\forall 1 \leq i \leq p : \mu_i \in \mathbb{R}^+) \Rightarrow (\forall 1 \leq i \leq p : (\mu_i = 0 \lor \vec{s}_i = \vec{r})).$$

- **Lines** $L$: vectors $\vec{l}$ such that both $\vec{l}$ and $-\vec{l}$ are rays of $P$:

$$\forall x \in P, \forall \mu \in \mathbb{R} : x + \mu \vec{l} \in P.$$
The Frame of a Convex Polyhedron

- **Extreme rays** $R$: rays $\vec{r}$ of $P$ (i.e. vectors such that there exists a half-line parallel to $\vec{r}$ and entirely included in $P$) which are not positive combinations of other rays $\vec{s}_1, \ldots, \vec{s}_p$ of $P$:

$$\vec{r} = \sum_{i=1}^{p} \mu_i \vec{s}_i \land (\forall 1 \leq i \leq p : \mu_i \in \mathbb{R}^+) \Rightarrow (\forall 1 \leq i \leq p : (\mu_i = 0 \lor \vec{s}_i = \vec{r})).$$

- **Lines** $L$: vectors $\vec{l}$ such that both $\vec{l}$ and $-\vec{l}$ are rays of $P$:

$$\forall \vec{x} \in P, \forall \mu \in \mathbb{R} : \vec{x} + \mu \vec{l} \in P.$$  

- Every point $\vec{x}$ of the polyhedron $P$ defined by the frame $F = (V, R, L)$ can be obtained from $V$, $R$ and $L$:

$$\vec{x} = \sum_{i=1}^{\sigma} \lambda_i \vec{v}_i + \sum_{j=1}^{\rho} \mu_j \vec{r}_j + \sum_{k=1}^{\delta} \nu_k \vec{l}_k$$

where $0 \leq \lambda_1, \ldots, \lambda_{\sigma} \leq 1, \sum_{i=1}^{\sigma} \lambda_i = 1, \mu_1, \ldots, \mu_{\rho} \in \mathbb{R}^+, \nu_1, \ldots, \nu_{\delta} \in \mathbb{R}$. 

24/30
Example of a Convex Polyhedron
Example of a Convex Polyhedron

System of linear inequations

\[ x_2 \geq 1 \]
Example of a Convex Polyhedron

System of linear inequations

\[ x_2 \geq 1 \]
\[ x_1 \geq 2 \]
Example of a Convex Polyhedron

System of linear inequations

\[
\begin{align*}
x_2 & \geq 1 \\
x_1 & \geq 2 \\
x_1 + 2x_2 & \geq 6
\end{align*}
\]
Example of a Convex Polyhedron

System of linear inequations

\[ \begin{align*} 
  x_2 & \geq 1 \\
  x_1 & \geq 2 \\
  x_1 + 2x_2 & \geq 6 \\
  x_1 - 2x_2 & \geq -6 
\end{align*} \]
Example of a Convex Polyhedron

System of linear inequations

\[
\begin{align*}
    x_2 & \geq 1 \\
    x_1 & \geq 2 \\
    x_1 + 2x_2 & \geq 6 \\
    x_1 - 2x_2 & \geq -6
\end{align*}
\]

Frame of a polyhedron

\[ F = (V, R, L) \]
Example of a Convex Polyhedron

System of linear inequations

\[
\begin{align*}
    x_2 & \geq 1 \\
    x_1 & \geq 2 \\
    x_1 + 2x_2 & \geq 6 \\
    x_1 - 2x_2 & \geq -6
\end{align*}
\]

Frame of a polyhedron

\[
F = (V, R, L)
\]

\[
V = \{ \vec{v}_1 = [4, 1], \vec{v}_2 = [2, 2], \vec{v}_3 = [2, 4] \}\]
Example of a Convex Polyhedron

System of linear inequations

\[
\begin{align*}
    x_2 & \geq 1 \\
    x_1 & \geq 2 \\
    x_1 + 2x_2 & \geq 6 \\
    x_1 - 2x_2 & \geq -6
\end{align*}
\]

