université
PARIS-SACLAY

UE14 · Vérification déductive de programmes

Frama-C • ACSL • WP

Master Informatique · FIIL · 2017 / 2018

Préliminaires

- Séances 15/11, 30/11, 7/12 : Andrei Paskevich (LRI)
- Séances 14/12, 21/12, 11/01 : Julien Signoles (CEA)
- Séance 18/01 : Andrei + Julien (TP noté)
- Examen : 1/02
- Évaluation :
  - 2 interrogations écrites, la 1ère dans quelques minutes
  - 1 TP noté
  - examen terminal
  - note finale : $\frac{1}{3}\text{CC} + \frac{2}{3}\text{ET}$
- Travaux pratiques (2ème partie) : outil Frama-C
  - `http://frama-c.com/` — site du projet
  - `http://why3.lri.fr/fiil-2017/` — les TPs pour ce cours (y compris Frama-C)
- Mon mail en cas de questions : `julien.signoles@cea.fr`

1. Frama-C

Contexte

Software is hard. — DONALD KNUTH

Contexte

Software is hard. — DONALD KNUTH

C is quirky, flawed, and an enormous success. — DENNIS M. RITCHIE

Contexte

Software is hard. — DONALD KNUTH

C is quirky, flawed, and an enormous success. — DENNIS M. RITCHIE

- le langage C est bizarre et imparfait
- le langage C est vieux (1972)
- le langage C est compliqué (ISO C 99 = 550 pages)

Contexte

Software is hard. — DONALD KNUTH

C is quirky, flawed, and an enormous success. — DENNIS M. RITCHIE

- le langage C est bizarre et imparfait
- le langage C est vieux (1972)
- le langage C est compliqué (ISO C 99 = 550 pages)

  ... mais
- beaucoup de systèmes industriels restent programmés en C
- très peu sont corrects

  ... alors que
- certaines erreurs sont dramatiques (voir cours 1)

Nécessaire de vérifier la correction des programmes C

Frama-C : introduction

- outil développé depuis 2004 au CEA LIST, en collaboration avec Inria
- plateforme d'analyse de codes sources écrits en ISO C 99

Frama-C : introduction

- outil développé depuis 2004 au CEA LIST, en collaboration avec Inria
- plateforme d'analyse de codes sources écrits en ISO C 99
- support de ACSL
  - langage de spécification formelle
  - programmes C annotés en ACSL

Frama-C : introduction

- outil développé depuis 2004 au CEA LIST, en collaboration avec Inria
- plateforme d'analyse de codes sources écrits en ISO C 99
- support de ACSL
  - langage de spécification formelle
  - programmes C annotés en ACSL
- plusieurs outils en 1 seul (« couteau suisse »)
- utilisations académiques et industrielles

`http://frama-c.com`

Frama-C : plateforme extensible et collaborative

- architecture à greffons (comme Eclipse ou Gimp)
- greffon = analyseur ou transformateur de code
- noyau de Frama-C
  - services généraux (e.g. options communes)
  - synthétise des informations (e.g. quelles propriétés sont prouvées ?)

