
Cours Modélisation et Vérification des Systèmes Informatiques (MVSI)

Le langage de spécification pour Frama-C ACSL

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2024-2025

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL
 - Variables dites *ghost*

Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

Listing 1 – slide0a.c

```
#include <limits.h>
void slide1(int a, int b, int c) {
    int x = a;
    int y = b;
    int z = c;
    //@ assert x + y == 4    && z == x + 2;
    z = x + y + 2 * z;
    //@ assert x + y == 4    && z + y == x + z;
}
```

```
frama-c  slide0a.c -wp  -report
```

Listing 2 – slide0b.c

```
#include <limits.h>
/*@ requires a+b==4 && b+c==6 && c== 4 && a >= 0 && b >= 0 && b <= INT_MAX && a <= INT_MAX;
 */
void slide1(int a, int b, int c) {
    int x = a;
    int y = b;
    int z = c;
    //@ assert x + y == 4 && z == x + 2;
    z = x + y + 2 * z;
    //@ assert x + y == 4 && z + y == x + z;
}
```

frama-c slide0b.c -wp -report

Exemple d'annotation

```
11: x+y=3 /\ z=x+2;  
z:=x+y+2*z;  
12: x+y=3 /\ z+y = x+z;
```

```
11:x+y=3 /\ z=x+2;  
z:=x+y+2*z;  
12:x+y=3 /\ z+y = x+z;
```

Listing 4 – slide1.c

```
#include <limits.h>  
/*@  
  requires  a >= 0 && b >= 0 && c >= 0 && a+b == 3 && c == a + 2;  
  assigns \nothing;  
*/  
void slide1(int a, int b, int c) {  
  int x = a;  
  int y = b;  
  int z = c;  
  //@ assert x + y == 3 && z == x + 2;  
  z = x + y + 2*z;  
  //@ assert x + y == 3 && z+y == x+z;  
}
```


Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

- ▶ Propriétés fonctionnelles : pré et post

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ 7/98 ↺ 🔍 ↻

- ▶ Propriétés fonctionnelles : pré et post
- ▶ Propriétés de sûreté :
 - accès à une adresse valide
 - pas d'overflow arithmétique

- ▶ Propriétés fonctionnelles : pré et post
- ▶ Propriétés de sûreté :
 - accès à une adresse valide
 - pas d'overflow arithmétique
 - pas de division par zéro.
 - ...

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ 7/98 ↺ 🔍 ↻

Current Subsection Summary

1 Prolégomènes

2 Analyse de programmes

Exemple 1

Exemple 2

3 Observations et commentaires

Observations sur la vérification

Commentaires

Plugin WP

Logique de Hoare

4 Vérification d'annotations avec Frama-C

Introduction

Mise en œuvre avec Frama-C

Annotations

TOP Mardi 5 décembre

Validation des annotations (type HOARE)

Validation des annotations (type memory model)

5 TD du mercredi 23

6 Programmation par contrat

Définition de contrats

Exemples

7 Éléments du langage ACSL

Exemple 0 : compilation simple

Listing 5 – example0.c

```
// returns the absolute value of x
int abs ( int x ) {
    if ( x >= 0 ) return x ;
    return -x ; }

// frama-c -wp file.c
```

Exemple 1 : définition du contrat

Listing 6 – example1.c

```
// returns the absolute value of x
/*@ ensures (x >= 0 ==> \result == x);
    ensures (x < 0 ==> \result == -x);
*/

int abs ( int x ) {
  if ( x >= 0 ) return x ;
  return -x ; }
```

Exemple 1 : vérification du contrat

Listing 7 – example 1

```
macdome:lectures-malg mery$ frama-c -wp exemple1.c
[kernel] Parsing FRAMAC.SHARE/libc/_fc_builtin_for_normalization.i (no p
[kernel] Parsing exemple1.c (with preprocessing)
[wp] warning: Missing RTE guards
[wp] 2 goals scheduled
[wp] Proved goals:      2 / 2
      Qed:              2
macdome:lectures-malg mery$
```


Exemple 2 : modification de la précondition

- ▶ Si $x = \text{INT_MIN}$, alors $-x$ n'est pas codable.
- ▶ Ajouter une condition sur x .
- ▶ Ajouter la bibliothèque `#include <limits.h>`

Listing 9 – exemple 2

```
#include <limits.h>
/*@ requires x > INT_MIN;
    requires x <= INT_MAX;
    ensures (x >= 0 ==> \result == x);
    ensures (x < 0 ==> \result == -x);
*/

int abs ( int x ) {
  if ( x >= 0 ) return x ;
  return -x ; }
```


Current Subsection Summary

① Prolégomènes

② Analyse de programmes

Exemple 1

Exemple 2

③ Observations et commentaires

Observations sur la vérification

Commentaires

Plugin WP

Logique de Hoare

④ Vérification d'annotations avec Frama-C

Introduction

Mise en œuvre avec Frama-C

Annotations

TOP Mardi 5 décembre

Validation des annotations (type HOARE)

Validation des annotations (type memory model)

