Cours MVSI
Modélisation et Vérification
des Systèmes Informatiques

Modélisation, spécification et vérification (II)

Dominique Méry
Telecom Nancy, Université de Lorraine

(4 décembre 2025 at 4:21 P.M.)

Plan

1. Le langage PlusCal
    - Définir des processus en PlusCal
    - Macros et procédures
2. Summation of the n first integers
3. Principe(s) d'induction
4. Méthode de preuves de propriétés d'invariance

PlusCal un langage algorithmique plongé dans TLA/TLA$^+$

- 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 parrtager 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;

{
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 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
=================================================================
```

Forme générale

```
—— 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; *)
```

Partage de variables et localisation des variables

- Les variables globales sont déclarées au niveau de l'algorithme et sont partagées par tous les processus qui peuvent lir et écrire en accord avec la rééègle d'atomicité.
- Les variables locales sont définies au niveau de chaque processus qui est un processus séquentiel.
- Le parallélisme est réalisé par entrelacement non-déterministe des actions des différents procesus.
- Le contrôle est introduit par la variable pc qui est un tableau de taille le nombre de processus concurrents possibles.

Exemple 2

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

Processus en PlusCal

- Un algorithme multiprocessus contient un ou plusieurs processus.
- Un processus commence de deux manières :
  - en définissant un ensemble de processus : `process ( ProcName` $\in$ `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
```

Définition des processus S et R

```
—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
```

Macros et Procedures

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

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

Annotation versus commentaire de programmes ou d'algorithmes

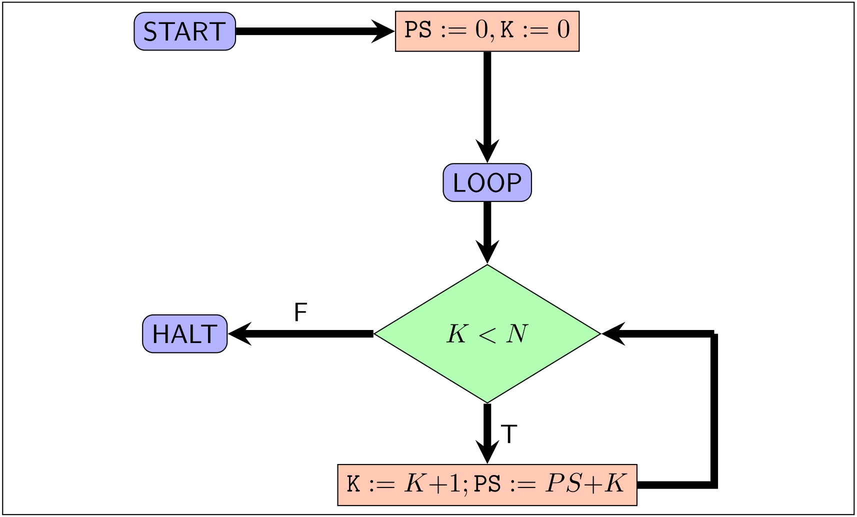
- Un programme ou un algorithme peuvent être annotés ou commentés.
- Un commentaire est une information pertinente destinée à être vue ou lue et qui a une importance relative dans l'esprit du concepteur.
- Un commentaire indique une information sur les données, sur les variables et donc sur l'état supposé 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)

```c
#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

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

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;
}
```

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.

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

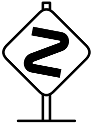
$$x_0 \xrightarrow{\star} x \tag{2}$$

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

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

i

Quelques simplifications et notations



On convient des notations suivantes équivalentes : $x \in E$ est équivalent à $E(x)$ pour toute valeur $x \in$ Vals. Cette simplification permet de 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 :

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

i

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

$$\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$$

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

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

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 :

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

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

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

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

Justification (complétude)

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

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

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

- Nous considérons la propriété suivante :
  $I(x) \;\widehat{=}\; \exists x_0 \in \text{VALS} \cdot Init(x_0) \wedge x_0 \xrightarrow[\text{Next}]{\star} x$.
- $I(x)$ 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 de l'algorithme $\mathcal{A}$.

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.

. . . .

$\boxtimes$ Definition(assertion)

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

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

. . . .

. . . .

$\boxtimes$ Definition(relation)

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

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

. . . .

