Cours MALG & MOVEX

Analyse des programmes

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2023-2024

Plan

Sommaire

Current Summary

Static Program Analysis

- 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

Foundational Ideas of Abstract Interpretation

- A Theory described in works of Patrick Cousot (1976 $\longrightarrow$ now), Father of the Abstract Interpretation : analysis of large codes for embeded software A380 with Astrée.
- ...and Radhia Cousot is the second AImate of the Abstract Interpretation Company.
- A Comprehensive Web Site is maintanined by Professor Patrick Cousot at ENS Paris.
- *abstract interpretation relies on an idea of discrete approximation which consists in replacing the reasoning on a concrete exact semantics by a computation on an abstract approximate semantics.*
- A theory unifying abstract and concrete objects with respect to a given semantics
- A theory providing a way to *transfer* statements from a concrete (complex) to an abstract (simpler) semantcical framework.
- A theory formalizing the approximated analyse of programs and allowing to compare relative precision of analyses

AI or Abstract Interpretation

- 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 pricniles (see Event B)
  - AI provides another solution by trabfering results from a *concret* framework to an *abstract* structure

Static Analysis of Program Properties

- $\mathcal{CS}(P)$ 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 arrors 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 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi = \varnothing$
- Case 2 : $\mathcal{CS}(P) \cap \varphi \neq \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$
- Case 3 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$

Static Analysis of Program Properties

- Case 1 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi = \varnothing$:
  - 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 : $\mathcal{CS}(P) \cap \varphi \neq \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$:
  - 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 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$:
  - 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

