

Cours MVSI
Modélisation et Vérification
des Systèmes Informatiques

Modélisation, spécification et vérification (III)

Dominique Méry
Telecom Nancy, Université de Lorraine
(4 décembre 2025 at 4:21 P.M.)

- ## ⑥ Conclusion et limites

Soit une propriété $I(x)$ telle que :

$$\forall x, x' \in \text{VALS} : \begin{cases} (1) \text{ Init}(x) \Rightarrow I(x) \\ (2) I(x) \Rightarrow A(x) \\ (3) I(x) \wedge \text{NEXT}(x, x') \Rightarrow I(x') \end{cases}$$

Alors $A(x)$ est une propriété de sûreté pour le système S modélisé par M .

Soient x et $x' \in \text{VALS}$ tels que $\text{INIT}(x) \wedge \text{NEXT}(x, x')$.

- ▶ On peut construire une suite telle que :

$$(x = x_0) \xrightarrow{\text{NEXT}} x_1 \xrightarrow{\text{NEXT}} x_2 \xrightarrow{\text{NEXT}} \dots \xrightarrow{\text{NEXT}} (x_i = x').$$

- ▶ L'hypothèse (1) nous permet de déduire $I(x_0)$.
- ▶ L'hypothèse (3) nous permet de déduire $I(x_1), I(x_2), I(x_3), \dots, I(x_i)$. En utilisant l'hypothèse (2) pour x' , nous en déduisons que x' satisfait A .

$$\forall x_0, x \cdot x, y \in \text{VALS} \wedge \text{Init}(x_0) \wedge x_0 \xrightarrow[\text{Next}]{\star} x \Rightarrow A(x)$$

PROUVONS QUE : il existe une propriété $I(x)$ telle que :

$$\forall x, x' \in \text{VALS} : \begin{cases} (1) \text{Init}(x) \Rightarrow I(x) \\ (2) I(x) \Rightarrow A(x) \\ (3) I(x) \wedge \text{NEXT}(x, x') \Rightarrow I(x') \end{cases}$$

- ▶ Nous considérons la propriété suivante :

$$I(x) \hat{=} \exists x_0 \in \text{VALS} \cdot \text{Init}(x_0) \wedge x_0 \xrightarrow[\text{Next}]{\star} x.$$

- ▶ $I(x)$ exprime que la valeur x est accessible à partir d'une valeur initiale x_0 .
- ▶ Les trois propriétés sont simples à vérifier pour $I(x)$. $I(x)$ est appelé le plus fort invariant de l'algorithme \mathcal{A} .

- ▶ P. et R. Cousot développent une étude complète des propriétés d'invariance et de sûreté en mettant en évidence correspondances entre les différentes méthodes ou systèmes proposées par Turing, Floyd, Hoare, Wegbreit, Manna ... et reformulent les principes d'induction utilisés pour définir ces méthodes de preuve (voir les deux cubes des 16 principes).
- ▶ Deux types de principes sont proposés : assertionnel et relationnel.
- ▶ Nous utilisons l'expression de propriété de sûreté, alors que généralement il s'agit d'une propriété d'invariance (\square propriété) et d'invariant au lieu d'invariant inductif.

.....

⊠ Definition(assertion)

Soit $(Th(s, c), X, VALS, INIT(x), \{r_0, \dots, r_n\})$ un modèle relationnel M d'un système \mathcal{S} . Une propriété A est une propriété assertionnelle de sûreté pour le système \mathcal{S} , si

$$\forall x_0, x \in VALS. Init(x_0) \wedge NEXT^*(x_0, x) \Rightarrow A(x).$$

.....

.....

⊠ Definition(relation)

Soit $(Th(s, c), X, VALS, INIT(x), \{r_0, \dots, r_n\})$ un modèle relationnel M d'un système \mathcal{S} . Une propriété R est une propriété relationnelle de sûreté pour le système \mathcal{S} , si

$$\forall x_0, x \in VALS. Init(x_0) \wedge NEXT^*(x_0, x) \Rightarrow R(x_0, x).$$

.....

Complétude et correction

$$\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x).$$

si, et seulement si,

il existe $I \in \mathcal{P}(\text{VALS})$

$$\forall x_0, x, x' \in \text{VALS} : \begin{cases} (1) \text{Init}(x_0) \Rightarrow I(x_0) \\ (2) I(x) \Rightarrow A(x) \\ (3) I(x) \wedge \text{NEXT}(x, x') \Rightarrow I(x') \end{cases}$$

si, et seulement si,

$$\exists i \in \mathcal{P}(\text{VALS}). \begin{cases} (1) \text{Init} \subseteq i \\ (2) i \subseteq A \\ (3) \forall x, x' \in \text{VALS}. i(x) \wedge \text{NEXT}(x, x') \Rightarrow i(x') \end{cases}$$

Complétude et correction

$$\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x_0, x).$$

si, et seulement si,

il existe $R \in \mathcal{P}(\text{VALS} \times \text{VALS})$

$$\forall x_0, x, x' \in \text{VALS} : \begin{cases} (1) \text{Init}(x_0) \Rightarrow R(x_0, x_0) \\ (2) R(x_0, x) \Rightarrow A(x_0, x) \\ (3) R(x_0, x) \wedge \text{NEXT}(x, x') \Rightarrow R(x_0, x') \end{cases}$$

si, et seulement si,

$\exists R \in \mathcal{P}(\text{VALS} \times \text{VALS}).$

$$\left[\begin{array}{l} (1) \text{Init} \times \text{Init} \subseteq R \\ (2) R \subseteq A \\ (3) \forall x, x' \in \text{VALS}. R(x_0, x) \wedge \text{NEXT}(x, x') \Rightarrow R(x_0, x') \end{array} \right.$$