⑤ TD du mercredi 23

⑥ Programmation par contrat

Définition de contrats

Exemples

⑦ Éléments du langage ACSL

Listing 11 – exemple 2

```
/*@ requires x >= 0 ;  
   @*/  
int f(int x){  
    return x+1;  
}
```


Listing 13 – exemple 2

```
#include <limits.h>
/*@  requires x >= 0 ;
   @  requires x < INT_MAX ;
   @*/
int f(int x){
    return x+1;
}
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ 19/98 ↺ 🔍 ↻

```
1 Completely validated
1 Total
```

Function f

19/98

Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

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

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

requires $\text{pre}(x_0)$
ensures $\text{post}(x_0, x_f)$
variables X

```
begin  
  0 :  $P_0(x_0, x)$   
  instruction0  
  ...  
   $i : P_i(x_0, x)$   
  ...  
  instruction $f-1$   
   $f : P_f(x_0, x)$   
end
```

- ▶ $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $\text{pre}(x_0) \wedge P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$
- ▶ Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $x, x' \in \text{MEMORY}$
$$\left(\begin{array}{l} \left(\text{pre}(x_0) \wedge P_\ell(x_0, x) \right) \\ \wedge \text{cond}_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \end{array} \right) \Rightarrow P_{\ell'}(x_0, x')$$
- ▶ Pour toute paire d'étiquettes m, n telle que $m \longrightarrow n$, on vérifie que,
 $\forall x, x' \in \text{MEMORY} : \text{pre}(x_0) \wedge P_m(x_0, x) \Rightarrow \text{DOM}(m, n)(x)$

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

Exemple $\mathbf{DOM}(m, n)(x)$

$DOM(\ell_0, \ell_1)(u) = u \in \text{minint}.. \text{maxint} \wedge 5 \in \text{minint}.. \text{maxint} \wedge u+5 \in \text{minint}.. \text{maxint}$ où $\ell_0 : P_{\ell_0}(u)U := U+5; \ell_1 : P_{\ell_0}(u)$

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

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

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

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

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

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

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

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

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

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

- ▶ le langage de programmation est C.
- ▶ extension du langage de programmation par des annotations et la programmation par contrat.
- ▶ prise en charge du langage et de son annotation par des outils de preuve automatiques.

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

- ▶ Le plugin WP met en œuvre le calcul WP pour la correction partielle et la correction totale.
- ▶ Révision de la logique de Hoare avec le plugin qui suit les règles de composition du calcul WP et aussi des règles de Hoare.

- ▶ S est une instruction de STATS.
- ▶ T est le type ou les types des variables et D est la constante ou les constantes Définie(s).
- ▶ P est un prédicat du langage Pred
- ▶ X est une variable de programme
- ▶ $E(X, D)$ (resp. $B(X, D)$) est une expression arithmétique (resp. booléenne) dépendant de X et de D .
- ▶ x est la valeur de X (X contient la valeur x).
- ▶ $e(x, d)$ (resp. $b(x, d)$) est l'expression arithmétique (resp. booléenne) du langage Pred associée à l'expression $E(X, D)$ (resp. $B(X, D)$) du langage des expressions arithmétiques (resp. booléennes) du langage de programmation Prog
- ▶ $b(x, d)$ est l'expression arithmétique du langage Pred associée à l'expression $E(X, D)$ du langage des expressions arithmétiques du langage de programmation Prog

Définition structurelle des transformateurs de prédicats

S	$wp(S)(P)$
$X := E(X, D)$	$P[e(x, d)/x]$
SKIP	P
$S_1; S_2$	$wp(S_1)(wp(S_2)(P))$
IF B S_1 ELSE S_2 FI	$(B \Rightarrow wp(S_1)(P)) \wedge (\neg B \Rightarrow wp(S_2)(P))$
WHILE B DO S OD	$\mu.(\lambda X. (B \Rightarrow wp(S)(X)) \wedge (\neg B \Rightarrow P))$

- ▶ $wp(X := X+5)(x \geq 8) \stackrel{def}{=} x+5 \geq 8 \approx x \geq 3$
- ▶ $wp(\text{WHILE } x > 1 \text{ DO } X := X+1 \text{ OD})(x = 4) = FALSE$
- ▶ $wp(\text{WHILE } x > 1 \text{ DO } X := X+1 \text{ OD})(x = 0) = x = 0$

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

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

requires $\text{pre}(x_0)$

ensures $\text{post}(x_0, x_f)$

variables X

begin

$0 : P_0(x_0, x)$

instruction₀

...

$i : P_i(x_0, x)$

...

instruction _{$f-1$}

$f : P_f(x_0, x)$

end

▶ $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$

▶ $P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$

▶ conditions de vérification pour toutes les paires $\ell \longrightarrow \ell'$

- ▶ $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- ▶ Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $x, x' \in \text{MEMORY}$
$$\left(\begin{array}{l} \left(pre(x_0) \wedge P_\ell(x_0, x) \right) \\ \wedge cond_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \end{array} \right) \Rightarrow P_{\ell'}(x_0, x')$$
- ▶ Pour toute paire d'étiquettes m, n telle que $m \longrightarrow n$, on vérifie que,
 $\forall x, x' \in \text{MEMORY} : pre(x_0) \wedge P_m(x_0, x) \Rightarrow \mathbf{DOM}(m, n)(x)$
- ▶ Pour toute instruction S et pour toute assertion Q, $\{WP(S)(Q)\}S\{Q\}$ est vérifiée par définition de $WP(S)(Q)$.
- ▶ Pour toute instruction S et pour toutes assertions P et Q, $\{P\}S\{Q\}$ est équivalent à $P \Rightarrow WP(S)(Q)$ vérifiée par définition de la sémantique de S.
- ▶ $pre(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow post(x_0, x_f)$ est équivalent à $pre(x_0) \wedge x = x_0 \Rightarrow WP(P)(post(x_0, x))$
- ▶ $\{pre(x_0) \wedge x = x_0\}P\{post(x_0, x)\}$

Listing 15 – slide5.c