Frame of a polyhedron

\[F = (V, R, L)\]

\[V = \{ \vec{v}_1 = [4, 1], \vec{v}_2 = [2, 2], \vec{v}_3 = [2, 4] \}\]

\[R = \{ \vec{r}_1 = (1, 0), \vec{r}_2 = (2, 1) \}\]
Example of a Convex Polyhedron

System of linear inequations

\[
\begin{align*}
x_2 & \geq 1 \\
x_1 & \geq 2 \\
x_1 + 2x_2 & \geq 6 \\
x_1 - 2x_2 & \geq -6
\end{align*}
\]

Frame of a polyhedron

\[F = (V, R, L)\]

\[V = \{\vec{v}_1 = [4, 1], \vec{v}_2 = [2, 2], \vec{v}_3 = [2, 4]\}\]

\[R = \{\vec{r}_1 = (1, 0), \vec{r}_2 = (2, 1)\}\]

\[L = \emptyset\]
Transformations of Convex Polyhedra

- Different types of nodes of the flowchart representation of a program perform distinct transformation on the polyhedron. The number of input and output polyhedra differs according to the type of the node.

- **Entries:** create a polyhedron according to restraints on the input values of variables (in case there are none for variable $x_i$, the polyhedron is unbounded in $i$-th dimension).
Assignments

• Performed operations vary according to assigned expression:
  
  • **Non-linear expression** $x_i := \text{<non-linear expression>}$: because these cannot be represented using convex polyhedra, any restraint on $x_i$ is dropped (we add line $d\cdot$ to frame such that $d_i = 1$ and $\forall 1 \leq j \leq n, i \neq j : d_j = 0$).
  
  • **Linear expression** $x_i := \sum_{j=1}^{n} a_j x_j + b$: the frame $F' = (V', R', L')$ of the output polyhedron can be computed from the frame $F = (V, R, L)$ of the input as
Assignments

- Performed operations vary according to assigned expression:
  - **Non-linear expression** \( x_i := \langle \text{non-linear expression} \rangle \): because these cannot be represented using convex polyhedra, any restraint on \( x_i \) is dropped (we add line \( \vec{d} \) to frame such that \( d_i = 1 \) and \( \forall 1 \leq j \leq n, i \neq j : d_j = 0 \)).
  - **Linear expression** \( x_i := \sum_{j=1}^{n} a_j x_j + b \): the frame \( F' = (V', R', L') \) of the output polyhedron can be computed from the frame \( F = (V, R, L) \) of the input as
    - \( V' = \{ \vec{v}_1', \ldots, \vec{v}_\sigma' \} \) where \( \vec{v}_j' \) is defined by \( v_{ji}' = \vec{a} \vec{v}_j + b \) and \( v_{jm}' = v_{jm} \) where \( \forall 1 \leq m \leq \sigma, m \neq i \).
• Performed operations vary according to assigned expression:

  • **Non-linear expression** $x_i := \langle \text{non-linear expression} \rangle$: because these cannot be represented using convex polyhedra, any restraint on $x_i$ is dropped (we add line $\vec{d}$ to frame such that $d_i = 1$ and $\forall 1 \leq j \leq n, i \neq j : d_j = 0$).

  • **Linear expression** $x_i := \sum_{j=1}^{n} a_j x_j + b$: the frame $F' = (V', R', L')$ of the output polyhedron can be computed from the frame $F = (V, R, L)$ of the input as

    - $V' = \{ \vec{v}'_1, \ldots, \vec{v}'_\sigma \}$ where $\vec{v}'_j$ is defined by $\vec{v}'_{ji} = \vec{a} \vec{v}_j + b$ and $\vec{v}'_{jm} = \vec{v}_jm$ where $\forall 1 \leq m \leq \sigma, m \neq i$.

    - $R' = \{ \vec{r}'_1, \ldots, \vec{r}'_\rho \}$ where $\vec{r}'_j$ is defined by $\vec{r}'_{ji} = \vec{a} \vec{r}_j$ and $\vec{r}'_{jm} = \vec{r}_jm$ for $\forall 1 \leq m \leq \rho, m \neq i$. 