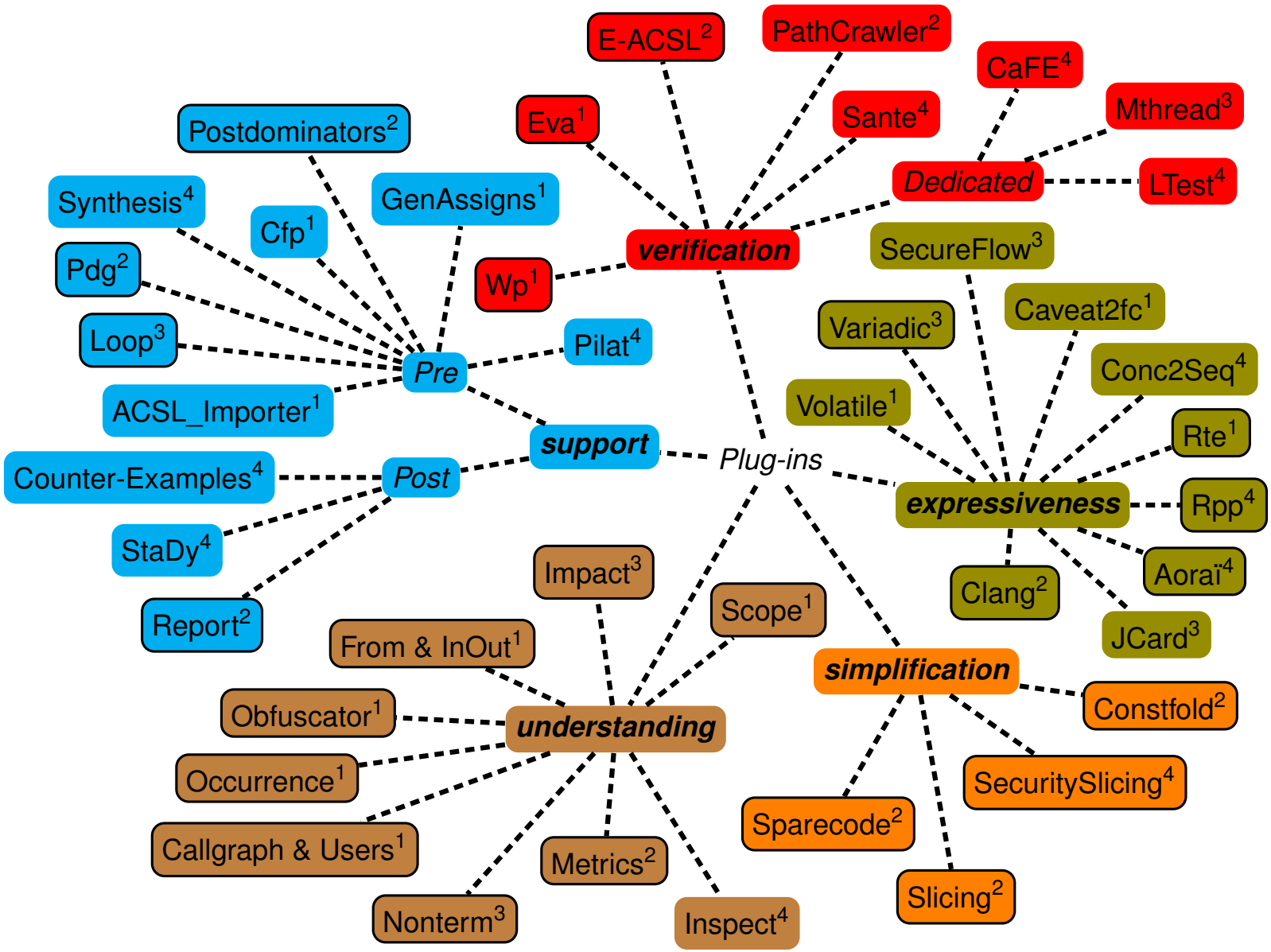
Frama-C : plateforme extensible et collaborative

- architecture à greffons (comme Eclipse ou Gimp)
- greffon = analyseur ou transformateur de code
- noyau de Frama-C
  - services généraux (e.g. options communes)
  - synthétise des informations (e.g. quelles propriétés sont prouvées ?)
- combinaison d'analyses
  - réutilisation des résultats
  - échange de propriétés ACSL à vérifier
  - ACSL = *langua franca* entre analyseurs

Frama-C : plateforme extensible et collaborative

- architecture à greffons (comme Eclipse ou Gimp)
- greffon = analyseur ou transformateur de code
- noyau de Frama-C
  - services généraux (e.g. options communes)
  - synthétise des informations (e.g. quelles propriétés sont prouvées ?)
- combinaison d'analyses
  - réutilisation des résultats
  - échange de propriétés ACSL à vérifier
  - ACSL = *langua franca* entre analyseurs
- développement possible de nouveaux greffons par des tiers
  - nouveaux analyseurs
  - adaptation/combinaison d'analyseurs existants

Frama-C : quelques greffons...



Frama-C : prise en main

Pour la version installée en TP (aka Phosphorus)...

- `frama-c -help` : greffons disponibles
  - WP : vérification de programmes fondée sur la logique de Hoare
  - Rte : génère des annotations ACSL pour chaque erreur à l'exécution potentielle

Frama-C : prise en main

Pour la version installée en TP (aka Phosphorus)...

- `frama-c -help` : greffons disponibles
  - WP : vérification de programmes fondée sur la logique de Hoare
  - Rte : génère des annotations ACSL pour chaque erreur à l'exécution potentielle
- `frama-c -kernel-help` : options du noyau

Frama-C : prise en main

Pour la version installée en TP (aka Phosphorus)...

