Cours MALG & MOVEX

Modélisation, spécification et vérification (III)

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2025-2026 (16 février 2026 12:51 A.M.)

Plan

Sommaire

Current Summary

Current Summary

Simple contrat (I)

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

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V
begin
$0 : P_0(v_0, v)$
$V := f(V)$
$f : P_f(v_0, v)$
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 $A(x_0, x) \stackrel{def}{=} (x_0 = (0, v_0) \wedge x = (f, v) \Rightarrow post(v_0, v))$.

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

Simple contrat (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 $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$.

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

Current Summary

Méthode de correction de propriétés de sûreté

Soit $\mathrm{A}(\ell_0, v_9, \ell, v)$ une propriété d'un programme P. Soit une famille d'annotations famille de propriétés $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ pour ce programme. Si les conditions suivantes sont vérifiées :
$\forall v0, v, v' \in \textsc{Memory}$:

$$\left\{\begin{array}{ll} (1) & \textbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v) \\ (2) & \forall \ell \in \textsc{Locations}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v) \\ (3) & \forall \ell, \ell' \in \textsc{Locations} : \\ \multicolumn{2}{l}{\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')} \end{array}\right. ,$$

alors $\mathrm{A}(\ell_0, v_9, \ell, v)$ est une propriété de sûreté pour le programme P.

1. Définir la précondition $\textbf{pre}(\mathrm{P})(v_0, v)$
2. Annoter le programme avec des prédicats $P_\ell(v_0, v)$ où $\ell \in \textsc{Locations}$
3. Vérifier que $\textbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v)$ où $\ell \in \textsc{Inputs}$ (ensemble des points d'entrée.
4. Vérifier que $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell, v)$ où $\ell \in \textsc{Locations}$
5. Pour chaque paire de points de contrôle (ℓ, ℓ') telle que $\ell \longrightarrow \ell'$ (successifs), vérifier que $\big(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')\big)$.

Conditions de vérification

1. Vérifier que **pre**$(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$ où $\ell \in \textsc{Inputs}$ (ensemble des points d'entrée.
2. Vérifier que $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$ où $\ell \in \textsc{Locations}$
3. Pour chaque paire de points de contrôle (ℓ, ℓ') telle que $\ell \longrightarrow \ell'$ (successifs), vérifier que $(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$.

Conditions de vérification

1. Vérifier que **pre**$(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$ où $\ell \in \textsc{Inputs}$ (ensemble des points d'entrée.
2. Vérifier que $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$ où $\ell \in \textsc{Locations}$
3. Pour chaque paire de points de contrôle (ℓ, ℓ') telle que $\ell \longrightarrow \ell'$ (successifs), vérifier que $(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$.

Exemples de propriétés de sûreté

- Correction partielle : $\mathrm{A}_1(\ell_0, v_0, \ell, v) \stackrel{def}{=} \ell = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$
- Absence d'erreurs à l'exécution : $\mathrm{A}_2(\ell_0, v_0, \ell, v) \stackrel{def}{=} \wedge_{\ell', \ell \rightarrow \ell'}$ **DOM**$(\ell, \ell')(v)$

Mécanisation et automatisation de la vérification

- 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

Current Subsection Summary

Traduction des annotations

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

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

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

Traduction des annotations

$$l0 : v = 3$$
$$v := v+2;$$
$$l1 : v = 5$$

- 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>>
al0l1 == pc="l0" /\ TRUE /\ pc'="l1" /\ v'=v+2
------------------------------
(* next relation *)
Next == skip \/ al0l1
------------------------------
(* 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
=====================================================
```

Méthode de vérification avec le toolset TLAPS et TLA/TLA$^+$

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

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

$$
\begin{array}{l}
\mathcal{E}(\ell, \ell') \triangleq \\
\qquad \wedge\ c = \ell \\
\qquad \wedge\ cond_{\ell,\ell'}(v) \\
\qquad \wedge\ c' = \ell' \\
\qquad \wedge\ v' = f_{\ell,\ell'}(v)
\end{array}
$$

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

$$\begin{array}{l}\mathcal{E}(\ell, \ell') \triangleq \\ \quad \wedge\ c = \ell \\ \quad \wedge\ cond_{\ell,\ell'}(v) \\ \quad \wedge\ c' = \ell' \\ \quad \wedge\ v' = f_{\ell,\ell'}(v)\end{array}$$

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

$$
\begin{array}{l}
i \triangleq \\
\quad \land\ c \in \textsc{Locations} \\
\quad \land\ v \in Type \\
\ldots \\
\quad \land\ c = \ell \Rightarrow P_{\ell}(v_0, v) \\
\quad \land\ c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\
\ldots \\
safety \triangleq\ S(c, v_0, v)
\end{array}
$$

$$
\begin{array}{l}
i \triangleq \\
\quad \land\ c \in \textsc{Locations} \\
\quad \land\ v \in Type \\
\ldots \\
\quad \land\ c = \ell \Rightarrow P_\ell(v_0, v) \\
\quad \land\ c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\
\ldots \\
safety \triangleq\ S(c, v_0, v)
\end{array}
$$

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

Méthode de vérification exhaustive ou algorithmique

Méthode de vérification exhaustive ou algorithmique

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

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

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

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

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

Current Summary

Exemple (I)