Current Summary

The Interproc Analyzer

- a web interface to the Interproc analyzer connected to the APRON Abstract Domain Library
- Analysis of programs uting 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
```

Current Summary

Static analysis

Static program analysis analyses computer software without actually executing programs :

- 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

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

Simple example

ℓ_0 :
$y := -11;$
ℓ_0 :
IF $x < y$ **THEN**
  ℓ_1 :
  $z := y;$
  ℓ_2 :
ELSE
  ℓ_3 :
  $z := x;$
  ℓ_4 :
FI
ℓ_5 :

Simple example

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

Simple example

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

Simple example

$\ell_0 :$
$y := -11;$
$\ell_0 : y < 0$
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
$\ell_5 :$

Simple example

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

Simple example

$\ell_0 :$

$y := -11;$

$\ell_0 : y < 0$

IF $\quad x < y \quad$ **THEN**

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

$\quad z := x;$

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

ELSE

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

$\quad z := x;$

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

FI

$\ell_5 :$

Simple example

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

Simple example

ℓ_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.

Verification by computing set of reachable states

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

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

$(\mathcal{P}(\text{VALS}), \subseteq, \varnothing, \cup, \cap)$ 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.
- $\text{REACHABLE}(\mathcal{MS}) = \mu\mathcal{F}$
- μF is defined as follows :
  - $F^0 = \varnothing$
  - $F^{i+1} = F(F^i)$, $\forall i \in \mathbb{N}$
  - $\mu F = Sup\{F^i | i \in \mathbb{N}\}$
  - 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 $x, y \in T, i \in \mathbb{N}$
$\ell_0 : \{x, y \in T\}$
$x := \bot;$
$y := \bot;$
$i := 0;$
$\ell_{11} : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i \leq Card(T) \wedge i = 0\};$
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;
$\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T){+}1\};$
$result := y;$
$\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T){+}1 \wedge result = y\};$

Verification in action

- Identify the safety property S to check.
- Run the algorithm for computing μF.
- Check that $\mu F \subseteq S$ or $\overline{S} \cap \mu F = \varnothing$.
- Check that $BUG \cap \mu F = \varnothing$, when BUG is a set of states that you identify as *bad states*.

Problem

- The general case is either infinite or large ... approximations of μF.
- Computing over abstract finite domain
- How to compute when it is not decidable ?
- Develop a framework for defining sound abstractions of software systems under analysis.

Current Summary

Standard, Collecting and Abstract Semantics

- 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.

Collecting semantics

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.

Summary of the technique

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

Small Programming Language

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

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

Defining the semantics of the small programming language

- Semantic Domains

$$\begin{array}{lcl} Mem & \stackrel{def}{=} & \mathbb{V} \longrightarrow \mathbb{Z} \\ States & \stackrel{def}{=} & \mathbb{C} \times Mem \end{array}$$

- Semantics for Expressions

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

$$\begin{array}{lcl} \mathcal{E}[\![v]\!](m) & \stackrel{def}{=} & \{v\} \\ \mathcal{E}[\![?]\!](m) & \stackrel{def}{=} & \mathbb{Z} \\ \mathcal{E}[\![x]\!](m) & \stackrel{def}{=} & \{m(x)\} \\ \mathcal{E}[\![e_1 \ op \ e_2]\!](m) & \stackrel{def}{=} & \{v | \exists ve_1, ve_2 . \left(\begin{array}{l} ve_1 \in \mathcal{E}[\![e_1]\!](m) \\ ve_2 \in \mathcal{E}[\![e_2]\!](m) \\ v = ve_1 \ o \ ve_2 \end{array} \right) \} \end{array}$$

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

$$
\begin{array}{lcl}
tt \in \mathcal{C}[\![e_1 \ relop \ e_2]\!](m) & \stackrel{def}{=} & \exists v_1, v2. \left(\begin{array}{l} v_1 \in \mathcal{E}[\![e_1]\!](m) \\ v_2 \in \mathcal{E}[\![\ e_2]\!](m) \\ v_1 \ relop \ v2 \end{array} \right) \\
ff \in \mathcal{C}[\![e_1 \ relop \ e_2]\!](m) & \stackrel{def}{=} & \exists v_1, v2. \left(\begin{array}{l} v_1 \in \mathcal{E}[\![e_1]\!](m) \\ v_2 \in \mathcal{E}[\![\ e_2]\!](m) \\ \neg(v_1 \ relop \ v2) \end{array} \right) \\
be_1 \ \wedge \ be_2 \in \mathcal{C}[\![be_1 \ \mathbf{and} \ be_2]\!](m) & \stackrel{def}{=} & and \left(\begin{array}{l} be_1 \in \mathcal{C}[\![be_1]\!](m) \\ be_2 \in \mathcal{C}[\![be_2]\!](m) \end{array} \right)
\end{array}
$$

Structural Operational Semantics : Small-step Semantics

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

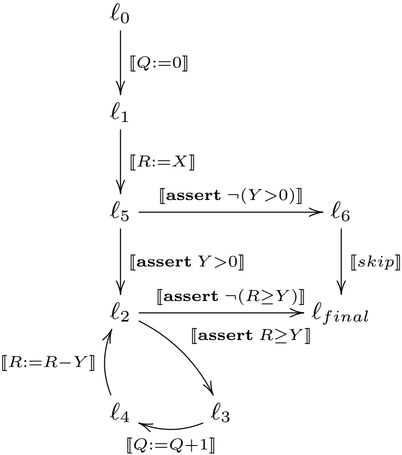
Generating Control Flowchart Graph from Program

- A *control flow graph* is generated from the program under consideration namely P.
- A control flow graph $\mathcal{CFG}[\![P]\!]$ is defined by nodes $(l \in \mathcal{C})$ which are program control points of P, $\mathcal{C}ontrol[\![P]\!]$ and by labelled edges with actions ($\mathcal{A}ctions[\![P]\!]$) defined by the following rules :

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

- A *control flow graph* is effectively defined by :
  - $\ell_{init} \in \mathcal{C}ontrol[\![P]\!]$: the entry point
  - $\ell_{end} \in \mathcal{C}ontrol[\![P]\!]$: the exit point
  - $\mathcal{E}dges[\![P]\!] \subseteq \mathcal{C}ontrol[\![P]\!] \times \mathcal{A}ctions[\![P]\!] \times \mathcal{C}ontrol[\![P]\!]$
