
Cours MALG & MOVEX

MALG

Analyse des programmes

Dominique Méry

Telecom Nancy, Université de Lorraine

Année universitaire 2023-2024

- Widening
Narrowing

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

Problem of Software Safety : Ariane 5 – Explosion after launch (1996)

- ▶ The Ariane 5 rocket explodes 37 seconds after launch.
- ▶ Loss of the rocket and its valuable cargo (500 million euros).
- ▶ Software error : Incorrect type conversion : a floating point number (64 bits) was converted to an integer (16 bits) without checking that the value fit in the space.
- ▶ Result : overflow \Rightarrow exception \Rightarrow guidance system shutdown \Rightarrow disorientation \Rightarrow self-destruction.
- ▶ The software came from Ariane 4 and was reused without modification, even though the speeds were much higher in Ariane 5.

- ▶ Several patients received massive doses of radiation (up to 100 times the intended dose).
- ▶ At least six patients were seriously irradiated, with several deaths.
- ▶ Software error : Race conditions between two internal processes : in some cases, a mechanical safety feature was disabled but the software thought it was active.
- ▶ Insufficient safeguards to check the actual status of the device.
- ▶ The software assumed that certain errors were 'impossible'. No alarm was raised.

- ▶ A battery of Patriot missiles failed to intercept an Iraqi Scud missile.
- ▶ Consequences : 28 American soldiers killed in Dhahran, Saudi Arabia.
- ▶ Software error* : Rounding error in floating point. The system used a time counter that accumulated a small error every second.
- ▶ After 100 hours of operation without restarting, the error had accumulated \Rightarrow incorrect calculation of the missile's position \Rightarrow interception failure.
- ▶ This precision bug caused an error of 0.34 seconds ... enough to miss a high-speed missile.

- ▶ Objectives of static program analysis
 - to prove properties about the run-time behaviour of a program
 - in a fully automatic way ie without interaction
 - without actually executing the program
- ▶ Applications
 - code optimisation
 - error detection (array out of bound access, null pointers)
 - proof support for generation of invariant

- ▶ Static Analysis *computes* approximations
- ▶ Abstract Interpretation (AI) provides a mathematical framework for relating *approximations*
- ▶ Properties of programs are generally non computable :
 - the halting problem is undecidable
 - Model checking is computing over finite structures
 - Proof assistant may be useful for proving partial correctness or total correctness by applying induction principles (see Event B)
 - AI provides another solution by transferring results from a *concret* framework to an *abstract* structure

- ▶ $\mathcal{CS}(P)$ is the concrete semantics of a program P : the set of reachable states of P .
- ▶ $\mathcal{AS}(P)$ is the approximation of $\mathcal{CS}(P)$: $\mathcal{CS}(P) \subseteq \mathcal{AS}(P)$.
- ▶ $\mathcal{CS}(P)$ is generally not computable and we will seek for *computable* approximation or abstract semantics $\mathcal{AS}(P)$.
- ▶ Problems : $\mathcal{AS}(P)$ may *loose* the expression of properties.

- ▶ φ is a program property stating the possible bugs or errors which we want to avoid.
- ▶ $\mathcal{CS}(P)$ is the concrete semantics of a program P : the set of reachable states of P .
- ▶ $\mathcal{AS}(P)$ is the approximation of $\mathcal{CS}(P)$: $\mathcal{CS}(P) \subseteq \mathcal{AS}(P)$.
- ▶ Case 1 : $\mathcal{CS}(P) \cap \varphi = \emptyset$ and $\mathcal{AS}(P) \cap \varphi = \emptyset$
- ▶ Case 2 : $\mathcal{CS}(P) \cap \varphi \neq \emptyset$ and $\mathcal{AS}(P) \cap \varphi \neq \emptyset$
- ▶ Case 3 : $\mathcal{CS}(P) \cap \varphi = \emptyset$ and $\mathcal{AS}(P) \cap \varphi \neq \emptyset$

- ▶ Case 1 : $\mathcal{CS}(P) \cap \varphi = \emptyset$ and $\mathcal{AS}(P) \cap \varphi = \emptyset$:
 - P is safe with respect to φ and no error specified by φ is possible for P .
 - Checking is computable on the approximation
- ▶ Case 2 : $\mathcal{CS}(P) \cap \varphi \neq \emptyset$ and $\mathcal{AS}(P) \cap \varphi \neq \emptyset$:
 - An error is detected on the approximation and on the concrete semantics.
 - P is unsafe with respect to φ
 - and an error is detected by the analyser.
- ▶ Case 3 : $\mathcal{CS}(P) \cap \varphi = \emptyset$ and $\mathcal{AS}(P) \cap \varphi \neq \emptyset$:
 - P is safe with respect to φ
 - but an error is detected by the analyser
 - A false alarm is provided by the analyzer
 - Approximation is over-approximating P with respect to φ
 - The analysis should be refined

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

- ▶ a web interface to the Interproc analyzer connected to the APRON Abstract Domain Library
- ▶ Analysis of programs using different *abstract domains*
- ▶ <http://pop-art.inrialpes.fr/interproc/>
- ▶ developed by Antoine Miné and his team.

Example 1 : Increment of a value

```
proc incr (x:int) returns (y:int)
begin
  y = x+1;
end
```

```
var i:int;
begin
  i = 0;
  while (i<=10) do
    i = incr(i);
  done;
end
```

Example 1 : Increment of a value

```
proc incr (x : int) returns (y : int) var ;
begin
  /* (L3 C5) [|x>=0; -x+10>=0|] */
  y = x + 1; /* (L4 C10)
                                     [|x>=0; -x+10>=0; y-1>=0; -y+11>=0|] */
end

var i : int;
begin
  /* (L8 C5) top */
  i = 0; /* (L9 C8) [|i>=0; -i+11>=0|] */
  while i <= 10 do
    /* (L10 C18) [|i>=0; -i+10>=0|] */
    i = incr(i); /* (L11 C16)
                                     [|i-1>=0; -i+11>=0|] */
  done; /* (L12 C7) [|i-11=0|] */
end
```

Example 2 : Division

```
var Q : int, R : int, X : int, Y : int;  
begin  
  
    Q = 0;  
    R = Y;  
  
    while R >= Y do  
        Q = Q + 1;  
        R = R - Y;  
    done;  
  
end
```

Example 2 : Division

Annotated program after forward analysis

```
var Q : int, R : int, X : int, Y : int;  
begin  
  /* (L3 C5) top */  
  Q = 0; /* (L5 C8) [|Q=0|] */  
  R = Y; /* (L6 C8) [|Q>=0|] */  
  while R >= Y do  
    /* (L8 C20) [|Q>=0|] */  
    Q = Q + 1; /* (L9 C17) [|Q-1>=0|] */  
    R = R - Y; /* (L10 C17) [|Q-1>=0|] */  
  done; /* (L11 C10) [|Q>=0|] */  
end
```

Example 3 : modified division

```
var Q : int, R : int, X : int, Y : int;  
begin  
  
  Q = 0;  
  R = Y;  
  if Y > 0 then  
    while R >= Y do  
      Q = Q + 1;  
      R = R - Y;  
    done;  
  else  
    skip;  
  endif;  
end
```


Example 3 : modified division

Annotated program after forward analysis

Annotated program after forward analysis

```
var Q : int, R : int, X : int, Y : int;
begin
  /* (L3 C5) top */
  Q = 0; /* (L5 C8) [|Q=0|] */
  R = Y; /* (L6 C8) [|Q=0|] */
  if Y > 0 then
    /* (L7 C15) [|Q>=0; Y-1>=0|] */
    while R >= Y do
      /* (L8 C20) [|Q>=0; R-1>=0; Y-1>=0|] */
      Q = Q + 1; /* (L9 C17)
                  [|Q-1>=0; R-1>=0; Y-1>=0|] */
      R = R - Y; /* (L10 C17)
                  [|Q-1>=0; Y-1>=0|] */
    done; /* (L11 C10) [|Q>=0; Y-1>=0|] */
  else
    /* (L12 C6) [|Q=0; -Y>=0|] */
    skip; /* (L13 C9) [|Q=0; -Y>=0|] */
  endif; /* (L14 C8) [|Q>=0|] */
end
```

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis**
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- 17 Conclusion

- ▶ absence of run time errors
 - ▶ detecting variables used before initialisation.
- ▶ Data flow analysis
 - ▶ Abstract interpretation
 - ▶ Use of property-preserving abstractions
 - ▶ Programs are interpreted in abstractions

- ▶ Sign analysis is used to determine the sign of variables
- ▶ x is an integer variable and has the following possible *abstract* states :
 - $x > 0$
 - $x \geq 0$
 - $x = 0$
 - $x < 0$
 - $x \leq 0$
 - $x \neq 0$

```
ℓ0 :  
y := -11;  
ℓ0 :  
IF   x < y  THEN  
    ℓ1 :  
    z := y;  
    ℓ2 :  
ELSE  
    ℓ3 :  
    z := x;  
    ℓ4 :  
FI  
ℓ5 :
```

```
ℓ0 :  
y := -11;  
ℓ0 : y < 0  
IF x < y THEN  
    ℓ1 :  
    z := y;  
    ℓ2 :  
ELSE  
    ℓ3 :  
    z := x;  
    ℓ4 :  
FI  
ℓ5 :
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
     $\ell_1 : y < 0 \quad x < 0$   
     $z := y;$   
     $\ell_2 :$   
ELSE  
     $\ell_3 :$   
     $z := x;$   
     $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 :$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```



```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := x;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := x;$   
   $\ell_4 : y < 0 \quad x \in \mathbb{Z}$   
FI  
 $\ell_5 :$ 
```

Simple example

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := x;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := x;$   
   $\ell_4 : y < 0 \quad x \in \mathbb{Z}$   
FI  
 $\ell_5 : y < 0 \quad x \in \mathbb{Z} \quad z \in \mathbb{Z}$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := x;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := x;$   
   $\ell_4 : y < 0 \quad x \in \mathbb{Z}$   
FI  
 $\ell_5 : y < 0 \quad x \in \mathbb{Z} \quad z \in \mathbb{Z}$ 
```

Result

$y < 0 \quad x \in \mathbb{Z} \quad z < 0$ means that $z < 0$ is an information resulting from the analysis over abstract domain of signs.

- ▶ \mathcal{MS} is $(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \dots, r_n\})$
- ▶ $\text{NEXT} \stackrel{\text{def}}{=} r_0 \vee \dots \vee r_n.$
- ▶ S is a safety property, when
 $\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow x \in S.$
- ▶ S is a safety property for \mathcal{MS} if, and only if, $\text{REACHABLE}(\mathcal{MS}) \subseteq S$

Characterisation of $\text{REACHABLE}(\mathcal{MS}) \subseteq S$ as a fixed-point

$(\mathcal{P}(\text{VALS}), \subseteq, \emptyset, \cup, \cap)$ is a complete lattice and

$F \in \mathcal{P}(\text{VALS}) \longrightarrow \mathcal{P}(\text{VALS})$ is defined as :

$F(X) = \{x | x \in \text{VALS} \wedge \text{INIT}(x)\} \cup X \cup \longrightarrow [X]$ and satisfies the following properties :

- ▶ F is a monotonic function.
- ▶ $\text{REACHABLE}(\mathcal{MS}) = \mu F$
- ▶ μF is defined as follows :
 - $F^0 = \emptyset$
 - $F^{i+1} = F(F^i), \forall i \in \mathbb{N}$
 - $\mu F = \text{Sup}\{F^i | i \in \mathbb{N}\}$
 - For any safety property S , $\mu F \subseteq S$.

