Projet MVSI

Groupe 1 : T. Adam, A. Cesari, T. Bagrel

Avril 2020

Table des matières

1 Triangle de Floyd (Timothée)

1.1 Code original

```c
#include <stdio.h>

void floyd_triangle(unsigned int n) {
    int i, c, a;
    a = 1;
    for (i = 1; i <= n; i++) {
        for (c = 1; c <= i; c++) {
            printf("%d ", a);
            a++;
        }
        printf("\n");
    }
}
```

1.2 Modélisation de l'affichage

Comme cet algorithme est un algorithme d'affichage, il est nécessaire de modéliser cet affichage par un composant mathématique que l'on peut utiliser dans la preuve (et surtout dans la post-condition). Ici, nous utiliserons une fonction partielle $display \in (\mathbb{N} \times \mathbb{N}) \pfun \mathbb{Z}$ pour modéliser la matrice d'affichage. Chaque ligne de la matrice *display* correspond à une ligne à l'écran. On considère que les nombres de notre triangle de Floyd seront placés dans des cases de la matrice d'affichage, où chaque case peut recevoir un nombre complet (et non pas seulement un caractère, ce qui obligerait à gérer la longueur en caractères de chaque nombre ainsi que le caractère espace entre chaque nombre). Dès lors, on peut modéliser les appels à la fonction `printf` par des manipulations sur la matrice *display* :

- on considère deux pseudo-variables u et v représentant la position actuelle du curseur (qui correspond à la prochaine case dans laquelle sera écrit un nombre). Au départ, la matrice *display* est vide ($display = \emptyset$), et le curseur est sur la première case en haut à gauche, donc $(u, v) = (0, 0)$;
- l'appel `printf("%d ", a);` est alors modélisé par `display(u, v) := a ; v := v + 1` : on place la valeur `a` à la position actuelle du curseur, et on déplace le curseur vers la droite ;
- l'appel `printf("\n");` est alors modélisé par `u := u + 1 ; v := 0` : on place le curseur sur la ligne suivante, et on le replace en début de ligne.

1.3 Réécriture et annotations

Lors de la réécriture du code C en algorithme prouvable, il est nécessaire de transformer les `for` en briques de bases que l'on sait prouver : les boucles `while`.

Ici, il n'y a pas de pré-condition (à part l'appartenance $n \in \mathbb{N}$). La post-condition, en revanche, correspond à l'assertion "les bons nombres sont à la bonne place dans l'affichage", et s'énonce mathématiquement avec : $\forall j \in 0..n-1, \forall k \in 0..j, (j, k) \in \mathrm{Dom}\,(display) \land display(j, k) = \dfrac{j(j+1)}{2} + k + 1$.

N.B. Les types des variables sont considérés comme des axiomes tout au long de la preuve et dans le système d'annotation (pour ne pas surcharger encore l'ensemble). Sous Rodin et TLA+, un invariant de vérification des types sera ajouté.

```
constants n
variables u, v, i, a, c, display
```