Complétude et correction

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

si, et seulement si,

il existe $I \in \mathcal{P}(\textsc{Vals})$

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

si, et seulement si,

$$\exists i \in \mathcal{P}(\textsc{Vals}). \left[\begin{array}{ll} (1) & Init \subseteq i \\ (2) & i \subseteq A \\ (3) & \forall x, x' \in \textsc{Vals}.i(x) \wedge \textsc{Next}(x, x') \Rightarrow i(x') \end{array} \right.$$

Complétude et correction

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

si, et seulement si,

il existe $R \in \mathcal{P}(\textsc{Vals}\times\textsc{Vals})$

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

si, et seulement si,

$\exists R \in \mathcal{P}(\textsc{Vals}\times\textsc{Vals})$.

$$\left[\begin{array}{ll} (1) & Init\times Init \subseteq R \\ (2) & R \subseteq A \\ (3) & \forall x, x' \in \textsc{Vals}.R(x_9, x) \wedge \textsc{Next}(x, x') \Rightarrow R(x_0, x') \end{array}\right.$$

Conséquences dur les méthodes pratiques

- La propriété invariante I est définie par
  $\mathsf{I}(x) \stackrel{def}{=} \exists x_0 \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0, x)$
- La propriété invariante R est définie par
  $\mathsf{R}(x_0, x) \stackrel{def}{=} Init(x_0) \wedge \textsc{Next}^\star(x_0, x)$

Propriétés de sûreté et d'invariance dans un modèle relationnel

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

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

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

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

Relation etre les deux types de propriétés assertionnelle et relationnelle

- $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow R(x_0, x)$ (R) est une propriété relationnelle de sûreté.
- Soit $y = (x_0, x)$, $y_0 = (x_0, x_0)$, et
  $\textsc{NextR}(y, y') \stackrel{def}{=} \textsc{Next}(x, x') \wedge y = (x_0, x) \wedge y' = (x_0, x')$
- (R) est réécrit comme suit :
  $$\forall y_0, y \in \textsc{Vals}\times\textsc{Vals}.Init(x_0) \wedge y_0 = (x_0, x_0) \wedge \textsc{NextR}^\star(y_0, y) \Rightarrow R(y) \text{ (R)}$$
- Par la propriété de correction et de complétude
- il existe une propriété d'état $IR(y)$, telle que :
  $$\forall y_0, y \in \textsc{Vals}\times\textsc{Vals}. \begin{cases} (1)\ Init(x_0) \wedge y_0 = (x_0, x_0) \Rightarrow IR(y) \\ (2)\ IR(y) \Rightarrow R(y) \\ (3)\ IR(y) \wedge \textsc{NextR}(y, y') \Rightarrow IR(y') \end{cases}$$
- il existe une propriété relationnelle $IR(x_0, x)$, telle que :
  $$\forall x_0, x \in \textsc{Vals}. \begin{cases} (1)\ Init(x_0) \wedge \Rightarrow IR(x_0, x) \\ (2)\ IR(x_0, x) \Rightarrow R(x_0, x) \\ (3)\ IR(x_0, x) \wedge \textsc{Next}(x, x') \Rightarrow IR(x_0, x') \end{cases}$$

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

Complétude et correction

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

si, et seulement si,

il existe $I \in \mathcal{P}(\textsc{Vals})$

$$\forall x_0, x, x' \in \textsc{Vals} \;:\; \left\{ \begin{array}{ll} (1) & Init(x_0) \Rightarrow I(x_0) \\ (2) & I(x) \Rightarrow A(x) \\ (3) & I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{array} \right.$$

- 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. principe

Complétude et correction

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

si, et seulement si,

il existe $R \in \mathcal{P}(\textsc{Vals} \times \textsc{Vals})$

$$\forall x_0, x, x' \in \textsc{Vals} \;:\; \left\{ \begin{array}{ll} (1) & Init(x_0) \Rightarrow R(x_0, x_0) \\ (2) & R(x_0, x) \Rightarrow A(x_0, x) \\ (3) & R(x_0, x) \wedge \textsc{Next}(x, x') \Rightarrow R(x_0, x') \end{array} \right.$$