- ▶ La propriété invariante I est définie par
$$I(x) \stackrel{def}{=} \exists x_0 \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x)$$
- ▶ La propriété invariante R est définie par
$$R(x_0, x) \stackrel{def}{=} \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x)$$

Complétude et correction

$\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x).$

si, et seulement si,

il existe $I \in \mathcal{P}(\text{VALS})$

$$\forall x_0, x, x' \in \text{VALS} : \begin{cases} (1) \text{Init}(x_0) \Rightarrow I(x_0) \\ (2) I(x) \Rightarrow A(x) \\ (3) I(x) \wedge \text{NEXT}(x, x') \Rightarrow I(x') \end{cases}$$

- L'absence d'erreurs à l'exécution est caractérisée comme une propriété assertionnelle, puisqu'elle porte sur le fait qu'un état est sans erreurs à l'exécution si les calculs sont définis en cet état.
principe

Complétude et correction

$$\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x_0, x).$$

si, et seulement si,

il existe $R \in \mathcal{P}(\text{VALS} \times \text{VALS})$

$$\forall x_0, x, x' \in \text{VALS} : \begin{cases} (1) \text{Init}(x_0) \Rightarrow R(x_0, x_0) \\ (2) R(x_0, x) \Rightarrow A(x_0, x) \\ (3) R(x_0, x) \wedge \text{NEXT}(x, x') \Rightarrow R(x_0, x') \end{cases}$$

- La correction partielle est caractérisée comme une relation entre l'état initial et l'état courant.

Principe assertionnel de sûreté ou d'invariance

$$\exists I(x) \in \mathcal{P}(\text{VALS}). \left[\begin{array}{l} \forall x, x' \in \text{VALS}. \\ (1) \text{ Init}(x) \Rightarrow I(x) \\ (2) I(x) \Rightarrow A(x) \\ (3) I(x) \wedge \text{NEXT}(x, x') \Rightarrow I(x') \end{array} \right.$$

Principe relationnel de sûreté ou d'invariance

$$\exists IR(x_0, x) \in \mathcal{P}(\text{VALS} \times \text{VALS}). \left[\begin{array}{l} \forall x_0, x, x' \in \text{VALS}. \\ (1) \text{ Init}(x_0) \Rightarrow IR(x_0, x_0) \\ (2) IR(x_0, x) \Rightarrow R(x_0, x) \\ (3) IR(x_0, x) \wedge \text{NEXT}(x, x') \Rightarrow IR(x_0, x') \end{array} \right.$$

Vérification du contrat : ce qui est la technique

Un programme P *remplit* un contrat (pre,post) :

- ▶ P transforme une variable x à partir d'une valeur initiale x_0 et produisant une valeur finale x_f : $x_0 \xrightarrow{P} x_f$
- ▶ x_0 satisfait pre : $\text{pre}(x_0)$ and x_f satisfait post : $\text{post}(x_0, x_f)$
- ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

```

requires  $pre(x_0)$ 
< ensures  $post(x_0, x_f)$ 
variables  $X$ 
  begin
     $0 : P_0(x_0, x)$ 
    instruction0
    ...
     $i : P_i(x_0, x)$ 
    ...
    instruction $f-1$ 
     $f : P_f(x_0, x)$ 
  end

```

- ▶ $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- ▶ conditions de vérification pour toutes les paires $\ell \longrightarrow \ell'$

- ▶ On considère un langage de programmation classique noté `PROGRAMS`
- ▶ et nous supposons que ce langage de programmation dispose de l'affectation, de la conditionnelle, de l'itération bornée, de l'itération non-bornée, de variables simples ou structurées comme les tableaux et de la définition de constantes.
- ▶ On se donne un programme `P` de `PROGRAMS` ; ce programme comprend
 - des variables notées globalement v ,
 - des constantes notées globalement pc ,
 - des types associés aux variables notés globalement `VALS` et identifiés à un ensemble de valeurs possibles des variables,
 - des instructions suivant un ordre défini par la syntaxe du langage de programmation.

- ▶ on définit un ensemble de points de contrôle LOCATIONS
- ▶ pour chaque programme ou algorithme P . LOCATIONS est un ensemble fini de valeurs et une variable cachée notée pc parcourt cet ensemble selon l'enchaînement.
- ▶ l'espace des valeurs possibles VALS est un produit cartésien de la forme $\text{LOCATIONS} \times \text{MEMORY}$
- ▶ les variables x du système se décomposent en deux entités indépendantes $x = (pc, v)$ avec comme conditions $pc \in \text{LOCATIONS}$ et $v \in \text{MEMORY}$.

$$x = (pc, v) \wedge pc \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \quad (1)$$

On considère un programme P annoté; on se donne un modèle relationnel

$\mathcal{MP} = (Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \dots, r_n\})$ où

- ▶ $Th(s, c)$ est une théorie définissant les ensembles, les constantes et les propriétés statiques de ce programme
- ▶ x est une liste de variables flexibles et x comprend une partie contrôle et une partie mémoire.
- ▶ $\text{LOCATIONS} \times \text{MEMORY}$ est un ensemble de valeurs possibles pour x .
- ▶ $\{r_0, \dots, r_n\}$ est un ensemble fini de relations reliant les valeurs avant x et les valeurs après x' et conformes à la relation de succession \longrightarrow entre les points de contrôle.
- ▶ $\text{INIT}(x)$ définit l'ensemble des valeurs initiales de (pc_0, v) et $x = (pc_0, v)$ avec $pre(v)$ qui caractérise les valeurs initiales de v au point initial.

