



# Cours MALG & MOVEX

# MALG **Analyse des programmes**

Dominique Méry Telecom Nancy, Université de Lorraine (21 avril 2025 at 10:54 P.M.)

Année universitaire 2024-2025

### Outline

1 Overview of the methodololy

2 Standard, Collecting and Abstract Semantics

## Main concepts

- ▶ Syntax of programs ( $P \in PL$ ) defines the class of programs for applying the analysis@
- ▶ **Semantics** ( $\llbracket P \rrbracket$ ) for each program ( $P \in PL$ )

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

#### Two examples of annotated programs

$$\begin{array}{l} \ell_0[X := 0]; \\ \ell_1[Y := Y + X]; \\ \ell_2[skip] \\ \ell_3[X := Y]; \end{array}$$

```
\begin{array}{l} \ell_0[Q:=0];\\ \ell_1[R:=X];\\ \textbf{IF}\ \ell_5[Y>0]\\  \qquad \qquad \textbf{WHILE}\ \ell_2[R\geq Y]\\ \ell_3[Q:=Q+1];\\ \ell_4[R:=R-Y]\\ \textbf{ENDWHILE}\\ \textbf{ELSE}\\ \ell_6[skip]\\ \textbf{ENDIF} \end{array}
```

## Defining the semantics of the small programming language

► Semantic Domains

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

Semantics for Expressions

$$\begin{split} & \mathcal{E}\llbracket v \rrbracket(m) \in \mathcal{P}(\mathbb{Z}), \ e \in Expr, m \in Mem, \ x \in \mathbb{V}, \ op \in \{+, -, \times, /\} \\ & \mathcal{E}\llbracket v \rrbracket(m) & \stackrel{def}{=} \quad \{v\} \\ & \mathcal{E}\llbracket v \rrbracket(m) & \stackrel{def}{=} \quad \mathbb{Z} \\ & \mathcal{E}\llbracket x \rrbracket(m) & \stackrel{def}{=} \quad \{m(x)\} \\ & \mathcal{E}\llbracket e_1 \ op \ e_2 \rrbracket(m) & \stackrel{def}{=} \quad \{v | \exists ve_1, ve_2. \left( \begin{array}{c} ve_1 \in \mathcal{E}\llbracket e_1 \rrbracket(m) \\ ve_2 \in \mathcal{E}\llbracket e_2 \rrbracket(m) \\ v = ve_1 \ o \ ve_2 \end{array} \right) \} \end{split}$$

## Defining the semantics of the small programming language

Semantics for conditions  $\mathcal{C}[[cond]](m) \in \mathcal{P}(\mathbb{B}), cond \in Cond, m \in Mem, x \in \mathbb{V},$  $op \in \{+, -, \times, /\}$  $\stackrel{def}{=} \exists v_1, v_2. \left( \begin{array}{c} v_1 \in \mathcal{E}\llbracket e_1 \rrbracket(m) \\ v_2 \in \mathcal{E}\llbracket e_2 \rrbracket(m) \\ v_1 \ relop \ v_2 \\ \end{array} \right)$   $\stackrel{def}{=} \exists v_1, v_2. \left( \begin{array}{c} v_1 \in \mathcal{E}\llbracket e_1 \rrbracket(m) \\ v_2 \in \mathcal{E}\llbracket e_2 \rrbracket(m) \\ v_2 \in \mathcal{E}\llbracket e_2 \rrbracket(m) \\ \neg (v_1 \ relop \ v_2) \end{array} \right)$  $tt \in \mathcal{C}[e_1 \ relop \ e_2](m)$ 

$$be_1 \wedge be_2 \in \mathcal{C}\llbracket be_1 \text{ and } be_2 \rrbracket(m) \stackrel{def}{=} and \begin{pmatrix} be_1 \in \mathcal{C}\llbracket be_1 \rrbracket(m) \\ be_2 \in \mathcal{C}\llbracket be_2 \rrbracket(m) \end{pmatrix}$$

 $ff \in \mathcal{C}[e_1 \ relop \ e_2](m)$ 

## **Structural Operational Semantics : Small-step Semantics**

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



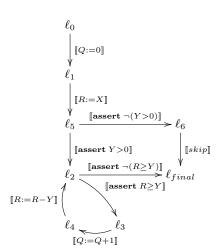
## Generating Control Flowchart Graph from Program

