

## Cours MALG & MOVEX

## Modélisation, spécification et vérification (III)

Dominique Méry  
Telecom Nancy, Université de Lorraine

Année universitaire 2025-2026 (26 janvier 2026 9:01 A.M.)

- ① Principe(s) d'induction
- ② Méthode de preuves de propriétés d'invariance
- ③ stop movex 3
- ④ Exemples de correction partielle (affectation simple)
- ⑤ Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- ⑥ Checking invariant and safety properties in PlusCal
- ⑦ Conclusion et limites

- 1 Principe(s) d'induction
- 2 Méthode de preuves de propriétés d'invariance
- 3 stop movex 3
- 4 Exemples de correction partielle (affectation simple)
- 5 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 6 Checking invariant and safety properties in PlusCal
- 7 Conclusion et limites

- 1 Principe(s) d'induction
- 2 Méthode de preuves de propriétés d'invariance
- 3 stop movex 3
- 4 Exemples de correction partielle (affectation simple)
- 5 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 6 Checking invariant and safety properties in PlusCal
- 7 Conclusion et limites

$$pre(v_0) \wedge v = v_0 \wedge v_f = f(v) \Rightarrow post(v_0, v_f) \quad (I)$$

requires  $pre(v_0)$   
ensures  $post(v_0, v_f)$   
variables  $V$

begin  
   $0 : P_0(v_0, v)$   
   $V := f(V)$   
   $f : P_f(v_0, v)$   
end

Liste des conditions à vérifier pour prouver  
(I)

- ▶  $v = v_0 \wedge pre(v_0) \Rightarrow P_0(v_0, v)$
- ▶  $pre(v_0) \wedge P_0(v_0, v) \wedge v' = f(v) \Rightarrow P_f(v_0, v')$
- ▶  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- ▶ (I) et (II) sont équivalents et (II) est la définition de l'invariance de  $A(x_0, x) \stackrel{def}{=} (x_0 = (0, v_0) \wedge x = (f, v) \Rightarrow post(v_0, v))$ .

$$\begin{aligned} x_0 = (0, v_0) \wedge pre(v_0) \wedge x_0 &\xrightarrow{[V := f(V)]} x \\ \Rightarrow \\ (x_0 = (0, v_0) \wedge x = (f, v) &\Rightarrow post(v_0, v)) \end{aligned} \quad (II)$$

$$v = v_0 \wedge pre(v_0) \wedge v_f = g(f(v)) \Rightarrow post(v_0, v_f) \text{ (I)}$$

requires  $pre(v_0)$   
ensures  $post(v_0, v_f)$   
variables  $V$

```
begin  
  0 :  $P_0(v_0, v)$   
   $V := f(V)$   
  1 :  $P_1(v_0, v)$   
   $V := g(V)$   
   $f : P_f(v_0, v)$   
end
```

Liste des conditions à vérifier pour prouver (I)

- ▶  $v = v_0 \wedge pre(v_0) \Rightarrow P_0(v_0, v)$
- ▶  $pre(v_0) \wedge P_0(v_0, v) \wedge v' = f(v) \Rightarrow P_1(v_0, v')$
- ▶  $pre(v_0) \wedge P_1(v_0, v) \wedge v' = g(v) \Rightarrow P_f(v_0, v')$
- ▶  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- ▶ (I) et (II) sont équivalents et (II) est la définition de l'invariance de  $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$ .

$$x_0 = (0, v_0) \wedge pre(v_0) \wedge x_0 \xrightarrow{[V := f(V); V := g(V)]} x \Rightarrow (x = (f, v) \Rightarrow post(v_0, v)) \text{ (II) } g$$

- 1 Principe(s) d'induction
- 2 Méthode de preuves de propriétés d'invariance
- 3 stop movex 3
- 4 Exemples de correction partielle (affectation simple)
- 5 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 6 Checking invariant and safety properties in PlusCal
- 7 Conclusion et limites

## Méthode de correction de propriétés de sûreté

Soit  $A(\ell_0, v_0, \ell, v)$  une propriété d'un programme  $P$ . Soit une famille d'annotations famille de propriétés  $\{P_\ell(v_0, v) : \ell \in \text{LOCATIONS}\}$  pour ce programme. Si les conditions suivantes sont vérifiées :

$\forall v_0, v, v' \in \text{MEMORY} :$

$$\left\{ \begin{array}{l} (1) \text{ pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v) \\ (2) \forall \ell \in \text{LOCATIONS}. P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v) \\ (3) \forall \ell, \ell' \in \text{LOCATIONS} : \\ \quad \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{array} \right. ,$$

alors  $A(\ell_0, v_0, \ell, v)$  est une propriété de sûreté pour le programme  $P$ .