```
// #include <limits.h>

int slide5(int a) {
    int i = a;
    //@ assert i == a;
    i++;
    //@ assert i == a+1;
    return i;
}
```

- ① $i == a \Rightarrow (i == a)$ (initialisation)
- ② $i == a \Rightarrow (i == a+1)[i \mapsto i+1]$ (assignment)



% frama-c slide5.c -wp -report

Listing 16 – slide4.c

```
#include <limits.h>
/*@
  requires  a >= 0;
  requires  (a+1)*(a+1) <= INT_MAX;
  assigns \nothing;
  ensures \result == (a+1)*(a+1);
*/
int slide4(int a) {
  int i = a;
  //@ assert i == a;
  i++;
  //@ assert i == a+1;
  i=i+a;
  //@ assert i == 2*a+1;
  i=i+a*a;
  //@ assert i == (a+1)*(a+1);
  return i;
}
```

Listing 17 – slide2.c

```
/*@
  requires  a >= 0  && a < 30;
  ensures  \result == 30;
*/
int slide2(int a) {
    int i = a;
    //@ assert i == a;
    /*@ loop invariant 0 <= i && i <= 30;
        loop assigns i;
        */

    while (i < 30) {
        //@ assert 0 <= i < 30;
        i++;
        //@ assert 0 <= i <= 30;
    };
    //@ assert i == 30;
    return i;
}
```

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

☒ **Definition**(Axiomes et règles d'inférence)

- ▶ Axiome d'affectation : $\{P(e/x)\} \mathbf{X} := \mathbf{E(X)} \{P\}$.
- ▶ Axiome du saut : $\{P\} \mathbf{skip} \{P\}$.
- ▶ Règle de composition : Si $\{P\} \mathbf{S_1} \{R\}$ et $\{R\} \mathbf{S_2} \{Q\}$, alors $\{P\} \mathbf{S_1 ; S_2} \{Q\}$.
- ▶ Si $\{P \wedge B\} \mathbf{S_1} \{Q\}$ et $\{P \wedge \neg B\} \mathbf{S_2} \{Q\}$, alors $\{P\} \mathbf{if\ B\ then\ S_1\ then\ S_2\ fi} \{Q\}$.
- ▶ Si $\{P \wedge B\} \mathbf{S} \{P\}$, alors $\{P\} \mathbf{while\ B\ do\ S\ od} \{P \wedge \neg B\}$.
- ▶ Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $\{P\} \mathbf{S} \{Q\}$, $Q \Rightarrow Q'$, alors $\{P'\} \mathbf{S} \{Q'\}$.

Exemple de preuve $\text{contrat}\{x = 1\} \mathbf{Z} := \mathbf{X}; \mathbf{X} := \mathbf{Y}; \mathbf{Y} := \mathbf{Z} \{y = 1\}$

- ▶ (1) $x = 1 \Rightarrow (z = 1)[x/z]$ (propriété logique)
- ▶ (2) $\text{contrat}\{(z = 1)[x/z]\} \mathbf{Z} := \mathbf{X} \{z = 1\}$ (axiome d'affectation)
- ▶ (3) $\text{contrat}\{x = 1\} \mathbf{Z} := \mathbf{X} \{z = 1\}$ (Règle de renforcement/affaiblissement avec (1) et (2))
- ▶ (4) $z = 1 \Rightarrow (z = 1)[y/x]$ (propriété logique)
- ▶ (5) $\text{contrat}\{(z = 1)[y/x]\} \mathbf{X} := \mathbf{Y} \{z = 1\}$ (axiome d'affectation)
- ▶ (6) $\text{contrat}\{z = 1\} \mathbf{X} := \mathbf{Y} \{z = 1\}$ (Règle de renforcement/affaiblissement avec (4) et (5))
- ▶ (7) $z = 1 \Rightarrow (y = 1)[z/y]$ (propriété logique)
- ▶ (8) $\text{contrat}\{(z = 1)[x/z]\} \mathbf{Y} := \mathbf{Z} \{y = 1\}$ (axiome d'affectation)
- ▶ (9) $\text{contrat}\{z = 1\} \mathbf{Y} := \mathbf{Z} \{y = 1\}$ (Règle de renforcement/affaiblissement avec (7) et (8))
- ▶ (10) $\text{contrat}\{x = 1\} \mathbf{Z} := \mathbf{X}; \mathbf{X} := \mathbf{Y}; \{z = 1\}$ (Règle de composition avec 3 et 6)
- ▶ (11) $\text{contrat}\{x = 1\} \mathbf{Z} := \mathbf{X}; \mathbf{X} := \mathbf{Y}; \mathbf{Y} := \mathbf{Z} \{y = 1\}$ (Règle de composition avec 11 et 9)