 27/30
Assignments

- Performed operations vary according to assigned expression:
  - Non-linear expression \( x_i := \langle \text{non-linear expression} \rangle \): because these cannot be represented using convex polyhedra, any restraint on \( x_i \) is dropped (we add line \( \vec{d} \) to frame such that \( d_i = 1 \) and \( \forall 1 \leq j \leq n, i \neq j : d_j = 0 \)).
  - Linear expression \( x_i := \sum_{j=1}^{n} a_j x_j + b \): the frame \( F' = (V', R', L') \) of the output polyhedron can be computed from the frame \( F = (V, R, L) \) of the input as
    - \( V' = \{ \vec{v}'_1, \ldots, \vec{v}'_\sigma \} \) where \( \vec{v}'_j \) is defined by \( \vec{v}'_{ji} = \vec{a} \vec{v}_j + b \) and \( \vec{v}'_{jm} = \vec{v}_{jm} \) where \( \forall 1 \leq m \leq \sigma, m \neq i \).
    - \( R' = \{ \vec{r}'_1, \ldots, \vec{r}'_\rho \} \) where \( \vec{r}'_j \) is defined by \( \vec{r}'_{ji} = \vec{a} \vec{r}_j \) and \( \vec{r}'_{jm} = \vec{r}_{jm} \) for \( \forall 1 \leq m \leq \rho, m \neq i \).
    - \( L' = \{ \vec{l}'_1, \ldots, \vec{l}'_\delta \} \) where \( \vec{l}'_j \) is defined by \( \vec{l}'_{ji} = \vec{a} \vec{l}_j \) and \( \vec{l}'_{jm} = \vec{l}_{jm} \) for \( \forall 1 \leq m \leq \delta, m \neq i \).
Tests

- The input polyhedron $P$ is transformed into two output polyhedra: $P_t$ for the true branch and $P_f$ for the false branch.
- For a Boolean condition $C$, it needs to hold that $P_t \supseteq P \cap T_C$, $P_f \supseteq P \setminus T_C$ where $T_C$ is the subset of $\mathbb{R}^n$ such that each point of $T_C$ satisfies $C$ (the right sides of the inclusions are not necessarily convex polyhedra).
- The operation that is performed varies according to the Boolean condition of the test:
Tests

- The input polyhedron $P$ is transformed into two output polyhedra: $P_t$ for the true branch and $P_f$ for the false branch.

- For a Boolean condition $C$, it needs to hold that $P_t \supseteq P \cap T_C$, $P_f \supseteq P \setminus T_C$ where $T_C$ is the subset of $\mathbb{R}^n$ such that each point of $T_C$ satisfies $C$ (the right sides of the inclusions are not necessarily convex polyhedra).

- The operation that is performed varies according to the Boolean condition of the test:
  - Non-linear tests: $P_t = P_f = P$ (can be refined for some cases).
Tests

- The input polyhedron $P$ is transformed into two output polyhedra: $P_t$ for the true branch and $P_f$ for the false branch.

- For a Boolean condition $C$, it needs to hold that $P_t \supseteq P \cap T_C, P_f \supseteq P \setminus T_C$ where $T_C$ is the subset of $\mathbb{R}^n$ such that each point of $T_C$ satisfies $C$ (the right sides of the inclusions are not necessarily convex polyhedra).

- The operation that is performed varies according to the Boolean condition of the test:
  - Non-linear tests: $P_t = P_f = P$ (can be refined for some cases).
  - Linear equality tests: Boolean condition $\vec{a} \vec{x} = b$ defines a hyperplane $H$. If $P$ is included in $H$, then $P_t = P, P_f = \emptyset$. If $P$ is not included in $H$, then $P_t = P \cap H$ and $P_f = P$. 

28/30
Tests

- The input polyhedron $P$ is transformed into two output polyhedra: $P_t$ for the true branch and $P_f$ for the false branch.

- For a Boolean condition $C$, it needs to hold that $P_t \supseteq P \cap T_C, P_f \supseteq P \setminus T_C$ where $T_C$ is the subset of $\mathbb{R}^n$ such that each point of $T_C$ satisfies $C$ (the right sides of the inclusions are not necessarily convex polyhedra).