On suppose qu'il existe un graphe sur l'ensemble des valeurs de contrôle définissant la relation de flux et nous notons cette structure $(\text{LOCATIONS}, \longrightarrow)$.

.....

☒ Definition

$$\ell_1 \longrightarrow \ell_2 \stackrel{\text{def}}{=} pc = \ell_1 \wedge pc' = \ell_2$$

.....

☒ Definition(Annotation d'un point de contrôle)

Soit une structure $(\text{LOCATIONS}, \longrightarrow)$ et une étiquette $\ell \in \text{LOCATIONS}$. Une annotation d'un point de contrôle ℓ est un prédicat $P_\ell(v)$ (version assertionnelle) ou $P_\ell(v_0, v)$ (version relationnelle).



$P_\ell(v_0, v)$ exprime une relation entre la valeur initiale de V notée v_0 et v la valeur courante de V au point ℓ et donc $P_\ell(v_0, v) \Rightarrow pre(v_0)$ précise que v_0 est une valeur initiale.

- ▶ Les étiquettes ℓ appartiennent à **LOCATIONS** : $\ell \in \text{LOCATIONS}$.
- ▶ Les variables v appartiennent à **MEMORY** : $v \in \text{MEMORY}$.
- ▶ $pre(v_0)$ spécifie les valeurs initiales de v .
- ▶ Chaque fois que le contrôle est en ℓ , v satisfait $P_\ell(v)$:
 $pc = \ell \Rightarrow P_\ell(v_0, v)$.
- ▶ A tout état (ℓ, v) du programme, la propriété suivante est vraie mais doit être prouvée :

$$J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge pc \in \text{LOCATIONS} \\ \wedge v \in \text{MEMORY} \\ \dots \\ \wedge pc = \ell \Rightarrow P_\ell(v_0, v) \\ \dots \end{array} \right]$$

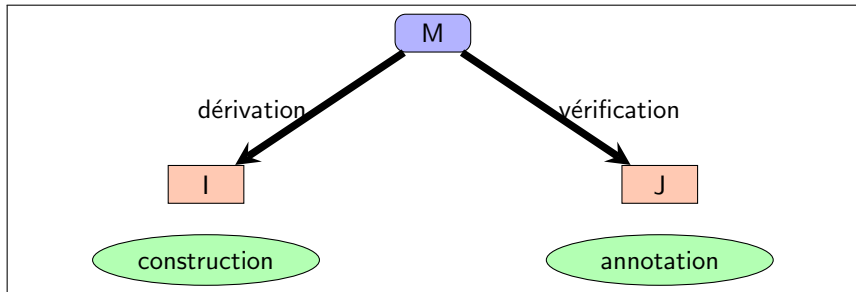
- ▶ $J(\ell_0, v_0, pc, v)$ est un invariant construit à partir des annotations produites mais il faut montrer que cet invariant permet de vérifier les trois conditions du principe d'induction.



ℓ_0 désigne l'étiquette marquant le début de l'algorithme et ℓ_f est la fin du programme. On pourra utiliser simplement 0 et f.

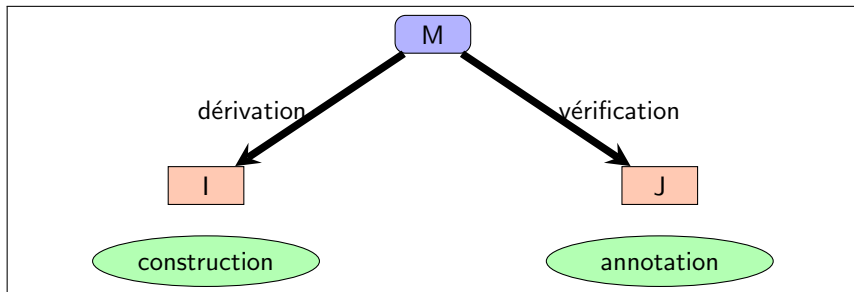
- ▶ Application de la correction du principe relationnel d'induction : si on vérifie les trois propriétés, alors A est une propriété de sûreté pour le modèle en question (vérification).
- ▶ Si on veut montrer que A est une propriété de sûreté, alors on doit utiliser l'invariant pour construire des annotations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

- Si on veut montrer que A est une propriété de sûreté, alors on doit utiliser l'invariant pour construire des annotations pour le modèle (dérivation).



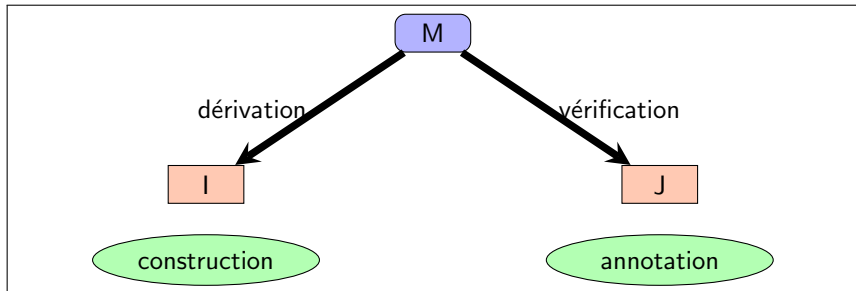
Utilisation du principe relationnel d'induction (RI)

- ▶ VALS = LOCATIONS \times MEMORY
- ▶ $J(pc_0, v_0, pc, v) \stackrel{def}{=} \exists x_0, x \in \text{VALS}. I(x_0, x) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (deduction)
- ▶ $I(x_0, x) \stackrel{def}{=} \exists pc_0, pc \in \text{LOCATIONS}, v_0, v \in \text{MEMORY}. J(pc_0, v_0, pc, v) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (induction)