- ▶ $\text{contrat}\{P\}\mathbf{S}\{Q\}$
- ▶ $\forall s, t \in STATES : P(s) \wedge \mathcal{D}(S)(s) = t \Rightarrow Q(t)$
- ▶ $\forall s \in STATES : P(s) \Rightarrow (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$

.....

Définition de wlp

$$\text{wlp}(S)(Q) \stackrel{\text{def}}{=} (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$$

.....

$$\text{wlp}(S)(Q) \equiv (\exists t \in STATES : \mathcal{D}(S)(s) = t \wedge \overline{Q}(t))$$

.....

Lien entre wp et wlp

- ▶ $\text{loop}(S) \equiv \overline{(\exists t \in STATES : \mathcal{D}(S)(s) = t)}$ (ensemble des états qui ne permettent pas à S de terminer)
 - ▶ $\text{wp}(S)(Q) \equiv \text{wlp}(S)(Q) \wedge \overline{\text{loop}(S)}$
-

.....

☒ Definition

$$WLP(S)(P) = \nu \lambda X. ((B \wedge wlp(BS)(X)) \vee (\neg B \wedge P))$$

.....

.....

☺ Property

► Si $P \Rightarrow Q$, then $wlp(S)(P) \Rightarrow wlp(S)(Q)$.

.....

☒ Definition triplets de Hoare

$$\{P\}\mathbf{S}\{Q\} \stackrel{def}{=} P \Rightarrow wlp(S)(Q)$$

.....

.....
☒ Definition triplets de Hoare Correction Totale

$$[P]\mathbf{S}[Q] \stackrel{def}{=} P \Rightarrow wp(S)(Q)$$

.....

.....

☒ Definition triplets de Hoare Correction Totale

$$[P]\mathbf{S}[Q] \stackrel{def}{=} P \Rightarrow wp(S)(Q)$$

.....

.....

☒ Definition (Axiomes et règles d'inférence)

- ▶ Axiome d'affectation : $[P(e/x)]\mathbf{X} := \mathbf{E}(\mathbf{X})[P]$.
 - ▶ Axiome du saut : $[P]\mathbf{skip}[P]$.
 - ▶ Règle de composition : Si $[P]\mathbf{S}_1[R]$ et $[R]\mathbf{S}_2[Q]$, alors $[P]\mathbf{if\ B\ then\ S_1\ then\ S_2\ fi}[Q]$.
 - ▶ Si $[P \wedge B]\mathbf{S}_1[Q]$ et $[P \wedge \neg B]\mathbf{S}_2[Q]$, alors $[P]\mathbf{if\ B\ then\ S_1\ then\ S_2\ fi}[Q]$.
 - ▶ Si $[P(n+1)]\mathbf{S}[P(n)]$, $P(n+1) \Rightarrow b$, $P(0) \Rightarrow \neg b$, alors $[\exists n \in \mathbb{N}. P(n)]\mathbf{while\ B\ do\ S\ od}[P(0)]$.
 - ▶ Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $[P]\mathbf{S}[Q]$, $Q \Rightarrow Q'$, alors $[P']\mathbf{S}[Q']$.
-

Correction

:

Si $[P]\mathbf{S}[Q]$ est dérivé selon les règles ci-dessus, alors $P \wp(S) \wp Q$.

- ▶ $[P(e/x)]\mathbf{X} := \mathbf{E}(\mathbf{X})[P]$ est valide : $wp(X := E)(P)/x = P(e/x)$.
- ▶ $[\exists n \in \mathbb{N}. P(n)]\mathbf{while\ B\ do\ S\ od}[P(0)]$: si s est un état de $P(n)$ alors au bout de n boucles on atteint un état s_f tel que $P(0)$ est vrai en s_f .

Complétude

:

Si $P \Rightarrow wp(S)(Q)$, alors il existe une preuve de $[P]\mathbf{S}[Q]$ construites avec les règles ci-dessus,

- ▶ $P \Rightarrow wp(X := E(X))(Q) : P \Rightarrow Q(e/x)$ et $[Q(e/x)]\mathbf{X} := \mathbf{E}(\mathbf{X})[Q]$ constituent une preuve.
- ▶ $P \Rightarrow wp(\text{while})(Q) :$
 - On construit la suite de $P(n)$ en définissant $P(n) = W_n$.
 - On vérifie que cela vérifie la règle du while.

- ▶ Revoir les wp
- ▶ Faire le calcul pour l'affectation et la conditionnelle

Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

Listing 18 – swap de deux contenus

