Cours Modélisation et Vérification des Systèmes Informatiques (MVSI)

Modélisation, spécification et vérification CM3

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2024-2025

Plan

Sommaire

Current Summary

1. Summation of the n first integers
2. Principe(s) d'induction
3. Méthode de preuves de propriétés d'invariance

Annotation versus commentaire de programmes ou d'algorithmes

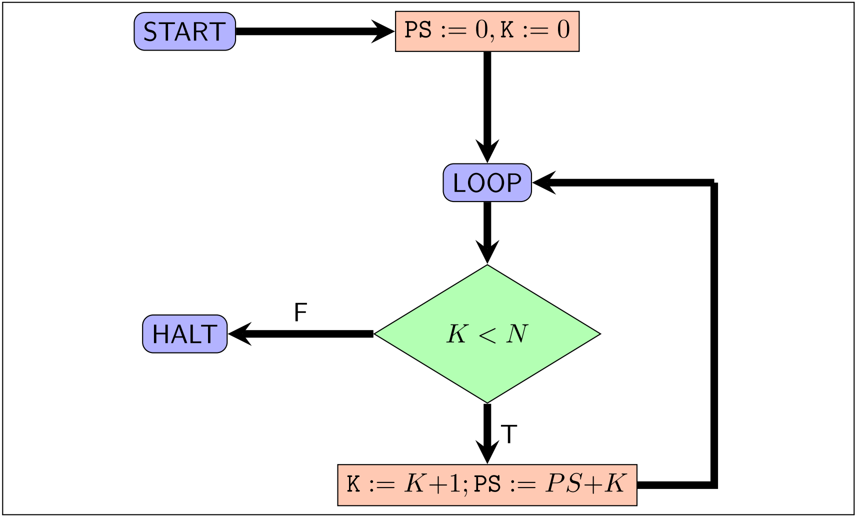
- Un programme ou un algorithme peuvent être annotés ou commentés.
- Un commentaire est une information pertinente destinée à être vue ou lue et qui a une importance relative dans l'esprit du concepteur.
- Un commentaire indique une information sur les données, sur les variables et donc sur l'état supposé du programme à l'exécution.
- Un commentaire est une annotation du texte du code qui nous permet de communiquer une information sémantique :
  - *à ce point, la variable k est plus petite sur n*
  - *l'indice e fait référence à une adresse licite de t et cette valeur est toujours positive*
  - *la somme des variables est positive*
- Les annotations peuvent être systématisées et obéir à une syntaxe spécifique définissant le langage d'annotations ;

```
/*@  assert l1:  z  >= 3 && y  == 3;  */
z = z +y;
/*@  assert l2:  z  >= 6 && y  == 3;  */
```

Calculer la somme des n premiers entiers (v0)

```
int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    k = k + 1;
    ps = ps + k;
  };
  return ps;
}
```

Calculer la somme des n premiers entiers (flowchart)



Calculer la somme des n premiers entiers (v0)

```
// pre n>=0;
// post ps == n*(n+1) / 2;

int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    k = k + 1;
    ps = ps + k;
  };
  return ps;
}


int main()
{

}
```

Calculer la somme des n premiers entiers (v0)

```
// pre n>=0;
// post ps == n*(n+1) / 2;

int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    // ps = k*(k+1) / 2;
    k = k + 1;
    ps = ps + k;
  };
    // ps = n*(n+1) / 2;
  return ps;
}


int main()
{

}
```

Calculer la somme des n premiers entiers (v0)

```
#include <stdio.h>

int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    k = k + 1;
    ps = ps + k;

  };
  return ps;
}

int main()
{
  int   z = 3;
  printf("Value for z=%d is %d\n",z,fS(z));
  return 0;
}
```

Definition of S(n) and IS(n)

$\forall n \in \mathbb{N} : S(n) = \sum_{k=0}^{n} k$

$$\left[\begin{array}{l} IS(0) = 0 \\ n \geq 0, IS(n+1) = IS(n)+(n+1) \end{array}\right.$$

Property for S(n) and IS(n)