Computing the least fixed-point over a finite lattice

INPUT $F \in T \longrightarrow T$
OUTPUT $result = \mu.F$
VARIABLES $x, y \in T, i \in \mathbb{N}$
 $\ell_0 : \{x, y \in T\}$
 $x := \perp;$
 $y := \perp;$
 $i := 0;$
 $\ell_{11} : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T) \wedge i = 0\};$
WHILE $i \leq Card(T)$
 $\ell_1 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)\};$
 $x := F(x);$
 $\ell_2 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)\};$
 $y := x \sqcup y;$
 $\ell_3 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0; k=i+1} F^k \wedge i \leq Card(T)\};$
 $i := i+1;$
 $\ell_4 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)+1\};$
OD;
 $\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i = Card(T)+1\};$
 $result := y;$
 $\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i = Card(T)+1 \wedge result = y\};$

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

- ▶ **Syntax of programs** ($P \in PL$) defines the class of programs for applying the analysis@
- ▶ **Semantics** ($\llbracket P \rrbracket$) for each program ($P \in PL$) which are defined by fixed-point equations and are organised through a hierarchy of semantics : trace semantics, wp semantics, denotational semantics, relational semantics ...
- ▶

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

- Abstract interpretation of programs is an approximation of programs semantics
- Correctness proof of the abstract interpretation requires the existence of the standard semantics describing the possible behaviours of programs during their execution.
- The class of properties of program executions is defined by a collecting semantics or static semantics.
- The collecting semantics can be an instrumented version of the standard semantics to gather information about programs executions.
- or the standard semantics reduced to essentials in order to ignore irrelevant details about program execution.
- The collecting semantics provides a sound and relatively complete proof method for the considered class of properties.
- It can be used subsequently as a reference semantics for proving the correctness of all other approximate semantics for that class of properties.
- The abstract semantics usually considers effectively computable properties of programs.
- The soundness of this abstract semantics is proved with respect to the collecting semantics.

- ▶ programs properties are expressed as sets of finite partial execution traces.
- ▶ forward collecting semantics : traces are finite sequences of states, starting with an initial state and such that two consecutive states on the trace satisfy \longrightarrow .
- ▶ backward collecting semantics : traces are finite sequences of states, starting with a final state and such that two consecutive states on the trace satisfy \longrightarrow^{-1} .
- ▶ the descendant states of the initial states are the collecting semantics.
- ▶ The abstraction consists in approximating a set of forward traces by the set of states which occur on any one of these traces.

Examples

- ▶ Computation Traces of Program
- ▶ Transitive Closure of the program transition relation
- ▶ Set of states

The collecting semantics is the semantics which is interesting our analysis

and we will consider as collecting semantics the set of states.

▶ *Collecting semantics*

- Static analysis of a program states a property of program executions defined by a *standard* semantics.
- Defining a so-called *collecting* semantics defining the strongest static property of interest
- Collecting semantics defines the class of static analysis, which approximates it
- State properties are subsets of $\mathbb{I} \times \mathbb{I} \times \mathbb{I} \times \mathbb{I}$ and abstract interpretation executes programs on these properties

▶ Approximation

- Spaces of values should be restricted to computable entities
- Over-approximation of concrete properties

$Expr$	$::=$	v $?$ x $Expr\ op\ Expr$	$v \in \mathbb{Z}$ $x \in \mathbb{V}$ $op \in \{+, -, \times, /\}$
$cond$	$::=$	$Expr\ relop\ Expr$ not $cond$ $cond$ and $cond$	$relop \in \{<, \leq, >, \geq, =, \neq\}$
$stmt$	$::=$	$\ell[x := Expr]$ $\ell[skip]$ if $\ell[cond]$ then $stmt$ else $stmt$ end if while $\ell[cond]$ do $stmt$ end do $stmt; stmt$	$\ell \in \mathbb{C}$

Two examples of annotated programs

```
 $\ell_0[X := 0];$   
 $\ell_1[Y := Y + X];$   
 $\ell_2[skip]$   
 $\ell_3[X := Y];$ 
```

```
 $\ell_0[Q := 0];$   
 $\ell_1[R := X];$   
IF  $\ell_5[Y > 0]$   
    WHILE  $\ell_2[R \geq Y]$   
         $\ell_3[Q := Q + 1];$   
         $\ell_4[R := R - Y]$   
    ENDWHILE  
ELSE  
     $\ell_6[skip]$   
ENDIF
```


► Semantic Domains

$$Mem \stackrel{def}{=} \mathbb{V} \longrightarrow \mathbb{Z}$$

$$States \stackrel{def}{=} \mathbb{C} \times Mem$$

► Semantics for Expressions

$$\mathcal{E}[[v]](m) \in \mathcal{P}(\mathbb{Z}), e \in Expr, m \in Mem, x \in \mathbb{V}, op \in \{+, -, \times, /\}$$

$$\mathcal{E}[[v]](m) \stackrel{def}{=} \{v\}$$

$$\mathcal{E}[[?]](m) \stackrel{def}{=} \mathbb{Z}$$

$$\mathcal{E}[[x]](m) \stackrel{def}{=} \{m(x)\}$$

$$\mathcal{E}[[e_1 \text{ op } e_2]](m) \stackrel{def}{=} \{v \mid \exists ve_1, ve_2. \left(\begin{array}{l} ve_1 \in \mathcal{E}[[e_1]](m) \\ ve_2 \in \mathcal{E}[[e_2]](m) \\ v = ve_1 \text{ o } ve_2 \end{array} \right) \}$$

► Semantics for conditions

$\mathcal{C}[\![cond]\!](m) \in \mathcal{P}(\mathbb{B})$, $cond \in Cond, m \in Mem, x \in \mathbb{V}$,
 $op \in \{+, -, \times, /\}$

$$\begin{aligned} tt \in \mathcal{C}[\![e_1 \text{ relop } e_2]\!](m) &\stackrel{def}{=} \exists v_1, v_2. \left(\begin{array}{l} v_1 \in \mathcal{E}[\![e_1]\!](m) \\ v_2 \in \mathcal{E}[\![e_2]\!](m) \\ v_1 \text{ relop } v_2 \end{array} \right) \\ ff \in \mathcal{C}[\![e_1 \text{ relop } e_2]\!](m) &\stackrel{def}{=} \exists v_1, v_2. \left(\begin{array}{l} v_1 \in \mathcal{E}[\![e_1]\!](m) \\ v_2 \in \mathcal{E}[\![e_2]\!](m) \\ \neg(v_1 \text{ relop } v_2) \end{array} \right) \\ be_1 \wedge be_2 \in \mathcal{C}[\![be_1 \text{ and } be_2]\!](m) &\stackrel{def}{=} \text{and} \left(\begin{array}{l} be_1 \in \mathcal{C}[\![be_1]\!](m) \\ be_2 \in \mathcal{C}[\![be_2]\!](m) \end{array} \right) \end{aligned}$$

- ▶ $(x := e, m) \longrightarrow m[x \mapsto v]$, **where** $v \in \mathcal{E}[[e]](m)$
- ▶ $(skip, m) \longrightarrow m$
- ▶ **If** $(S_1, m) \longrightarrow m'$, **then** $(S_1; S_2, m) \longrightarrow (S_2, m')$.
- ▶ **If** $tt \in \mathcal{C}[[be]]$, **then** $(\text{if } be \text{ then } S_1 \text{ else } S_2 \text{ end if}, m) \longrightarrow (S_1, m)$.
- ▶ **If** $ff \in \mathcal{C}[[be]]$, **then**
 $(\text{if } be \text{ then } S_1 \text{ else } S_2 \text{ end if}, m) \longrightarrow (S_2, m)$.
- ▶ **If** $tt \in \mathcal{C}[[be]]$, **then**
 $(\text{while } be \text{ do } S \text{ end do}, m) \longrightarrow (S; \text{while } be \text{ do } S \text{ end do}, m)$.
- ▶ **If** $ff \in \mathcal{C}[[be]]$, **then** $(\text{while } be \text{ do } S \text{ end do}, m) \longrightarrow m$.

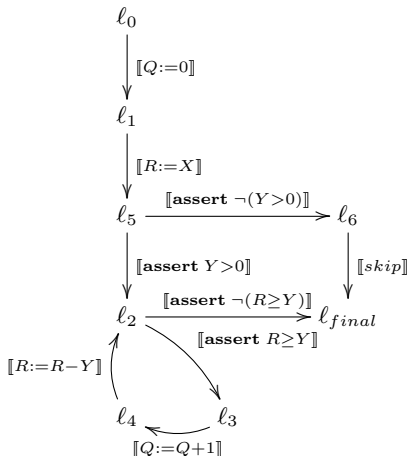
- ▶ A *control flow graph* is generated from the program under consideration namely P .
- ▶ A control flow graph $\mathcal{CFG}\llbracket P \rrbracket$ is defined by nodes ($l \in \mathcal{C}$) which are program control points of P , $\text{Control}\llbracket P \rrbracket$ and by labelled edges with actions ($\text{Actions}\llbracket P \rrbracket$) defined by the following rules :

$$\begin{array}{lcl} \text{actions} & ::= & v := \text{exp} \\ & | & \text{skip} \\ & | & \mathbf{assert} \text{ be} \end{array}$$

- ▶ A *control flow graph* is effectively defined by :
 - $\ell_{init} \in \text{Control}\llbracket P \rrbracket$: the entry point
 - $\ell_{end} \in \text{Control}\llbracket P \rrbracket$: the exit point
 - $\mathcal{Edges}\llbracket P \rrbracket \subseteq \text{Control}\llbracket P \rrbracket \times \text{Actions}\llbracket P \rrbracket \times \text{Control}\llbracket P \rrbracket$
- ▶ $\mathcal{CFG}\llbracket P \rrbracket = (\ell_{init}, \mathcal{Edges}\llbracket P \rrbracket, \ell_{end})$

From program to flowchart

```
ℓ0[Q := 0];  
ℓ1[R := X];  
IF ℓ5[Y > 0]  
    WHILE ℓ2[R ≥ Y]  
        ℓ3[Q := Q + 1];  
        ℓ4[R := R - Y]  
    ENDWHILE  
ELSE  
    ℓ6[skip]  
ENDIF
```



- ▶ $Mem \stackrel{def}{=} \mathbb{V} \longrightarrow \mathbb{Z}$
- ▶ Semantics of actions : $\xrightarrow{a} \subseteq Mem \times Mem$
 - $m \xrightarrow{x:=e} m[x \mapsto v]$ if there is a value $v \in \mathcal{E}[e](m)$
 - $m \xrightarrow{skip} m$
 - $m \xrightarrow{\text{assert } be} m$ if $tt \in \mathcal{C}[be](m)$
- ▶ Semantics for $\mathcal{CFG}[P]$: $\xrightarrow{P} \subseteq States \times States$
 - If $m \xrightarrow{a} m'$ and $(\ell_1, a, \ell_2) \in \mathcal{Edges}[P]$, then $(\ell_1, m) \xrightarrow{P} (\ell_2, m')$
 - The set of initial states is $\{\ell_{init}\} \times Mem$
 - The set of reachable states for P is denoted $\text{REACHABLE}(P)$ and defined by $\llbracket P \rrbracket = \{s \mid \exists s_0 \in \{\ell_{init}\} \times Mem : s_0 \xrightarrow[\star]{P} s\}$.