```
/*@  
    requires \valid(a) && \valid(b);  
    assigns *a, *b;  
    ensures *a == \old(*b);  
    ensures *b == \old(*a);  
*/  
static void swap(int* a, int* b) {  
    int temp;  
    temp = (*a);  
    (*a) = (*b);  
    (*b) = temp;  
}
```


Listing 20 – prepostframa3.cl

```
int main(int argc , char* argv []) {  
    int a = 21;  
    int b = 42;  
  
    swap(&a, &b);  
    //@ assert a == 42 && b == 21;  
  
    int c = difference(b, a);  
    //@ assert c == 22;  
  
    return 0;  
}
```

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

- ▶ Assertions à un point du programme :

```
/*@ assert pred; */
```

- ▶ Assertions à un point du programme selon les comportements.

```
/*@ for id1,id2, ..., idn: assert pred; */
```


Listing 21 – anno0.c

```
int main(void){
    signed long int x,y,z;
    x = 1;
    /*@ assert x == 1; */
    y = 2;
    /*@ assert x == 1 && y == 2; */
    z = x * y;
    /*@ assert x == 1 && y == 1 && z==2; */
    return 0;
}
```

Listing 22 – anno00.c

```
int main(void){
    signed long int x,y,z; //    int x,y,z;
    x = 1;
    /*@ assert x == 1;      */
    y = 2;
    /*@ assert x == 1 && y == 2;
    z = x *y;
    /*@ assert  x == 1 && y == 2 && z == 2;
    return 0;
}
```

```
/*@ loop invariant I;  
   @ loop assigns L;  
   @*/
```

Listing 23 – anno5.c

```
/*@ requires a >= 0 && b >= 0;  
   ensures 0 <= \result;  
   ensures \result < b;  
   ensures \exists integer k; a == k * b + \result;  
   */  
int rem(int a, int b) {  
    int r = a;  
    /*@  
       loop invariant  
       (\exists integer i; a == i * b + r) &&  
       r >= 0;  
       loop assigns r;  
    */  
    while (r >= b) { r = r - b; };  
    return r;  
}
```

Listing 24 – anno6.c

```
/*@ requires a >= 0 && b >= 0;  
    ensures 0 <= \result;  
    ensures \result < b;  
    ensures \exists integer k; a == k * b + \result;  
*/  
int rem(int a, int b) {  
    int r = a;  
    /*@  
        loop invariant  
        (\exists integer i; a == i * b + r) &&  
        r >= 0;  
        loop assigns r;  
    */  
    while (r >= b) { r = r - b; };  
    return r;  
}
```

Echec de la preuve

L'invariant est insuffisamment informatif pour être prouvé et il faut ajouter une information sur y.

```
frama-c -wp anno6.c
[kernel] Parsing anno6.c (with preprocessing)
[wp] Warning: Missing RTE guards
[wp] anno6.c:8: Warning: Missing assigns clause (assigns 'everything' i
[wp] 2 goals scheduled
[wp] [Alt-Ergo 2.3.3] Goal typed_f_loop_invariant_preserved : Timeout (
[wp] [Cache] found:1
[wp] Proved goals:      1 / 2
    Qed:                1 (0.57ms)
    Alt-Ergo 2.3.3:      0 (interrupted: 1) (cached: 1)
[wp:pedantic-assigns] anno6.c:1: Warning:
    No 'assigns' specification for function 'f'.
    Callers assumptions might be imprecise.
```

Analyse avec succès

L'invariant est plus précis et donne des conditions liant x et y .

Listing 25 – anno7.c

```
int f() {  
  int x = 0;  
  int y = 10;  
  /*@  
    loop invariant  
    0 <= x < 11 && x+y == 10;  
  */  
  while (y > 0) {  
    x++;  
    y--;  
  }  
  return 0;  
}
```

```
frama-c -wp anno7.c
[kernel] Parsing anno7.c (with preprocessing)
[wp] Warning: Missing RTE guards
[wp] anno7.c:8: Warning: Missing assigns clause (assigns 'everything' i
[wp] 2 goals scheduled
[wp] [Cache] found:1
[wp] Proved goals:      2 / 2
    Qed:                  1 (0.32ms-3ms)
    Alt-Ergo 2.3.3:      1 (6ms) (8) (cached: 1)
[wp:pedantic-assigns] anno7.c:1: Warning:
    No 'assigns' specification for function 'f'.
    Callers assumptions might be imprecise.
```

- ▶ Un variant est une quantité qui décroît au cours de la boucle.
- ▶ Deux possibilités d'analyse sont possibles :
 - Terminaison d'une boucle (variant)
 - Terminaison de l'appel d'une fonction récursive (decrease)

Listing 26 – variant2.c

```
//@ loop variant e;
//@ decreases e;
```

- ▶ La terminaison est assurée en montrant que chaque boucle termine.
- ▶ Une boucle est caractérisée par une expression `expvariant(x)` appelée `variant` qui doit décroître à chaque exécution du corps de la boucle `S` où x_1 et x_2 sont les valeurs de X respectivement au début de la boucle `S` et à la fin de `S` :

$$\forall x_1, x_2. b(x_1) \wedge x_1 \xrightarrow{S} x_2 \Rightarrow \text{expvariant}(x_1) > \text{expvariant}(x_2)$$

Listing 27 – variant1.c