l_0 : VRAI
```
u := 0  // printing starts in the upper-left corner
v := 0
```
$l_1 : u = 0 \wedge v = 0$
```
i := 1
```
$l_2 : i = 1 \wedge u = i - 1 \wedge v = 0$
```
a := 1
```
$l_3 : i = 1 \wedge a = 1 \wedge u = i - 1 \wedge v = 0$
```
while( i <= n) {
```
$l_4 : 1 \leqslant i \leqslant n \wedge a = \frac{i(i-1)}{2} + 1 \wedge u = i - 1 \wedge v = 0$
$\wedge \forall j \in 0..i-2, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
```
c := 1
```
$l_5 : 1 \leqslant i \leqslant n \wedge c = 1 \wedge a = \frac{i(i-1)}{2} + c \wedge u = i - 1 \wedge v = c - 1$
$\wedge (\forall j \in 0..i-2, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
```
while (c <= i) {
```
$l_6 : 1 \leqslant i \leqslant n \wedge 1 \leqslant c \leqslant i \wedge a = \frac{i(i-1)}{2} + c \wedge u = i - 1 \wedge v = c - 1$
$\wedge \forall j \in 0..i-2, (\forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
$\wedge (\forall k \in 0..c-2, (i-1,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{(i-1)i}{2} + k + 1)$
```
display(u, v) := a  // print a
v := v + 1
```
$l_7 : 1 \leqslant i \leqslant n \wedge 1 \leqslant c \leqslant i \wedge a = \frac{i(i-1)}{2} + c \wedge u = i - 1 \wedge v = c$
$\wedge (\forall j \in 0..i-2, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
$\wedge (\forall k \in 0..c-1, (i-1,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{(i-1)i}{2} + k + 1)$
```
a := a + 1
```
$l_8 : 1 \leqslant i \leqslant n \wedge 1 \leqslant c \leqslant i \wedge a = \frac{i(i-1)}{2} + c + 1 \wedge u = i - 1 \wedge v = c$
$\wedge (\forall j \in 0..i-2, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
$\wedge (\forall k \in 0..c-1, (i-1,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{(i-1)i}{2} + k + 1)$
```
c := c + 1
```
$l_9 : 1 \leqslant i \leqslant n \wedge 1 \leqslant c \leqslant i + 1 \wedge a = \frac{i(i-1)}{2} + c \wedge u = i - 1 \wedge v = c - 1$
$\wedge (\forall j \in 0..i-2, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
$\wedge (\forall k \in 0..c-2, (i-1,k) \in \mathrm{Dom}\,(display) \wedge display(i-1,k) = \frac{(i-1)i}{2} + k + 1)$
```
}
```
$l_{10} : 1 \leqslant i \leqslant n \wedge c = i + 1 \wedge a = \frac{i(i+1)}{2} + 1 \wedge u = i - 1 \wedge v = c - 1$
$\wedge (\forall j \in 0..i-1, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
```
u := u + 1  // print \n
v := 0
```
$l_{11} : 1 \leqslant i \leqslant n \wedge c = i + 1 \wedge a = \frac{i(i+1)}{2} + 1 \wedge u = i \wedge v = 0$
$\wedge (\forall j \in 0..i-1, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
```
i := i + 1
```
$l_{12} : 1 \leqslant i \leqslant n + 1 \wedge c = i \wedge a = \frac{(i-1)i}{2} + 1 \wedge u = i - 1 \wedge v = 0$
$\wedge (\forall j \in 0..i-2, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1)$
```
}
```
$l_{13} : \forall j \in 0..n-1, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \frac{j(j+1)}{2} + k + 1$

1.4 Absence d'erreurs à l'exécution

Pour vérifier l'absence d'erreur à l'exécution, Frama-C va tenter de prouver l'absence d'overflow/underflow, ainsi que la validité des cases mémoires lors des déréférences de pointeurs. Cependant, pour que Frama-C y arrive, il doit prouver un invariant (ou au moins une forme allégée) semblable à celui présenté dans ce document. C'est une tâche ardue, qui nécessite donc qu'on l'aide avec des annotations.

Cependant, en analysant le problème de manière plus détaillée, on peut se passer des capacités de Frama-C :

— la validité des cases mémoires (les accès aux cellules de *display*) a été prouvée avec Rodin, avec toutes les annotations (validées) évoquant $\mathrm{Dom}\,(display)$;

— les annotations prouvées avec Rodin montrent que i, c sont toujours dans l'intervalle $0..n-1$, de plus Rodin valide la post-condition : $\forall j \in 0..n-1, \forall k \in 0..j, (j,k) \in \mathrm{Dom}\,(display) \wedge display(j,k) = \dfrac{j(j+1)}{2} + k + 1$, ainsi on en déduit que a reste toujours dans la plage $0..\dfrac{n(n+1)}{2}$

— enfin, u et v sont des pseudo-variables qui ne sont utilisées que pour modéliser l'affichage, et qui n'apparaissent pas dans le code réel.