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

Principe assertionnel de sûreté ou d'invariance

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

Principe relationnel de sûreté ou d'invariance

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

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

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

```
requires pre(x0)
< ensures post(x0, xf)
variables X
      begin
      0 : P0(x0, x)
      instruction0
      ...
      i : Pi(x0, x)
      ...
      instructionf-1
      f : Pf(x0, 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.

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 $\mathcal{MP} = (Th(s, c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ 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.
- $\textsc{Init}(x)$ 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 $(\textsc{Locations}, \longrightarrow)$.

. . . .

⊠ Definition

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

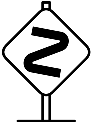
. . . .

. . . .

⊠ 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 valur 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.

Vérification à faire

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

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x_0, x)$ est une propriété de sûreté pour P, si $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}. Init(x_0) \wedge x_0 \xrightarrow[\textsc{Next}]{} x \Rightarrow A(x)$.

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 :

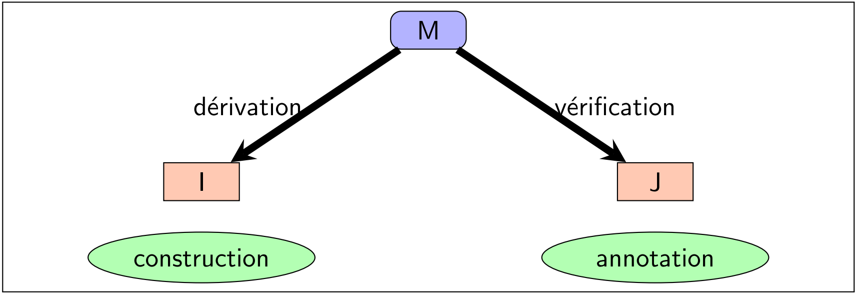
$\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

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

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

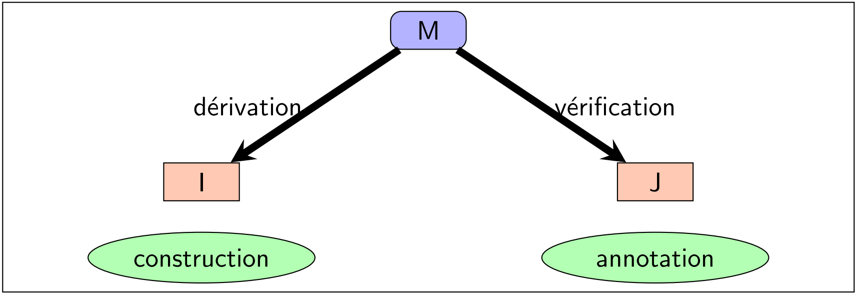
Vérification à faire

- 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 (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).



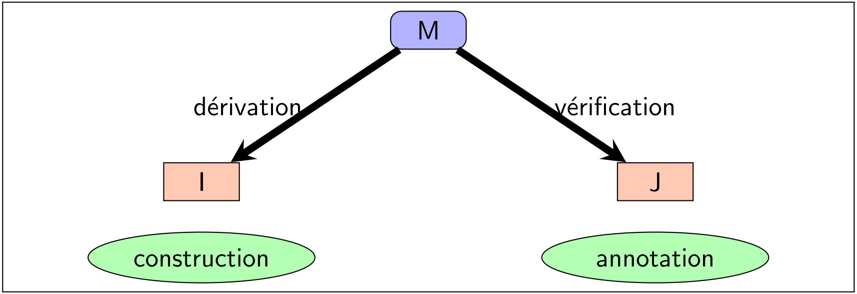
Utilisation du principe relationnel d'induction (RI)

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

- $\textsc{Vals} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc,v) \stackrel{def}{=} \exists x \in \textsc{Vals}.I(x) \wedge x = (pc,v)$ (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) $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0,x) \Rightarrow A(x)$ $(I(x))$
(2) $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0,x) \Rightarrow R(x_0,x)$ $(IR(x))$

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

$$\begin{cases} pc = \ell_1 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_1 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \\ pc = \ell_3 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_3 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \end{cases}$$

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

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

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

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \dots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x)$ est une propriété de sûreté pour P, si $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}.Init(x_0) \wedge x_0 \xrightarrow[\textsc{Next}]{} x \Rightarrow A(x)$. 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 :

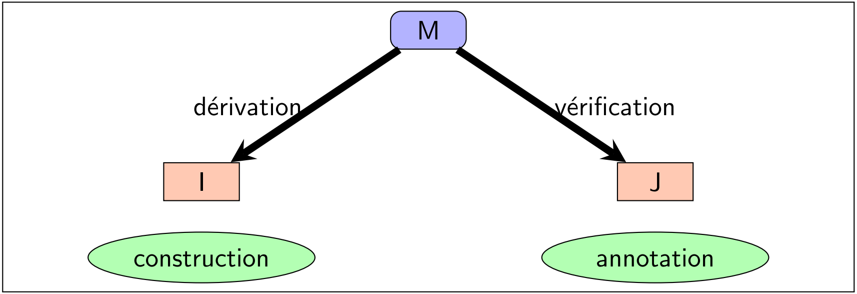
$\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

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

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

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 (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).



Vérification à faire : retour sur l'exemple

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

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

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

Vérification sur l'exemple

1. $\textsc{Init}(0, u_0, v_0,) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
   $pc = 0 \wedge u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
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 $[A \wedge (B = C \Rightarrow U) \wedge (B = D \wedge B = C \Rightarrow V) wedge C \neq D \wedge E] \longrightarrow [A \wedge B = C \wedge U \wedge C \neq D \wedge E]$

$J(pc,u,v) \wedge r01(pc,u,v,pc',u',v') \Rightarrow J(pc',u',v')$ **(1)**

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

$$
\Rightarrow
$$

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

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

Initialisation

Le modèle relationnel $M(P)$ pour le programme P annoté est donc défini comme suit :
$M(P) \stackrel{def}{=}$
$(Th(s,c),(pc,v),\textsc{Locations}\times\textsc{Memory},Init(\ell,v),\{r_{\ell,\ell'}|\ell,\ell' \in \textsc{Locations} \wedge \ell \longrightarrow \ell'\})$.
La définition de $Init(x)$ est dépendante de la précondition de P :

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

Conditions initiales

Les deux propriétés suivantes sont équivalentes :

- $\forall x_0 \in \textsc{Vals} : \mathit{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0)$
- $\forall v \in \textsc{Memory}.\textbf{pre}(\mathrm{P})(v) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$

Pas d'induction

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

Pas d'induction

Les deux propriétés suivantes sont équivalentes :

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x \ r_i \ x' \Rightarrow \mathrm{J}(x_0, x')$