$\forall n \in \mathbb{N} : S(n) = IS(n)$

- base 0 : $S(0) = 0 = IS(0)$
- induction $i+1$: $\forall j \in 0..i : S(j) = IS(j)$
  - $S(i+1) = \sum_{k=0}^{i+1} k$ (definition)
  - $S(i+1) = (\sum_{k=0}^{i} k)+i+1$ (property of summation)
  - $S(i+1) = S(i)+(i+1)$ (by definition of S(i))
  - $S(i+1) = IS(i)+(i+1)$ (by inductive assumption on S(i) et IS(j))
  - $S(i+1) = IS(i)+(i+1) = IS(i+1)$ (by defintion of IS(i+1))
- conclusion : $\forall i \in \mathbb{N} : S(i) = IS(i)$ (by induction principle)

Reformulation algorithmique du calcul de la somme des n premiers entiers

- $\forall n \in \mathbb{N} : S(n) = \sum_{k=0}^{n} k$
- $\left[\begin{array}{l} IS(0) = 0 \\ n \geq 0, IS(n+1) = IS(n)+(n+1) \end{array} \right.$
- $\forall n \in \mathbb{N} : S(n) = IS(n)$
-
  - base 0 : $S(0) = 0$
  - induction $k+1$: $S(k+1) = S(k)+k+1$
  - step $k+1$: $S(k+1) = S(k)+k+1$
- $S(k) = ps$: current value of ps is S(k)
- $S(k-1) = ops$: current value of ops is $S(k-1)$
- step $k+1$: $ps = D(k+1) = S(k)+k+1 = ops+k+1$

Calculer la somme des n premiers entiers (v1)

```
#include <stdio.h>

int fS(int n) {
  int ps = 0;
  int k = 0;
  int ok=k, ops = 0;
  while (k < n) {
    ok=k;ops=ps;
    k = ok + 1;
    ps = ops + k;
  };
  return ps;
}

int main()
{
  int   z = 3;
  printf("Value for z=%d is %d\n",z,fS(z));
  return 0;
}
```

Calculer la somme des n premiers entiers (v2)

```
int fS(int n) {
  int ps = 0;
  int k = 0;
  int ok=k, ops = 0;
  while (k < n) {
 /*@ assert 0 <= k && k < n
   && ps == S(k) && ops == S(ok);     */
    ok=k;ops=ps;
    k = ok + 1;
    ps = ops + k;
 /*@ assert 0 <= k && k <= n && ps == S(k)
   && ops == S(ok);     */
  };
  return ps;
}
```

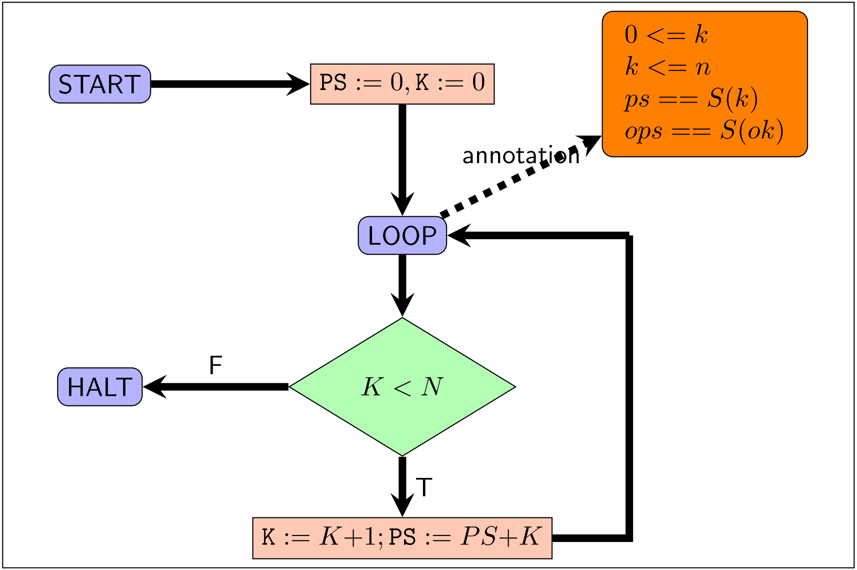
Calculer la somme des n premiers entiers

```
/*@ axiomatic S {
  @ logic integer S(integer n);
  @ axiom S_0: S(0) == 0;
  @ axiom S_i: \forall integer i; i > 0 ==> S(i) == S(i-1)+i;
  @ } */

/*@ requires n >= 0;
   assigns \nothing ;
   ensures \result == S(n);
*/
int fS(int n) {
  int ps = 0;
  int k = 0;
  int ok=k,ops=ps;
  /*@ loop invariant 0 <= k && k <= n && ps == S(k) && ops == S(ok) ;
    loop assigns ps, k,ops,ok;
   */
  while (k < n) {
/*@ assert l0: 0 <= k && k < n && ps == S(k) && ops == S(ok);    */
    ops=ps;ok=k;
     k = ok + 1;
    ps = ops + k;
 /*@ assert l1:  0 <= k && k <= n && ps == S(k) && ops == S(ok);    */
  };
 /*@ assert  ps == S(n);    */
  return ps;}
```

```
frama-c -wp sum.c
[kernel] Parsing sum.c (with preprocessing)
[wp] Warning: Missing RTE guards
[wp] 8 goals scheduled
[wp] Proved goals:    8 / 8
  Qed:               6  (2ms-4ms-10ms)
  Alt-Ergo 2.3.3:    2  (8ms-15ms) (38)
```