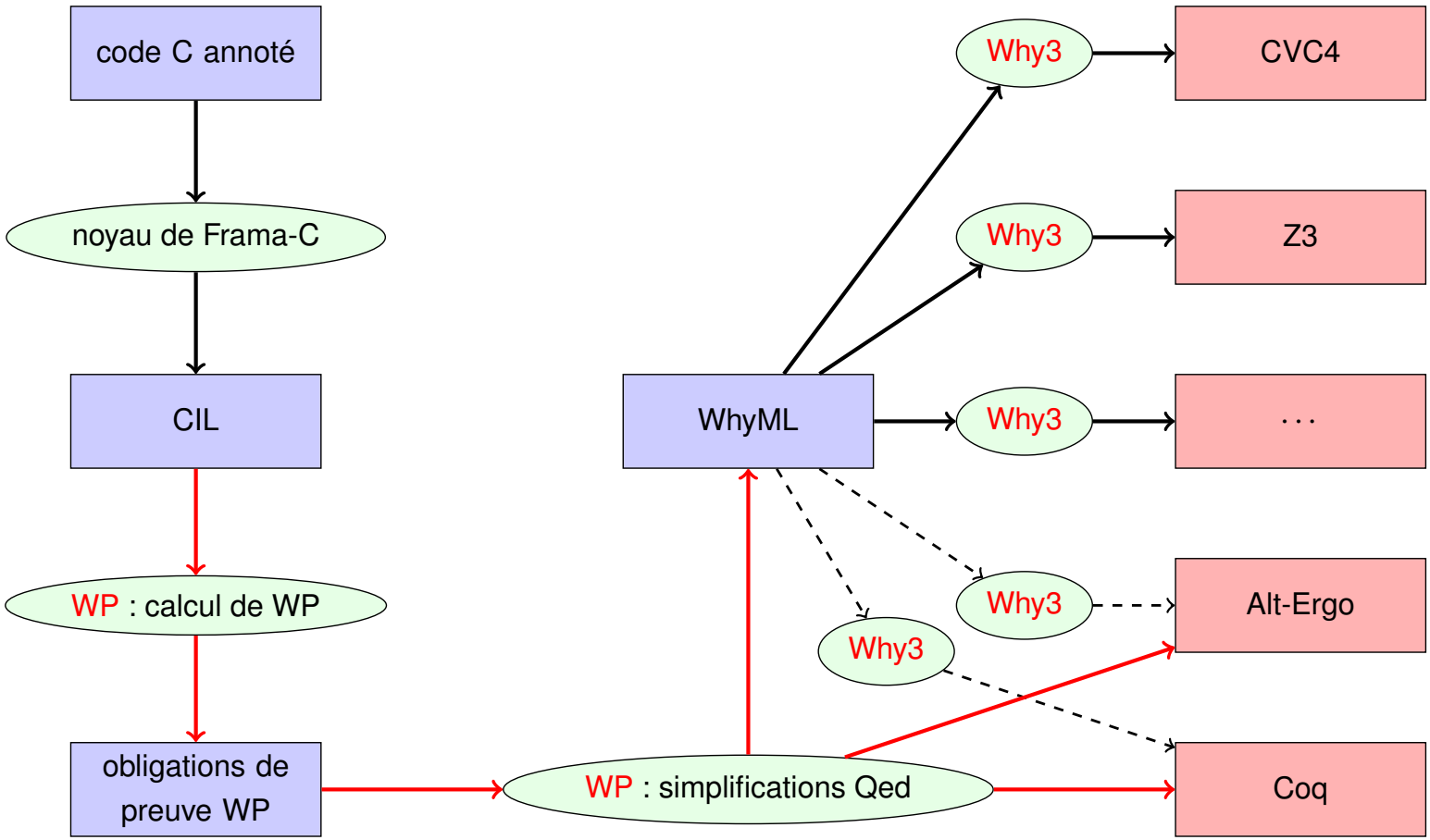
- `frama-c -help` : greffons disponibles
  - WP : vérification de programmes fondée sur la logique de Hoare
  - Rte : génère des annotations ACSL pour chaque erreur à l'exécution potentielle
- `frama-c -kernel-help` : options du noyau
- `frama-c -wp-help`
  - `-wp` : vérifie tout le programme
  - `-wp-fct <f, ...>` : vérifie certaines fonctions seulement
  - `-wp-rte` : lance le greffon RTE avant le WP
  - `-wp-timeout <n>` : temps limite pour les prouveurs

Frama-C : prise en main

Pour la version installée en TP (aka Phosphorus)...