- 1 Définir la précondition  $\text{pre}(P)(v_0, v)$
- 2 Annoter le programme avec des prédicats  $P_\ell(v_0, v)$  où  $\ell \in \text{LOCATIONS}$
- 3 Vérifier que  $\text{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v)$  où  $\ell \in \text{INPUTS}$  (ensemble des points d'entrée).
- 4 Vérifier que  $P_\ell(v_0, v) \Rightarrow A(\ell, v)$  où  $\ell \in \text{LOCATIONS}$
- 5 Pour chaque paire de points de contrôle  $(\ell, \ell')$  telle que  $\ell \longrightarrow \ell'$  (successifs), vérifier que  $(P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ .



- 1 Vérifier que  $\mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$  où  $\ell \in \text{INPUTS}$   
(ensemble des points d'entrée).
- 2 Vérifier que  $P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v)$  où  $\ell \in \text{LOCATIONS}$
- 3 Pour chaque paire de points de contrôle  $(\ell, \ell')$  telle que  $\ell \longrightarrow \ell'$   
(successifs), vérifier que  
 $(P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ .

- 1 Vérifier que  $\mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$  où  $\ell \in \text{INPUTS}$  (ensemble des points d'entrée).
- 2 Vérifier que  $P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v)$  où  $\ell \in \text{LOCATIONS}$
- 3 Pour chaque paire de points de contrôle  $(\ell, \ell')$  telle que  $\ell \longrightarrow \ell'$  (successifs), vérifier que  $(P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ .

### Exemples de propriétés de sûreté

- ▶ Correction partielle :  $A_1(\ell_0, v_0, \ell, v) \stackrel{\text{def}}{=} \ell = \ell_f \Rightarrow \mathbf{post}(P)(v_0, v)$
- ▶ Absence d'erreurs à l'exécution :  
 $A_2(\ell_0, v_0, \ell, v) \stackrel{\text{def}}{=} \bigwedge_{\ell', \ell \rightarrow \ell'} \mathbf{DOM}(\ell, \ell')(v)$

- ▶ Les vérifications sont longues et nombreuses
- ▶ Les vérifications sont parfois élémentaires et assez faciles à prouver
- ▶ Approche par vérification algorithmique via TLA et ses outils
- ▶ Approche par mécanisation du raisonnement symbolique via Event-B et ses outils

- 1 Principe(s) d'induction
- 2 Méthode de preuves de propriétés d'invariance
- 3 stop movex 3
- 4 Exemples de correction partielle (affectation simple)
- 5 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 6 Checking invariant and safety properties in PlusCal
- 7 Conclusion et limites

$$\begin{array}{l} l0 : v = 3 \\ v := v+2; \\ l1 : v = 5 \end{array}$$

$$\begin{array}{l} l0 : v = 3 \\ v := v+2; \\ l1 : v = 5 \end{array}$$

- ▶ Annotation du code
- ▶ Traduction de l'invariant à vérifier
- ▶ Expression de la propriété de correction partielle
- ▶ Vérification de la propriété

$$\begin{array}{l} l0 : v = 3 \\ v := v+2; \\ l1 : v = 5 \end{array}$$

- ▶ Annotation du code
- ▶ Traduction de l'invariant à vérifier
- ▶ Expression de la propriété de correction partielle
- ▶ Vérification de la propriété

```

-----MODULE an0-----
EXTENDS Integers, TLC
-----
CONSTANTS v0,pc0
VARIABLES v,pc

(* extra definitions *)
min == -2^{31}
max == 2^{31}-1
D == min..max

-----
(* precondition pre(x0,y0,z0,pc0) *)
pre(fv) == fv=3
ASSUME pre(v0)

-----
(* initial conditions *)
Init == pc = "l0" /\ v=3

-----
(* actions *)
skip == UNCHANGED <<pc,v>>
al011 == pc="l0" /\ TRUE /\ pc'="l1" /\ v'=v+2

-----
(* next relation *)
Next == skip \/ al011

-----
(* invariant properties *)
i ==
  /\ pc \in {"l0","l1"}
  /\ pc="l0" => v=3
  /\ pc="l1" => v=5

-----
(* safety properties *)
suretecorrectionpartielle == pc="l1" => v=5
sureteabsencederreurs == v \in D /\ v+2 \in D

-----
tocheck == i

```

- ▶ Le programme ou l'algorithme est annoté à des points de contrôle  $\ell \in \text{LOCATIONS}$  et à chaque point de contrôle  $\ell$  se trouve une assertion  $P_\ell(v_0, v)$ .