Utilisation du principe assertionnel d'induction (AI)

- ▶ $\text{VALS} = \text{LOCATIONS} \times \text{MEMORY}$
- ▶ $J(pc, v) \stackrel{def}{=} \exists x \in \text{VALS}. I(x) \wedge x = (pc, v)$ (deduction)
- ▶ $I(x) \stackrel{def}{=} \exists pc \in \text{LOCATIONS}, v \in \text{MEMORY}. J(pc, v) \wedge x = (pc, v)$ (induction)



- (1) $\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x) \ (I(x))$
- (2) $\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow R(x_0, x) \ (IR(x))$

Relations et définitions

$x = (\ell, v)$, $x_0 = (\ell_0, v_0)$, $I(x)$, $IR(x_0, x)$ et les annotations $P_\ell(v)$, $RP_\ell(v_0, v)$ sont liées ainsi :

- ▶ $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶ $P_\ell(v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge I(x))$
- ▶ $IR(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge RP_\ell(v_0, v))$
- ▶ $RP_\ell(v_0, v) \stackrel{def}{=} \exists x, x_0. (x, x_0 \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge IR(x_0, x))$

La transformation est fondée la relation de transition définie pour chaque couple d'étiquettes de contrôle qui se suivent est exprimée très simplement par la forme relationnelle suivante :

$$x \ r_{\ell, \ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$$

- ▶ La transition de ℓ à ℓ' est possible, quand la condition $cond_{\ell,\ell'}(v)$ est vraie pour V et quand le contrôle est en ℓ ($pc = \ell$).
- ▶ Quand la transition est observée, les variables V sont transformées comme suit $v' = f_{\ell,\ell'}(v)$.
- ▶ La définition de la transition n'exprime aucune hypothèse liée à une stratégie d'exécution comme l'équité par exemple.
- ▶ $cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)$ est une expression où les expressions $cond_{\ell,\ell'}(v)$ et $v' = f_{\ell,\ell'}(v)$ posent des questions de définition :
 - $DOM(\ell, \ell')(v) \stackrel{def}{=} DEF(cond_{\ell,\ell'}(v))(v) \wedge DEF(f_{\ell,\ell'}(v))$
 - $DEF(E(X))(x)$, signifie que l'expression $E(X)$ est définie pour x la valeur courante de X .
- ▶ Certaines transitions peuvent conduire à des catastrophes :
 - $DEF(X+1)(x) \stackrel{def}{=} x+1 \in D$ où D est le domaine de codage de X par exemple $D = -2^{31} \dots 2^{31}-1$ pour un codage sur 32 bits.
 - $DEF(T(I+1) < V)(t, x, v) \stackrel{def}{=} i+1 \in dom(t) \wedge v \in D \wedge t(i+1) \in D$

$$\begin{aligned}\ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v)\end{aligned}$$

Traduction

- ▶ $(pc = \ell \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶ $cond_{\ell, \ell'}(v) \stackrel{def}{=} TRUE$

$$\begin{aligned}\ell_1 &: P_{\ell_1}(v_0, v) \\ \textbf{WHILE } B(V) \textbf{ DO} \\ &\ell_2 : P_{\ell_2}(v_0, v) \\ &\dots \\ &\ell_3 : P_{\ell_3}(v_0, v) \\ \textbf{END} \\ \ell_4 &: P_{\ell_4}(v_0, v)\end{aligned}$$

Traduction

$$\left\{ \begin{array}{l} pc = \ell_1 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_1 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \\ pc = \ell_3 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_3 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \end{array} \right.$$

Un programme P *remplit* un contrat $(pre, post)$:

- ▶ P transforme une variable v à partir d'une valeur initiale v_0 et produisant une valeur finale v_f : $v_0 \xrightarrow{P} v_f$
- ▶ v_0 satisfait pre : $pre(v_0)$ and v_f satisfait $post$: $post(v_0, v_f)$
- ▶ $pre(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow post(v_0, v_f)$

requires $pre(v_0)$

ensures $post(v_0, v_f)$

variables V

begin

$0 : P_0(v_0, v)$

instruction₀

...

$i : P_i(v_0, v)$

...

instruction _{$f-1$}

$f : P_f(v_0, v)$

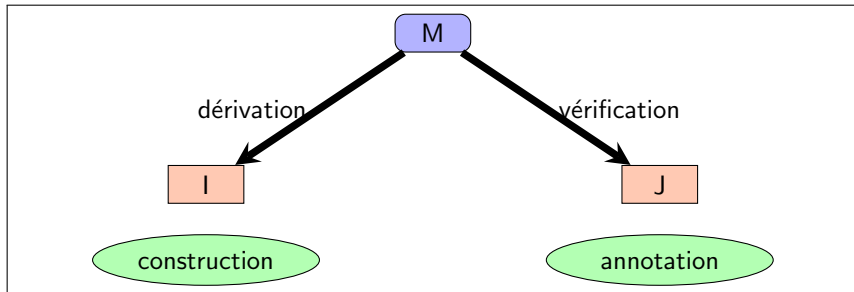
end

- ▶ $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- ▶ $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- ▶ conditions sur les transitions ℓ, ℓ' à définir à partir des principes d'induction.

```
variables  $U, V$   
requires  $u_0, v_0 \in \mathbb{N}$   
ensures  $u_f + v_f = u_0 + v_0$   
begin  
  0 :  $u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N}$   
   $U := U + 2$   
  1 :  $u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N}$   
   $V := V - 2$   
  2 :  $u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N}$   
end
```


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

- Si on veut montrer que A est une propriété de sûreté, alors on doit utiliser l'invariant pour construire des annotations pour le modèle (dérivation).