- $\forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Pas d'induction (explication)

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

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x \ r_{\ell,\ell'} \ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \ r_{\ell,\ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \;\stackrel{def}{=}\; (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x \ r_{\ell,\ell'} \ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \ r_{\ell,\ell'} \ x' \ \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'\ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \; \Rightarrow \; pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \; \Rightarrow \; pc = \ell'$ (Tautologie)

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow P_{\ell'}(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0nx) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0,x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0nx) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0,x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0,v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0,v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow P_{\ell'}(v_0, v')$
- $P_\ell(v_0,v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Conclusion

- $J(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x, x_0 \in \textsc{Vals} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge J(x_0), x)$
- $\mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x)$
- $\exists \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell \in \textsc{Locations}, v, v_0 \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$

Conclusion

Les deux propriétés suivantes sont équivalentes :

- $\mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell \in \textsc{Locations}, v, v_0 \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$

Traduction des conditions de vérifications

Les conditions de vérification suivantes sont équivalentes :

- $\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:
$$\begin{cases} (1) & \textsc{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x \; r_i \; x' \Rightarrow \mathrm{J}(x_0, x') \end{cases}$$
- $\forall v_0, v, v' \in \textsc{Memory}$:
$$\begin{cases} (1) & \textbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v) \\ (2) & \forall \ell \in \textsc{Locations}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v) \\ (3) & \forall \ell, \ell' \in \textsc{Locations} : \\ & \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{cases}$$

Méthode relationnelle de preuve de correction de propriétés de sûreté

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

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

Floyd and Hoare

- $\forall v_0, v, v' \in \textsc{Memory}.\forall \ell,\ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0,v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0,v')$ est équivalent à $\forall \ell,\ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \textsc{Memory}.P_\ell(v_0,v) \wedge cond_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$
- $\forall v_0, v, v' \in \textsc{Memory}.\forall \ell,\ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0,v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0,v')$ est équivalent à $\forall \ell,\ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \textsc{Memory}.\left(\exists v \in \textsc{Memory}.P_\ell(v_0,v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)\right) \Rightarrow P_{\ell'}(v_0,v')$

Condition de vérification pour l'affectation

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

- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

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

Condition de vérification pour l'affectation

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

- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

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

Condition de vérification pour l'affectation

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

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

- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Condition de vérification pour l'affectation

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

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

- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (l'axiomatique de Hoare).

Condition de vérification pour l'affectation

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

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

- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (l'axiomatique de Hoare).
- $\forall v \in \textsc{Memory}.\,(\exists v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v_0, v')$ correspond à la règle d'affectation de Floyd.

Conditions de vérification pour l'itération

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

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

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

Conditions de vérification pour la conditionnelle

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

Pour la structure de conditionnelle, les conditions suivantes :

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

Soit v une variable d'état de P. **pre**$(\mathrm{P})(v)$ est la précondition de P pour v ; elle caractérise les valeurs initiales de v. **post**$(\mathrm{P})(v_0, v)$ est la postcondition de P pour v ; elle caractérise les valeurs finales de v en relation avec la valeur initiale v_0