- ▶ Defining for each control point ℓ of P the set of reachable values :

$$\llbracket P \rrbracket_{\ell}^{coll} = \{s \mid s \in States \wedge s \in \llbracket P \rrbracket \wedge \exists m \in Mem : s = (\ell, m)\}$$

- ▶ Characterizing $\llbracket P \rrbracket_{\ell}^{coll}$: it satisfies the system of equations

$$\forall \ell \in \mathcal{C}(P). X_{\ell} = X_{\ell}^{init} \cup \bigcup_{(\ell_1, a, \ell) \in \mathcal{E}dges[P]} \llbracket a \rrbracket(X_{\ell_1}) \quad (1)$$

- ▶ Let $a \in \mathcal{A}ctions[\llbracket P \rrbracket]$ and $x \subseteq Mem$.

$$\llbracket a \rrbracket(x) = \{e \mid e \in States \wedge \exists f. f \in x \wedge f \xrightarrow{a} e\}$$

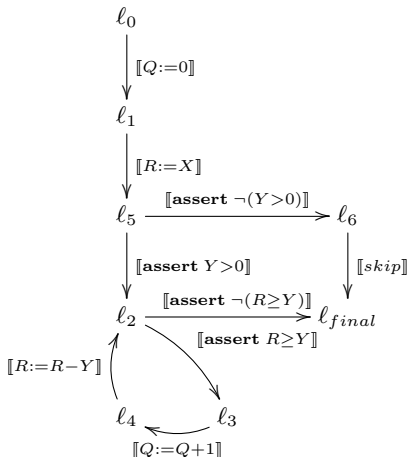
- ▶ $\forall \ell \in \mathcal{C}(P). \left(\begin{array}{l} \ell = \ell_{init} \Rightarrow X_{\ell}^{init} = Mem \\ \ell \neq \ell_{init} \Rightarrow X_{\ell}^{init} = \emptyset \end{array} \right)$

.....
☺ Théorème Let F the function defined as follows :

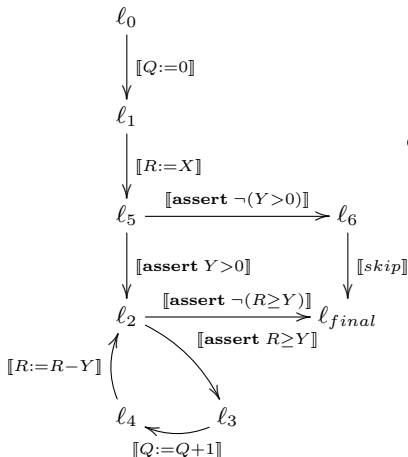
- ▶ n is the cardinality of $\mathcal{C}(P)$.
- ▶ $F \in \mathcal{P}(\text{States})^n \longrightarrow \mathcal{P}(\text{States})^n$
- ▶ If $X \in \mathcal{P}(\text{States})^n$, then $F(X) = (\dots, F_\ell(X), \dots)$
- ▶ $\forall \ell \in \mathcal{C}(P). F_\ell(X) = X_\ell^{init} \cup \bigcup_{(\ell_1, a, \ell) \in \text{Edges}[P]} \llbracket a \rrbracket(X_{\ell_1})$

The function F is monotonic over the complete lattice $(\mathcal{P}(\text{States})^n, \subseteq)$ and has a least fixed-point μF defining the collecting semantics.

From flowchart to equational system

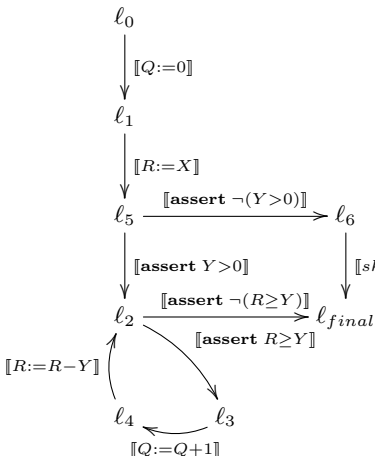


From flowchart to equational system



Defining equational systems for the collecting semantics :

From flowchart to equational system



Defining equational systems for the collecting semantics :

$$\left\{ \begin{array}{l} X_0 = Mem \\ X_1 = \llbracket Q := 0 \rrbracket(X_0) \\ X_5 = \llbracket R := X \rrbracket(X_1) \\ X_2 = \llbracket assert(Y > 0) \rrbracket(X_5) \cup \llbracket R := R - Y \rrbracket(X_4) \\ X_3 = \llbracket assert R \geq Y \rrbracket(X_2) \\ X_4 = \llbracket Q := Q + 1 \rrbracket(X_3) \\ X_6 = \llbracket assert \neg(Y > 0) \rrbracket(X_5) \\ X_7 = X_6 \cup \llbracket assert \neg(R \geq Y) \rrbracket(X_2) \end{array} \right.$$

- ▶ The collecting semantics is the least fixed-point of the system of equations, which exists by fixed-point theorems.
- ▶ Questions :
 - How to compute the solution ?
 - Computing over finite structures, when it is possible....
 - Using an approximation of fixed-points ?
 - What is an approximation ?
 - What is an abstraction ?
 - What is the best abstraction ?

Next step

Defining a framework for computing lfp solution of these equational systems in any case.

Example for computing reachable states (I)

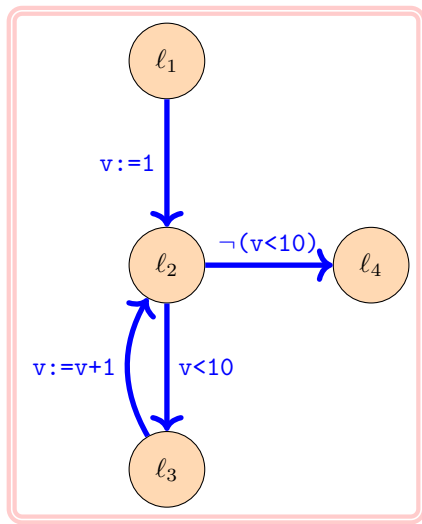
```
begin  
   $v := 1$ ;  
  while  $v < 10$  {  
     $v := v + 1$ ;  
  };  
end
```

Example for computing reachable states (I)

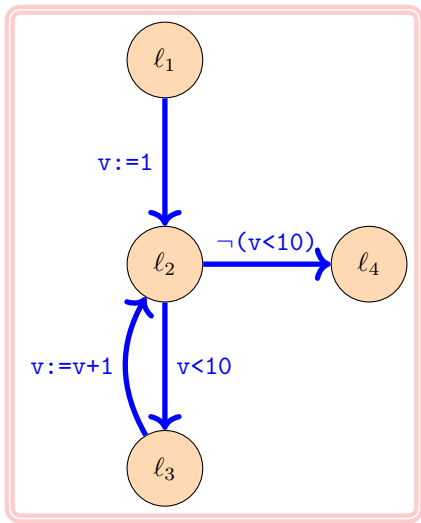
```
begin
  v := 1;
  while v < 10 {
    v := v+1;
  };
end
```

```
begin
   $\ell_1[v := 1]$ ;
  while  $\ell_2[v < 10]$  {
     $\ell_3[v := v+1]$ ;
  };
   $\ell_4$  :
end
```

Example for computing reachable states(I)



Example for computing reachable states(I)



- ▶ System of equations over $(\mathcal{P}(\mathbb{Z}), \subseteq)$

- $X_1 = \mathbb{Z}$
- $X_2 = \{1\} \cup \{v \mid v \in \mathbb{Z} \wedge v-1 \in X_3\}$
- $X_3 = \{v \mid v \in X_2 \wedge v < 10\}$
- $X_4 = \{v \mid v \in X_2 \wedge v \geq 10\}$

► Reachability

- $X_1 = \mathbb{Z}$
- $X_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- $X_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $X_4 = \{10\}$

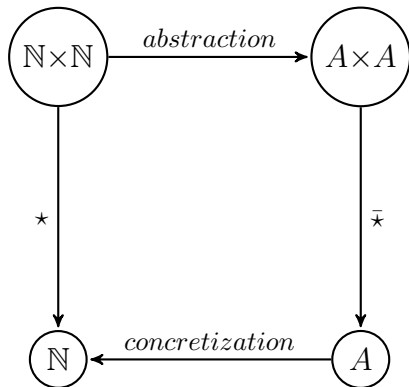
- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

Finding Sound Abstractions for Computing

- ▶ $n \in \mathbb{N}$: abstraction for n is defined as $\text{modulo}(n, 9)$.
- ▶ \mathbb{A} is the set of possible abstract values $\bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}, \bar{7}, \bar{8}$
- ▶ $n \star m = \bar{n} \star \bar{m}$
- ▶ $25 \star 25 = 625$, $\bar{25} \star \bar{25} = 6\bar{2}5$, $\bar{7} \star \bar{7} = \bar{4}$, $\overline{7 \star 7} = \bar{4}$, $\overline{49} = \bar{4}$, $\bar{4} = \bar{4}$,
- ▶ But $25 \star 25 \neq 265$ and $\overline{25 \star 25} = \overline{265}$

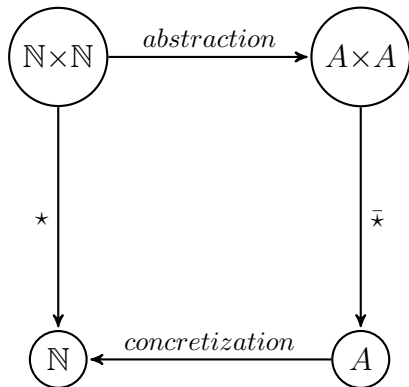
Finding Sound Abstractions for Computing

- ▶ $n \in \mathbb{N}$: abstraction for n is defined as $\text{modulo}(n, 9)$.
- ▶ \mathbb{A} is the set of possible abstract values $\bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}, \bar{7}, \bar{8}$
- ▶ $n \star m = \bar{n} \star \bar{m}$
- ▶ $25 \star 25 = 625$, $\bar{25} \star \bar{25} = 6\bar{25}$, $\bar{7} \star \bar{7} = \bar{4}$, $\overline{7 \star 7} = \bar{4}$, $\overline{49} = \bar{4}$, $\bar{4} = \bar{4}$,
- ▶ But $25 \star 25 \neq 265$ and $\overline{25 \star 25} = \overline{265}$



Finding Sound Abstractions for Computing

- ▶ $n \in \mathbb{N}$: abstraction for n is defined as $\text{modulo}(n, 9)$.
- ▶ \mathbb{A} is the set of possible abstract values $\bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}, \bar{7}, \bar{8}$
- ▶ $n \star m = \bar{n} \star \bar{m}$
- ▶ $25 \star 25 = 625$, $\bar{25} \star \bar{25} = 6\bar{25}$, $\bar{7} \star \bar{7} = \bar{4}$, $\overline{7 \star 7} = \bar{4}$, $\overline{49} = \bar{4}$, $\bar{4} = \bar{4}$,
- ▶ But $25 \star 25 \neq 265$ and $\overline{25 \star 25} = \overline{265}$



.....
☺ Rule

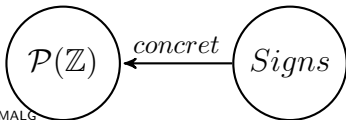
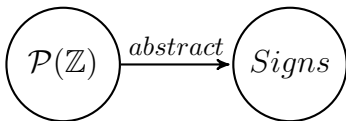
If $\overline{a \star b} \neq \overline{a} \star \overline{b}$,

then the operation is not correct.

- ▶ A number $z \in \mathbb{Z}$ is soundly approximated by an abstract value $abstract(z) \in Signs$.
- ▶ 2 is approximated by $pos : \{2\} \subseteq concrete(pos)$
- ▶ $\{2, 8\}$ is approximated by $pos : \{2, 8\} \subseteq concrete(pos)$
- ▶ -2 is approximated by $neg : \{2\} \subseteq concrete(neg)$
- ▶ 0 is approximated by $zero : \{0\} \subseteq concrete(zero)$
- ▶ $\{-2, -8\}$ is approximated by $pos : \{-2, -8\} \subseteq concrete(neg)$
- ▶ $\{-2, 2, 8\}$ is approximated by $pos : \{-2, 2, 8\} \subseteq concrete(nonzero)$

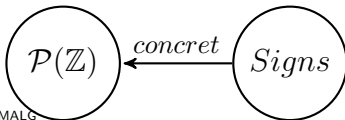
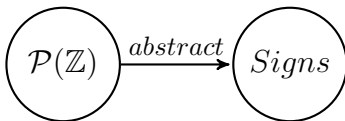
Finding Sound Abstractions for Computing

- ▶ A number $z \in \mathbb{Z}$ is soundly approximated by an abstract value $\text{abstract}(z) \in \text{Signs}$.
- ▶ 2 is approximated by $\text{pos} : \{2\} \subseteq \text{concrete}(\text{pos})$
- ▶ $\{2, 8\}$ is approximated by $\text{pos} : \{2, 8\} \subseteq \text{concrete}(\text{pos})$
- ▶ -2 is approximated by $\text{neg} : \{2\} \subseteq \text{concrete}(\text{neg})$
- ▶ 0 is approximated by $\text{zero} : \{0\} \subseteq \text{concrete}(\text{zero})$
- ▶ $\{-2, -8\}$ is approximated by $\text{pos} : \{-2, -8\} \subseteq \text{concrete}(\text{neg})$
- ▶ $\{-2, 2, 8\}$ is approximated by $\text{pos} : \{-2, 2, 8\} \subseteq \text{concrete}(\text{nonzero})$