► $x = (pc, u, v)$

► $J(0, u_0, v_0, pc, u, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge pc \in \{0, 1, 2\} \\ \wedge u, v \in \mathbb{Z} \\ \wedge pc = 0 \Rightarrow u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\ \wedge pc = 1 \Rightarrow u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\ \wedge pc = 2 \Rightarrow u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N} \end{array} \right.$

► $A(0, u_0, v_0, pc, u, v) \stackrel{def}{=} (pc = 2 \Rightarrow u + v = u_0 + v_0 - 2 \wedge u_0, v_0 \in \mathbb{N})$

$\forall pc, u, v, pc', u', v' \in \{0, 1, 2\} \times \mathbb{Z} :$

$$\left\{ \begin{array}{l} (1) \text{ INIT}(0, u_0, v_0) \Rightarrow J(0, u_0, v_0, 0, u_0, v_0) \\ (2) J(0, u_0, v_0, pc, u, v) \Rightarrow A(0, u_0, v_0, pc, u, v) \\ (3) \forall i \in \{0, \dots, n\} : J(0, u_0, v_0, pc, u, v) \wedge x \text{ } r_i \text{ } pc', u', v' \Rightarrow J(0, u_0, v_0, pc', u', v') \end{array} \right.$$

- $$\begin{array}{l} r01(pc, u, v, pc', u', v') \stackrel{def}{=} pc = 0 \wedge u' = u+2 \wedge pc' = 1 \wedge v' = v \\ r12(pc, u, v, pc', u', v') \stackrel{def}{=} pc = 1 \wedge v' = v-2 \wedge pc' = 2 \wedge u' = u \end{array}$$

lification 1 $[A \wedge (A \Rightarrow B)] \longrightarrow [A \wedge B]$

lification 2 $[A \wedge (B = C) \wedge D \Rightarrow E \wedge (B = F) \wedge G] \longrightarrow [A \wedge (B = C) \wedge D \Rightarrow E \wedge (C = F) \wedge G]$

lification 3 $[A \wedge (B = C) \wedge D \Rightarrow E \wedge (F = F) \wedge G] \longrightarrow [A \wedge (B = C) \wedge D \Rightarrow E \wedge TRUE \wedge G]$

lification 4 $[A \Rightarrow B \wedge TRUE \wedge C] \longrightarrow [A \Rightarrow B \wedge C]$



lification 5 $[A \wedge (B = C \Rightarrow U) \wedge (B = D \wedge B = C \Rightarrow V)] \wedge C \neq D \wedge E \longrightarrow [A \wedge B = C \wedge U \wedge C \neq D \wedge E]$

Le modèle relationnel $M(P)$ pour le programme P annoté est donc défini comme suit :

$$M(P) \stackrel{def}{=} (Th(s, c), (pc, v), \text{LOCATIONS} \times \text{MEMORY}, \text{Init}(\ell, v), \{r_{\ell, \ell'} \mid \ell, \ell' \in \text{LOCATIONS} \wedge \ell \longrightarrow \ell'\}).$$

La définition de $\text{Init}(x)$ est dépendante de la précondition de P :

$$\text{Init}(x) \stackrel{def}{=} .x = (\ell_0, v) \wedge \mathbf{pre}(P)(v).$$

Conditions initiales

Les deux propriétés suivantes sont équivalentes :

- ▶ $\forall x_0 \in \text{VALS} : \text{Init}(x_0) \Rightarrow J(x_0, x_0)$
- ▶ $\forall v \in \text{MEMORY}. \mathbf{pre}(P)(v) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$

- ▶ Les relations r_i correspondent aux transitions satisfaisant $\ell \longrightarrow \ell'$ et on associe à chaque r_i la relation $r_{\ell,\ell'}$
 - ▶ $x \ r_{\ell,\ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell')$
- ▶ $J(x_0, x) \stackrel{def}{=} \exists v_0, \ell, v. (\ell \in \text{LOCATIONS} \wedge v_0, v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v_0, v))$
- ▶ $P_\ell(v_0, v) \stackrel{def}{=} \exists x_0, x. (x_0, x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

Pas d'induction

Les deux propriétés suivantes sont équivalentes :

- ▶ $\forall i \in \{0, \dots, n\} : J(x_0, x) \wedge x \ r_i \ x' \Rightarrow J(x_0, x')$
- ▶ $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

► $J(x_0 n x) \wedge x \ r_{\ell, \ell'} \ x' \Rightarrow J(x_0, x')$

► $J(x_0 n x) \wedge x \ r_{\ell, \ell'} \ x' \Rightarrow J(x_0, x')$

► $x \ r_{\ell, \ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$

- ▶ $J(x_0, x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- ▶ $x \text{ r}_{\ell, \ell'} x' \stackrel{\text{def}}{=} (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶ $I(x) \stackrel{\text{def}}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$

- ▶ $J(x_0, x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- ▶ $x \text{ r}_{\ell, \ell'} x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶ $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶ $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

- ▶ $J(x_0nx) \wedge x \ r_{\ell,\ell'} \ x' \Rightarrow J(x_0, x')$
- ▶ $x \ r_{\ell,\ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell')$
- ▶ $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶ $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- ▶ $J(x_0nx) \equiv pc = \ell \wedge P_\ell(v_0, v)$