- $\mathcal{CFG}[\![P]\!] = (\ell_{init}, \mathcal{E}dges[\![P]\!], \ell_{end})$

$\ell_0[Q := 0];$
$\ell_1[R := X];$
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



Small-step Semantics for Control Flowcharts

- $Mem \stackrel{def}{=} \mathbb{V} \longrightarrow \mathbb{Z}$
- Semantics of actions : $\xrightarrow{a} \subseteq Mem \times Mem$
  $m \xrightarrow{x:=e} m[x \mapsto v]$ **if** *there is a value* $v \in \mathcal{E}[\![e]\!](m)$
  $m \xrightarrow{skip} m$
  $m \xrightarrow{\mathbf{assert}\ be} m]$ $if\ tt \in \mathcal{C}[\![be]\!](m)$
- Semantics for $\mathcal{CFG}[\![P]\!]$: $\xrightarrow{P} \subseteq States \times States$
  - **If** $m \xrightarrow{a} m'$ and $(\ell_1, a, \ell_2) \in \mathcal{E}dges[\![P]\!]$, 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 $[\![P]\!] = \{s | \exists s_0 \in \{\ell_{init} \times Mem : s_0 \xrightarrow[\star]{P} s\}$.

- Defining for each control point ℓ of P the set of reachables values :
$$[\![P]\!]^{coll}_{\ell} = \{s|s \in States \wedge s \in [\![P]\!] \wedge \exists m \in Mem : s = (\ell, m)\}$$
- Characterizing $[\![P]\!]^{coll}_{\ell}$: it satisfies the system of equations
$$\forall \ell \in \mathcal{C}(P).X_{\ell} = X^{init}_{\ell} \cup \bigcup_{(\ell_1,a,\ell) \in \mathcal{E}dges[\![P]\!]} [\![a]\!](X_{\ell_1}) \qquad (1)$$
- Let $a \in \mathcal{A}ctions[\![P]\!]$ and $x \subseteq Mem$.
$$[\![a]\!](x) = \{e|e \in States \wedge \exists f.f \in x \wedge f \xrightarrow{a} e\}$$
- $\forall \ell \in \mathcal{C}(P). \left(\begin{array}{l} \ell = \ell_{init} \Rightarrow X^{init}_{\ell} = Mem \\ \ell \neq \ell_{init} \Rightarrow X^{init}_{\ell} = \varnothing \end{array} \right)$

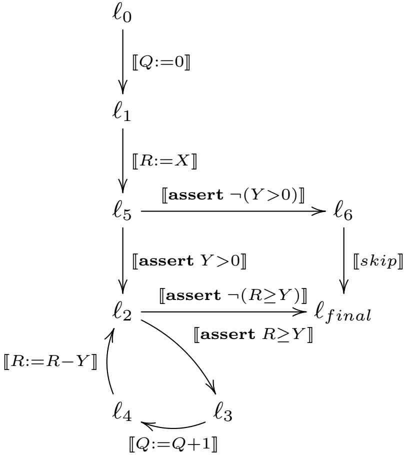
. . . .

☺ Théorème Let F the function defined as follows :

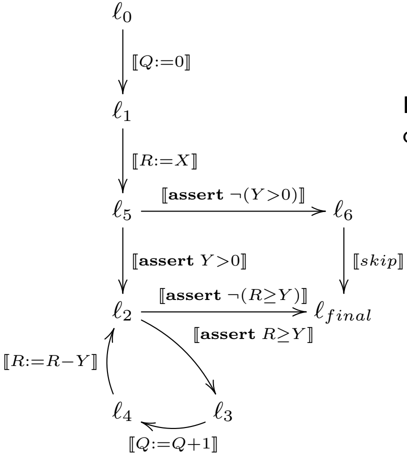
- n is the cardinality of $\mathcal{C}(P)$.
- $F \in \mathcal{P}(States)^n \longrightarrow \mathcal{P}(States)^n$
- If $X \in \mathcal{P}(States)^n$, then $F(X) = (\ldots, F_\ell(X), \ldots)$
- $\forall \ell \in \mathcal{C}(P).F_\ell(X) = X_\ell^{init} \cup \bigcup_{(\ell_1,a,\ell) \in \mathcal{E}dges[\![P]\!]} [\![a]\!](X_{\ell_1})$

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

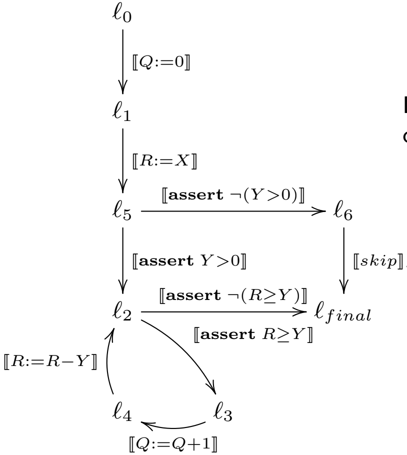
From flowchart to equational system



From flowchart to equational system



Defining equational systems for the collecting semantics :



Defining equational systems for the collecting semantics :

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

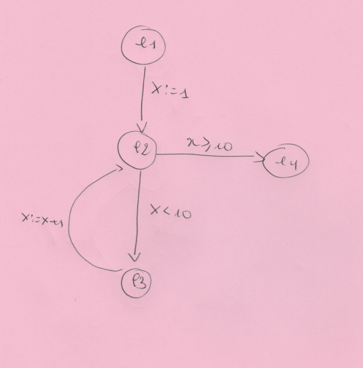
Solving the equational system

- 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



- System of equations over $(\mathcal{P}(\mathbb{Z}, \subseteq)$
  - $X_1 = \mathbb{Z}$
  - $X_2 = \{1\} \cup \{v | v \in \mathbb{Z} \wedge v - 1 \in X_3\}$
  - $X_3 = \{v | v \in X_2 \wedge v < 10 \}$
  - $X_4 = \{v | v \in X_2 \wedge v \geq 10 \}$
- Reachability
  - $X_1 = \mathbb{Z}$
  - $X_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
  - $X_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $X_4 = \{10\}$

Current Summary

Finding Sound Abstractions for Computing

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

$$
\begin{tikzcd}
\mathbb{N} \times \mathbb{N} \arrow[r, "abstraction"] \arrow[d, "\star"'] & A \times A \arrow[d, "\bar{\star}"] \\
\mathbb{N} & A \arrow[l, "concretization"']
\end{tikzcd}
$$

Finding Sound Abstractions for Computing

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

$$
\begin{tikzcd}
\mathbb{N}\times\mathbb{N} \arrow[r, "abstraction"] \arrow[d, "\star"'] & A\times A \arrow[d, "\bar{\star}"] \\
\mathbb{N} & A \arrow[l, "concretization"']
\end{tikzcd}
$$

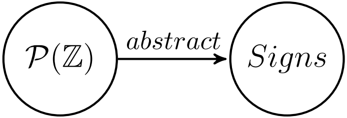
☺ Rule
If $\overline{a \star b} \neq \overline{a}\bar{\star}\overline{b}$,
then the operation is not correct.

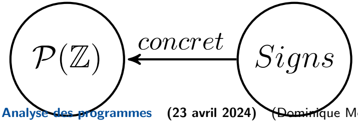
Finding Sound Abstractions for Computing

- 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 $abstract(z) \in Signs$.
- 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 $abstract(z) \in Signs$.
- 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 : $\{-2,2,8\} \subseteq concrete(nonzero)$

$$\mathcal{P}(\mathbb{Z}) \xrightarrow{abstract} Signs$$

$$\mathcal{P}(\mathbb{Z}) \xleftarrow{concret} Signs$$

- $concret(pos) = \{z | z \in \mathbb{Z} \wedge z > 0\}$
- $concret(neg) = \{z | z \in \mathbb{Z} \wedge z < 0\}$
- $concret(non) = \varnothing$
- $concret(nonzero) = \{z | z \in \mathbb{Z} \wedge z \neq 0\}$

Finding Sound Abstractions for Computing

- 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 :
  - $\{-2, 2, 8\} \subseteq concrete(nonzero)$
    $abstract(\{-2, 2, 8\}) = nonzero$

Finding Sound Abstractions for Computing

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
(C, \sqsubseteq) \arrow[r, "abstraction"] \arrow[d, "concrete\ operator"'] & (A, \sqsubseteq) \arrow[d, "abstract\ operator"] \\
(C, \sqsubseteq) & (A, \sqsubseteq) \arrow[l, "concretization"']
\end{tikzcd}
$$

Current Summary

Defining good abstractions

- Let $(A, \sqsubseteq, \sqcup, \sqcap)$ be a complete lattice. Let Q a subset of A. Q is a Moore family, if for each part Q' of Q, $\sqcap Q' \in Q$.
- **Property** : Let $(A, \sqsubseteq, \sqcup, \sqcap)$ be a complete lattice and $B \subseteq A$.
  1. If, for any $p \in A$, $\{q \in B | p \sqsubseteq q\}$ has a least element, then B is Moore family.
  2. If B is Moore family, then for any $p \in A$, $\{q \in B | p \sqsubseteq q\}$ has a least element.

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

Defining good abstractions

- 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 operatorρ such that $B = \rho(A)$
- B is a *good* abstraction of C, if it satisfies $B = \rho(C)$ where ρ is a closure operator.

Defining good abstractions

- 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.

Finding Sound Abstractions for Computing