- `frama-c -help` : greffons disponibles
  - WP : vérification de programmes fondée sur la logique de Hoare
  - Rte : génère des annotations ACSL pour chaque erreur à l'exécution potentielle
- `frama-c -kernel-help` : options du noyau
- `frama-c -wp-help`
  - `-wp` : vérifie tout le programme
  - `-wp-fct <f, ...>` : vérifie certaines fonctions seulement
  - `-wp-rte` : lance le greffon RTE avant le WP
  - `-wp-timeout <n>` : temps limite pour les prouveurs
- `frama-c-gui` : GUI pour Frama-C (mêmes options)
- exemples :
  - $ frama-c foo.c -wp -wp-fct my_function
  - $ frama-c-gui -wp foo.c

WP : principe de fonctionnement



WP : usages industriels

- systèmes avioniques
  - programmes A380, A350 et A400M
  - vérification de propriétés fonctionnelles
  - remplacement des tests unitaires par de la preuve unitaire
  - qualification et certification pour DO178 B/C, niveau A

WP : usages industriels

- systèmes avioniques
  - programmes A380, A350 et A400M
  - vérification de propriétés fonctionnelles
  - remplacement des tests unitaires par de la preuve unitaire
  - qualification et certification pour DO178 B/C, niveau A
- système SCADA (Supervisory Control And Data Acquisition)
  - vérification de propriétés temporelles
  - 100+ kloc de C
  - justification et certification pour norme IEC60880, classe 1
  - en complément de l'interprétation abstraite

2. ACSL

Introduction

- ACSL = langage de spécifications formelles pour le C
- Langages similaires :
  - JML pour Java
  - Spec# pour C#
  - Spark2014 pour Ada
- fondé sur la notion de contrats (cours 2), héritée de Eiffel