- ▶ $J(x_0 n x) \wedge x \ r_{\ell, \ell'} \ x' \Rightarrow J(x_0, x')$
- ▶ $x \ r_{\ell, \ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶ $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_{\ell}(v))$
- ▶ $P_{\ell}(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- ▶ $J(x_0 n x) \equiv pc = \ell \wedge P_{\ell}(v_0, v)$
- ▶ $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell}(v_0, v')$

- ▶ $J(x_0nx) \wedge x \ r_{\ell,\ell'} \ x' \Rightarrow J(x_0, x')$
- ▶ $x \ r_{\ell,\ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell')$
- ▶ $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶ $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- ▶ $J(x_0nx) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- ▶ $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$
- ▶ $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell') \Rightarrow pc = \ell' \wedge P_{\ell'}(v_0, v')$

- $J(x_0 n x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- $x \text{ r}_{\ell, \ell'} x' \stackrel{\text{def}}{=} (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- $I(x) \stackrel{\text{def}}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{\text{def}}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $J(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_{\ell'}(v_0, v'))$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \text{ (Tautologie)})$
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow P_{\ell'}(v_0, v'))$

- ▶ Le programme est annoté.
- ▶ Les annotations définissent un invariant à vérifier selon les conditions de vérification.
- ▶ $A(\ell, v)$ est l'énoncé de la propriété de sûreté à vérifier.

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

☒ Definition Condition de vérification

L'expression $P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ où ℓ, ℓ' sont deux étiquettes liées par la relation \longrightarrow , est appelée une condition de vérification.

Floyd and Hoare

- ▶ $\forall v_0, v, v' \in \text{MEMORY}. \forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow$
 $P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ est équivalent à
 $\forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow \forall v' \in$
 $\text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell, \ell'}(v))$
- ▶ $\forall v_0, v, v' \in \text{MEMORY}. \forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow$
 $P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ est équivalent à
 $\forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow \forall v' \in$
 $\text{MEMORY}. (\exists v \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v)) \Rightarrow$
 $P_{\ell'}(v_0, v')$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

$$\blacktriangleright \forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$$

$$\begin{array}{l} \ell : P_\ell(v_0, v) \\ V := f_{\ell, \ell'}(V) \\ \ell' : P_{\ell'}(v_0, v) \end{array}$$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{TRUE} \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$$\begin{aligned} \ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v) \end{aligned}$$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{TRUE} \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$$\begin{aligned} \ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v) \end{aligned}$$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

$$\begin{aligned}\ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v)\end{aligned}$$

- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{TRUE} \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶ $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶ $\forall v \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell, \ell'}(v))$
(l'axiomatique de Hoare).


```
 $\ell_1 : P_{\ell_1}(v_0, v)$   
WHILE  $B(v)$  DO  
   $\ell_2 : P_{\ell_2}(v_0, v)$   
  ...  
   $\ell_3 : P_{\ell_3}(v_0, v)$   
END  
 $\ell_4 : P_{\ell_4}(v_0, v)$ 
```

Pour la structure d'itération, les conditions de vérification sont les suivantes :

- ▶ $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- ▶ $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$
- ▶ $P_{\ell_3}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- ▶ $P_{\ell_3}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$

```
 $\ell_1 : P_{\ell_1}(v_0, v)$   
IF  $B(v)$  THEN  
   $\ell_2 : P_{\ell_2}(v_0, v)$   
  ...  
   $\ell_3 : P_{\ell_3}(v_0, v)$   
ELSE  
   $m_2 : P_{\ell_2}(v_0, v)$   
  ...  
   $m_3 : P_{\ell_3}(v_0, v)$   
FI  
 $\ell_4 : P_{\ell_4}(v_0, v)$ 
```

Pour la structure de conditionnelle, les conditions suivantes :

- ▶ $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- ▶ $P_{\ell_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$
- ▶ $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{m_2}(v_0, v)$
- ▶ $P_{m_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$

La correction partielle vise à établir qu'un programme P est partiellement correct par rapport à sa précondition et à sa postcondition.

- ▶ la spécification des données de P **pre**(P)(v_0)
- ▶ la spécification des résultats de P **post**(P)(v_0, v)
- ▶ une famille d'annotations de propriétés $\{P_\ell(v_0, v) : \ell \in \text{LOCATIONS}\}$ pour ce programme.
- ▶ une propriété de sûreté définissant la correction partielle $pc = \ell_f \Rightarrow \mathbf{post}(P)(v_0, v_f)$ où ℓ_f est l'étiquette marquant la fin du programme P

.....

☒ Definition

Le programme P est partiellement correct par rapport à **pre**(P)(v_0) et **post**(P)(v_0, v), si la propriété $pc = \ell_f \Rightarrow \mathbf{post}(P)(v_0, v)$ est une propriété de sûreté pour ce programme.

.....

Si les conditions suivantes sont vérifiées :

- ▶ $\forall v_0, v \in \text{MEMORY} : \mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- ▶ $\forall v_0, v \in \text{MEMORY} : P_{\ell_f}(v_0, v) \Rightarrow \mathbf{post}(P)(v_0, v)$
- ▶ $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in \text{MEMORY}. (P_{\ell}(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$,

alors le programme P est partiellement correct par rapport à $\mathbf{pre}(P)(v_0)$ et $\mathbf{post}(P)(v_0, v)$.

- ▶ La correction partielle indique que si le programme termine normalement, alors la postcondition est vérifiée par les variables courantes.
- ▶ La sémantique du contrat est donc assez simple à donner :

An Early Program Proof by Alan Turing

Turing, A. M. 1949. "Checking a Large Routine." In Report of a Conference on High Speed Automatic Calculating Machines, Univ. Math. Lab., Cambridge, pp. 67-69.