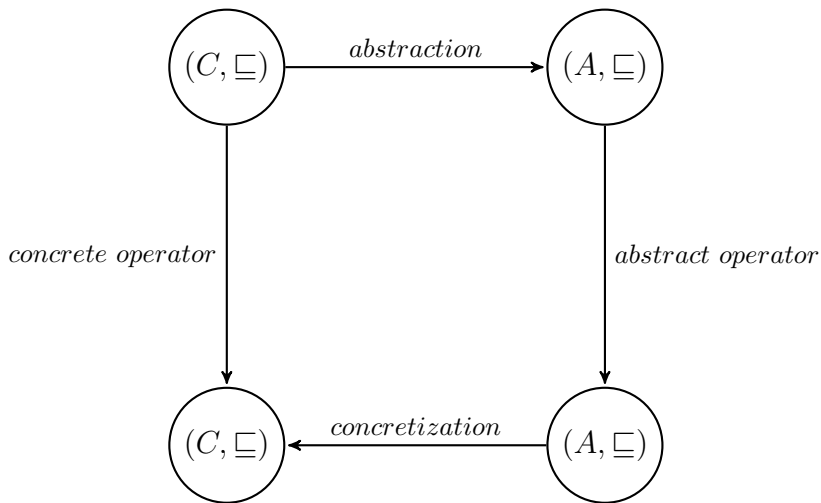
Finding Sound Abstractions for Computing

- ▶ A number $z \in \mathbb{Z}$ is soundly approximated by an abstract value $\text{abstract}(z) \in \text{Signs}$.
- ▶ 2 is approximated by $\text{pos} : \{2\} \subseteq \text{concret}(\text{pos})$
- ▶ $\{2, 8\}$ is approximated by $\text{pos} : \{2, 8\} \subseteq \text{concret}(\text{pos})$
- ▶ -2 is approximated by $\text{neg} : \{2\} \subseteq \text{concret}(\text{neg})$
- ▶ 0 is approximated by $\text{zero} : \{0\} \subseteq \text{concret}(\text{zero})$
- ▶ $\{-2, -8\}$ is approximated by $\text{pos} : \{-2, -8\} \subseteq \text{concret}(\text{neg})$
- ▶ $\{-2, 2, 8\}$ is approximated by $\text{pos} : \{-2, 2, 8\} \subseteq \text{concret}(\text{nonzero})$



- ▶ $\text{concret}(\text{pos}) = \{z \mid z \in \mathbb{Z} \wedge z > 0\}$
- ▶ $\text{concret}(\text{neg}) = \{z \mid z \in \mathbb{Z} \wedge z < 0\}$
- ▶ $\text{concret}(\text{non}) = \emptyset$
- ▶ $\text{concret}(\text{nonzero}) = \{z \mid z \in \mathbb{Z} \wedge z \neq 0\}$

- ▶ A number $z \in \mathbb{Z}$ is soundly approximated by an abstract value $abstract(z) \in Signs$.
- ▶ 2 is approximated by pos :
 - $\{2\} \subseteq concrete(pos)$
 - $abstract(\{2\}) = pos$
- ▶ $\{2, 8\}$ is approximated by pos :
 - $\{2, 8\} \subseteq concrete(pos)$
 - $abstract(\{2, 8\}) = pos$
- ▶ $\{-2, 2, 8\}$ is approximated by pos :
 - $\{-2, 2, 8\} \subseteq concrete(nonzero)$
 - $abstract(\{-2, 2, 8\}) = nonzero$

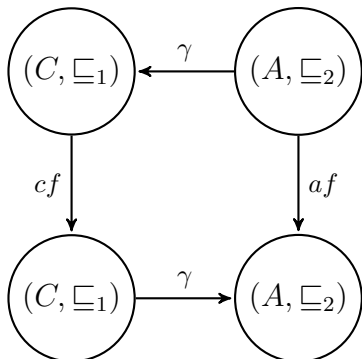


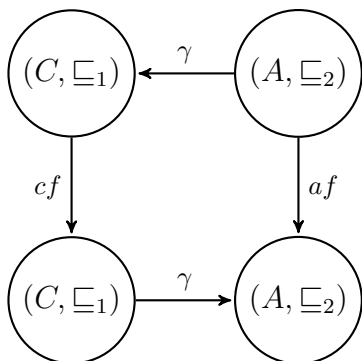
- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 **Galois Connections**
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- 17 Conclusion

- If B is a Moore family, then it is a good abstraction.**

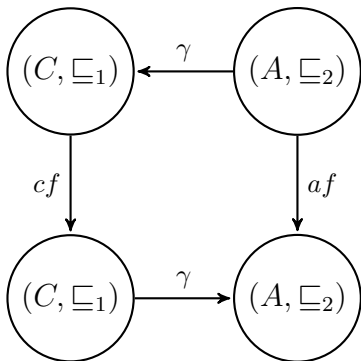
- ▶ Let $(A, \sqsubseteq, \sqcup, \sqcap)$ be a complete lattice.
- ▶ $\rho \in A \longrightarrow A$ is a upper closure operator, if it satisfies the following properties :
 - ρ is monotonic : $\forall x, y \in A. x \sqsubseteq y \Rightarrow \rho(x) \sqsubseteq \rho(y)$.
 - ρ is extensive : $\forall x \in A. x \sqsubseteq \rho(x)$.
 - ρ is idempotent : $\forall x \in A. \rho(x) = \rho(\rho(x))$.
- ▶ **Property** : Let $(A, \sqsubseteq, \sqcup, \sqcap)$ be a complete lattice. and $B \subseteq A$.
 B is a Moore family if, and only if, there exists a closure operator ρ such that $B = \rho(A)$
- ▶ B is a *good* abstraction of C , if it satisfies $B = \rho(C)$ where ρ is a closure operator.

- ▶ Two complete lattices $(C, \sqsubseteq_1, \sqcup_1, \sqcap_1)$ and $(A, \sqsubseteq_2, \sqcup_2, \sqcap_2)$ are supposed to be given.
- ▶ Two functions α and γ are supposed to be defined as follows :
 - $\alpha \in C \longrightarrow A$
 - $\gamma \in A \longrightarrow C$
- ▶ The pair (α, γ) is a Galois connection, if it satisfies the following property : $\forall x_1 \in C, x_2 \in A. \alpha(x_1) \sqsubseteq_2 x_2 \Leftrightarrow x_1 \sqsubseteq_1 \gamma(x_2)$
- ▶ A complete lattice A is a good abstraction of L , when there is a Galois connection between A and L .

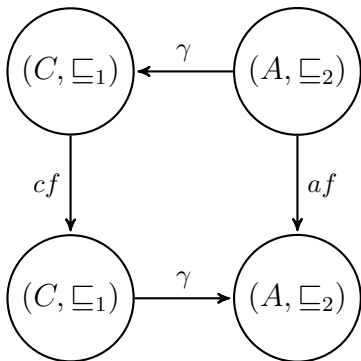




► a is a sound abstraction of c , if $c \sqsubseteq_1 \gamma(a)$.



- ▶ a is a sound abstraction of c , if $c \sqsubseteq_1 \gamma(a)$.
- ▶ functional operator : af is a sound abstraction of cf , if $\forall a \in A. cf(\gamma(a)) \sqsubseteq_1 \gamma(af(a))$



- ▶ a is a sound abstraction of c , if $c \sqsubseteq_1 \gamma(a)$.
- ▶ functional operator : af is a sound abstraction of cf , if $\forall a \in A. cf(\gamma(a)) \sqsubseteq_1 \gamma(af(a))$
- ▶ relational operator : ar is a sound abstraction of cr , if $\forall a \in A. cr(\gamma(a_1), \dots, \gamma(a_n)) \sqsubseteq_1 \gamma(ac(a_1, \dots, a_n))$

Galois Connections

The pair (α, γ) is a Galois connection, if it satisfies the following property : $\forall x_1 \in L, x_2 \in L. \alpha(x_1) \sqsubseteq' x_2 \Leftrightarrow x_1 \sqsubseteq \gamma(x_2)$

Notation : $L \xrightleftharpoons[\alpha]{\gamma} L'$

Properties of a Galois connection $L \xrightleftharpoons[\alpha]{\gamma} L'$

- ▶ α and γ are monotonic over the lattices.
- ▶ $\text{id}(L) \subseteq \gamma \circ \alpha : \gamma \circ \alpha$ is extensive.
- ▶ $\alpha \circ \gamma \subseteq \text{id}(L') : \alpha \circ \gamma$ is retractive.
- ▶ $\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$
- ▶ $\alpha(x) = \bigcap' \{y \in L' \mid x \sqsubseteq \gamma(y)\}$
- ▶ $\gamma(y) = \bigcup \{x \in L \mid \alpha(x) \sqsubseteq' y\}$

Properties

- ▶ $\gamma \circ \alpha \circ \gamma \circ \alpha = \gamma \circ \alpha$
- ▶ $\alpha \circ \gamma \circ \alpha \circ \gamma = \alpha \circ \gamma$
- ▶ We assume that $\{(\alpha_i, \gamma_i) | i \in \{1 \dots n\}\}$ is a family of Galois connections :

$$L_1 \xleftrightarrow[\alpha_1]{\gamma_1} L_2 \xleftrightarrow[\alpha_2]{\gamma_2} \dots L_{n-1} \xleftrightarrow[\alpha_{n-1}]{\gamma_{n-1}} L_n$$

Then $(\alpha_1; \dots; \alpha_i; \dots; \alpha_{n-1}, \gamma_{n-1}; \dots; \gamma_i; \dots; \gamma_1)$ is a Galois connection. or equivalently

$L_1 \xleftrightarrow[\alpha_{n-1} \circ \dots \circ \alpha_i \circ \dots \circ \alpha_1]{\gamma_1 \circ \dots \circ \gamma_i \circ \dots \circ \gamma_{n-1}} \text{ is a Galois connection.}$

- ▶ We assume that $\{(\alpha_i, \gamma_i) | i \in \{1, 2\}\}$ two Galois connections :
 $\alpha_1 = \alpha_2$ if, and only if, $\gamma_1 = \gamma_2$

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

- ▶ We consider a transition system (S, I, t) where S is the set of states, I is the set of initial states and t is a binary relation over S .
- ▶ A property P of the transition system is a subset of S : $P \subseteq S$.
- ▶ P holds in $s \in S$, when $s \in P$.
- ▶ Four operators over properties can be defined as follows :

- $\text{pre}[t]P \stackrel{\text{def}}{=} \{s | s \in S \wedge \exists s'. ((s, s') \in t \wedge s' \in P)\}$
- $\tilde{\text{pre}}[t]P \stackrel{\text{def}}{=} \{s | s \in S \wedge \forall s'. ((s, s') \in t \Rightarrow s' \in P)\}$
- $\text{post}[t]P \stackrel{\text{def}}{=} \{s | s \in S \wedge \exists s'. ((s', s) \in t \wedge s' \in P)\}$
- $\tilde{\text{post}}[t]P \stackrel{\text{def}}{=} \{s | s \in S \wedge \forall s'. ((s', s) \in t \Rightarrow s' \in P)\}$

- ▶ Duality of operators :

- ① $\tilde{\text{pre}}[t]\neg P = \neg \text{pre}[t]P$
- ② $\tilde{\text{post}}[t]\neg P = \neg \text{post}[t]P$

- ▶ Galois connections over \mathcal{P} , the set of subsets of S :

$$(\mathcal{P}, \subseteq) \xrightleftharpoons[\text{pre}[t]]{\tilde{\text{post}}[t]} (\mathcal{P}, \subseteq)$$

$$(\mathcal{P}, \subseteq) \xrightleftharpoons[\text{post}[t]]{\tilde{\text{pre}}[t]} (\mathcal{P}, \subseteq)$$

- ▶ Let two sets \mathcal{L} standing for labels et \mathcal{M} standing for memories.
- ▶ First step :
 - \sqsubseteq is the partial ordering over functions using the subset relationship over function graphs : $f \sqsubseteq g$ means that $\text{Graph}(f) \subseteq \text{Graph}(g)$.
 - $\alpha_1 = \lambda P. \lambda l. \{m \mid (l, m) \in P\}$
 - $\gamma_1 = \lambda Q. \{(l, m) \mid l \in \mathcal{L} \wedge m \in Q(l)\}$
 - $(\mathcal{P}(\mathcal{L} \times \mathcal{M}), \sqsubseteq) \xrightleftharpoons[\alpha_1]{\gamma_1} (\mathcal{L} \longrightarrow \mathcal{P}(\mathcal{M}), \sqsubseteq)$ is a Galois connection
- ▶ Second step :
 - Let two sets $Pred$, set of predicates, and \mathcal{M} , a set of memories.
 - The relationship between both sets is stating as follows : For any given predicate p and any given memory m , p holds in m .
 - We define $B(p) = \{m \mid m \in \mathcal{M} \wedge p(m)\}$, set of memories in which p holds.
 - Next we define :
 - ▶ $\alpha_2 = \lambda Q. \{p \mid p \in Pred \wedge Q \subseteq B(p)\}$
 - ▶ $\gamma_2 = \lambda P. \bigcap \{B(p) \mid p \in P\}$
 - $(\mathcal{P}(\mathcal{M}), \sqsubseteq) \xrightleftharpoons[\alpha_2]{\gamma_2} (\mathcal{P}(Pred), \Rightarrow)$ is a Galois connection.