$$
\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "cf"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "af"] \\
(C, \sqsubseteq_1) \arrow[r, "\gamma"] & (A, \sqsubseteq_2)
\end{tikzcd}
$$

Finding Sound Abstractions for Computing

$$
\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "cf"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "af"] \\
(C, \sqsubseteq_1) \arrow[r, "\gamma"] & (A, \sqsubseteq_2)
\end{tikzcd}
$$

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

Finding Sound Abstractions for Computing

$$\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "cf"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "af"] \\
(C, \sqsubseteq_1) \arrow[r, "\gamma"] & (A, \sqsubseteq_2)
\end{tikzcd}$$

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

Finding Sound Abstractions for Computing

$$\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "cf"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "af"] \\
(C, \sqsubseteq_1) \arrow[r, "\gamma"] & (A, \sqsubseteq_2)
\end{tikzcd}$$

- 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 $\forall a \in A.cr(\gamma(a_1), \ldots, \gamma(a_n)) \sqsubseteq_1 \gamma(ac(a_1, \ldots, a_n)))$

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 : $L \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L'$

Properties of a Galois connection $L \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L'$

- 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 $\gamma \circ \alpha \circ \gamma = \gamma$
- $\alpha(x) = \bigcap'\{y \in L' | x \sqsubseteq \gamma(y)\}$
- $\gamma(y) = \bigcup\{x \in L | \alpha(x) \sqsubseteq' y\}$

Properties

- $\gamma \circ \alpha \circ \gamma \circ \alpha = \gamma \circ \alpha$
- $\alpha \circ \gamma \circ \alpha \circ \gamma = \alpha \circ \gamma$
- We assume that $\{(\alpha_i, \gamma_i) | i \in \{1 \ldots n\}\}$ is a family of Galois connections :
$$L_1 \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} L_2 \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \ldots L_{n-1} \underset{\alpha_{n-1}}{\overset{\gamma_{n-1}}{\leftrightarrows}} L_n$$
  Then $(\alpha_1; \ldots; \alpha_i; \ldots; \alpha_{n-1}, \gamma_{n-1}; \ldots \gamma_i; \ldots; \gamma_1)$ is a Galois connection. or equivalently
  $L_1 \underset{\alpha_{n-1} \circ \ldots \circ \alpha_i \circ \ldots \circ \alpha_1}{\overset{\gamma_1 \circ \ldots \gamma_i \circ \ldots \circ \gamma_{n-1}}{\leftrightarrows}}$ 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, $\gamma_1 = \gamma_2$

Current Summary

Examples

- 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 :
  - $\mathsf{pre}[t]P \stackrel{def}{=} \{s | s \in S \wedge \exists s'.((s,s') \in t \wedge s' \in P)\}$
  - $\widetilde{\mathsf{pre}}\,[t]P \stackrel{def}{=} \{s | s \in S \wedge \forall s'.((s,s') \in t \Rightarrow s' \in P)\}$
  - $\mathsf{post}[t]P \stackrel{def}{=} \{s | s \in S \wedge \exists s'.((s',s) \in t \wedge s' \in P)\}$
  - $\widetilde{\mathsf{post}}\,[t]P \stackrel{def}{=} \{s | s \in S \wedge \forall s'.((s',s) \in t \Rightarrow s' \in P)\}$
- Duality of operators :
  1. $\widetilde{\mathsf{pre}}\,[t]\neg P = \neg \mathsf{pre}[t]P$
  2. $\widetilde{\mathsf{post}}\,[t]\neg P = \neg \mathsf{post}[t]P$
- Galois connections over $\mathcal{P}$, the set of subsets of S :

$$(\mathcal{P}, \subseteq) \underset{\mathsf{pre}[t]}{\overset{\widetilde{\mathsf{post}}[t]}{\leftrightarrows}} (\mathcal{P}, \subseteq) \qquad (\mathcal{P}, \subseteq) \underset{\mathsf{post}[t]}{\overset{\widetilde{\mathsf{pre}}[t]}{\leftrightarrows}} (\mathcal{P}, \subseteq)$$

Examples

- 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 $\mathbb{G}raph(f) \subseteq \mathbb{G}raph(g)$.
  - $\alpha_1 = \lambda P.\lambda l.\{m|(l,m) \in P\}$
  - $\gamma_1 = \lambda Q.\{(l,m)|l \in \mathcal{L} \wedge m \in Q(l)\}$
  - $(\mathcal{P}(\mathcal{L}\times\mathcal{M}), \subseteq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\mathcal{L} \longrightarrow \mathcal{P}(\mathcal{M}), \subseteq)$ 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 $B(p) = \{m|m \in \mathcal{M} \wedge p(m)\}$, set of predicates in which p holdsd.
  - Next we define :
    - $\alpha_2 = \lambda Q.\{p|p \in Pred \wedge Q \subseteq B(p)\}$
    - $\gamma_2 = \lambda P.\cap\{B(p)|p \in P\}$
  - $(\mathcal{P}(\mathcal{M}), \subseteq) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (\mathcal{P}(Pred), \Rightarrow)$ is a Galois connection.

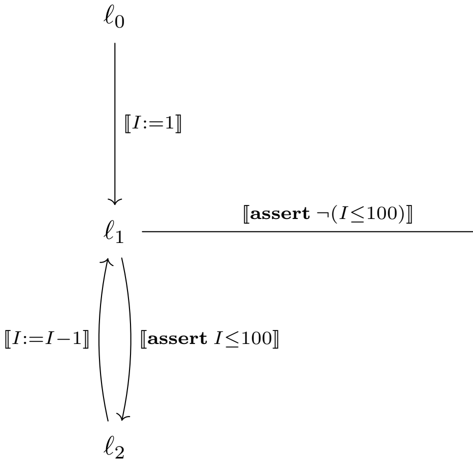
Examples

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

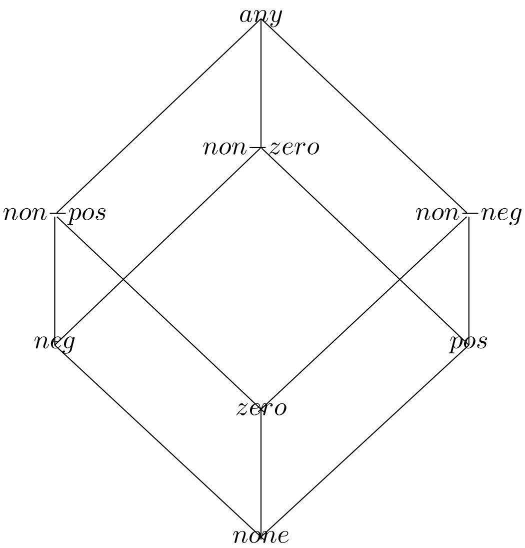
Current Summary

Examples of Abstractions

$\ell_0[I := 1];$
while $\ell_1[I \leq 100]$ **do**
  $\ell_2[I := I+1];$
end while
$\ell_{final}[skip]$



Domain of Signs



- Defining an abstraction for integers $\alpha \in \mathcal{P}(\mathbb{Z}) \longrightarrow \mathbb{S}igns$
- $\left\{ \begin{array}{ll} z & \alpha(z) \\ z < 0 & neg \\ z > 0 & pos \\ z = 0 & zero \end{array} \right.$

Composing Galois Connections

- Abstraction by projection :
  $(\mathcal{P}(Var \longrightarrow \mathbb{Z}), \subseteq) \underset{\alpha_\pi}{\overset{\gamma_\pi}{\leftrightarrows}} (Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq)$
- Abstraction of signs
  $(Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha_{sign}}{\overset{\gamma_{sign}}{\leftrightarrows}} (Var \longrightarrow \mathbb{S}igns), \subseteq)$
- Composition of abstractions :
  $(\mathcal{P}(Var \longrightarrow \mathbb{Z}), \subseteq) \underset{\alpha_{sign} \circ \alpha_\pi}{\overset{\gamma_\pi \circ \gamma_{sign}}{\leftrightarrows}} (Var \longrightarrow \mathbb{S}igns), \subseteq)$
- $\alpha = \alpha_{sign} \circ \alpha_\pi$ and $\gamma = \gamma_\pi \circ \gamma_{sign}$

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

$$
\begin{tikzcd}
L \arrow[r, "f"] & L \arrow[d, "\alpha"] \\