```
/*@ requires n > 0;
    ensures \result == 0;
*/
int code(int n) {
    int x = n;
    /*@ loop invariant x >= 0 && x <= n;
        loop assigns x;
        loop variant x;
    */
    while (x != 0) {
        x = x - 1;
    };
    return x;
}
```


Listing 28 – variant3.c

```

int f() {
int x = 0;
int y = 10;
/*@
    loop invariant
    0 <= x < 11 && x+y == 10;
    loop variant y;
*/
while (y > 0) {
    x++;
    y--;
}
return 0;
}

```

Listing 29 – variant4.c

```
/*@ requires n <= 12;
   @ decreases n;
   @*/
int fact(int n){
    if (n <= 1) return 1;
    return n*fact(n-1);
}
```

- ▶ Pas de gestion de la mémoire comme les pointeurs
- ▶ Affectation à chaque variable une variable logique
- ▶ $x++$ avec x de type `int` et la C-variable est affectée à deux L-variables $x2 = x1 + 1$.

Listing 30 – wp2.c

```
/*@CONSOLE
#include <LIMITS.h>
int q1() {
    int x=10,y=30,z=20;
    //@ assert x== 10 && y == z+x && z==2*x;
    y= z+x;
    //@ assert x== 10 && y == x+2*10;
    x = x+1;
    //@ assert x-1== 10 && y == x-1+2*10;
    return (0);
}
```

Listing 31 – wp3.c

```
int q1() {
    int c = 2 ;
    /*@ assert c == 2; */
    int x;
    /*@ assert c == 2; */
    x = 3 * c ;
    /*@ assert x == 6; */
    return (0);
}
```

Listing 32 – wp4.c

```
int main()
{
    int a = 42; int b = 37;
    int c = a+b; // i:1
    //@assert b == 37 ;
    a -= c; // i:2
    b += a; // i:3
    //@assert b == 0 && c == 79;
    return(0);
}
```

Listing 33 – wp5.c

```
int main()
{
    int z; // instruction 8
    int a = 4; // instruction 7
    //@assert a == 4 ;
    int b = 3; // instyruction 6
    //@assert b == 3 && a == 4;
    int c = a+b; // instruction 4
    /*@ assert b == 3 && c == 7 && a == 4 ; */
    a += c; // instruction 3
    b += a; // instruction 2
    //@ assert a == 11 && b == 14 && c == 7 ;
    //@ assert a +b == 25 ;
    z = a*b; // instruction 1
    //@assert a == 11 && b == 14 && c == 7 && z == 154;
    return(0);
}
```

Listing 34 – wp6.c

```
int main()
{
    int a = 4;
    int b = 3;
    int c = a+b; // i:1
    a += c; // i:2
    b += a; // i:3
    //@assert a == 11 && b == 14 && c == 7 ;
    return (0);
}
```

Listing 35 – wp7.c

```
/*@ ensures x == a;  
    ensures y == b;  
    */  
void swap1(int a, int b) {  
    int x = a;  
    int y = b;  
    //@ assert x == a && y == b;  
    int tmp;  
    tmp = x;  
    x = y;  
    y = tmp;  
    //@ assert x == a && y == a;  
}  
  
void swap2(int a, int b) {  
    int x = a;  
    int y = b;  
    //@ assert x == a && y == b;  
    x = x + y;  
    y = x - y;  
    x = x - y;  
    //@ assert x == b && y == a;  
}  
  
/*@ requires \valid(a);  
    requires \valid(b);  
    ensures *a == \old(*b);  
    ensures *b == \old(*a);  
    */  
void swap3(int *a, int *b) {  
    int tmp;  
    tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

Listing 36 – wp8.c

```
int main()
{
    int x = -1;
    int *p;
    //@assert x == -1;
    p = &x ;
    x = x+1;
    //@assert x >= 0 && *p >= 0 ;
    return (0);
}
```


Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

- ▶ Revoir la notion de contrat
- ▶ Revoir la notion d'invariant de boucle.
- ▶ Gestion des labels
- ▶ notion de théorie

Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

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

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

requires $\text{pre}(x_0)$
ensures $\text{post}(x_0, x_f)$
variables X

```
begin  
  0 :  $P_0(x_0, x)$   
  instruction0  
  ...  
  i :  $P_i(x_0, x)$   
  ...  
  instructionf-1  
  f :  $P_f(x_0, x)$   
end
```

- ▶ $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $\text{pre}(x_0) \wedge P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$
- ▶ Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $x, x' \in \text{MEMORY}$
$$\left(\begin{array}{l} \left(\text{pre}(x_0) \wedge P_\ell(x_0, x) \right) \\ \wedge \text{cond}_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \end{array} \right) \Rightarrow P_{\ell'}(x_0, x')$$
- ▶ Pour toute paire d'étiquettes m, n telle que $m \longrightarrow n$, on vérifie que,
 $\forall x, x' \in \text{MEMORY} : \text{pre}(x_0) \wedge P_m(x_0, x) \Rightarrow \text{DOM}(m, n)(x)$