► Third step

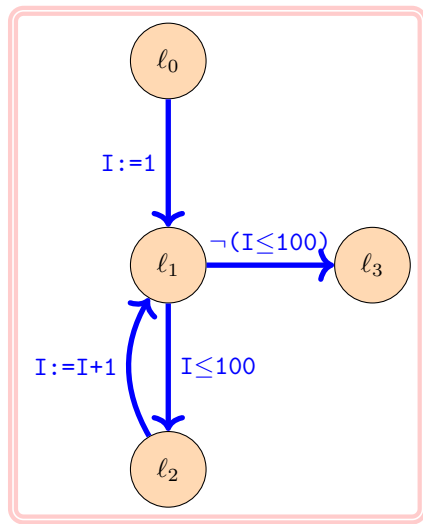
- $\alpha_3 = \lambda\ell.\alpha_2(Q_\ell) : Q \subseteq_1 Q' \stackrel{def}{=} \forall \ell \in \mathcal{L}. Q_\ell \subseteq Q'_\ell.$
- $\gamma_3 = \lambda\ell.\gamma_2(P_\ell) : P \Rightarrow_1 P' \stackrel{def}{=} \forall \ell \in \mathcal{L}. P_\ell \Rightarrow P'_\ell.$
- $(\mathcal{L} \longrightarrow \mathcal{P}(\mathcal{M}), \subseteq_1) \xrightleftharpoons[\alpha_3]{\gamma_3} (\mathcal{L} \longrightarrow \mathcal{P}(\text{Pred}), \Rightarrow_1)$ is a Galois connection.

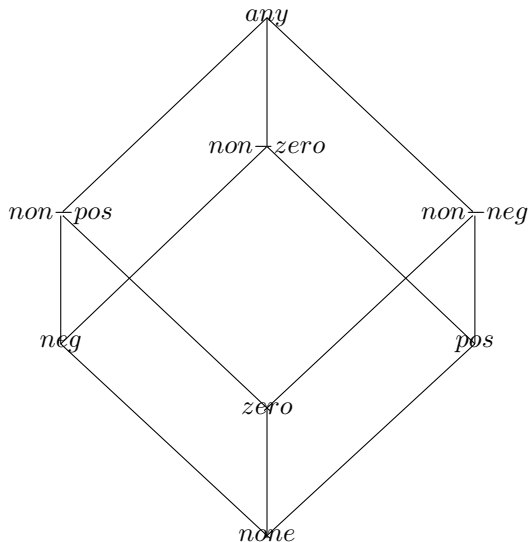
- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

- ① Introduction
- ② Example of analysis
- ③ Static analysis
- ④ Overview of the methodology
- ⑤ Standard, Collecting and Abstract Semantics
- ⑥ TOP
- ⑦ Finding Sound Abstractions for Computing
- ⑧ Galois Connections
- ⑨ Examples of Galois connections
- ⑩ TOP2
- ⑪ Domain of Signs
- ⑫ Domain of intervals
- ⑬ Abstraction and approximation
- ⑭ TOP3
- ⑮ Widening and Narrowing
 - Widening
 - Narrowing
- ⑯ Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- ⑰ Conclusion

Examples of Abstractions

```
 $\ell_0[I := 1];$   
while  $\ell_1[I \leq 100]$  do  
   $\ell_2[I := I+1];$   
end while  
 $\ell_3[skip]$ 
```





- ▶ Defining an abstraction for integers

$\alpha \in \mathcal{P}(\mathbb{Z}) \longrightarrow \text{Signs}$

- ▶
$$\begin{cases} z & \alpha(z) \\ z < 0 & \text{neg} \\ z > 0 & \text{pos} \\ z = 0 & \text{zero} \end{cases}$$

- ▶ Abstraction by projection :

$$(\mathcal{P}(Var \longrightarrow \mathbb{Z}), \subseteq) \xleftrightarrow[\alpha_\pi]{\gamma_\pi} (Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq)$$

- ▶ Abstraction of signs

$$(Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq) \xleftrightarrow[\alpha_{sign}]{\gamma_{sign}} (Var \longrightarrow Signs), \subseteq)$$

- ▶ Composition of abstractions :

$$(\mathcal{P}(Var \longrightarrow \mathbb{Z}), \subseteq) \xleftrightarrow[\alpha_{sign} \circ \alpha_\pi]{\gamma_\pi \circ \gamma_{sign}} (Var \longrightarrow Signs), \subseteq)$$

- ▶ $\alpha = \alpha_{sign} \circ \alpha_\pi$ and $\gamma = \gamma_\pi \circ \gamma_{sign}$

- L is the concrete domain and L' is the abstract model :

$$\begin{array}{ccc} L & \xrightarrow{f} & L \\ \gamma \uparrow & & \downarrow \alpha \\ L' & \xrightarrow{f'} & L' \end{array}$$

$$f' = \alpha \circ f \circ \gamma \quad (2)$$

f' is the best approximation of f

- ▶ Concrete states : $cv \in Var \longrightarrow \mathcal{P}(\mathbb{Z})$: if X is in Var , then $cv(X) \in \mathcal{P}(\mathbb{Z})$.
- ▶ Abstract states : $av \in Var \longrightarrow Signs$: if X is in Var , then $av(X) \in Signs$.
- ▶ (α, γ) is extended as :
 (α_1, γ_1) entre $(Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq)$ et $(Var \longrightarrow Signs, \sqsubseteq)$. En particulier, $\alpha_1(cv) = av$ et, pour tout X de Var , $av(X) = \alpha(cv(X))$; $\gamma_1(av) = cv$ et, pour tout X de Var , $cv(X) = \gamma(av(X))$.
- ▶ Any expression e can be evaluated on each domain :
 - concrete domain : $States = Var \longrightarrow \mathcal{P}(\mathbb{Z})$:
 $\llbracket e \rrbracket \in (Var \longrightarrow \mathcal{P}(\mathbb{Z})) \longrightarrow \mathcal{P}(\mathbb{Z})$ and $\llbracket e \rrbracket(cv)$
 - abstract domain : $AStates = Var \longrightarrow Signs$:
 $\llbracket e \rrbracket_a \in (Var \longrightarrow Signs) \longrightarrow Signs$ and $\llbracket e \rrbracket_a(av)$.

- ▶ The best abstraction is simply dedined as follows :

$$\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av).$$

- ▶ Applying the best approximation for assignment :

$$\llbracket x := e \rrbracket_{best}(av) = \begin{cases} av(y), y \neq x \\ \llbracket e \rrbracket_{best}(av) \end{cases}$$

- ▶ $(\mathcal{P}(Var \longrightarrow \mathbb{Z}), \subseteq) :$

$$A, B \in \mathcal{P}(\mathbb{Z}) : A+B = \{a+b | a \in A \wedge b \in B\}$$

- ▶ $(Var \longrightarrow Signs), \subseteq) :$

$$x, y \in Signs : x \oplus y = \alpha(\gamma(x) + \gamma(y))$$

- ▶ examples :

- $pos \oplus neg = \alpha(\gamma(pos) + \gamma(neg)) = \alpha((1, +\infty) + (-\infty, -1)) = \alpha((-\infty, +\infty)) = any$
- $pos \oplus zero = \alpha(\gamma(pos) + \gamma(zero)) = \alpha((1, +\infty) + (0)) = \alpha((1, +\infty)) = pos$
- Building a table for the abstract operation \oplus .

- ▶ Applying the analysis on the example

$\ell_0[X := 1];$
 $\ell_1[Y := 5];$
 $\ell_2[X := X+1];$
 $\ell_3[Y := Y-1];$
 $\ell_4[X := Y+X];$
 $\ell_{final}[skip];$

ℓ	X	Y
ℓ_0	<i>any</i>	<i>any</i>
ℓ_1	<i>pos</i>	<i>any</i>
ℓ_2	<i>pos</i>	<i>pos</i>
ℓ_3	<i>pos</i>	<i>pos</i>
ℓ_4	<i>pos</i>	<i>non-neg</i>
ℓ_{final}	<i>non-neg</i>	<i>non-neg</i>

- ▶ ℓ_3 to ℓ_4 : abstract value of Y is *pos* and by γ , we obtain $(1, +\infty)$
and now we can compute in concrete domain \mathbb{Z}
 $(1, +\infty) + (-1) = (0, +\infty)$. By reapplying α we obtain *non-neg*.
- ▶ Computations may be not computable and one should use techniques for accelerating the convergence like widening.
- ▶ Computing is still costly : computing now in the abstraction and defining a sound approximation of f .

- ▶ Evaluation is using the *best* approximation :

$$\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$$

- ▶ Evaluation is using the *best* approximation :
$$\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$$
- ▶ Computing over the concrete domain is remaining complex

- ▶ Evaluation is using the *best* approximation :
$$\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$$
- ▶ Computing over the concrete domain is remaining complex
- ▶ Idea : approximation of the *best* approximation : $\llbracket e \rrbracket_a$ and, for any av abstract state, $\llbracket e \rrbracket_{best}(av) \sqsubseteq \llbracket e \rrbracket_a(av)$.
- ▶ Abstract semantics is defined as follows :
 $av \in Var \longrightarrow Signs :$

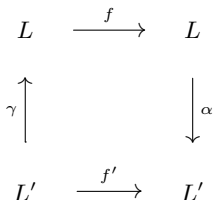
- ▶ Evaluation is using the *best* approximation :
$$\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$$
- ▶ Computing over the concrete domain is remaining complex
- ▶ Idea : approximation of the *best* approximation : $\llbracket e \rrbracket_a$ and, for any av abstract state, $\llbracket e \rrbracket_{best}(av) \sqsubseteq \llbracket e \rrbracket_a(av)$.
- ▶ Abstract semantics is defined as follows :
 $av \in Var \longrightarrow Signs :$
 - $\llbracket const \rrbracket_a(av) = \alpha(\{c\})$
 - $\llbracket x \rrbracket_a(av) = av(x)$
 - $\llbracket e_1 + e_2 \rrbracket_a(av) = \llbracket e_1 \rrbracket_a(av) \oplus \llbracket e_2 \rrbracket_a(av)$
 - $\llbracket e_1 * e_2 \rrbracket_a(av) = \llbracket e_1 \rrbracket_a(av) \otimes \llbracket e_2 \rrbracket_a(av)$