- The operation that is performed varies according to the Boolean condition of the test:
  - Non-linear tests: $P_t = P_f = P$ (can be refined for some cases).
  - Linear equality tests: Boolean condition $C : \vec{a} \vec{x} = b$ defines a hyperplane $H$. If $P$ is included in $H$, then $P_t = P, P_f = \emptyset$. If $P$ is not included in $H$, then $P_t = P \cap H$ and $P_f = P$.
  - Linear inequality tests: for Boolean condition $\vec{a} \vec{x} \leq b$, the outputs are $P_t = P \cap \vec{a} \vec{x} \leq b$ and $P_f = P \cap \vec{a} \vec{x} \geq b$. 
Junctions

- Junctions correspond to merge of several program paths so the output polyhedron $P$ is union of all input polyhedra $P_i$. It is computed according to the kind of the junction:
Junctions

- Junctions correspond to merge of several program paths so the output polyhedron $P$ is union of all input polyhedra $P_i$. It is computed according to the kind of the junction:
  - **Simple junctions**: for input polyhedra $P_1, \ldots, P_m$, we compute the convex hull of $P_1 \cup \cdots \cup P_m$. 
  - **Loop junctions**: for input polyhedra $P_1, \ldots, P_m$, let $Q$ be the convex hull of $P_1 \cup \cdots \cup P_m$. Then $P' = P \triangledown Q$ is the convex polyhedron consisting of linear restraints of $P$s satisfied by every element of $Q$. 

Junctions

- Junctions correspond to merge of several program paths so the output polyhedron $P$ is union of all input polyhedra $P_i$. It is computed according to the kind of the junction:
  - **Simple junctions**: for input polyhedra $P_1, \ldots, P_m$, we compute the convex hull of $P_1 \cup \cdots \cup P_m$.
  - **Loop junctions**: for input polyhedra $P_1, \ldots, P_m$, let $Q$ be the convex hull of $P_1 \cup \cdots \cup P_m$. Then $P' = P \triangledown Q$ is the convex polyhedron consisting of linear restraints of $P$ satisfied by every element of $Q$. 
Example

I := 2, J := 0;
L:
if ... then
   I := I + 4
else
   J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ...
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ... then
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;
Example

\begin{verbatim}
I := 2, J := 0;
L:
  if ... then
    I := I + 4
  else
    J := J + 1, I := I + 2;
fi;
go to L;
\end{verbatim}
Example

I := 2, J := 0;
L:
if ... then
    I := I + 4
else
    J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ... then
    I := I + 4
else
    J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ... then
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;
Example

\begin{align*}
I &:= 2, J := 0; \\
L: & \\
\text{if ... then} & \\
\quad I := I + 4 & \\
\text{else} & \\
\quad J := J + 1, I := I + 2; & \\
\text{fi; } & \\
\text{go to L;}
\end{align*}
Example

I := 2, J := 0;
L:
  if ... then
    I := I + 4
  else
    J := J + 1, I := I + 2;
  fi;
  go to L;
Example

I := 2, J := 0;
L:
if ...
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ... then
    I := I + 4
else
    J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ... then
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;
Example

I := 2, J := 0;
L:
if ... then
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
goto L;
Example

I := 2, J := 0;
L:
\[ \text{if \ldots then} \]
\[ \text{I := I + 4} \]
\[ \text{else} \]
\[ \text{J := J + 1, I := I + 2;} \]
\[ \text{fi;} \]
\[ \text{go to L;} \]
Example

I := 2, J := 0;
L:
  if ... then
    I := I + 4
  else
    J := J + 1, I := I + 2;
  fi;
go to L;
I := 2, J := 0;
L:
if ... then
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;

$P_0^3 = P_3^3 = P_5^3$
Example

\[
I := 2, J := 0; \\
L: \\
if \ldots \text{ then} \\
\quad I := I + 4 \\
\text{else} \\
\quad J := J + 1, I := I + 2; \\
\text{fi;} \\
go \text{ to } L;
\]
Example

\[I := 2, \quad J := 0;\]

\[L: \]

\[if \ldots \ then\]
\[\quad I := I + 4\]
\[else\]
\[\quad J := J + 1, \quad I := I + 2;\]
\[fi;\]
\[go to L;\]
Example

\begin{verbatim}
I := 2, J := 0;
L:
if ... then
  I := I + 4
else
  J := J + 1, I := I + 2;
fi;
go to L;
\end{verbatim}