L' \arrow[u, "\gamma"] \arrow[r, "f'"] & L'
\end{tikzcd}
$$

$$f' = \alpha \circ f \circ \gamma \tag{2}$$

f' is the best approximation of f

Best evaluation and abstract evaluation

- Concrete states : $cv \in Var \longrightarrow \mathcal{P}(\mathbb{Z})$: if if X is in Var, then $cv(X) \in \mathcal{P}(\mathbb{Z})$.
- Abstract states : $av \in Var \longrightarrow \mathbb{S}igns$: if X is in Var, then $av(X) \in \mathbb{S}igns$.
- (α, γ) is extended as :
  (α_1, γ_1) entre $(Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq)$ et $(Var \longrightarrow \mathbb{S}igns, \sqsubseteq)$. 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 : $States = Var \longrightarrow \mathcal{P}(\mathbb{Z})$:
    $[\![e]\!] \in (Var \longrightarrow \mathcal{P}(\mathbb{Z})) \longrightarrow \mathcal{P}(\mathbb{Z})$ and $[\![e]\!](cv)$
  - abstract domain : $AStates = Var \longrightarrow \mathbb{S}igns$:
    $[\![e]\!]_a \in (Var \longrightarrow \mathbb{S}igns) \longrightarrow \mathbb{S}igns$ and $[\![e]\!]_a(av)$.

Domain of signs

- The best abstraction is simply dedined as follows : $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$.
- Applying the best approximation for assignment : $[\![x := e]\!]_{best}(av) = \begin{cases} av(y), y \neq x \\ [\![e]\!]_{best}(av) \end{cases}$
- $(\mathcal{P}(Var \longrightarrow \mathbb{Z}), \subseteq)$: $A, B \in \mathcal{P}(\mathbb{Z}) : A{+}B = \{a{+}b | a \in A \wedge b \in B\}$
- $(Var \longrightarrow \mathbb{S}igns), \subseteq)$: $x, y \in \mathbb{S}igns : x{\oplus}y = \alpha(\gamma(x){+}\gamma(y))$
- 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 $\oplus$.

Forward analysis in the domain of signs : first way

- 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	any	any
ℓ_1	pos	any
ℓ_2	pos	pos
ℓ_3	pos	pos
ℓ_4	pos	$non-neg$
ℓ_{final}	$non-neg$	$non-neg$

- ℓ_3 to ℓ_4 : abstract value of Y is pos and by γ, we obtain $(1, +\infty)$ a,d 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 accelarating the convergence like widening.
- Computing is still costly : computing now in the abstraction and defining a sound approximation of f.

Forward analysis in the domain of signs : second way

- Evaluation is using the *best* approximation :
  $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$

Forward analysis in the domain of signs : second way

- Evaluation is using the *best* approximation :
  $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$
- Computing over the concrete domain is remaining complex

Forward analysis in the domain of signs : second way

- Evaluation is using the *best* approximation :
  $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$
- Computing over the concrete domain is remaining complex
- Idea : approximation of the *best* approximation :

Forward analysis in the domain of signs : second way

- Evaluation is using the *best* approximation :
  $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$
- Computing over the concrete domain is remaining complex
- Idea : approximation of the *best* approximation :$[\![e]\!]_a$ and, for any av abstract state, $[\![e]\!]_{best}(av) \sqsubseteq [\![e]\!]_a(av)$.

Forward analysis in the domain of signs : second way

- Evaluation is using the *best* approximation : $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$
- Computing over the concrete domain is remaining complex
- Idea : approximation of the *best* approximation :$[\![e]\!]_a$ and, for any av abstract state, $[\![e]\!]_{best}(av) \sqsubseteq [\![e]\!]_a(av)$.
- Abstract semantics is defined as follows : $av \in Var \longrightarrow \mathbb{S}igns$:

Forward analysis in the domain of signs : second way

- Evaluation is using the *best* approximation :
  $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$
- Computing over the concrete domain is remaining complex
- Idea : approximation of the *best* approximation :$[\![e]\!]_a$ and, for any av abstract state, $[\![e]\!]_{best}(av) \sqsubseteq [\![e]\!]_a(av)$.
- Abstract semantics is defined as follows :
  $av \in Var \longrightarrow \mathbb{S}igns$:
  - $[\![const]\!]_a(av) = \alpha(\{c\})$
  - $[\![x]\!]_a(av) = av(x)$
  - $[\![e_1+e_2]\!]_a(av) = [\![e_1]\!]_a(av) \oplus [\![e_2]\!]_a(av)$
  - $[\![e_1+e_2]\!]_a(av) = [\![e_1]\!]_a(av) \otimes [\![e_2]\!]_a(av)$

Forward analysis in the domain of signs : second way

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

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 :

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

$$\begin{tikzcd}
L \arrow[r, "f"] & L \arrow[d, "\alpha"] \\
L' \arrow[u, "\gamma"] \arrow[r, "f'"] & L'
\end{tikzcd}$$

The four statements are equivalent

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

Defining an abstract semantics of expressions

- $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$ provide the best abstraction but is costly.
- Another solution is to define an abstract semantics for expressions : $[\![e]\!]_a$ such that for any av, $[\![e]\!]_{best}(av) \sqsubseteq [\![e]\!]_a(av)$.
- $av \in Var \longrightarrow \mathbb{S}igns$:
  - $[\![const]\!]_a(v) = \alpha(\{c\})$
  - $[\![x]\!]_a(v) = v(x)$
  - $[\![e_1{+}e_2]\!]_a(v) = [\![e_1]\!]_a(v) \oplus [\![e_2]\!]_a(v)$
  - $[\![e_1{+}e_2]\!]_a(v) = [\![e_1]\!]_a(v) \otimes [\![e_2]\!]_a(v)$

$\ell[X := E]$

- $[\![Y-1]\!]_a(av) = [\![Y]\!]_a(av) \oplus [\![-1]\!](av)_a = pos \oplus neg = may$
- $[\![Y-1]\!]_{best}(av) = \alpha_1 \circ [\![Y-1]\!] \circ \gamma_1(av) == \alpha_1([\![Y-1]\!](\gamma_1(av))) = \alpha_1([\![Y-1]\!](\{Y \mapsto (1, +\infty)\}) = \alpha_1((1+\infty)+(-1)) = \alpha_1((0, +\infty)) = non-neg$

Forward analysis in the domain of signs using the approximation

- 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	any	any
ℓ_1	pos	any
ℓ_2	pos	pos
ℓ_3	pos	pos
ℓ_4	pos	any
ℓ_{final}	any	any