Introduction

- ACSL = langage de spécifications formelles pour le C
- Langages similaires :
  - JML pour Java
  - Spec# pour C#
  - Spark2014 pour Ada
- fondé sur la notion de contrats (cours 2), héritée de Eiffel
- annotations ACSL ajoutées au programme C par des commentaires dédiés
  `/*@ ... */` ou `//@ ...`
- ```
  /*@ ensures \result == x; */
  int identity(int x) { return x; }
  ```
  ```

Clauses, termes et prédicats

- clauses
  - assertion : /*@ assert x == 0 ; */
  - postcondition : /*@ ensures \result == x ; */
  - ...
- mots-clés commencent par un backslash '\'

Clauses, termes et prédicats

- clauses
  - assertion : /*@ assert x == 0 ; */
  - postcondition : /*@ ensures \result == x ; */
  - ...
- mots-clés commencent par un backslash '\'
- logique du premier ordre
- termes
  - expressions C pures, i.e. sans effet de bord : `(int)x->a`
  - mots-clés : \result, ...
  - fonctions logiques : \sqrt, ...
- prédicats
  - relations : ==, <=, ...
  - connecteurs : &&, ||, ==>, <==>, ...
  - quantifications : `\forall integer x,y; \exists integer z; x+y==z`
  - prédicats prédéfinis : \valid(p), ...

Typage

- ACSL est un langage typé
- types du langage C
- `boolean` : `\true` et `\false`
- `integer` : $\mathbb{Z}$, entiers mathématiques (précision arbitraire)
- `real` : $\mathbb{R}$, réels mathématiques
- les opérations arithmétiques sont dans $\mathbb{Z}$ ou $\mathbb{R}$
  - `3 / 2 == 1`
  - `3. / 2. == 1.5`
  - `\sqrt(4.1) >= 2.01`
- sous-typage (avec coercions implicites) :
  - types C integraux (*integral C types*) sont sous-types de `integer` : x + 1 avec `x:int`
  - `integer` est sous-type de `real` : `3. / 2 == 1.5`
  - `float` et `double` sont sous-types de `real`
- promotion implicite d'`integer` à `boolean` : `(x && y) == (z || t)`

Entiers machines *vs* entiers mathématiques

- en C
  - entiers machines bornés
  - possibilité de débordements
  - quel est le résultat de `INT_MAX + 1` ?

Entiers machines *vs* entiers mathématiques

- en C
  - entiers machines bornés
  - possibilité de débordements
  - quel est le résultat de `INT_MAX + 1` ?
  - souvenez-vous d'Ariane 5 en 1996...

Entiers machines *vs* entiers mathématiques

- en C
  - entiers machines bornés
  - possibilité de débordements
  - quel est le résultat de `INT_MAX + 1` ?
  - souvenez-vous d'Ariane 5 en 1996...
- en ACSL
  - entiers mathématiques
  - jamais de débordement
  - fonctions totales sous-spécifiées
    - fonctions définies en tout point de leur domaine
    - mais sans nécessairement connaître exactement la valeur associée
  - jamais d'erreur : `1/0` est non spécifié mais `1/0 == 1/0` est un prédicat valide
  - possibilité de casts vers un type intégral τ du C

$$(\tau)n \equiv n \bmod 2^{8\times \text{sizeof}(\tau)}$$

Entiers machines *vs* entiers mathématiques

- en C
  - entiers machines bornés
  - possibilité de débordements
  - quel est le résultat de `INT_MAX + 1` ?
  - souvenez-vous d'Ariane 5 en 1996...
- en ACSL
  - entiers mathématiques
  - jamais de débordement
  - fonctions totales sous-spécifiées
    - fonctions définies en tout point de leur domaine
    - mais sans nécessairement connaître exactement la valeur associée
  - jamais d'erreur : `1/0` est non spécifié mais `1/0 == 1/0` est un prédicat valide
  - possibilité de casts vers un type intégral τ du C

$$(\tau)n \equiv n \bmod 2^{8\times \text{sizeof}(\tau)}$$

- petits exercices : les prédicats ACSL suivants sont-ils valides ?
  - `\forall integer x,y; \exists integer z; x+y == z`

Entiers machines *vs* entiers mathématiques

- en C
  - entiers machines bornés
  - possibilité de débordements
  - quel est le résultat de `INT_MAX + 1` ?
  - souvenez-vous d'Ariane 5 en 1996...
- en ACSL
  - entiers mathématiques
  - jamais de débordement
  - fonctions totales sous-spécifiées
    - fonctions définies en tout point de leur domaine
    - mais sans nécessairement connaître exactement la valeur associée
  - jamais d'erreur : `1/0` est non spécifié mais `1/0 == 1/0` est un prédicat valide
  - possibilité de casts vers un type intégral τ du C

$$(\tau)n \equiv n \bmod 2^{8\times \text{sizeof}(\tau)}$$

- petits exercices : les prédicats ACSL suivants sont-ils valides ?
  - `\forall integer x,y; \exists integer z; x+y == z`
  - `\forall int x,y; \exists int z; x+y == z`

Erreurs à l'exécution

$\{P\}\ s\ \{Q\}$ est valide si et seulement si
quand on exécute le programme s
dans un état de mémoire initial qui satisfait P,
alors il n'y a pas d'erreurs pendant l'exécution
et, si l'exécution s'arrête, l'état final satisfait Q.

Erreurs à l'exécution

$\{P\}\ s\ \{Q\}$ est valide si et seulement si
quand on exécute le programme s
dans un état de mémoire initial qui satisfait P,
alors il n'y a pas d'erreurs pendant l'exécution
et, si l'exécution s'arrête, l'état final satisfait Q.

- le greffon WP ne vérifie pas de lui-même l'absence d'erreur à l'exécution
- ... mais suppose qu'il n'y en a pas

Erreurs à l'exécution

$\{P\}\ s\ \{Q\}$ est valide si et seulement si
quand on exécute le programme s
dans un état de mémoire initial qui satisfait P,
alors il n'y a pas d'erreurs pendant l'exécution
et, si l'exécution s'arrête, l'état final satisfait Q.

- le greffon WP ne vérifie pas de lui-même l'absence d'erreur à l'exécution
- ... mais suppose qu'il n'y en a pas
- il faut vérifier cette condition par ailleurs
- c'est le rôle du greffon RTE :
  - génère des annotations ACSL pour chaque *RTE* potentielle
- d'où l'utilisation de l'option `-wp-rte` (ou `-rte -then`)

Exercice 1

1. Spécifier et prouver la fonction suivante
   (fichier `C4/abs.c` dans `~jsignol/fiil/C4`) :

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

2. Vérifier l'absence de débordements en utilisant l'option `-wp-rte` en modifiant votre spécification si nécessaire

Clause `assigns`

- `assigns lv1, lv2, ..., lvN`
- fait partie de la postcondition
- la clause `assigns` spécifie les valeurs gauches qui sont *possiblement* modifiées

Clause `assigns`

- `assigns lv1, lv2, ..., lvN`
- fait partie de la postcondition
- la clause `assigns` spécifie les valeurs gauches qui sont *possiblement* modifiées
- par contraposée, le reste de la mémoire est *obligatoirement* inchangée
- évite d'expliciter, pour chaque variable inchangée `ensures v == \old(v)`

Clause `assigns`

- `assigns lv1, lv2, ..., lvN`
- fait partie de la postcondition
- la clause `assigns` spécifie les valeurs gauches qui sont *possiblement* modifiées
- par contraposée, le reste de la mémoire est *obligatoirement* inchangée
- évite d'expliciter, pour chaque variable inchangée `ensures v == \old(v)`
- si rien ne peut être modifié (fonction en lecture seule) : `assigns \nothing`

Clause `assigns`

- `assigns lv1, lv2, ..., lvN`
- fait partie de la postcondition
- la clause `assigns` spécifie les valeurs gauches qui sont *possiblement* modifiées
- par contraposée, le reste de la mémoire est *obligatoirement* inchangée
- évite d'expliciter, pour chaque variable inchangée `ensures v == \old(v)`
- si rien ne peut être modifié (fonction en lecture seule) : `assigns \nothing`
-