- ▶ Le programme ou l'algorithme est annoté à des points de contrôle  $\ell \in \text{LOCATIONS}$  et à chaque point de contrôle  $\ell$  se trouve une assertion  $P_\ell(v_0, v)$ .
- ▶ Si les deux points de contrôle  $\ell, \ell'$  définissent un calcul élémentaire, alors on définit une action  $\mathcal{E}(\ell, \ell')$  comme suit :

$$\begin{aligned}\mathcal{E}(\ell, \ell') &\triangleq \\ &\wedge c = \ell \\ &\wedge \text{cond}_{\ell, \ell'}(v) \\ &\wedge c' = \ell' \\ &\wedge v' = f_{\ell, \ell'}(v)\end{aligned}$$

- ▶ Le programme ou l'algorithme est annoté à des points de contrôle  $\ell \in \text{LOCATIONS}$  et à chaque point de contrôle  $\ell$  se trouve une assertion  $P_\ell(v_0, v)$ .
- ▶ Si les deux points de contrôle  $\ell, \ell'$  définissent un calcul élémentaire, alors on définit une action  $\mathcal{E}(\ell, \ell')$  comme suit :

$$\begin{aligned}\mathcal{E}(\ell, \ell') &\triangleq \\ &\wedge c = \ell \\ &\wedge \text{cond}_{\ell, \ell'}(v) \\ &\wedge c' = \ell' \\ &\wedge v' = f_{\ell, \ell'}(v)\end{aligned}$$

- $v$  est la variable de l'état mémoire ou la liste des variables de l'état mémoire ;  $v$  inclut les variables locales et les variables résultat.
- $c$  est une nouvelle variable qui modélise le flot de contrôle de type LOCATIONS.
- $\mathcal{E}(\ell, \ell')$  simule le calcul débutant en  $\ell$  et terminant en  $\ell'$  ;  $v$  est mise à jour.

$$\begin{aligned} i &\triangleq \\ &\quad \wedge c \in \text{LOCATIONS} \\ &\quad \wedge v \in \text{Type} \\ &\quad \dots \\ &\quad \wedge c = \ell \Rightarrow P_\ell(v_0, v) \\ &\quad \wedge c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\ &\quad \dots \\ \text{safty} &\triangleq S(c, v_0, v) \end{aligned}$$

$$\begin{aligned} i &\triangleq \\ &\wedge c \in \text{LOCATIONS} \\ &\wedge v \in \text{Type} \\ \dots \\ &\wedge c = \ell \Rightarrow P_\ell(v_0, v) \\ &\wedge c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\ \dots \\ \text{safety} &\triangleq S(c, v_0, v) \end{aligned}$$

- ▶ *Type* est le type des variables  $v$  et est un ensemble de valeurs possibles.
- ▶ L'annotation donne gratuitement les conditions satisfaites par  $v$  quand le contrôle est en  $\ell$ , (resp. en  $\ell'$ ).
- ▶  $S(c, v_0, v)$  est une propriété de sûreté à vérifier et est un théorème dans le cas de *Event-B*.



- La relation de transition  $Next$  est définie par :

$$Next \triangleq \dots \vee \mathcal{E}(\ell, \ell') \vee \dots$$

- ▶ La relation de transition  $Next$  est définie par :

$$Next \triangleq \dots \vee \mathcal{E}(\ell, \ell') \vee \dots$$

- ▶ Les conditions initiales des variables sont à définir par un prédicat  $Init$

- 1 Principe(s) d'induction
- 2 Méthode de preuves de propriétés d'invariance
- 3 stop movex 3
- 4 Exemples de correction partielle (affectation simple)
- 5 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 6 Checking invariant and safety properties in PlusCal
- 7 Conclusion et limites



```

----- MODULE exemple -----
EXTENDS Naturals, Integers, TLC
CONSTANTS x0,y0,z0,min,max,undef
-----

(* precondition *)
ASSUME x0 = y0 + 3*z0
-----

(*
--algorithm ex {
  variables x=x0,
           y = y0,
           z=z0;

{
10: assert x = y + 3*z /\ /\ y=y0 /\ z=z0 ;
   x := y+3*z;
11: assert x = y0+3*z0 /\ y=y0 /\ z=z0 ;
}
}