Listing 37 – factoriel

```
/*@ axiomatic mathfact {  
  @ logic integer mathfact(integer n);  
  @ axiom mathfact.1: mathfact(1) == 1;  
  @ axiom mathfact.rec: \forall integer n; n > 1  
    ==> mathfact(n) == n * mathfact(n-1);  
  @ } */  
  
/*@ requires n > 0;  
   ensures \result == mathfact(n);  
*/  
int codefact(int n) {  
  //@ assert n >= 1 && n <= n &&   mathfact(n) == 1 * mathfact(n);  
  int y = 1;  
  //@ assert n >= 1 && n <= n &&   mathfact(n) == y * mathfact(n);  
  int x = n;  
  //@ assert x >= 1 && x <= n &&   mathfact(n) == y * mathfact(x);  
  
  //@ loop invariant x >= 1 &&  
    mathfact(n) == y * mathfact(x);  
  loop assigns x, y;  
  loop variant x-1;  
  */  
  while (x != 1) {  
    //@ assert mathfact(n) == y * mathfact(x) && x > 1;  
    y = y * x;  
    x = x - 1;  
  };  
  //@ assert x >= 1 && x <= n &&   mathfact(n) == y * mathfact(x) && x == 1;  
  //@ assert y == mathfact(n);  
  return y;  
  //@ @ assert \result == y && y == mathfact(n);  
}
```

Définition du contrat et des axiomes

```
/*@ axiomatic mathfact {  
  @ logic integer mathfact(integer n);  
  @ axiom mathfact_1: mathfact(1) == 1;  
  @ axiom mathfact_rec: \forall integer n; n > 1  
    ==> mathfact(n) == n * mathfact(n-1);  
  @ } */
```

- ▶ La spécification d'une fonction à calculer nécessite de la définir mathématiquement.
- ▶ Cette définition axiomatique est fondée sur une définition inductive de la fonction `mfact` qui sera utilisée dans les assertions pour le contrat définissant la fonction informatique de calcul.

```
/*@ requires n > 0;  
   ensures \result == mathfact(n);  
  */  
int codefact(int n) {  
  int y = 1;  
  int x = n;  
  /*@ loop invariant x >= 1 &&  
    mathfact(n) == y * mathfact(x);  
    loop assigns x, y;  
  */  
  while (x != 1) {  
    y = y * x;  
    x = x - 1;  
  };  
  return y;  
}
```


Listing 39 – schema de contrat

```
/*@ requires P1 && ... && Pn;  
   @ assigns L1, ..., Lm;  
   @ ensures E1 && ... && Ep;  
   @*/
```

- ▶ $\backslash old(x)$ fait référence à la valeur de x à l'appel.
- ▶ $\backslash result$ fait référence à la valeur du résultat de l'appel.

Current Subsection Summary

1 Prolégomènes

② Analyse de programmes

Example 1

Exemple 2

3 Observations et commentaires

Observations sur la vérification

Commentaires

Plugin WP

Logique de Hoare

Introduction

Mise en œuvre avec Framac

Annotations

TOP Mardi 5 décembre

Validation des annotations (type HOARE)

Validation des annotations (type memory model)

⑤ TD du mercredi 23

⑥ Programmation par contrat

Définition de contrats

Examples

7 Éléments du langage ACSL

- ▶ Précondition $x \geq 0$
- ▶ Postcondition $result \cdot result \leq x < (result+1) \cdot (result+1)$

Listing 40 – contrat squareroot

```
/*@ requires x >= 0;  
   @ ensures \result >= 0;  
   @ ensures \result * \result <= x;  
   @ ensures (\result+1) * (\result + 1) > x;  
   @*/  
int squareroot(int x);
```

- ▶ Précondition $def(p)$
- ▶ Postcondition $\cdot p = \underline{\cdot p} + 1$

Listing 41 – contrat increment

```
/*@ requires \valid(p);  
   @ assigns *p;  
   @ ensures *p == \old(*p) + 1;  
   @*/  
void increment(int *p);
```

Listing 42 – fschema3.c

```
/*@ requires P;  
  @ behavior B1;  
  @ assumes A1;  
  @ requires R1;  
  @ assigns L1;  
  @ ensures E1;  
  @ behavior B2;  
  @ assumes A2;  
  @ requires R2;  
  @ assigns L2;  
  @ ensures E2;  
  @*/
```

- ▶ La fonction appelante doit garantir que $P \wedge (A1 \Rightarrow R1) \wedge (A2 \Rightarrow R2)$ est vraie à l'appel.
- ▶ La fonction appelante renvoie un état satisfaisant le prédicat $\text{old}(A1) \Rightarrow E1$ et $\text{old}(A2) \Rightarrow E2$
- ▶ Les variables qui ne figurent pas dans l'ensemble $L1 \cup \dots \cup Lp$ ne sont pas modifiées.