Calculer la somme des n premiers entiers (annotated flowchart)



Observations sur le calcul

- Définition des fonctions mathématiques nécessaires pour exprimer le calcul de la somme des n premiers nombres entiers.
- Expression des résultats intermédiaires appelés *sommes partielles*
- Relation entre la preuve par induction et la forme du corps de l'itération.
- Induction et calcul sont liés.

Observations sur le calcul

- Définition des fonctions mathématiques nécessaires pour exprimer le calcul de la somme des n premiers nombres entiers.
- Expression des résultats intermédiaires appelés *sommes partielles*
- Relation entre la preuve par induction et la forme du corps de l'itération.
- Induction et calcul sont liés.

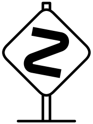
$$x_0 \xrightarrow{P} x \tag{1}$$

$$x_0 \xrightarrow{\star} x \tag{2}$$

$$x_0 \longrightarrow x_1 \longrightarrow \ldots \longrightarrow x \tag{3}$$

$$x_0 \longrightarrow x_1 \longrightarrow \ldots \longrightarrow x_n \xrightarrow{step} x \tag{4}$$

Current Summary



On convient des notations suivantes équivalentes : $x \in E$ est équivalent à $E(x)$ pour toute valeur $x \in$ Vals. Cette simplification permet relier un ensemble $U \subseteq$ Vals à une assertion $U(x)$ en considérant que $U(x)$ et $x \in U$ désigne le même concept.

Les deux expressions suivantes sont équivalentes :

- $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow A(x)$
- $\forall x \in \textsc{Vals}.(\exists x_0.x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \mathsf{Next}^\star(x_0, x)) \Rightarrow A(x)$

- $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow A(x)$
- $\forall x \in \textsc{Vals}.(\exists x_0.x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \mathsf{Next}^\star(x_0, x)) \Rightarrow A(x)$.
- $\textsc{reachable}(M) = \{u | u \in \textsc{Vals} \wedge (\exists x_0.x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \mathsf{Next}^\star(x_0, u))\}$ est l'ensemble des états accessibles à partir des états initiaux et on doit montrer la propriété de sûreté $A(x)$ en montrant l'inclusion des ensembles (model-checking) :
$$\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$$

Principe assertionnel d'induction

Soit $(Th(s,c), x, \textsc{Vals}, Init(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système S.
Une propriété $A(x)$ est une propriété de sûreté pour le système S,

$$\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow A(x)$$

si et seulement si,
il existe une propriété d'état $I(x)$, telle que :

$$\forall x, x' \in \textsc{Vals} \ : \ \left\{ \begin{array}{l} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \mathsf{Next}(x, x') \Rightarrow I(x') \end{array} \right.$$

La propriété $I(x)$ est appelée un invariant inductif de S et est une propriété de sûreté particulière plus forte que les autres propriétés de sûreté.

Justification (correction)

Soit une propriété $I(x)$ telle que :

$$\forall x, x' \in \textsc{Vals} \ : \left\{ \begin{array}{l} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \mathsf{Next}(x,x') \Rightarrow I(x') \end{array} \right.$$

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

Soient x_0 et $x \in \textsc{Vals}$ tels que $Init(x_0) \wedge \mathsf{Next}^\star(x_0, x)$.