- ▶ Evaluation is using the *best* approximation :
$$\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$$
- ▶ Computing over the concrete domain is remaining complex
- ▶ Idea : approximation of the *best* approximation : $\llbracket e \rrbracket_a$ and, for any av abstract state, $\llbracket e \rrbracket_{best}(av) \sqsubseteq \llbracket e \rrbracket_a(av)$.
- ▶ Abstract semantics is defined as follows :
 $av \in Var \longrightarrow Signs :$
 - $\llbracket const \rrbracket_a(av) = \alpha(\{c\})$
 - $\llbracket x \rrbracket_a(av) = av(x)$
 - $\llbracket e_1 + e_2 \rrbracket_a(av) = \llbracket e_1 \rrbracket_a(av) \oplus \llbracket e_2 \rrbracket_a(av)$
 - $\llbracket e_1 * e_2 \rrbracket_a(av) = \llbracket e_1 \rrbracket_a(av) \otimes \llbracket e_2 \rrbracket_a(av)$
- ▶ $\ell[X := E] : \llbracket E \rrbracket_a$ in av ou encore $\llbracket E \rrbracket_a(av) :$
$$\llbracket Y + X + 6 \rrbracket_a(av) = \llbracket Y \rrbracket_a(av) +_a \llbracket X \rrbracket_a(av) +_a \llbracket 6 \rrbracket_a(av).$$
 - $\llbracket Y - 1 \rrbracket_a(av) = \llbracket Y \rrbracket_a(av) \oplus \llbracket -1 \rrbracket_a(av) = pos \oplus neg = any$
 - $\llbracket Y - 1 \rrbracket_{best}(av) = \alpha \circ \llbracket Y - 1 \rrbracket \circ \gamma_1(av) = \alpha(\llbracket Y - 1 \rrbracket(\gamma_1(av))) = \alpha(\llbracket Y - 1 \rrbracket(\{Y \mapsto (1, +\infty)\})) = \alpha((1 + \infty) + (-1)) = \alpha((0, +\infty)) = non-neg$

Sound approximations of f with respect to a Galois connection

A sound approximation of f with respect to a Galois connection f' satisfies the following property :

$$\forall x \in L, y \in L'. \alpha(x) \sqsubseteq y \Rightarrow \alpha(f(x)) \sqsubseteq f'(y)$$



The four statements are equivalent

- ▶ f' is a sound approximation of f with respect to a Galois connection
- ▶ $\alpha \circ f \sqsubseteq' f' \circ \alpha$
- ▶ $\alpha \circ f \circ \gamma \sqsubseteq' f'$
- ▶ $f \circ \gamma \sqsubseteq' \gamma \circ f'$

- ▶ $\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$ provide the best abstraction but is costly.
- ▶ Another solution is to define an abstract semantics for expressions : $\llbracket e \rrbracket_a$ such that for any av , $\llbracket e \rrbracket_{best}(av) \sqsubseteq \llbracket e \rrbracket_a(av)$.
- ▶ $av \in Var \longrightarrow Signs :$
 - $\llbracket const \rrbracket_a(v) = \alpha(\{c\})$
 - $\llbracket x \rrbracket_a(v) = v(x)$
 - $\llbracket e_1 + e_2 \rrbracket_a(v) = \llbracket e_1 \rrbracket_a(v) \oplus \llbracket e_2 \rrbracket_a(v)$
 - $\llbracket e_1 * e_2 \rrbracket_a(v) = \llbracket e_1 \rrbracket_a(v) \otimes \llbracket e_2 \rrbracket_a(v)$

- ▶ $\llbracket Y-1 \rrbracket_a(av) = \llbracket Y \rrbracket_a(av) \oplus \llbracket -1 \rrbracket(av)_a = pos \oplus neg = may$
- ▶ $\llbracket Y-1 \rrbracket_{best}(av) = \alpha_1 \circ \llbracket Y-1 \rrbracket \circ \gamma_1(av) = \alpha_1(\llbracket Y-1 \rrbracket(\gamma_1(av))) = \alpha_1(\llbracket Y-1 \rrbracket(\{Y \mapsto (1, +\infty)\})) = \alpha_1((1+\infty)+(-1)) = \alpha_1((0, +\infty)) = non-neg$

- ▶ Applying the analysis on the example

$\ell_0[X := 1];$
 $\ell_1[Y := 5];$
 $\ell_2[X := X + 1];$
 $\ell_3[Y := Y - 1];$
 $\ell_4[X := Y + X];$
 $\ell_{final}[skip];$

ℓ	X	Y
ℓ_0	<i>any</i>	<i>any</i>
ℓ_1	<i>pos</i>	<i>any</i>
ℓ_2	<i>pos</i>	<i>pos</i>
ℓ_3	<i>pos</i>	<i>pos</i>
ℓ_4	<i>pos</i>	<i>any</i>
ℓ_{final}	<i>any</i>	<i>any</i>

- ▶ The new analysis is less precise but more efficient since we compute in the domain of signs.

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- 17 Conclusion

- ▶ $\mathbb{I}(\mathbb{Z}) = \{\perp\} \cup \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{\infty\}, l \leq u\}$
- ▶ $[l_1, u_1] \sqsubseteq [l_2, u_2]$ si, et seulement si, $l_2 \leq l_1$ et $u_1 \leq u_2$.
- ▶ $(\mathbb{I}(\mathbb{Z}), \sqsubseteq)$ est une structure partiellement ordonnée.
- ▶
 - ① $[l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)]$
 - ② $[l_1, u_1] \sqcap [l_2, u_2] = \begin{cases} [\max(l_1, l_2), \min(u_1, u_2)] \\ \perp, \text{ si } \max(l_1, l_2) > \min(u_1, u_2) \end{cases}$
- ▶ $(\mathbb{I}(\mathbb{Z}), \sqcup)$ is a complete lattice.
- ▶
 - ① $\alpha(X) = \begin{cases} [\min(X), \max(X)] \\ \perp, \text{ si } X = \emptyset \end{cases}$
 - ② $\gamma([l, u]) = [l..u]$ et $\gamma(\perp) = \emptyset$
- ▶ (α, γ) is a Galois connexion.
- ▶
 - ① $i_1 \oplus i_2 = [l_1 + l_2, u_1 + u_2]$
 - ② $i_1 \ominus i_2 = [l_1 - u_2, u_1 - l_2]$
 - ③ $i_1 \otimes i_2 = [\min(l_1 \cdot l_2, l_1 \cdot u_2, u_1 \cdot l_2, u_1 \cdot u_2), \max(l_1 \cdot l_2, l_1 \cdot u_2, u_1 \cdot l_2, u_1 \cdot u_2)]$
 - ④ $i_1 \oslash i_2 = [\min(l_1 / l_2, l_1 / u_2, u_1 / l_2, u_1 / u_2), \max(l_1 / l_2, l_1 / u_2, u_1 / l_2, u_1 / u_2)]$

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples
- 17 Conclusion

- ▶ Computing *collecting semantics* is generally undecidable :
 - S is a safety property for \mathcal{MS} if, and only if, $\text{REACHABLE}(\mathcal{MS}) \subseteq S$.
 - Finding a sound approximation of $\text{REACHABLE}(\mathcal{MS})$, denoted $\alpha(\text{REACHABLE}(\mathcal{MS}))$, and satisfying $\gamma(\alpha(\text{REACHABLE}(\mathcal{MS}))) \subseteq S$.
 - $\text{REACHABLE}(\mathcal{MS}) \subseteq \gamma(\alpha(\text{REACHABLE}(\mathcal{MS})))$ and $\gamma(\alpha(\text{REACHABLE}(\mathcal{MS}))) \subseteq S$.
- ▶ Abstract domains can be finite as the domain of Signs but the domain of intervals is infinite : computing $\text{REACHABLE}(\mathcal{MS})$ remains undecidable but we can approximate its computation.
- ▶ Abstract domains can be infinite : we have to accelerate the computations of fixed-points in the case of loops for instance : widening and narrowing.

- L is the concrete domain and L' is the abstract model :

$$\begin{array}{ccc} L & \xrightarrow{f} & L \\ \gamma \uparrow & & \downarrow \alpha \\ L' & \xrightarrow{f'} & L' \end{array}$$

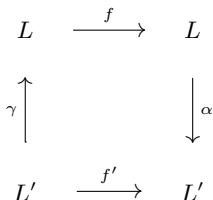
$$f' = \alpha \circ f \circ \gamma \quad (3)$$

f' is the best approximation of f

Sound approximations of f with respect to a Galois connection

A sound approximation of f with respect to a Galois connection f' satisfies the following property :

$$\forall x \in L, y \in L'. \alpha(x) \sqsubseteq y \Rightarrow \alpha(f(x)) \sqsubseteq f'(y)$$



The four statements are equivalent

- ▶ f' is a sound approximation of f with respect to a Galois connection
- ▶ $\alpha \circ f \sqsubseteq' f' \circ \alpha$
- ▶ $\alpha \circ f \circ \gamma \sqsubseteq' f'$
- ▶ $f \circ \gamma \sqsubseteq' \gamma \circ f'$

- ▶ $\llbracket e \rrbracket_{best}(av) = \alpha \circ \llbracket e \rrbracket \circ \gamma_1(av)$ provides the best abstraction but is costly.
- ▶ Another solution is to define an abstract semantics for expressions :
hide $\llbracket e \rrbracket_a$ such that for any av , $\llbracket e \rrbracket_{best}(av) \sqsubseteq \llbracket e \rrbracket_a(av)$.
- ▶ $av \in Var \longrightarrow Signs :$
 - $\llbracket const \rrbracket_a(v) = \alpha(\{c\})$
 - $\llbracket x \rrbracket_a(v) = v(x)$
 - $\llbracket e_1 + e_2 \rrbracket_a(v) = \llbracket e_1 \rrbracket_a(v) \oplus \llbracket e_2 \rrbracket_a(v)$
 - $\llbracket e_1 * e_2 \rrbracket_a(v) = \llbracket e_1 \rrbracket_a(v) \otimes \llbracket e_2 \rrbracket_a(v)$

- ▶ Suppose that $C \xrightleftharpoons[\alpha]{\gamma} A$ is a Galois connection
- ▶ a function $f \in C \rightarrow C$: to find a function g

$$\begin{array}{ccc} C & \xrightarrow{f} & C \\ \gamma \uparrow & & \downarrow \alpha \\ A & \xrightarrow{g} & A \end{array}$$

- ▶ f is monotone
- ▶ $g = R(\alpha, \gamma, f)$ and $f \sqsubseteq \gamma \circ g \circ \alpha$
- ▶ $f \sqsubseteq \gamma \circ g \circ \alpha$ or equivalently $\alpha \circ f \circ \gamma \sqsubseteq g$
- ▶ $g = \alpha \circ f \circ \gamma$ is the *best* approximation.

Definition of a sound approximation of a function f

A function $g \in A \rightarrow A$ is a sound approximation of a function $f \in C \rightarrow C$, if it satisfies the following condition :

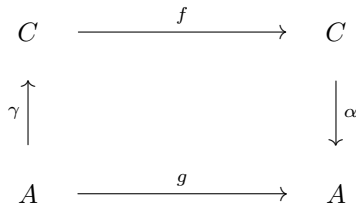
$$\forall c \in C : \forall a \in A : \alpha(c) \sqsubseteq a \Rightarrow \alpha(f(c)) \sqsubseteq g(a)$$

Properties

Suppose that $C \xrightleftharpoons[\alpha]{\gamma} A$ is a Galois connection.

The four statements are equivalent

- ① g is a sound approximation of f with respect to a Galois connection
- ② $\alpha \circ f \sqsubseteq g \circ \alpha$
- ③ $\alpha \circ f \circ \gamma \sqsubseteq g$
- ④ $f \circ \gamma \sqsubseteq \gamma \circ g$
- ⑤ $f \sqsubseteq \gamma \circ g \circ \alpha$



Best abstraction

Suppose that :

- ▶ $C \xrightleftharpoons[\alpha]{\gamma} A$ is a Galois connection.
- ▶ $f \in C \rightarrow C$ is monotonous
- ▶ $g = \alpha \circ f \circ \gamma$

Then $lfp(f) \sqsubseteq \gamma(lfp(g))$ and $\alpha(lfp(f)) \sqsubseteq lfp(g)$
or equivalently rewritten as $\mu f \sqsubseteq \gamma(\mu g)$ and $\alpha(\mu f) \sqsubseteq \mu g$

First theorem

- ▶ Suppose that $C \xleftrightarrow[\alpha]{\gamma} A$ is a Galois connection
- ▶ Two functions $f \in C \rightarrow C$ and $g \in A \rightarrow A$:

$$\begin{array}{ccc} C & \xrightarrow{f} & C \\ \gamma \uparrow & & \downarrow \alpha \\ A & \xrightarrow{g} & A \end{array}$$

- ▶ f and g are monotone
- ▶ $\alpha \circ f = g \circ \alpha$.