```
/* returns the maximum of [x] and [y]. */

/*@ ensures \result >= x && \result >= y;
  @ ensures \result == x || \result == y;
  @ assigns \nothing; */
int max(int x, int y) {
  if (x >= y) return x;
  return y;
}
```

Exercice 2

1. Spécifier et prouver la fonction suivante (fichier `C4/max.c`) :

```
/* returns the maximum of [x] and [y]. */

int G;

int max(int x, int y) {
  if (x >= y) return x;
  return y;
}
```

Comportements : spécifications par cas

```
/*@ requires R;      // précondition globale
  @
  @ behavior b1:     // comportement b1
  @   assumes A1;    // garde d'activation de b1
  @   requires R1;   // précondition de b1
  @   ensures E1;    // postcondition de b1
  @   assigns L1;    // zone mémoire possiblement modifiée par b1
  @ ...
  @ behavior bn:
  @   assumes An;
  @   requires Rn;
  @   ensures En;
  @   assigns Ln;
  @
  @ complete behaviors;
  @ disjoint behaviors; */
```

Comportements : vérification

- la fonction appelante doit vérifier

$$R \wedge \bigwedge_{i \in [1..n]} (A_i \implies R_i)$$

Comportements : vérification

- la fonction appelante doit vérifier

$$R \wedge \bigwedge_{i \in [1..n]} (A_i \implies R_i)$$

- la fonction appelée doit garantir

$$\bigwedge_{i \in [1..n]} (\texttt{\old}(A_i) \implies E_i)$$

- pour tout $i \in [1..n]$, si A_i est valide dans l'état initial, alors les seules zones mémoires modifiables sont les L_i.

Comportements : vérification

- la fonction appelante doit vérifier

$$R \wedge \bigwedge_{i \in [1..n]} (A_i \implies R_i)$$

- la fonction appelée doit garantir

$$\bigwedge_{i \in [1..n]} (\texttt{\textbackslash old}(A_i) \implies E_i)$$

- pour tout $i \in [1..n]$, si A_i est valide dans l'état initial, alors les seules zones mémoires modifiables sont les L_i.
- `complete behaviors` : tous les cas sont couverts ($\bigvee_{i \in [1..n]} A_i$ est valide dans l'état initial)
- `disjoint behaviors` : tous les cas sont 2 à 2 mutuellement exclusifs

Exercice 3

1. Modifier la solution de `abs.c` pour y introduire des comportements

Appels de fonction

Un calcul de WP est modulaire (cours 2)

- prouver la postcondition d'une fonction indépendemment de tout site d'appel
  - en supposant que sa précondition est vraie

Appels de fonction

Un calcul de WP est modulaire (cours 2)

- prouver la postcondition d'une fonction indépendemment de tout site d'appel
  - en supposant que sa précondition est vraie
- si on veut prouver une fonction *g* qui appelle une fonction *f* :
  - la précondition de l'appelée *f* doit être vérifiée par l'appelant *g*
  - la postcondition de l'appelée *f* est supposée vraie
    - on pose en hypothèse cette postcondition
    - le code de *f* n'est pas vérifié en ce point
    - seuls un contrat et une déclaration de *f* sont requis

Exercice 4

1. En utilisant les solutions des exercices précédants, prouver la fonction suivante (fichier `C4/max_abs.c`) :