- On peut construire une suite telle que :
  $x_0 \xrightarrow[\mathsf{Next}]{} x_1 \xrightarrow[\mathsf{Next}]{} x_2 \xrightarrow[\mathsf{Next}]{} \ldots \xrightarrow[\mathsf{Next}]{} (x_i = x)$.
- L'hypothèse (1) nous permet de déduire $I(x_0)$.
- L'hypothèse (3) nous permet de déduire $I(x_1)$, $I(x_2)$, $I(x_3)$, $\ldots$, $I(x_i)$. En utilisant l'hypothèse (2) pour x, nous en déduisons que x satisfait A.

Justification (complétude)

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

$\textsc{Prouvons que}$: il existe une propriété $I(x)$ telle que :

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

- Nous considérons la propriété suivante :
  $I(x) \ \widehat{=}\ \exists x_0 \in \textsc{Vals} \cdot Init(x_0) \wedge x_0 \xrightarrow[\text{Next}]{\star} x.$
- $I(x)$ 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 du système de transition défini par *Init* et Next

Assertion versus relation

- 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 ($\Box$ propriété) et d'invariant au lieu d'invariant inductif.

Propriétés de sûreté et d'invariance dans un modèle relationnel

. . . .

⊠ Definition(assertion)

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

$$\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow A(x).$$

. . . .

. . . .

⊠ Definition(relation)

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

$$\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow R(x_0, x).$$

. . . .

Relation etre les deux types de propriétés assertionnelle et relationnelle

- $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow R(x_0, x)$ (R) est une propriété relationnelle de sûreté.
- Soit $y = (x_0, x)$, $y_0 = (x_0, x_0)$, et $\mathsf{NextR}(y, y') \stackrel{def}{=} \mathsf{Next}(x, x') \wedge y = (x_0, x) \wedge y' = (x_0, x')$
- (R) est réécrit comme suit :

$$\forall y_0, y \in \textsc{Vals}{\times}\textsc{Vals}.Init(x_0) \wedge y_0 = (x_0, x_0) \wedge \mathsf{NextR}^\star(y_0, y) \Rightarrow R(y) \text{ (R)}$$

- Par la propriété de correction et de complétude
- il existe une propriété d'état $IR(y)$, telle que :

$$\forall y_0, y \in \textsc{Vals}{\times}\textsc{Vals}. \begin{cases} (1)\ Init(x_0) \wedge y_0 = (x_0, x_0) \Rightarrow IR(y) \\ (2)\ IR(y) \Rightarrow R(y) \\ (3)\ IR(y) \wedge \mathsf{NextR}(y, y') \Rightarrow IR(y') \end{cases}$$

- il existe une propriété relationnelle $IR(x_0, x)$, telle que :

$$\forall x_0, x \in \textsc{Vals}. \begin{cases} (1)\ Init(x_0) \wedge y_0 = (x_0, x_0) \Rightarrow IR(x_0, x_0) \\ (2)\ IR(x_0, x) \Rightarrow R(x_0, x) \\ (3)\ IR(x_0, x) \wedge \mathsf{Next}(x, x') \Rightarrow IR(x_0, x') \end{cases}$$

On obtient donc deux types de principes d'induction selon les propriétés de sûreté.

Complétude et correction

$\forall x_0, x \in \text{VALS}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow A(x)$.

si, et seulement si,

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

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

- 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 ls calculs sont définis en cet état.
- $A{-}RTE(x)$ signifie qu'on peut évaluer la relation $\mathsf{Next}(x, x')$ dans le modèle de calcul : $A{-}RTE(x) = \exists x' \in \mathcal{P}(\text{VALS}).\mathsf{Next}(x, x')$

Principes relationnels d'induction

Complétude et correction

$\forall x_0, x \in \text{VALS}.Init(x_0) \wedge \mathsf{Next}^{\star}(x_0, x) \Rightarrow A(x_0, x)$.

si, et seulement si,

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

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

- La correction partielle est caractérisée comme une relation entre l'état initial et l'état courant.
- $A{-}PC(x)$ signifie que le programme se termine dans des états définissant la postcondition dans le modèle de calcul : $A{-}PC(x_0, x) = term(x) \Rightarrow PRE(x_0) \wedge POSTCOND(x_0, x)$.