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

Current Summary

Abstract Domain of Intervals

- $\mathbb{I}(\mathbb{Z}) = \{\perp\} \cup \{[l,u] | l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{\infty\}, l \leq u\}$
- $[l_1,u_1] \sqsubseteq [l_2,u_2]$ si, et seulement si, $l2 \leq l1$ 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. $[l_1,u_1] \sqcap [l_2,u_2] = \begin{cases} [max(l_1,l_2), min(u_1,u_2)] \\ \perp, si\ max(l_1,l_2) > min(u_1,u_2) \end{cases}$
- $(\mathbb{I}(\mathbb{Z}), \sqcup)$ is a complete lattice.
- 1. $\alpha(X) = \begin{cases} [min(X), max(X)] \\ \perp, si\ X = \varnothing \end{cases}$
  2. $\gamma([l,u]) = [l..u]$ et $\gamma(\perp] = \varnothing$
- (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. $i_1 \oslash i_2 =$ $[min(l_1/l_2, l_1/u_2, u_1/l_2, u_1/u_2, max(l_1/l_2, l_1/u_2, u_1/l_2, u_1/u_2)]$

Current Summary

Operators for approximating fixed-point computations

- Computing *collecting semantics* is generally undecidable :
  - S is a safety property for $\mathcal{MS}$ if, and only if, $\textsc{reachable}(\mathcal{MS}) \subseteq \mathcal{S}$.
  - Finding a sound approximation of $\textsc{reachable}(\mathcal{MS})$, denoted $\alpha(\textsc{reachable}(\mathcal{MS}))$, and satisfying $\gamma(\alpha(\textsc{reachable}(\mathcal{MS}))) \subseteq \mathcal{S}$.
  - $\textsc{reachable}(\mathcal{MS}) \subseteq \gamma(\alpha(\textsc{reachable}(\mathcal{MS})))$ and $\gamma(\alpha(\textsc{reachable}(\mathcal{MS}))) \subseteq \mathcal{S}$.
- Abstract domains can be finite as the domain of Signs but the domain of intervals is infinite : computing $\textsc{reachable}(\mathcal{MS})$ remains undecidable but we can approximate its computation.
- Abstract domains can be infinite : we have to accelarate 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 :

$$
\begin{tikzcd}
L \arrow[r, "f"] & L \arrow[d, "\alpha"] \\
L' \arrow[u, "\gamma"] \arrow[r, "f'"] & L'
\end{tikzcd}
$$

$$f' = \alpha \circ f \circ \gamma \tag{3}$$

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 :

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

$$\begin{tikzcd}
L \arrow[r, "f"] & L \arrow[d, "\alpha"] \\
L' \arrow[u, "\gamma"] \arrow[r, "f'"] & L'
\end{tikzcd}$$

The four statements are equivalent

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

Defining an abstract semantics of expressions

- $[\![e]\!]_{best}(av) = \alpha \circ [\![e]\!] \circ \gamma_1(av)$ provides the best abstraction but is costly.
- Another solution is to define an abstract semantics for expressions : hide $[\![e]\!]_a$ such that for any av, $[\![e]\!]_{best}(av) \sqsubseteq [\![e]\!]_a(av)$.
- $av \in Var \longrightarrow \mathbb{S}igns$:
  - $[\![const]\!]_a(v) = \alpha(\{c\})$
  - $[\![x]\!]_a(v) = v(x)$
  - $[\![e_1+e_2]\!]_a(v) = [\![e_1]\!]_a(v) \oplus [\![e_2]\!]_a(v)$
  - $[\![e_1+e_2]\!]_a(v) = [\![e_1]\!]_a(v) \otimes [\![e_2]\!]_a(v)$

Approximation of a function f

- Suppose that $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ is a Galois connection
- a function $f \in C \to C$: to find a function g

$$
\begin{tikzcd}
C \arrow[r, "f"] & C \arrow[d, "\alpha"] \\
A \arrow[u, "\gamma"] \arrow[r, "g"] & A
\end{tikzcd}
$$

- 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 $g \in A \longrightarrow A$ is a sound approximation of a function $f \in C \longrightarrow C$, 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 $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ 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. $f \sqsubseteq \gamma \circ g \circ \alpha$

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

Best abstraction

Suppose that :

- $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ is a Galois connection.
- $f \in C \longrightarrow C$ 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 $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ is a Galois connection
- Two functions $f \in C \rightarrow C$ and $g \in A \rightarrow A$:

$$
\begin{tikzcd}
C \arrow[r, "f"] & C \arrow[d, "\alpha"] \\
A \arrow[u, "\gamma"] \arrow[r, "g"] & A
\end{tikzcd}
$$

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

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

Proof

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 $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ is a Galois connection
- Two functions $f \in C \to C$ and $g \in A \to A$:

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

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

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

Example of computation

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

Widening

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 $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_3 \ldots \sqsubseteq x_i \sqsubseteq x_{i+1} \ldots$, 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.

Widening

Theorem

If $\bigtriangledown$ is a widening operator over $(L, \sqsubseteq)$ ($\bigtriangledown \in L \times L \rightarrow L$), then the ascending sequence $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_3 \ldots \sqsubseteq x_i \sqsubseteq x_{i+1} \ldots$ defined by :

- $x_0 = \bot$
- $x_{i+1} = x_i \bigtriangledown f(x_i)$

is eventually stationary and its limit satisfies $lfp(f) \sqsubseteq \sqsubseteq \{x_i | i \in \mathbb{N}\}$ stabilizes after a finite amount of time.

- Using $\bigtriangledown$ instead of $\sqsubseteq$ for computing approximation of upper bound.

Defining Widening Operator

Intervals

- $\bot \bigtriangledown \bot = \bot$
- $\bot \bigtriangledown (l, u) = (l, u) \bigtriangledown \bot = (l, u)$
- $(l1, u1) \bigtriangledown (l2, u2) = \left(\left(\begin{array}{l} -\infty \ if \ l2 < l1 \\ l1 \end{array}\right), \left(\begin{array}{l} \infty \ if \ u2 > u1 \\ u1 \end{array}\right)\right)$

Examples of widening