- ▶ Turing se pose une question fondamentale de la correction des routines ou programmes en 1949.
- ▶ Il s'agit sans doute (Jones!) de la méthode d'annotation et d'induction sur les programmes qui sera finalisée par Floyd en 1967.

Méthode de Floyd

- ▶ Au point 0, $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ Annotations : au point i , l'assertion $P_i(x_0, x)$ est vraie.
- ▶ Au point final f , $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$

- ▶ La transition à exécuter est celle allant de ℓ à ℓ' et caractérisée par la condition ou garde $cond_{\ell,\ell'}(v)$ sur v et une transformation de la variable v , $v' = f_{\ell,\ell'}(v)$.
- ▶ Une condition d'absence d'erreur est définie par $\mathbf{DOM}(\ell, \ell')(v)$ pour la transition considérée. $\mathbf{DOM}(\ell, \ell')(v)$ signifie que la transition $\ell \longrightarrow \ell'$ est possible et ne conduit pas à une erreur.
- ▶ Une erreur est un débordement arithmétique, une référence à un élément de tableau qui n'existe pas, une référence à un pointeur nul, ...

exemple

- 1 La transition correspond à une affectation de la forme $x := x+y$ ou $y := x+y$:
$$\mathbf{DOM}(x+y)(x, y) \stackrel{def}{=} \mathbf{DOM}(x)(x, y) \wedge \mathbf{DOM}(y)(x, y) \wedge x+y \in int$$
- 2 La transition correspond à une affectation de la forme $x := x+1$ ou $y := x+1$:
$$\mathbf{DOM}(x+1)(x, y) \stackrel{def}{=} \mathbf{DOM}(x)(x, y) \wedge x+2 \in int$$

Définition RTE

L'absence d'erreurs à l'exécution vise à établir qu'un programme P ne va pas produire des erreurs durant son exécution par rapport à sa précondition et à sa postcondition.

- ▶ la spécification des données de P **pre**(P)(v)
- ▶ la spécification des résultats de P **post**(P)(v_0, v)
- ▶ une famille d'annotations de propriétés $\{P_\ell(v) : \ell \in \text{LOCATIONS}\}$ pour ce programme.
- ▶ une propriété de sûreté définissant l'absence d'erreurs à l'exécution :

$$\bigwedge_{\ell \in \text{LOCATIONS} - \{\text{output}\}, n \in \text{LOCATIONS}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$$

.....

☒ Definition

Le programme P ne produira pas d'erreurs à l'exécution par rapport à **pre**(P)(v) et **post**(P)(v_0, v), si la propriété

$$\bigwedge_{\ell \in \text{LOCATIONS} - \{\text{output}\}, n \in \text{LOCATIONS}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$$
 est une propriété de sûreté pour ce programme.

RTE = Run Time Error

Si les conditions suivantes sont vérifiées :

- ▶ $\forall v_0, v \in \text{MEMORY} : \mathbf{pre(P)}(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- ▶ $\forall m \in \text{LOCATIONS} - \{\ell_f\}, n \in \text{LOCATIONS}, \forall v_0, v, v' \in \text{MEMORY} : m \longrightarrow n : \mathbf{pre(P)}(v_0) \wedge P_m(v_0, v) \Rightarrow \mathbf{DOM}(m, n)(v)$
- ▶ $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in \text{MEMORY}. (P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$,

alors le programme P ne produira pas d'erreurs à l'exécution par rapport à $\mathbf{pre(P)}(v_0)$ et $\mathbf{post(P)}(v_0, v)$.

- ▶ On doit d'abord vérifier la correction partielle puis renforcer les assertions de la correction partielle par des conditions de domaine.
- ▶ On peut donc en déduire un contrat qui intègre aussi la vérification de l'absence d'erreurs à l'exécution.

Un programme P *remplit* un contrat (pre,post) :

- ▶ P transforme une variable v à partir d'une valeur initiale v_0 et produisant une valeur finale v_f : $v_0 \xrightarrow{P} v_f$
- ▶ v_0 satisfait pre : $\text{pre}(v_0)$ and v_f satisfait 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 :  $P_0(v_0, v)$ 
  instruction0
  ...
  i :  $P_i(v_0, v)$ 
  ...
  instructionf-1
  f :  $P_f(v_0, v)$ 
end
```

- ▶ $\text{pre}(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- ▶ $\text{pre}(x_0) \wedge P_f(v_0, v) \Rightarrow \text{post}(v_0, v)$
- ▶ Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $v, v' \in \text{MEMORY}$

$$\left(\begin{array}{c} P_\ell(v_0, v) \\ \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
- ▶ $\forall m \in \text{LOCATIONS} - \{\ell_f\}, n \in \text{LOCATIONS}, \forall v_0, v, v' \in \text{MEMORY} :$
 $m \longrightarrow n :$
 $\text{pre}(v_0) \wedge P_m(v_0, v) \Rightarrow \text{DOM}(m, n)(v)$

- 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 (ℓ, ℓ') 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 (ℓ, ℓ') 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 : $A_2(\ell_0, v_0, \ell, v) \stackrel{\text{def}}{=} \wedge_{\ell', \ell \rightarrow \ell'} \mathbf{DOM}(\ell, \ell')(v)$

- ▶ 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

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

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

- ▶ 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 ℓ 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 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 :

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

- ▶ Le programme ou l'algorithme est annoté à des points de contrôle $\ell \in \text{LOCATIONS}$ et à chaque point de contrôle 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 :

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

- 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 ℓ et terminant en ℓ' ; 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 ℓ , (resp. en ℓ').
- ▶ $S(c, v_0, v)$ est une propriété de sûreté à vérifier et est un théorème dans le cas de *Event-B*.

Méthode de vérification exhaustive ou algorithmique

- ▶ 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 $Init$

- ▶ Définition d'un langage algorithmique simple.
- ▶ Commentaire spécifique dans le texte du module TLA⁺ entre (* et *)
- ▶ Définition d'un algorithme comme une suite de définitions de processus séquentiels et pouvant partager des variables :