Sommaire sur les deux principes d'induction

Principe assertionnel de sûreté ou d'invariance

$$\exists I(x) \in \mathcal{P}(\textsc{Vals}). \left[\begin{array}{l} \forall x, x' \in \textsc{Vals}. \\ (1) \quad Init(x) \Rightarrow I(x) \\ (2) \quad I(x) \Rightarrow A(x) \\ (3) \quad I(x) \wedge \mathsf{Next}(x, x') \Rightarrow I(x') \end{array}\right.$$

Principe relationnel de sûreté ou d'invariance

$$\exists IR(x_0, x) \in \mathcal{P}(\textsc{Vals}\times\textsc{Vals}).$$

$$\left[\begin{array}{l} \forall x_0, x, x' \in \textsc{Vals}. \\ (1) \quad Init(x_0) \Rightarrow IR(x_0, x_0) \\ (2) \quad IR(x_0, x) \Rightarrow R(x_0, x) \\ (3) \quad IR(x_0, x) \wedge \mathsf{Next}(x, x') \Rightarrow IR(x_0, x') \end{array}\right.$$

Current Summary

1. Summation of the n first integers
2. Principe(s) d'induction
3. Méthode de preuves de propriétés d'invariance

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 : $x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X

begin
$0 : P_0(x_0, x)$
`instruction`$_0$
$\ldots$
$i : P_i(x_0, x)$
$\ldots$
`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 $\textsc{Programs}$
- 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 $\textsc{Programs}$; ce programme comprend
  - des variables notées globalement v,
  - des constantes notées globalement pc,
  - des types associés aux variables notés globalement $\textsc{Vals}$ et identifiés à un ensemble de valeurs possibles des variables,
  - des instructions suivant un ordre défini par la syntaxe du langage de programmation.

Hypothèses

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

$$x = (pc, v) \wedge pc \in \textsc{Locations} \wedge v \in \textsc{Memory} \tag{5}$$

On considère un programme P annoté ; on se donne un modèle relationnel $\mathcal{MP} = (Th(s,c), x, \textsc{Vals}, \mathsf{Init}(x), \{r_0, \ldots, r_n\})$ 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.
- $\textsc{Locations} \times \textsc{Memory}$ est un ensemble de valeurs possibles pour x.
- $\{r_0, \ldots, r_n\}$ 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.
- $\mathsf{Init}(x)$ 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.

Hypothèses syr le calcul

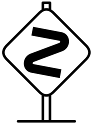
On suppose qu'il existe un graphe sur l'ensemble des valeurs de contrôle définissant la relation de flux et nous notons cette structure $(\textsc{Locations}, \longrightarrow)$.

⊠ Definition

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

⊠ Definition(Annotation d'un point de contrôle)

Soit une structure $(\textsc{Locations}, \longrightarrow)$ et une étiquette $\ell \in \textsc{Locations}$. 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.**

Relation entre annotation et invariant

- Les étiquettes ℓ appartiennent à $\textsc{Locations}$: $\ell \in \textsc{Locations}$.
- Les variables v appartiennent à $\textsc{Memory}$: $v \in \textsc{Memory}$.
- $pre(v_0)$ 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 :

$$J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array} \right.$$

- $J(\ell_0, v_0, pc, v)$ 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'algoritrhme et ℓ_f est la fin du programme. On pourra utiliser simplement 0 et f.

$$x = (pc, v) \text{ et } J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \land\ pc \in \textsc{Locations} \\ \land\ v \in \textsc{Memory} \\ \dots \\ \land\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \dots \end{array} \right.$$

Soit $(Th(s,c), x, \textsc{Vals}, \mathsf{Init}(x), \{r_0, \dots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x_0, x)$ est une propriété de sûreté pour P, si $\forall x_0, x \in \textsc{Locations}\times\textsc{Memory}.Init(x_0) \land x_0 \xrightarrow[\text{Next}]{} x \Rightarrow A(x)$. On sait que cette propriété implique qu'il existe une propriété d'état $\mathrm{I}(x_0, x)$ telle que les trois propriétés sont vérifiées mais on applique cette vérification pour J :