```
int abs(int x);
int max(int x, int y);

/* returns the maximum of absolute values of [x] and [y] */
int max_abs(int x, int y) {
  x = abs(x);
  y = abs(y);
  return max(x, y);
}
```

Boucles

$$\mathrm{WP}(\texttt{while } t \texttt{ invariant } J \texttt{ variant } s \cdot \prec \texttt{ do } e \texttt{ done}, Q) \equiv$$
$$J \wedge$$
$$\forall x_1 \dots x_k.$$
$$(J \wedge \quad t \rightarrow \mathrm{WP}(e, J \wedge s \prec w)[w \mapsto s]) \wedge$$
$$(J \wedge \neg t \rightarrow Q)$$

$x_1, \dots, x_k$ références modifiées dans e

w variable fraîche (le variant au début de l'itération)

Boucles

$$\begin{aligned}&\mathrm{WP}(\texttt{while } t \texttt{ invariant } J \texttt{ variant } s \cdot \prec \texttt{ do } e \texttt{ done}, Q) \equiv \\ &\quad J \wedge \\ &\quad \forall x_1 \dots x_k. \\ &\qquad (J \wedge \quad t \rightarrow \mathrm{WP}(e, J \wedge s \prec w)[w \mapsto s]) \wedge \\ &\qquad (J \wedge \neg t \rightarrow Q)\end{aligned}$$

$x_1, \dots, x_k$ références modifiées dans e

w variable fraîche (le variant au début de l'itération)

avec Frama-C, pour une boucle `for(i = 0; i < n; i++);` :

- donner un invariant J : `loop invariant 0 <= i <= n;`
- donner un variant $s \prec w$ pour la correction totale :

Boucles

$$\mathrm{WP}(\texttt{while } t \texttt{ invariant } J \texttt{ variant } s \cdot \prec \texttt{ do } e \texttt{ done}, Q) \equiv$$
$$J \wedge$$
$$\forall x_1 \dots x_k.$$
$$(J \wedge \quad t \rightarrow \mathrm{WP}(e, J \wedge s \prec w)[w \mapsto s]) \wedge$$
$$(J \wedge \neg t \rightarrow Q)$$

$x_1, \dots, x_k$ références modifiées dans e

w variable fraîche (le variant au début de l'itération)

avec Frama-C, pour une boucle `for(i = 0; i < n; i++);` :

- donner un invariant J : `loop invariant 0 <= i <= n;`
- donner un variant $s \prec w$ pour la correction totale :`loop variant n-i;`
- connaître les variables $\vec{x}$ modifiées par la boucle : `loop assigns i;`

Exercice 5

1. Spécifier et prouver la fonction suivante (fichier `code/m.c`) :

```
int m(int x, int y) {
  int res = 0;
  int a = 0;
  while (a < x) {
    res += y;
    a++;
  }
  return res;
}
```

Prédicats, lemmes et fonctions logiques

- lemme

```
/*@ lemma add_even: \forall integer x,y;
      a % 2 == 0 ==> y % 2 == 0 ==> (x+y) % 2 == 0; */
```

- prédicat : `/*@ predicate is_even(integer n) = n % 2 == 0; */`
- fonction logique
  - explicite : `/*@ logic integer next(integer n) = n+1; */`
  - axiomatisée :

```
/*@ axiomatic fib {
      logic integer fib(integer n);
      axiom fib0: fib(0) = 0;
      axiom fib1: fib(1) = 1;
      axiom fibn: \forall integer b;
        n >= 2 ==> fib(n) == fib(n-1) + fib(n-2);
  } */
```

Aide aux spécifications et aux preuves

Exercice 6

1. définir une axiomatique caractérisant $\sum_{i=0}^{n} i$
2. à l'aide de la question 1, spécifier et vérifier le programme suivant (fichier sum.c) :

```
int sum(int n) {
  int i = 0;
  int res = 0;
  for(int i = 0; i < n; i++) res += i;
  return res;
}
```