Ainsi, il semble que l'absence de RTE soit déjà validée grâce à Rodin, à condition que $\dfrac{n(n+1)}{2}$ ne dépasse pas la valeur maximale pour un entier signé, à savoir 2147483647 en 32 bit ou 9223372036854775807 en 64 bit. C'est à dire que n ne dépasse pas $\left\lfloor \frac{-1+\sqrt{1+8 \cdot \mathrm{INT\_MAX}}}{2} \right\rfloor$ soit 65535 en 32 bit et 4294967295 en 64 bit.

2 Modélisation des tableaux pour une preuve de tri

Deux des trois programmes du projet à prouver sont des algorithmes de tri sur des tableaux. Il convient donc dans un premier temps d'introduire un formalisme pouvant être utilisé pour modéliser les tableaux à la fois dans les preuves de programmes manuelles mais également mécanisées avec TLA+ et Rodin.

2.1 Définition de la pré-condition et post-condition pour un algorithme de tri

Si les algorithmes de tri étudiés travaillent bien souvent en place (modification du tableau passé en paramètres pour le trier), il est nécessaire dans la preuve de considérer que le tableau passé en entrée n'est pas modifié, afin de pouvoir trouver une formulation de "le tableau est bien trié" au niveau de la post-condition.

Du point de vue code, un algorithme de tri générique prendrait en entrée un tableau (pointeur vers un entier) `tab` ainsi qu'un entier non signé `n`, et si ce tableau est de taille au moins `n`, trierait en place les valeurs de ce tableau par ordre croissant.

Côté preuve, l'algorithme disposerait en entrée des constantes :

— $tab \in \mathbb{N} \nrightarrow \mathbb{Z}$

— $n \in \mathbb{N}$

et serait chargé de construire $tab_{out} \in \mathbb{N} \nrightarrow \mathbb{Z}$ qui représenterait le tableau trié en sortie.

Pré-condition

La pré-condition correspond alors à la formulation mathématique de "le tableau tab est de taille au moins n", et est facile à énoncer grâce à l'utilisation de fonctions partielles :

$$\text{PRE} : 0..n-1 \subseteq \text{Dom}\,(tab)$$

Post-condition

Intuitivement, la post-condition comporte deux parties :

— le tableau en sortie est de taille au moins n

— le tableau en sortie contient les mêmes éléments que celui en entrée, mais triés par ordre croissant.

La première partie s'énonce facilement (toujours avec le formalisme des fonctions partielles) :

$$0..n-1 \subseteq \text{Dom}\,(tab_{out})$$

Pour formaliser la deuxième partie, nous allons introduire l'opérateur Enum_n.
Soit $n \in \mathbb{N}$ et $tab \in \mathbb{N} \nrightarrow \mathbb{Z}$ tel que $0..n-1 \subseteq \text{Dom}\,(tab)$

$$\text{Enum}_n\,(tab) : \left\{ \begin{array}{lcl} \mathbb{Z} & \longmapsto & \mathbb{N} \\ m & \rightarrow & \text{card}\,\{i \in 0..n-1 | tab(i) = m\} \end{array} \right.$$

La conservation des éléments entre le tableau initial tab et le tableau final tab_{out} sera donc modélisée par $0..n-1 \subseteq \text{Dom}\,(tab_{out}) \wedge \text{Enum}_n\,(tab_{out}) = \text{Enum}_n\,(tab)$.

Intuitivement, $\text{Enum}_n\,(tab_{out}) = \text{Enum}_n\,(tab)$ vérifie que chaque valeur apparaissant dans le tableau tab entre les indices 0 et $n-1$ inclus est présente en même nombre dans tab_{out}. Il est enfin facile de prouver que $\text{Enum}_n\,(tab_{out}) = \text{Enum}_n\,(tab)$ si et seulement si tab et tab_{out} restreints à $0..n-1$ sont en permutation.