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

Approximation of a fixed-point operator

Let us assume that $(L, \sqsubseteq)$ is a complete lattice and f is a monotonic function defined from L to L.

Theorem

If $\bigtriangledown \in L \times L \rightarrow L$ is a widening operator, then the sequence $\{x_i | i \in \mathbb{N}\}$ defined by

- $x_0 = \perp$
- $x_{i+1} = \begin{cases} x_i \ if \ f(x_i) \sqsubseteq x_i \\ x_i \bigtriangledown f(x_i) \end{cases}$

is eventually stationary and its limit satisfies $lfp(f) \sqsubseteq \bigcup \{x_i | i \in \mathbb{N}\}$

Current Summary

Definition of a sound approximation of a function f

A function $g \in A \longrightarrow A$ is a sound approximation of a function $f \in C \longrightarrow C$, 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 $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ 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 : $av \in Var \longrightarrow \mathbb{S}igns$: if X is in Var, then $av(X) \in \mathbb{S}igns$.
- (α, γ) is extended as : (α_1, γ_1) entre $(Var \longrightarrow \mathcal{P}(\mathbb{Z}), \subseteq)$ et $(Var \longrightarrow \mathbb{S}igns, \sqsubseteq)$. 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 TS

- $Init \subseteq C$ is the set of initial states.
- $\textsc{Next}$ defines the transition over concrete states
- $\textsc{reachable}(TS) = \{u | u \in C \wedge (\exists x_0 . x_0 \in C \wedge (x_0 \in Init) \wedge \textsc{Next}^\star(x_0, x))\}$
- pour toute partie U de Σ, $U = \mathrm{FP}(U)$
- pour toute partie U de Σ, $\mathrm{FP}(U) = Init_{\mathrm{S}} \cup \longrightarrow [U]$

Sound approximation of fixed-point

$$
\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "f"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "g"] \\
(C, \sqsubseteq_1) & (A, \sqsubseteq_2) \arrow[l, "\gamma"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "f"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "g"] \\
(C, \sqsubseteq_1) & (A, \sqsubseteq_2) \arrow[l, "\gamma"']
\end{tikzcd}
$$

First Theorem

- Suppose that $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ is a Galois connection
- Two functions $f \in C \to C$ and $g \in A \to A$:
- f and g are monotone
- $\alpha \circ f = g \circ \alpha$.

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

Proof

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
   - $\mu f \sqsubseteq \gamma(y)$
   - $\alpha(\mu f) \sqsubseteq y$
   - $\alpha(\mu f) \sqsubseteq \mu g$

Sound approximation of fixed-point

$$\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "f"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "g"] \\
(C, \sqsubseteq_1) & (A, \sqsubseteq_2) \arrow[l, "\gamma"']
\end{tikzcd}$$

Sound approximation of fixed-point

$$\begin{tikzcd}
(C, \sqsubseteq_1) \arrow[d, "f"'] & (A, \sqsubseteq_2) \arrow[l, "\gamma"'] \arrow[d, "g"] \\
(C, \sqsubseteq_1) & (A, \sqsubseteq_2) \arrow[l, "\gamma"']
\end{tikzcd}$$

Second Theorem

- Suppose that $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ 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 $\alpha(\mu f) \sqsubseteq \mu g$.

Example of computation

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

Current Summary

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 $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_3 \ldots \sqsubseteq x_i \sqsubseteq x_{i+1} \ldots$, 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.

Widening

Theorem

If $\bigtriangledown$ is a widening operator over $(L, \sqsubseteq)$ ($\bigtriangledown \in L \times L \rightarrow L$), then the ascending sequence $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_3 \ldots \sqsubseteq x_i \sqsubseteq x_{i+1} \ldots$ defined by :

- $x_0 = \bot$
- $x_{i+1} = x_i \bigtriangledown f(x_i)$

is eventually stationary and its limit satisfies $lfp(f) \sqsubseteq \sqsubseteq \{x_i | i \in \mathbb{N}\}$ stabilizes after a finite amount of time.

- Using $\bigtriangledown$ instead of $\sqsubseteq$ for computing approximation of upper bound.

Defining Widening Operator

Intervals

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

Examples of widening

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

Approximation of a fixed-point operator

Let us assume that $(L, \sqsubseteq)$ is a complete lattice and f is a monotonic function defined from L to L.

Theorem

If $\bigtriangledown \in L \times L \rightarrow L$ is a widening operator, then the sequence $\{x_i | i \in \mathbb{N}\}$ defined by

- $x_0 = \bot$
- $x_{i+1} = \begin{cases} x_i \ if \ f(x_i) \sqsubseteq x_i \\ x_i \bigtriangledown f(x_i) \end{cases}$

is eventually stationary and its limit satisfies $lfp(f) \sqsubseteq \bigcup \{x_i | i \in \mathbb{N}\}$

Current Summary

Current Subsection Summary

Simple example

ℓ_0 :
$y := -11;$
ℓ_0 :
IF $x < y$ **THEN**
  ℓ_1 :
  $z := y;$
  ℓ_2 :
ELSE
  ℓ_3 :
  $z := x;$
  ℓ_4 :
FI
ℓ_5 :

Simple example

ℓ_0 :
$y := -11;$
$\ell_0 : y < 0$
IF $x < y$ **THEN**
  ℓ_1 :
  $z := y;$
  ℓ_2 :
ELSE
  ℓ_3 :
  $z := x;$
  ℓ_4 :
FI
ℓ_5 :

Simple example

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

Simple example

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

Simple example

ℓ_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
  $\ell_3 : y < 0 \quad x \in \mathbb{Z}$
  $z := x;$
  ℓ_4 :
FI
ℓ_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
$\ell_3 : y < 0 \quad x \in \mathbb{Z}$
$z := y;$
$\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$
FI
$\ell_5 :$

Simple example

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

Simple example

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

Current Subsection Summary

Verification by computing set of reachable states

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

Computing the least fixed-point over a finite lattice