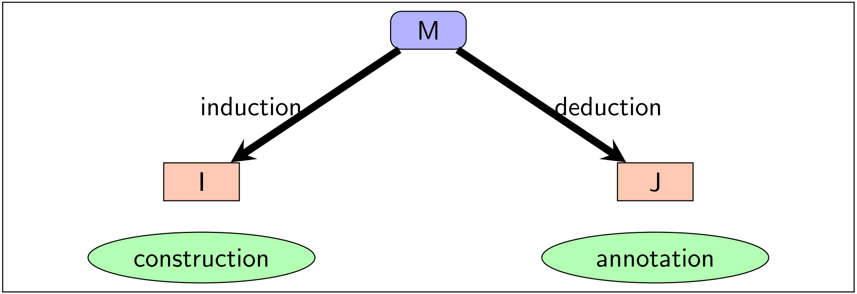
$\forall x_0, x, x' \in \textsc{Locations}\times\textsc{Memory}$:

$$\left\{ \begin{array}{ll} (1) & \mathsf{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \dots, n\} : \mathrm{J}(x_0, x) \land x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x') \end{array} \right.$$

$\forall i \in \{0, \dots, n\} : \mathbf{J}(x_0, x) \land x\ r_i\ x' \Rightarrow \mathbf{J}(x_0, x')$ **est équivalent à** $\mathbf{J}(x_0, x) \land (\exists i \in \{0, \dots, n\} : x\ r_i\ x') \Rightarrow \mathbf{J}(x_0, x')$

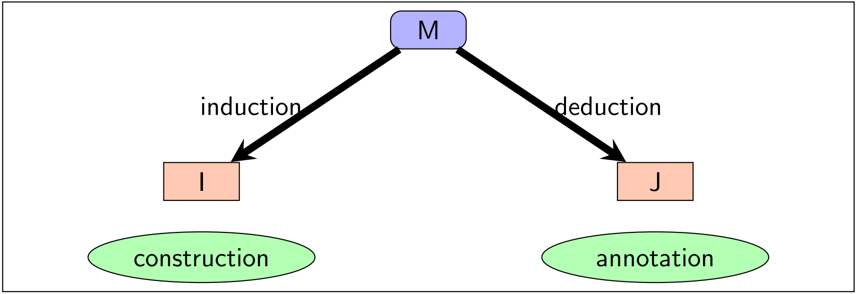
Vérification à faire

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



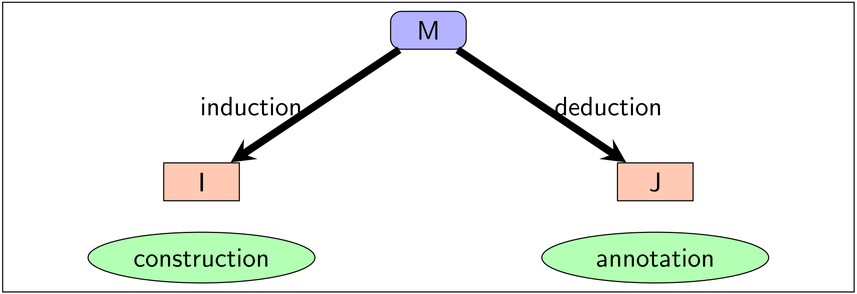
Utilisation du principe relationnel d'induction (RI)

- $\textsc{Vals} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc_0, v_0, pc, v) \stackrel{def}{=} \exists x_0, x \in \textsc{Vals}.I(x_0, x) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (deduction)
- $I(x_0, x) \stackrel{def}{=} \exists pc_0, pc \in \textsc{Locations}, v_0, v \in \textsc{Memory}.J(pc_0, v_0, pc, v) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (induction)



Utilisation du principe assertionnel d'induction (AI)

- $\textsc{Vals} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc, v) \stackrel{def}{=} \exists x \in \textsc{Vals}.I(x) \wedge x = (pc, v)$ (deduction)
- $I(x) \stackrel{def}{=} \exists pc \in \textsc{Locations}, v \in \textsc{Memory}.J(pc, v) \wedge x = (pc, v)$ (induction)



Adaptation aux programmes

(1) $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow A(x)$ $(I(x))$
(2) $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \mathsf{Next}^\star(x_0, x) \Rightarrow R(x_0, x)$ $(IR(x))$

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 :

- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge I(x))$
- $IR(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge RP_\ell(v_0, v))$
- $RP_\ell(v_0, v) \stackrel{def}{=} \exists x, x_0.(x, x_0 \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge IR(x_0, x))$

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 :