```
--algorithm nom { definitions }
```
- ▶ Génération d'une spécification TLA⁺ avec introduction d'une nouvelle variable pc modélisant le contrôle.
- ▶ Introduction d'une variable pc pour modéliser le contrôle de l'algorithme.
- ▶ L'outil ToolBox dispose d'une fonctionnalité de traduction.

Exemple (I)

```

----- MODULE exemple1 -----
EXTENDS Naturals, Integers, TLC
CONSTANTS x0,y0,z0,min,max,undef

-----

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

-----

(*
--algorithm exemple1 {
  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 exemple1 -----  
.....  
  
-----  
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  
=====
```

```
—— MODULE module_name ——  
\* TLA+ code  
  
(* —algorithm algorithm_name  
variables global_variables  
  
process p_name = ident  
variables local_variables  
begin  
  \* pluscal code  
end process  
  
process p_group \in set  
variables local_variables  
begin  
  \* pluscal code  
end process  
  
end algorithm; *)
```


Exemple 2

```
process pro = "test"  
begin  
  print<<" test">>;  
end process
```

- ▶ Un algorithme multiprocessus contient un ou plusieurs processus.
- ▶ Un processus commence de deux manières :
 - en définissant un ensemble de processus : `process (ProcName ∈ IdSet)`
 - en définissant un processus avec un identifiant `process (ProcName = Id)`
- ▶ `self` désigne le processus actuel

Un processus S envoie un message à un processus R.

```

—algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
  /* defining macros
  process (Sender = "S")
  {

  }; /* end Sender process block
  process (Receiver = "R")
  {

  }; /* end Receiver process block

} /* end algorithm

```

```

—algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
  macro Send(m, chan) {
    chan := Append(chan, m);
  }
  macro Recv(v, chan) {
    await chan # <<>>;
    v := Head(chan);
    chan := Tail(chan);
  }
}

```

- * Processes S and R

```
} \* end algorithm
```



```
—algorithm ex_process {  
  variables  
    input = <<>>, output = <<>>,  
    msgChan = <<>>, ackChan = <<>>,  
    newChan = <<>>;  
  /* defining macros  
    process (Sender = "S")  
      variables msg;  
      {  
        sending: Send("Hello", msgChan);  
        printing: print <<"Sender", input>>;  
      }; /* end Sender process block  
    process (Receiver = "R")  
      {  
        waiting: Recv(msg, msgChan);  
        adding: output := Append(output, msg);  
        printing: print <<"Receiver", output>>;  
      }; /* end Receiver process block  
  } /* end algorithm
```

```
macro Name(var1, ...)  
begin  
  \* something to write  
end macro;
```

```
procedure Name(arg1, ...)  
variables var1 = ... \* not \in, only =  
begin  
  Label:  
  \* something  
  return;  
end procedure;
```


- ▶ \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/>
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.
- ▶ Vérification des conditions de vérification avec un outil de preuve formelle.

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

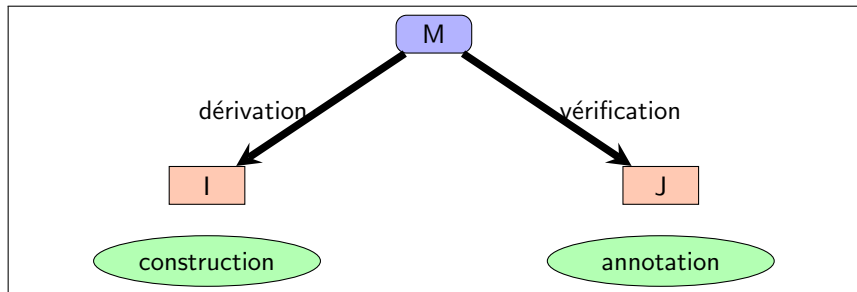
- ▶ Vérifier les énoncés de la forme $\Gamma \vdash P$ (séquents)
- ▶ Énoncer ou calculer les invariants d'un modèle : $\text{REACHABLE}(M)$.

- ▶ Vérifier les énoncés de la forme $\Gamma \vdash P$ (séquents)
- ▶ Énoncer ou calculer les invariants d'un modèle : $\text{REACHABLE}(M)$.
- ▶ 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
 - ▶ Éditeurs de modèles : TLA^+ et Event-B
 - ▶ Model-Checking : TLA^+ et Event-B
 - ▶ Assistant de preuve : Event-B
 - ▶ Animateur et Model-Checker ProB

- ▶ 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.



- ▶ 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 : $P_0(v_0, v)$

instruction₀

...

$i : P_i(v_0, v)$

...

instruction _{$f-1$}

$f : P_f(v_0, v)$

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 ℓ, ℓ'
such that $\ell \longrightarrow \ell'$, one verifies that, pour
any values $v, v' \in \text{MEMORY}$

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

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

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

Ontology Language Knowledge Language Documentation Language

Theory Domain

Assertion Language Programming Language

integrates integrates

Annotated Programming Language

TLA⁺/PlusCal Event-B C Boogie L Timed-Automata

ToolBox Rodin Frama-C BoogieT Uppaal