```

## Exemple (II)

---

```
----- MODULE exemple -----  
  
-----  
ISDEF(X,Y) == X # undef => X \in Y  
DD(X) == X # undef => X \in min..max  
-----  
  
i ==  
  /\ pc \in {"l0","l1","Done"}  
  /\ ISDEF(x,Int) /\ ISDEF(x,Int) /\ ISDEF(z,Int)  
  /\ pc = "l0" => x = y + 3*z  
  /\ pc = "l1" => x+y+z \geq y  
post ==      x = y0+3*z0 /\ y=y0 /\ z=z0  
  
safetyrte == DD(x) /\ DD(y) /\ DD(z)  
safetypc == pc="Done" => post  
=====
```

- 1 Principe(s) d'induction
- 2 Méthode de preuves de propriétés d'invariance
- 3 stop movex 3
- 4 Exemples de correction partielle (affectation simple)
- 5 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 6 Checking invariant and safety properties in PlusCal
- 7 Conclusion et limites



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

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

- ▶  $\mathcal{R}$  : 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.

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



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

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

- ▶  $\mathcal{R}$  : 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

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



Vérification de conditions de vérification

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

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



### Vérification de conditions de vérification

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



- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)

- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)
- ▶ Énoncer ou calculer les invariants d'un modèle :  $\text{REACHABLE}(M)$ .



- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)
- ▶ Énoncer ou calculer les invariants d'un modèle :  $\text{REACHABLE}(M)$ .
- ▶  $\text{TLA}^+$  versus Event-B
  - Plate-formes :  $\text{TLA}^+$  avec TLAPS et Toolbox, Event-B avec Rodin
  - Langage de la théorie des ensembles avec quelques différences
  - Fonctionnalités des outils
    - ▶ Éditeurs de modèles :  $\text{TLA}^+$  et Event-B
    - ▶ Model-Checking :  $\text{TLA}^+$  et Event-B
    - ▶ Assistant de preuve : Event-B
    - ▶ Animateur et Model-Checker ProB

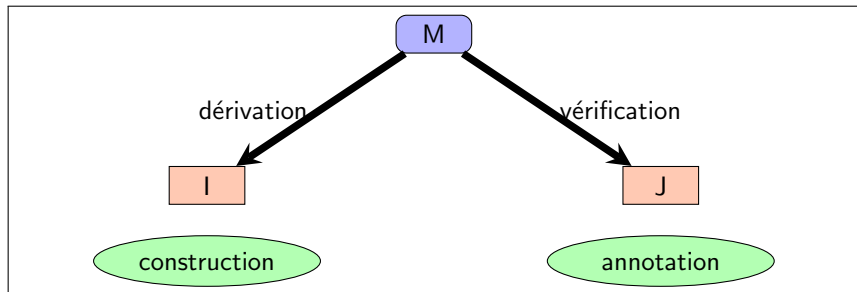
- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)
- ▶ Enoncer ou calculer les invariants d'un modèle :  $\text{REACHABLE}(M)$ .
- ▶  $\text{TLA}^+$  versus Event-B
  - Plate-formes :  $\text{TLA}^+$  avec TLAPS et Toolbox, Event-B avec Rodin
  - Langage de la théorie des ensembles avec quelques différences
  - Fonctionnalités des outils
    - ▶ Editeurs de modèles :  $\text{TLA}^+$  et Event-B
    - ▶ Model-Checking :  $\text{TLA}^+$  et Event-B
    - ▶ Assistant de preuve : Event-B
    - ▶ Animateur et Model-Checker ProB
- ▶ Développement d'outils symboliques comme les solveurs SMT ou des procédures de décision

- ▶ TLA<sup>+</sup> et TLA Toolbox : logique temporelle, théorie des ensembles, calcul des prédicats, model-checker
- ▶ Event-B et Rodin : théorie des ensembles, assistant de preuve, model-checker, animateur
- ▶ B et Event-B et ProB : théorie des ensembles, model-checker, animateur, validation
- ▶ Promela et SPIN : logique temporelle, model-checking
- ▶ C et Frama-C : analyse sémantique des programmes, assistants de preuve, solveurs SMT.
- ▶ Spec# et Rise4fun : pre/post, contrats
- ▶ PAT : cadre générique pour créer son propre model-checker (classique, temps réel, probabiliste, stochastique)
- ▶ C et cppcheck : analyse statique de programmes C ou C++



## Vérification à faire mais comment automatiquement ?

- ▶ Application de la correction du principe d'induction : si on vérifie les trois propriétés, alors  $A$  est une propriété de sûreté pour le modèle en question : outil de vérification.
- ▶ Si on veut montrer que  $A$  est une propriété de sûreté, alors on doit utiliser l'invariant pour induire des annotations pour le modèle : outil de dérivation.