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

Transition d'une étiquette vers une autre étiquette

- 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 $= f_{\ell,\ell'}(v)$ 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 $D = -2^{31}|..2^{31}-1$ pour un codage sur 32 bits.
  - $DEF(T(I+1) < V)(t,x,v) \stackrel{def}{=} i+1 \in dom(t) \wedge v \in D \wedge t(i+1) \in D$

Relations pour des instructions de programmes

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

Traduction

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

$\ell_1 : P_{\ell_1}(v_0, v)$
WHILE $B(V)$ **DO**
  $\ell_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $\ell_3 : P_{\ell_3}(v_0, v)$
END
$\ell_4 : P_{\ell_4}(v_0, v)$

Traduction

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

Vérification du contrat : ce qui sera la technique

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 : $v_0 \xrightarrow{\mathsf{P}} v_f$
- v_0 satisfait pre : pre(v_0) and v_f satisfait post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow$ post(v_0, v_f)

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

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

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

Vérification du contrat : un exemple simple

variables U, V
requires $u_0, v_0 \in \mathbb{N}$
ensures $u_f + v_f = u_0 + v_0$

$$\left[\begin{array}{l}
\text{begin} \\
0 : u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\
U := U + 2 \\
1 : u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\
V := V - 2 \\
2 : u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N} \\
\text{end}
\end{array}\right.$$

Vérification à faire

$$x = (pc, v) et J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array} \right.$$

Soit $(Th(s, c), x, \textsc{Vals}, \mathsf{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x)$ est une propriété de sûreté pour P, si $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}. Init(x_0) \wedge x_0 \xrightarrow[\mathsf{Next}]{} x \Rightarrow A(x)$. On sait que cette propriété implique qu'il existe une propriété d'état $\mathrm{I}(x)$ telle que les trois propriétés sont vérifiées mais on applique cette vérification pour J :

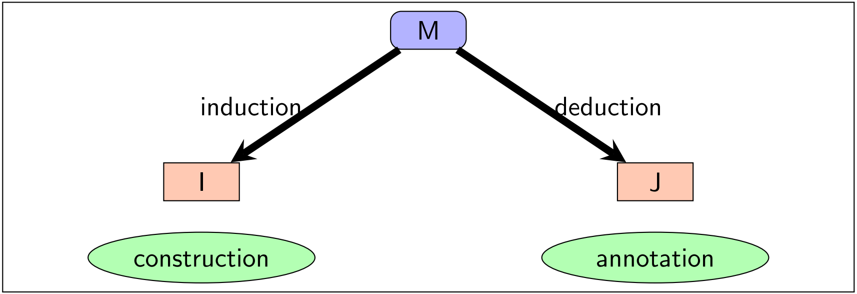
$\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

$$\left\{ \begin{array}{ll} (1) & \mathsf{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x') \end{array} \right.$$

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x')$ est équivalent à $\mathrm{J}(x) \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x') \Rightarrow \mathrm{J}(x')$

Vérification à faire

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



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

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

$$\left\{\begin{array}{ll} (1) & \mathsf{Init}(0, u_0, v_0) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0) \\ (2) & \mathrm{J}(0, u_0, v_0, pc, u, v) \Rightarrow \mathrm{A}(0, u_0, v_0, pc, u, v) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(0, u_0, v_0, pc, u, v) \wedge x\ r_i\ pc', u', v' \Rightarrow \mathrm{J}(0, u_0, v_0, pc', \end{array}\right.$$

Vérification sur l'exemple

1 $\mathsf{Init}(0, u_0, v_0,) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
$pc = 0 \wedge u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:

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

3 $\forall i \in \{0, \ldots, n\} : \mathrm{J}(0, u_0, v_0, pc, u, v) \wedge x \; r_i \; pc', u', v' \Rightarrow \mathrm{J}(0, u_0, v_0, pc', u', v')$

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

- $J(0, u_0, v_0, pc, u, v) \wedge r01(pc, u, v, pc', u', v') \Rightarrow J(pc', u', v')$
- $J(0, u_0, v_0, pc, u, v) \wedge r12(pc, u, v, pc', u', v') \Rightarrow J(0, u_0, v_0, pc', u', v')$

Quelques règles de calcul logique

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

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

$$
\Rightarrow
$$

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

Utilisation de TLA Toolbox pour vérifier ces éléments : cours1.tla