- ▶ 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,  $Control[\![P]\!]$  and by labelled edges with actions  $(Actions[\![P]\!])$  defined by the following rules :

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

- 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
  - $\mathcal{E}dges[\![P]\!] \subseteq \mathcal{C}ontrol[\![P]\!] \times \mathcal{A}ctions[\![P]\!] \times \mathcal{C}ontrol[\![P]\!]$
- $\triangleright \ \mathcal{CFG}[\![P]\!] = (\ell_{init}, \mathcal{E}dges[\![P]\!], \ell_{end})$

## From program to flowchart



## **Small-step Semantics for Control Flowcharts**

- $ightharpoonup Mem \stackrel{def}{=} \mathbb{V} \longrightarrow \mathbb{Z}$
- ▶ Semantics for  $\mathcal{CFG}\llbracket P \rrbracket : \xrightarrow{P} \subseteq States \times States$ 
  - If  $m \stackrel{a}{\longrightarrow} m'$  and  $(\ell_1, a, \ell_2) \in \mathcal{E} dges \llbracket P \rrbracket$ , then  $(\ell_1, m) \stackrel{P}{\longrightarrow} (\ell_2, m')$
  - The set of initial states is  $\{\ell_{init}\} \times Mem$
  - The set of reachable states for P is denoted REACHABLE(P) and defined by  $[\![P]\!] = \{s | \exists s_0 \in \{\ell_{init} \times Mem : s_0 \xrightarrow{P} s\}.$



## **Collecting Semantics for Programs**

lackbox Defining for each control point  $\ell$  of P the set of reachables values :

$$[\![P]\!]^{coll}_\ell = \{s | s \in States \land s \in [\![P]\!] \land \exists m \in Mem : s = (\ell, m)\}$$

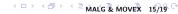
lackbox Characterizing  $[\![P]\!]_\ell^{coll}$  : it satisfies the system of equations

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

▶ Let  $a \in Actions[P]$  and  $x \subseteq Mem$ .

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

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



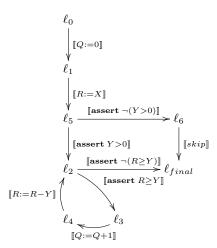
## **Collecting Semantics for Programs**

.....

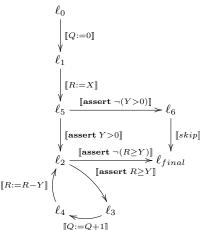
- © Théorème Let F the function defined as follows :
  - ightharpoonup n is the cardinality of  $\mathcal{C}(P)$ .
  - $ightharpoonup F \in \mathcal{P}(States)^n \longrightarrow \mathcal{P}(States)^n$
  - ▶ If  $X \in \mathcal{P}(States)^n$ , then  $F(X) = (\dots, F_{\ell}(X), \dots)$
  - $\blacktriangleright \ \forall \ell \in \mathcal{C}(P).F_{\ell}(X) = X_{\ell}^{init} \cup \bigcup_{(\ell_1, a, \ell) \in \mathcal{E}dges\llbracket P \rrbracket} \ \llbracket a \rrbracket(X_{\ell_1})$

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

## From flowchart to equational system

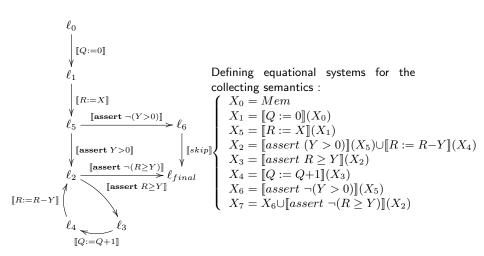


### From flowchart to equational system



Defining equational systems for the collecting semantics :

## From flowchart to equational system



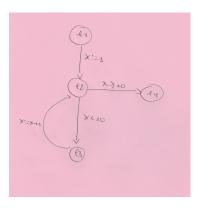
## Solving the equational system

- ► 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 Ifp solution of these equational systems in any case.

## **Example for computing reachable states**



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