Then $\alpha(\mu.f) = \mu.g$.

- ① $\mu g \sqsubseteq \alpha(\mu f)$
 - $f(\mu f) = \mu f$ (fixed-point property)
 - $\alpha(f(\mu f)) = \alpha(\mu f)$ (applying the relation over f and g)
 - $\alpha(f(\mu f)) = g(\alpha(\mu f)) = \alpha(\mu f)$
 - $\alpha(\mu f)$ is a fixed-point of g and $\mu g \sqsubseteq \alpha(\mu f)$
- ② $\alpha(\mu f) \sqsubseteq \mu g$
 - Consider y a fixed-point of $g : g(y) = y$ and $\mu g \sqsubseteq y$.
 - $\gamma(y)$ is a fixed-point of f
 - $\mu f \sqsubseteq \gamma(y)$
 - $\alpha(\mu f) \sqsubseteq y$
 - $\alpha(\mu f) \sqsubseteq \mu g$

Second theorem

- ▶ Suppose that $C \xleftrightarrow[\alpha]{\gamma} A$ is a Galois connection
- ▶ Two functions $f \in C \rightarrow C$ and $g \in A \rightarrow A$:

$$\begin{array}{ccc} C & \xrightarrow{f} & C \\ \gamma \uparrow & & \downarrow \alpha \\ A & \xrightarrow{g} & A \end{array}$$

- ▶ f and g are monotone
- ▶ $\alpha \circ f \sqsubseteq g \circ \alpha$.

Then $\alpha(\mu f) \sqsubseteq \mu g$.

- ▶ $\alpha \in \mathcal{P}(\mathbb{Z}) \rightarrow \text{Signs} : \begin{cases} z & \alpha(z) \\ z < 0 & \text{neg} \\ z > 0 & \text{pos} \\ z = 0 & \text{zero} \end{cases}$
- ▶ $f \in \mathcal{P}(\mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{Z})$ where $f(X) = \{0\} \cup \{x+2 \mid x \in \mathbb{Z} \wedge x \in X\}$
- ▶ $g = \alpha \circ f \circ \gamma$
- ▶ $f^0 = \emptyset, f^1 = \{0\}, f^2 = \{0, 2\}, \dots$
- ▶ $g(\perp) = \perp, g^1 = \alpha \circ f \circ \gamma(\perp) = [0, \infty[, g^2 = [0, \infty[, \dots$ and $\forall i \geq 2 : g^i = [0, \infty[.$
- ▶ $\mu.g = [0, \infty[$

Definition

∇ is a widening operator over (L, \sqsubseteq) ($\nabla \in L \times L \rightarrow L$)

- ▶ For any x and y in L : $x \sqcup y \sqsubseteq x \nabla y$
- ▶ For any sequence $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_3 \dots \sqsubseteq x_i \sqsubseteq x_{i+1} \dots$, the sequence $\{y_i | i \in \mathbb{N}\}$
 - $y_0 = x_0$
 - $y_{i+1} = y_i \nabla x_{i+1}$

stabilizes after a finite amount of time.

- ▶ $\mathbb{I}(\mathbb{Z}) = \{\perp\} \cup \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{\infty\}, l \leq u\}$
- ▶ $(\mathbb{I}(\mathbb{Z}), \sqsubseteq)$ est une structure partiellement ordonnée.
- ▶ $[l_1, u_1] \nabla [l_2, u_2] = [\text{cond}(l_2 < l_1, -\infty, l_1), \text{cond}(u_1 < u_2, \infty, u_1)]$
- ▶ $[2, 3] \nabla [1, 4] = [-\infty, \infty]$
- ▶ $[0, 1] \sqsubseteq [0, 3]$
- ▶ $[0, 1] \nabla [0, 3] = [0, \infty]$.
- ▶ $[0, 3] \nabla [0, 2] = [0, 3]$.
- ▶ $[0, 2] \nabla ([0, 1] \nabla [0, 2]) = [0, \infty]$
- ▶ $([0, 2] \nabla [0, 1]) \nabla [0, 2] = [0, 2]$

Definition of a sound approximation of a function f

A function $g \in A \rightarrow A$ is a sound approximation of a function $f \in C \rightarrow C$, if it satisfies the following condition :

$$\forall c \in C : \forall a \in A : \alpha(c) \sqsubseteq a \Rightarrow \alpha(f(c)) \sqsubseteq g(a)$$

Properties

Suppose that $C \xrightleftharpoons[\alpha]{\gamma} A$ is a Galois connection.

The four statements are equivalent

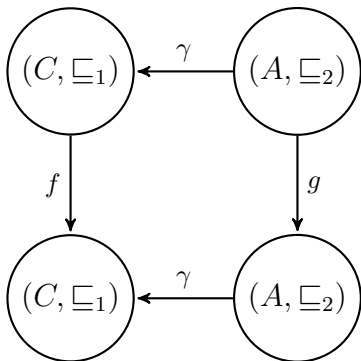
- ① g is a sound approximation of f with respect to a Galois connection
- ② $\alpha \circ f \sqsubseteq g \circ \alpha$
- ③ $\alpha \circ f \circ \gamma \sqsubseteq g$
- ④ $f \circ \gamma \sqsubseteq \gamma \circ g$
- ⑤ $f \sqsubseteq \gamma \circ g \circ \alpha$

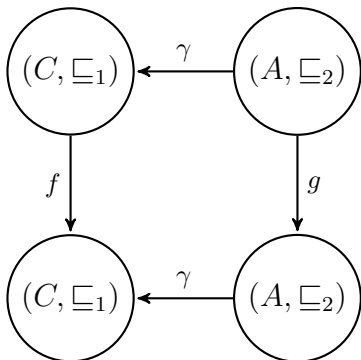
Example of a sound approximation of the invariant of a system

- ▶ C is the set of concrete states : $cv \in Var \longrightarrow \mathcal{P}(\mathbb{Z})$: if X is in Var , then $cv(X) \in \mathcal{P}(\mathbb{Z})$.
- ▶ A is the set of abstract states : $av \in Var \longrightarrow Signs$: if X is in Var , then $av(X) \in Signs$.
- ▶ (α, γ) is extended as :
 (α_1, γ_1) entre $(Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq)$ et $(Var \longrightarrow Signs, \sqsubseteq)$. En particulier, $\alpha_1(cv) = av$ et, pour tout X de Var ,
 $av(X) = \alpha(cv(X))$; $\gamma_1(av) = cv$ et, pour tout X de Var ,
 $cv(X) = \gamma(av(X))$.

Computing the set of computing states of a transition system

- ▶ $Init \subseteq C$ is the set of initial states.
- ▶ **NEXT** defines the transition over concrete states
- ▶ $REACHABLE(TS) = \{u | u \in C \wedge (\exists x_0. x_0 \in C \wedge (x_0 \in Init) \wedge NEXT^*(x_0, x))\}$
- ▶ pour toute partie U de Σ , $U = FP(U)$
- ▶ pour toute partie U de Σ , $FP(U) = Init_S \cup \longrightarrow [U]$



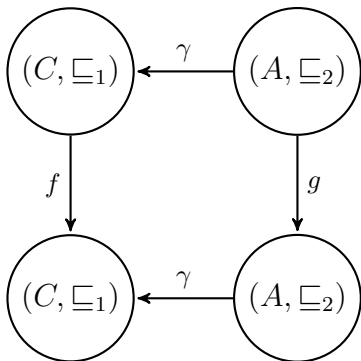


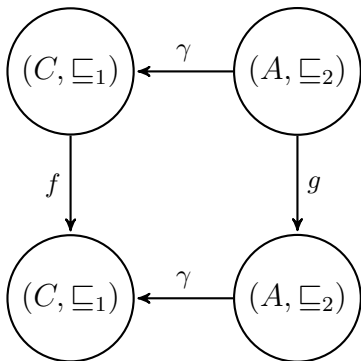
First Theorem

- ▶ Suppose that $C \xleftrightarrow[\alpha]{\gamma} A$ is a Galois connection
- ▶ Two functions $f \in C \rightarrow C$ and $g \in A \rightarrow A$:
- ▶ f and g are monotone
- ▶ $\alpha \circ f = g \circ \alpha$.

Then $\alpha(\mu.f) = \mu.g$.

- ① $\mu g \sqsubseteq \alpha(\mu f)$
 - $f(\mu f) = \mu f$ (fixed-point property)
 - $\alpha(f(\mu f)) = \alpha(\mu f)$ (applying the relation over f and g)
 - $\alpha(f(\mu f)) = g(\alpha(\mu f)) = \alpha(\mu f)$
 - $\alpha(\mu f)$ is a fixed-point of g and $\mu g \sqsubseteq \alpha(\mu f)$
- ② $\alpha(\mu f) \sqsubseteq \mu g$
 - Consider y a fixed-point of $g : g(y) = y$ and $\mu g \sqsubseteq y$.
 - $\gamma(y)$ is a fixed-point of f
 - $\mu f \sqsubseteq \gamma(y)$
 - $\alpha(\mu f) \sqsubseteq y$
 - $\alpha(\mu f) \sqsubseteq \mu g$





Second Theorem

- ▶ Suppose that $C \xleftrightarrow[\alpha]{\gamma} A$ is a Galois connection
- ▶ Two functions $f \in C \rightarrow C$ and $g \in A \rightarrow A$:
- ▶ f and g are monotone
- ▶ $\alpha \circ f \sqsubseteq g \circ \alpha$.

Then $\alpha(\mu f) \sqsubseteq \mu g$.

- ▶ $\alpha \in \mathcal{P}(\mathbb{Z}) \rightarrow \text{Signs} : \begin{cases} z & \alpha(z) \\ z < 0 & \text{neg} \\ z > 0 & \text{pos} \\ z = 0 & \text{zero} \end{cases}$
- ▶ $f \in \mathcal{P}(\mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{Z})$ where $f(X) = \{0\} \cup \{x+2 \mid x \in \mathbb{Z} \wedge x \in X\}$
- ▶ $g = \alpha \circ f \circ \gamma$
- ▶ $f^0 = \emptyset, f^1 = \{0\}, f^2 = \{0, 2\}, \dots$
- ▶ $g(\perp) = \perp, g^1 = \alpha \circ f \circ \gamma(\perp) = [0, \infty[, g^2 = [0, \infty[, \dots$ and $\forall i \geq 2 : g^i = [0, \infty[.$
- ▶ $\mu.g = [0, \infty[$

- ## 17 Conclusion

Current Subsection Summary

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing**
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples

Definition

∇ is a widening operator over (L, \sqsubseteq) ($\nabla \in L \times L \rightarrow L$)

- ▶ For any x and y in $L : x \sqcup y \sqsubseteq x \nabla y$
- ▶ For any sequence $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_3 \dots \sqsubseteq x_i \sqsubseteq x_{i+1} \dots$, the sequence $\{y_i | i \in \mathbb{N}\}$
 - $y_0 = x_0$
 - $y_{i+1} = y_i \nabla x_{i+1}$

stabilizes after a finite amount of time.