Exemple

1. **pre**$(\mathrm{P})(x, y, z)$=$x, y, z \in \mathbb{N}$ et **post**$(\mathrm{P})(x_0, y_0, z_0, x, y, z)$=$z = x_0 \cdot y_0$
2. **pre**$(\mathrm{Q})(x, y, z)$=$x, y, z \in \mathbb{N}$ et **post**$(\mathrm{Q})(x_0, y_0, z_0, x, y, z)$=$z = x_0 + y_0$

$$\begin{array}{l}
\forall \underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q}. \\
\quad \mathbf{pre}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}) \wedge (\underline{x}, \underline{y}, \underline{r}, \underline{q}) \xrightarrow{\mathrm{P}} (\overline{x}, \overline{y}, \overline{r}, \overline{q}) \\
\qquad \Rightarrow \mathbf{post}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q})
\end{array}$$

Correction partielle d'un programme

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**$(\mathrm{P})(\mathsf{v}_0)$
- la spécification des résultats de P **post**$(\mathrm{P})(\mathsf{v}_0,\mathsf{v})$
- une famille d'annotations de propriétés $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant la correction partielle $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v_f)$ où ℓ_f est l'étiquette marquant la fin du programme P

. . . .

$\boxtimes$ Definition

Le programme P est partiellement correct par rapport à **pre**$(\mathrm{P})(\mathsf{v}_0)$ et **post**$(\mathrm{P})(\mathsf{v}_0,\mathsf{v})$, si la propriété $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$ est une propriété de sûreté pour ce programme.

. . . .

Correction partielle d'un programme

Si les conditions suivantes sont vérifiées :

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

alors le programme P est partiellement correct par rapport à **pre**(P)(v_0) et **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 :

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

  (expression de la correction partielle)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \xrightarrow{\textsc{Next}^\star} (pc, v) \wedge pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f)$

  (big-step semantics et small-step semantics equivalence)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \xrightarrow{\textsc{Next}^\star} (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$

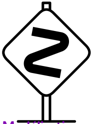
  (implication and conjunction property)

- $Init(x_0) \wedge x_0 \xrightarrow{\textsc{Next}^\star} x \Rightarrow \mathsf{PC}(x_0, x)$

  $(Init(x_0) \stackrel{def}{=} pc_0 = \ell_0 \wedge \mathsf{pre}(v_0)$

  $x_0 \stackrel{def}{=} (\ell_0, v_0)$ and $x \stackrel{def}{=} (pc, v)$

  $\mathsf{PC}(x_0 x) \stackrel{def}{=} x_0 = (\ell_0, v_0) \wedge x = (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$



Partial correctness is a safety property and the relational method for safety properties is applied.

Vérification du contrat

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

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

$$\left[\begin{array}{l}
\text{begin} \\
0 : P_0(v_0, v) \\
\texttt{instruction}_0 \\
\ldots \\
i : P_i(v_0, v) \\
\ldots \\
\texttt{instruction}_{f-1} \\
f : P_f(v_0, v) \\
\text{end}
\end{array}\right.$$

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $v, v' \in \textsc{Memory}$
$$\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

Méthode des assertions intermittentes de Floyd/Turing

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, $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$

Absence d'erreurs à l'exécution

- 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 $\textbf{DOM}(\ell,\ell')(v)$ pour la transition considérée. $\textbf{DOM}(\ell,\ell')(v)$ 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 'existe pas, une référence à un pointeur nul, ...

exemple

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

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 $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant l'absence d'erreurs à l'exécution :
$$\bigwedge_{\ell \in \textsc{Locations}-\{output\}, n \in \textsc{Locations}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$$

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

Principe d'induction pour RTE

RTE = Run Time Error

Si les conditions suivantes sont vérifiées :

- $\forall v_0, v \in \textsc{Memory} : \textbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- $\forall m \in \textsc{Locations}-\{\ell_f\}, n \in \textsc{Locations}, \forall v_0, v, v' \in \textsc{Memory} :$ $m \longrightarrow n : \textbf{pre}(\mathrm{P})(v_0) \wedge P_m(v_0, v) \Rightarrow \textbf{DOM}(m, n)(v)$
- $\forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in$ $\textsc{Memory}. (P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$,

alors le programme P ne produira pas d'erreurs à l'exécution par rapport à $\textbf{pre}(\mathrm{P})(\mathsf{v}_0)$ et $\textbf{post}(\mathrm{P})(\mathsf{v}_0,\mathsf{v})$.

- 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.

Méthode relationnelle de vérification pour RTE

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)
- $\mathbb{D}$ 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

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(x_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $v, v' \in \textsc{Memory}$
$$\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
- $\forall m \in \textsc{Locations} - \{\ell_f\}, n \in \textsc{Locations}, \forall v_0, v, v' \in \textsc{Memory}$: $m \longrightarrow n$: $pre(v_0) \wedge P_m(v_0, v) \Rightarrow \mathbf{DOM}(m, n)(v)$