Listing 43 – contrat3.c

```
/*@ behavior change-p:
   @   assumes n > 0;
   @   requires \valid(p);
   @   assigns *p;
   @   ensures *p == n;
   @ behavior change-q:
   @   assumes n <= 0;
   @   requires \valid(q);
   @   assigns *q;
   @   ensures *q == n;
   @*/
void f(int n, int *p, int *q) {
    if (n > 0) *p = n; else *q = n;
}
```

Listing 44 – contrat1.c

```
/*@ requires x >= 0;  
@ ensures \result >= 0;  
@ ensures \result * \result <= x;  
@ ensures x < (\result + 1) * (\result + 1); @*/  
int squareroot(int x);
```


Listing 45 – contrat2.c

```
/*@ requires \valid(p);
   @ assigns *p;
   @ ensures *p == \old(*p) + 1;
   @*/
void increment(int *p);
```

Listing 46 – contrat3.c

```

/*@ behavior change-p:
@ assumes n > 0;
@ requires \valid(p);
@ assigns *p;
@ ensures *p == n;
@ behavior change-q:
@ assumes n <= 0;
@ requires \valid(q);
@ assigns *q;
@ ensures *q == n;
@*/
void f(int n, int *p, int *q) {
    if (n > 0) *p = n; else *q = n;
}

```

Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

Current Subsection Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

- ▶ Une variable dite *ghost* permet de désigner de manière cachée ou masquée une valeur calculée et utile pour exprimer une propriété.
- ▶ Elle ne doit pas changer la sémantique des autres variables et on ne modifie pas le code dans les instructions ghost.

Listing 47 – ghost2.c

```
int f (int x, int y) {  
    //@ghost int z=x+y;  
    switch (x) {  
    case 0: return y;  
    //@ ghost case 1: z=y;  
    // above statement is correct.  
    //@ ghost case 2: { z++; break; }  
    // invalid, would bypass the non-ghost default  
    default: y++; }  
    return y; }  
  
int g(int x) { //@ ghost int z=x;  
    if (x>0){return x;}  
    //@ ghost else { z++; return x; }  
    // invalid, would bypass the non-ghost return  
    return x+1; }
```

Listing 48 – ghost1.c

```
/*@ requires a >= 0 && b > 0;
    ensures 0 <= \result;
    ensures \result < b;
    ensures \exists integer k; a == k * b + \result;
*/
int rem(int a, int b) {
    int r = a;
    /*@ ghost    int q=0;
    */
    /*@
        loop invariant
        a == q * b + r &&
        r >= 0 && r <= a
        ;
        loop assigns r;
        loop assigns q;
    */
    while (r >= b) {
        r = r - b;
    /*@ ghost
        q = q+1;
    */
    };
    return r;
}
```

Current Subsection Summary

⑦ Éléments du langage ACSL

Le langage de spécification pour Frama-C
ACSL (18 novembre 2024) (Dominique Méry)

- ▶ $\backslash old(x)$ désigne la valeur de la variable x au moment de l'appel de la fonction.
- ▶ Cette expression est utilisable dans la postcondition *ensures*

Listing 49 – old1.c

```
/*@  
  requires a >= 0 && b >= 0;  
  ensures  a == \old(a)+2;  
  ensures  b == \old(b)+\old(a)+2;  
*/  
int old(int a, int b) {  
  a = a + 1;  
  a = a + 1;  
  b = b + a;  
  return 0;  
}
```

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ 95/98 ↺ 🔍 ↻

Listing 50 – prevaleur

```
/*@  
  requires a >= 0 && b >= 0;  
  assigns \nothing;  
  ensures \result == \old(a)+2;  
*/  
int at(int a, int b) {  
  a = a +1;  
  //@ assert a == \at(a, Pre)+1;  
  a = a +1;  
  //@ assert a == \at(a, Pre)+2;  
  b = b +a;  
  //@ assert a == \at(a, Pre)+2;  
  return a ;  
}
```


Listing 51 – prevaleur

```
void f (int n) {  
  for (int i = 0; i < n; i++) {  
    /*@ assert  $\backslash at(i, LoopEntry) = 0;$  */  
    int j=0;  
    while (j++ < i) {  
      /*@ assert  $\backslash at(j, LoopEntry) = 0;$  */  
      /*@ assert  $\backslash at(j, LoopCurrent) + 1 = j;$  */  
    } }  
}
```

Current Summary

- ① Prolégomènes
- ② Analyse de programmes
 - Exemple 1
 - Exemple 2
- ③ Observations et commentaires
 - Observations sur la vérification
 - Commentaires
 - Plugin WP
 - Logique de Hoare
- ④ Vérification d'annotations avec Frama-C
 - Introduction
 - Mise en œuvre avec Frama-C
 - Annotations
 - TOP Mardi 5 décembre
 - Validation des annotations (type HOARE)
 - Validation des annotations (type memory model)
- ⑤ TD du mercredi 23
- ⑥ Programmation par contrat
 - Définition de contrats
 - Exemples
- ⑦ Éléments du langage ACSL