```
INPUT tf ∈ T ⟶ T
OUTPUT result = μ.f
VARIABLES x, y ∈ T, i ∈ ℕ
ℓ0 : {x, y ∈ T}
x := ⊥;
y := ⊥;
i := 0;
ℓ11 : {x, y ∈ T ∧ x = F^i ∧ y = ⋃_{k=0;k=i} F^k ∧ i ≤ Card(T) ∧ i = 0};
WHILE i ≤ Card(T)
  ℓ1 : {x, y ∈ T ∧ x = F^i ∧ y = ⋃_{k=0;k=i} F^k ∧ i ≤ Card(T)};
  x := f(x);
  ℓ2 : {x, y ∈ T ∧ x = F^{i+1} ∧ y = ⋃_{k=0;k=i} F^k ∧ i ≤ Card(T)};
  y := x ⊔ y;
  ℓ3 : {x, y ∈ T ∧ x = F^{i+1} ∧ y = ⋃_{k=0;k=i+1} F^k ∧ i ≤ Card(T)};
  i := i+1;
  ℓ4 : {x, y ∈ T ∧ x = F^i ∧ y = ⋃_{k=0;k=i} F^k ∧ i ≤ Card(T)+1};
OD;
ℓ5 : {x, y ∈ T ∧ x = F^i ∧ y = ⋃_{k=0;k=i} F^k ∧ i = Card(T)+1};
result := y;
ℓ6 : {x, y ∈ T ∧ x = F^i ∧ y = ⋃_{k=0;k=i} F^k ∧ i = Card(T)+1 ∧ result = y};
```

Conclusion and Comments

- 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 (diomain 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 Summary

Simple example

ℓ_0 :
$y := -11;$
ℓ_0 :
IF $x < y$ **THEN**
  ℓ_1 :
  $z := y;$
  ℓ_2 :
ELSE
  ℓ_3 :
  $z := x;$
  ℓ_4 :
FI
ℓ_5 :

Simple example

ℓ_0 :
$y := -11;$
$\ell_0 : y < 0$
IF $x < y$ **THEN**
  ℓ_1 :
  $z := y;$
  ℓ_2 :
ELSE
  ℓ_3 :
  $z := x;$
  ℓ_4 :
FI
ℓ_5 :

Simple example

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

Simple example

$\ell_0 :$

$y := -11;$

$\ell_0 : y < 0$

IF $\quad x < y \quad$ **THEN**

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

$\quad z := y;$

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

ELSE

$\quad \ell_3 :$

$\quad z := x;$

$\quad \ell_4 :$

FI

$\ell_5 :$

Simple example

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

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
$\ell_3 : y < 0 \quad x \in \mathbb{Z}$
$z := y;$
$\ell_4 : y < 0 \quad x \in \mathbb{Z} \quad z < 0$
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

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

  $z := y;$

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

FI

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

Simple example

ℓ_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

Verification by computing set of reachable states

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

i

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;
$\ell_5 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T)+1\};$
$result := y;$
$\ell_6 : \{x, y \in T \wedge x = F^i \wedge y = \bigcup_{k=0;k=i} F^k \wedge i = Card(T)+1 \wedge result = y\};$

Conclusion and Comments

- 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 (diomain 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.

Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

- $\mathcal{R}$: exigences du système.

Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

- $\mathcal{R}$: exigences du système.
- $\mathcal{D}$: domaine du problème.

Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

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

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

- $\mathcal{R}$: 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, ...

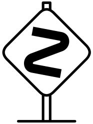
Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$

$$\boxed{\begin{array}{l} \mathcal{D} \\ \hline \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \\ \hline \textsc{Alg} \end{array}}$$

Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textsf{\textbf{pre}}(\textsc{Alg})(v) \\ \textsf{\textbf{post}}(\textsc{Alg})(v_0, v) \end{cases}$$



Vérification de conditions de vérification

$\mathcal{D}$
$\textsf{\textbf{pre}}(\textsc{Alg})(v)$ $\textsf{\textbf{post}}(\textsc{Alg})(v_0, v)$
$\textsc{Alg}$

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \begin{cases} \textbf{pre}(\text{ALG})(v) \\ \textbf{post}(\text{ALG})(v_0, v) \end{cases}$$

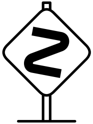
Vérification de conditions de vérification

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

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

Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$



Vérification de conditions de vérification

$\mathcal{D}$
$\textbf{pre}(\textsc{Alg})(v)$ $\textbf{post}(\textsc{Alg})(v_0, v)$
$\textsc{Alg}$

- 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.

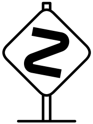
Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \texttt{requires}\ \textsc{Alg}(v) \\ \texttt{ensures}\ \textsc{Alg}(v_0, v) \end{cases}$$

$\mathcal{D}$
`requires` $\textsc{Alg}(v)$ `ensures` $\textsc{Alg}(v_0, v)$
$\textsc{Alg}$

Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \left\{ \begin{array}{l} \texttt{requires } \textsc{Alg}(v) \\ \texttt{ensures } \textsc{Alg}(v_0, v) \end{array} \right.$$

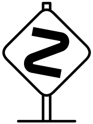


Vérification de conditions de vérification

$\mathcal{D}$
$\texttt{requires } \textsc{Alg}(v)$ $\texttt{ensures } \textsc{Alg}(v_0, v)$
$\textsc{Alg}$

Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \left\{ \begin{array}{l} \texttt{requires } \textsc{Alg}(v) \\ \texttt{ensures } \textsc{Alg}(v_0, v) \end{array} \right.$$



Vérification de conditions de vérification

$\mathcal{D}$
$\texttt{requires } \textsc{Alg}(v)$ $\texttt{ensures } \textsc{Alg}(v_0, v)$
$\textsc{Alg}$

- 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