```
--------------------- 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;

{
l0: assert x = y + 3*z/\ /\ y=y0 /\ z=z0 ;
    x := y+3*z;
l1: 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
=================================================================
```

Current Summary

1. stop movex 4
2. Exemples de correction partielle (affectation simple)
3. Annotation et vérification outillée avec TLA/TLA$^+$
   - Vérification avec TLA et ses outils
4. Checking invariant and safety properties in PlusCal
5. Conclusion et limites

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.

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

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

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

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

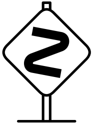
Sommaire des résultats

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

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

Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \left\{ \begin{array}{l} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{array} \right.$$

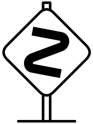


Vérification de conditions de vérification

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

Sommaire des résultats

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



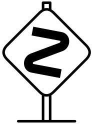
Vérification de conditions de vérification

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

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

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.

Observations

Observations

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

Observations

- Vérifier les énoncés de la forme $\Gamma \vdash P$ (séquents)
- Enoncer ou calculer les invariants d'un modèle : $\textsc{reachable}(M)$.

Observations

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

Observations

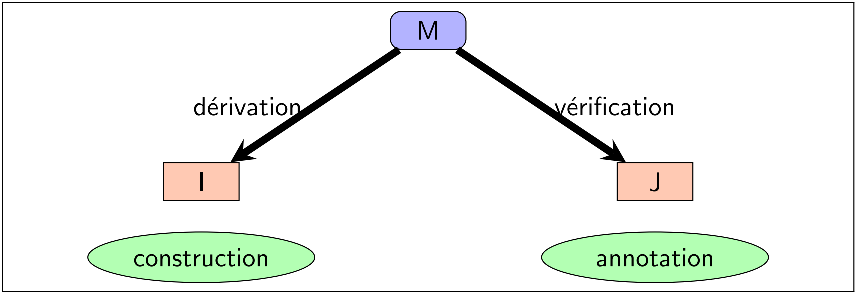
- Vérifier les énoncés de la forme $\Gamma \vdash P$ (séquents)
- Enoncer ou calculer les invariants d'un modèle : $\text{REACHABLE}(M)$.
- TLA$^+$ versus Event-B
  - Plate-formes : TLA$^+$ avec TLAPS et Toolbox, Event-B avec Rodin
  - Langage de la théorie des ensembles avec quelques différences
  -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

Revue des outils

- TLA$^+$ 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++

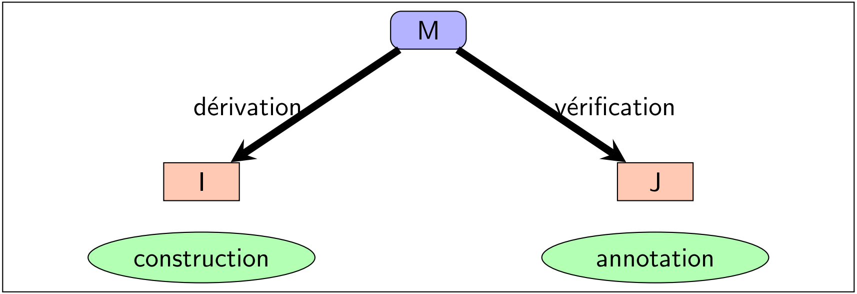
Vérification à faire mais comment automatiquement ?

- 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.



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.



Point d'étape

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

  $(\mathcal{P}(\textsc{Vals}), \subseteq) \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} (D, \sqsubseteq)$
  $\alpha(\textsc{reachable}(M)) \sqsubseteq A$ ssi $\textsc{reachable}(M) \sqsubseteq \gamma(A)$
  Si $\gamma(A) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$, alors
  $\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$

Analyse automatique

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

$$(\mathcal{P}(\textsc{Vals}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (D, \sqsubseteq)$$
$$\alpha(\textsc{reachable}(M)) \sqsubseteq A \text{ ssi } \textsc{reachable}(M) \sqsubseteq \gamma(A)$$

Si A vérifie $\gamma(A) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$, alors
$\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$

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 : $v_0 \xrightarrow{\mathsf{P}} v_f$
- v_0 satisfies pre : pre(v_0) and v_f satisfies post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow$ post(v_0, v_f)
- $\mathbb{D}$ est le domaine RTE de V

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

begin
$0 : P_0(v_0, v)$
`instruction`$_0$
$\ldots$
$i : P_i(v_0, v)$
$\ldots$
`instruction`$_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(x_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labelsℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \textsc{Memory}$ $\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \\ \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right)$,
- For any pair of labels m, n such that $m \longrightarrow n$, one verifies that, $\forall v, v' \in \textsc{Memory}$: $pre(v_0) \wedge P_m(v_0, v) \Rightarrow \mathbf{DOM}(m, n)(v)$

Summry of concepts