Intervals

- ▶ $\perp \nabla \perp = \perp$
- ▶ $\perp \nabla (l, u) = (l, u) \nabla \perp = (l, u)$
- ▶ $(l1, u1) \nabla (l2, u2) = \left(\begin{pmatrix} -\infty \text{ if } l2 < l1 \\ l1 \end{pmatrix}, \begin{pmatrix} \infty \text{ if } u2 > u1 \\ u1 \end{pmatrix} \right)$

- ▶ $\mathbb{I}(\mathbb{Z}) = \{\perp\} \cup \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{\infty\}, l \leq u\}$
- ▶ $(\mathbb{I}(\mathbb{Z}), \sqsubseteq)$ est une structure partiellement ordonnée.
- ▶ $[l_1, u_1] \nabla [l_2, u_2] = [\text{cond}(l_2 < l_1, -\infty, l_1), \text{cond}(u_1 < u_2, \infty, u_1)]$
- ▶ $[2, 3] \nabla [1, 4] = [-\infty, \infty]$
- ▶ $[0, 1] \sqsubseteq [0, 3]$
- ▶ $[0, 1] \nabla [0, 3] = [0, \infty]$.
- ▶ $[0, 3] \nabla [0, 2] = [0, 3]$.
- ▶ $[0, 2] \nabla ([0, 1] \nabla [0, 2]) = [0, \infty]$
- ▶ $([0, 2] \nabla [0, 1]) \nabla [0, 2] = [0, 2]$

Current Subsection Summary

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing**
 - Widening
 - Narrowing
- 16 Analysis of Programs
 - Example
 - Analysing Iterative Programs
 - Examples

- ## Example
- ### Analysing Iterative Programs
- ### Examples

Current Subsection Summary

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs**
 - Example**
 - Analysing Iterative Programs
 - Examples

```
ℓ0 :  
y := -11;  
ℓ0 :  
IF   x < y  THEN  
    ℓ1 :  
    z := y;  
    ℓ2 :  
ELSE  
    ℓ3 :  
    z := x;  
    ℓ4 :  
FI  
ℓ5 :
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
     $\ell_1 :$   
     $z := y;$   
     $\ell_2 :$   
ELSE  
     $\ell_3 :$   
     $z := x;$   
     $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 :$   
ELSE  
   $\ell_3 :$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 :$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := x;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := y;$   
   $\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$   
FI  
 $\ell_5 :$ 
```



```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
     $\ell_1 : y < 0 \quad x < 0$   
     $z := x;$   
     $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
     $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
     $z := y;$   
     $\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$   
FI  
 $\ell_5 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := x;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := y;$   
   $\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$   
FI  
 $\ell_5 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$ 
```

Result : $y < 0 \quad x \in \mathbb{Z} \quad z < 0$ means that $z < 0$ is an information resulting from the analysis over abstract domain of

Current Subsection Summary

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs**
 - Example
 - Analysing Iterative Programs
 - Examples

- ▶ \mathcal{MS} is $(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \dots, r_n\})$
- ▶ $\text{NEXT} \stackrel{\text{def}}{=} r_0 \vee \dots \vee r_n.$
- ▶ S is a safety property, when
 $\forall y, x \in \text{VALS}. \text{Init}(y) \wedge \text{NEXT}^*(y, x) \Rightarrow x \in S.$
- ▶ $(\mathcal{P}(\text{VALS}), \subseteq, \emptyset, \cup, \cap)$ is a complete lattice.
- ▶ μF is defined as follows :
 - $F^0 = \emptyset$
 - $F^{i+1} = F(F_i), \forall i \in \mathbb{N}$
 - $\mu F = \text{Sup}\{F^i | i \in \mathbb{N}\}$
 - For any safety property S , $\mu F \subseteq S.$

Computing the least fixed-point over a finite lattice

```
INPUT   $tf \in T \longrightarrow T$ 
OUTPUT  $result = \mu.f$ 
VARIABLES   $x, y \in T, i \in \mathbb{N}$ 
 $\ell_0 : \{x, y \in T\}$ 
 $x := \perp;$ 
 $y := \perp;$ 
 $i := 0;$ 
 $\ell_{11} : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T) \wedge i = 0\};$ 
WHILE   $i \leq Card(T)$ 
   $\ell_1 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)\};$ 
   $x := f(x);$ 
   $\ell_2 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)\};$ 
   $y := x \sqcup y;$ 
   $\ell_3 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0; k=i+1} F^k \wedge i \leq Card(T)\};$ 
   $i := i+1;$ 
   $\ell_4 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)+1\};$ 
OD;
 $\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i = Card(T)+1\};$ 
 $result := y;$ 
 $\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i = Card(T)+1 \wedge result = y\};$ 
```

- ▶ Abstract interpretation is a general framework for defining sound approximation of the semantics of computer programs, based on monotonic functions over ordered sets, especially lattices.
- ▶ Main concrete application is formal static analysis, the automatic extraction of information about the possible executions of computer programs.
- ▶ When defining an abstract domain, it can be finite (domain of signs) or infinite (domain of intervals) : it means that we have to manage undecidability questions for computing fixed-points.
- ▶ interproc is a tool that can be used for analysing recursive programs and for playing with abstract interpretation.

Current Subsection Summary

- 1 Introduction
- 2 Example of analysis
- 3 Static analysis
- 4 Overview of the methodology
- 5 Standard, Collecting and Abstract Semantics
- 6 TOP
- 7 Finding Sound Abstractions for Computing
- 8 Galois Connections
- 9 Examples of Galois connections
- 10 TOP2
- 11 Domain of Signs
- 12 Domain of intervals
- 13 Abstraction and approximation
- 14 TOP3
- 15 Widening and Narrowing
 - Widening
 - Narrowing
- 16 Analysis of Programs**
 - Example
 - Analysing Iterative Programs
 - Examples

```
ℓ0 :  
y := -11;  
ℓ0 :  
IF   x < y  THEN  
    ℓ1 :  
    z := y;  
    ℓ2 :  
ELSE  
    ℓ3 :  
    z := x;  
    ℓ4 :  
FI  
ℓ5 :
```



```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
     $\ell_1 :$   
     $z := y;$   
     $\ell_2 :$   
ELSE  
     $\ell_3 :$   
     $z := x;$   
     $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 :$   
ELSE  
   $\ell_3 :$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
     $\ell_1 : y < 0 \quad x < 0$   
     $z := y;$   
     $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
     $\ell_3 :$   
     $z := x;$   
     $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := y;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := x;$   
   $\ell_4 :$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
   $\ell_1 : y < 0 \quad x < 0$   
   $z := x;$   
   $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
   $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
   $z := y;$   
   $\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$   
FI  
 $\ell_5 :$ 
```

```
 $\ell_0 :$   
 $y := -11;$   
 $\ell_0 : y < 0$   
IF  $x < y$  THEN  
     $\ell_1 : y < 0 \quad x < 0$   
     $z := x;$   
     $\ell_2 : y < 0 \quad x < 0 \quad z < 0$   
ELSE  
     $\ell_3 : y < 0 \quad x \in \mathbb{Z}$   
     $z := y;$   
     $\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$   
FI  
 $\ell_5 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$ 
```

$\ell_0 :$

$y := -11;$

$\ell_0 : y < 0$

IF $x < y$ **THEN**

$\ell_1 : y < 0 \quad x < 0$

$z := x;$

$\ell_2 : y < 0 \quad x < 0 \quad z < 0$

ELSE

$\ell_3 : y < 0 \quad x \in \mathbb{Z}$

$z := y;$

$\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$

FI

$\ell_5 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$

Result

$y < 0, x \in \mathbb{Z}, z < 0$ means that $z < 0$ is an information

- ▶ \mathcal{MS} is $(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \dots, r_n\})$
- ▶ $\text{NEXT} \stackrel{\text{def}}{=} r_0 \vee \dots \vee r_n.$
- ▶ S is a safety property, when
 $\forall y, x \in \text{VALS}. \text{Init}(y) \wedge \text{NEXT}^*(y, x) \Rightarrow x \in S.$
- ▶ $(\mathcal{P}(\text{VALS}), \subseteq, \emptyset, \cup, \cap)$ is a complete lattice.
- ▶ μF is defined as follows :
 - $F^0 = \emptyset$
 - $F^{i+1} = F(F_i), \forall i \in \mathbb{N}$
 - $\mu F = \text{Sup}\{F^i | i \in \mathbb{N}\}$
 - For any safety property S , $\mu F \subseteq S.$

Computing the least fixed-point over a finite lattice

```
INPUT   $tf \in T \longrightarrow T$ 
OUTPUT  $result = \mu.f$ 
VARIABLES   $x, y \in T, i \in \mathbb{N}$ 
 $\ell_0 : \{x, y \in T\}$ 
 $x := \perp;$ 
 $y := \perp;$ 
 $i := 0;$ 
 $\ell_{11} : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T) \wedge i = 0\};$ 
WHILE   $i \leq Card(T)$ 
   $\ell_1 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)\};$ 
   $x := f(x);$ 
   $\ell_2 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)\};$ 
   $y := x \sqcup y;$ 
   $\ell_3 : \{x, y \in T \wedge x = F^{i+1} \wedge y = \bigcup_{k=0; k=i+1} F^k \wedge i \leq Card(T)\};$ 
   $i := i+1;$ 
   $\ell_4 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i \leq Card(T)+1\};$ 
OD;
 $\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i = Card(T)+1\};$ 
 $result := y;$ 
 $\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0; k=i} F^k \wedge i = Card(T)+1 \wedge result = y\};$ 
```


- ▶ Abstract interpretation is a general framework for defining sound approximation of the semantics of computer programs, based on monotonic functions over ordered sets, especially lattices.
- ▶ Main concrete application is formal static analysis, the automatic extraction of information about the possible executions of computer programs.
- ▶ When defining an abstract domain, it can be finite (domain of signs) or infinite (domain of intervals) : it means that we have to manage undecidability questions for computing fixed-points.
- ▶ interproc is a tool that can be used for analysing recursive programs and for playing with abstract interpretation.

- \mathcal{R} : exigences du système.

- ▶ \mathcal{R} : exigences du système.
- ▶ \mathcal{D} : domaine du problème.
- ▶ \mathcal{S} : système répondant aux spécifications.

$$\mathcal{D}, \mathcal{S} \text{ SATISFAIT } \mathcal{R}$$

- ▶ \mathcal{R} : exigences du système.
- ▶ \mathcal{D} : domaine du problème.
- ▶ \mathcal{S} : système répondant aux spécifications.

$$\mathcal{D}, \mathcal{S} \quad \text{SATISFAIT} \quad \mathcal{R}$$

- ▶ \mathcal{R} : pre/post.
- ▶ \mathcal{D} : entiers, réels, ...
- ▶ \mathcal{S} : code, procédure, programme, ...

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \mathbf{pre}(\text{ALG})(v) \\ \mathbf{post}(\text{ALG})(v_0, v) \end{array} \right.$$

\mathcal{D}
<hr/>
$\mathbf{pre}(\text{ALG})(v)$
$\mathbf{post}(\text{ALG})(v_0, v)$
<hr/>
ALG

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \mathbf{pre}(\text{ALG})(v) \\ \mathbf{post}(\text{ALG})(v_0, v) \end{array} \right.$$



Vérification de conditions de vérification

\mathcal{D}
<hr/>
$\mathbf{pre}(\text{ALG})(v)$
$\mathbf{post}(\text{ALG})(v_0, v)$
<hr/>
ALG

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \text{pre}(\text{ALG})(v) \\ \text{post}(\text{ALG})(v_0, v) \end{array} \right.$$



Vérification de conditions de vérification

\mathcal{D}
$\text{pre}(\text{ALG})(v)$
$\text{post}(\text{ALG})(v_0, v)$
ALG

- Vérification des conditions de vérification avec un model-checker par exploration de tous les états.

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \text{pre}(\text{ALG})(v) \\ \text{post}(\text{ALG})(v_0, v) \end{array} \right.$$



Vérification de conditions de vérification

\mathcal{D}
$\text{pre}(\text{ALG})(v)$
$\text{post}(\text{ALG})(v_0, v)$
ALG

- ▶ Vérification des conditions de vérification avec un model-checker par exploration de tous les états.
- ▶ Vérification des conditions de vérification avec un outil de preuve formelle.

\mathcal{D}, ALG SATISFAIT $\left\{ \begin{array}{l} \text{requires ALG}(v) \\ \text{ensures ALG}(v_0, v) \end{array} \right.$



Vérification de conditions de vérification

\mathcal{D}
requires ALG(v)
ensures ALG(v_0, v)
ALG

$$\mathcal{D}, \text{ALG} \text{ SATISFAIT } \left\{ \begin{array}{l} \text{requires ALG}(v) \\ \text{ensures ALG}(v_0, v) \end{array} \right.$$



Vérification de conditions de vérification

\mathcal{D}
<hr/>
$\text{requires ALG}(v)$
$\text{ensures ALG}(v_0, v)$
<hr/>
ALG

- ▶ Vérification des conditions de vérification avec un outil de preuve formelle QED
- ▶ Vérification des conditions de vérification avec un outil de preuve formelle Alt-Ergo