- ▶  $\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x)$
- ▶  $\forall x \in \text{VALS}. (\exists x_0. x_0 \in \text{VALS} \wedge \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x)) \Rightarrow A(x).$
- ▶  $\text{REACHABLE}(M) = \{u | u \in \text{VALS} \wedge (\exists x_0. x_0 \in \text{VALS} \wedge \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, u))\}$  est l'ensemble des états accessibles à partir des états initiaux.
- ▶ Model Checking : on doit montrer l'inclusion  $\text{REACHABLE}(M) \subseteq \{u | u \in \text{VALS} \wedge A(u)\}.$
- ▶ Preuves : définir un invariant  $I(\ell, v) \equiv \bigvee_{\ell \in \text{LOCATIONS}} \left( \bigvee_{v \in \text{MEMORY}} P_\ell(v) \right)$  avec la famille d'annotations  $\{P_\ell(v) : \ell \in \text{LOCATIONS}\}$  et démontrer les conditions de vérification.
- ▶ Analyse automatique :
  - Mécaniser la vérification des conditions de vérification
  - Calculer  $\text{REACHABLE}(M)$
  - Calculer une valeur approchée de  $\text{REACHABLE}(M)$

$$(\mathcal{P}(\text{VALS}), \subseteq) \xleftrightarrow[\alpha]{\gamma} (D, \sqsubseteq)$$

$$\alpha(\text{REACHABLE}(M)) \sqsubseteq A \text{ ssi } \text{REACHABLE}(M) \subseteq \gamma(A)$$

Si  $\gamma(A) \subseteq \{u | u \in \text{VALS} \wedge A(u)\}$ , alors

$$\text{REACHABLE}(M) \subseteq \{u | u \in \text{VALS} \wedge A(u)\}$$

- ▶ Mécaniser la vérification des conditions de vérification
- ▶ Calculer  $\text{REACHABLE}(M)$  comme un point-fixe.
- ▶ Calculer une valeur approchée de  $\text{REACHABLE}(M)$

$$\begin{aligned} (\mathcal{P}(\text{VALS}), \subseteq) &\xleftrightarrow[\alpha]{\gamma} (D, \subseteq) \\ \alpha(\text{REACHABLE}(M)) &\subseteq A \text{ ssi } \text{REACHABLE}(M) \subseteq \gamma(A) \end{aligned}$$

Si  $A$  vérifie  $\gamma(A) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$ , alors  
 $\text{REACHABLE}(M) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$

## Method for verifying program properties

correctness and Run Time Errors

A program  $P$  satisfies a (pre,post) contract :

- ▶  $P$  transforms a variable  $v$  from initial values  $v_0$  and produces a final value  $v_f : v_0 \xrightarrow{P} v_f$
- ▶  $v_0$  satisfies pre :  $\text{pre}(v_0)$  and  $v_f$  satisfies post :  $\text{post}(v_0, v_f)$
- ▶  $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- ▶  $\mathbb{D}$  est le domaine RTE de  $V$

requires  $\text{pre}(v_0)$

ensures  $\text{post}(v_0, v_f)$

variables  $V$

begin

0 :  $P_0(v_0, v)$

instruction<sub>0</sub>

...

$i : P_i(v_0, v)$

...

instruction <sub>$f-1$</sub>

$f : P_f(v_0, v)$

end

▶  $\text{pre}(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$

▶  $\text{pre}(x_0) \wedge P_f(v_0, v) \Rightarrow \text{post}(v_0, v)$

▶ For any pair of labels  $\ell, \ell'$   
such that  $\ell \longrightarrow \ell'$ , one verifies that, pour  
any values  $v, v' \in \text{MEMORY}$

$$\left( \begin{array}{l} P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \\ \wedge v' = f_{\ell, \ell'}(v) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

▶ For any pair of labels  $m, n$   
such that  $m \longrightarrow n$ , one verifies that,  
 $\forall v, v' \in \text{MEMORY} :$

$$\text{pre}(v_0) \wedge P_m(v_0, v) \Rightarrow \text{DOM}(m, n)(v)$$



## Summary of concepts