Regroupement

Finalement, un algorithme de tri (en place) peut être défini côté code par une fonction de signature

```
void sort(int *tab, unsigned int n)
```

et côté preuve par

$$\begin{array}{ll}
\text{CONSTANTS} & tab \in \mathbb{N} \nrightarrow \mathbb{Z} \\
& n \in \mathbb{N} \\
\text{VARIABLES} & tab_{out} \in \mathbb{N} \nrightarrow \mathbb{Z} \\
\text{PRE} & 0..n-1 \subseteq \mathrm{Dom}\,(tab) \\
\text{POST} & 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) \\
& \wedge \forall k \in 0..n-2, tab_{out}(k) \leqslant tab_{out}(k+1)
\end{array}$$

Enfin, il sera nécessaire de réaliser dans l'adaptation en algorithme du code C une copie du tableau initial tab dans le tableau tab_{out}.

2.2 Modélisation de l'affectation d'une cellule du tableau

Soit $tab \in \mathbb{N} \nrightarrow \mathbb{Z}$ avec $\mathrm{D} \subseteq \mathrm{Dom}\,(tab)$. Soit $k \in \mathbb{N}$ et $v \in \mathbb{Z}$.
L'opération `tab(k) := v` est modélisée par l'assertion logique

$$\mathrm{D} \cup \{k\} \subseteq \mathrm{Dom}\,(tab') \wedge tab' : \left\{ \begin{array}{ccc} \mathbb{N} & \nrightarrow & \mathbb{Z} \\ i & \longmapsto & \left\{ \begin{array}{l} v \text{ si } i = k \\ tab(i) \text{ sinon} \end{array} \right. \end{array} \right.$$

2.3 Théorèmes sur la notation Enum

Lemme 2.1.
Soit $n \in \mathbb{N}$. Soit $tab \in \mathbb{N} \nrightarrow \mathbb{Z}$ tel que $0..n-1 \subseteq \mathrm{Dom}\,(tab)$.
Soit $k \in 0..n-1$ et $v \in \mathbb{Z}$.
Soit $tab' : \left\{ \begin{array}{ccc} \mathbb{N} & \nrightarrow & \mathbb{Z} \\ i & \longmapsto & \left\{ \begin{array}{l} v \text{ si } i = k \\ tab(i) \text{ sinon} \end{array} \right. \end{array} \right.$

$$\mathrm{Enum}_n\,(tab') = \mathrm{Enum}_n\,(tab) + \chi_{\{v\}} - \chi_{\{tab(k)\}}$$

Preuve du lemme 2.1.
Soit $m \in \mathbb{Z}$.

$$\mathrm{Enum}_n\,(tab')\,(m) = \mathrm{card}\{i \in 0..n-1 | tab'(i) = m\}$$

or

$$\begin{array}{rcl}
\forall i \in 0..n-1, \quad tab'(i) = m & \iff & (i \neq k \wedge tab'(i) = m) \vee (i = k \wedge tab'(i) = m) \\
& \iff & (i \neq k \wedge tab(i) = m) \vee (i = k \wedge v = m)
\end{array}$$

donc

$$\{i \in 0..n-1 | tab'(i) = m\} = \{i \in 0..n-1 | i \neq k \wedge tab(i) = m\} \bigsqcup \{i \in 0..n-1 | i = k \wedge v = m\}$$

de plus,

$$\{i \in 0..n-1 | i \neq k \wedge tab(i) = m\} \bigsqcup \{i \in 0..n-1 | i = k \wedge tab(i) = m\} = \{i \in 0..n-1 | tab(i) = m\}$$

donc

$$\mathrm{card}\{i \in 0..n-1 | i \neq k \wedge tab(i) = m\} = \mathrm{card}\{i \in 0..n-1 | tab(i) = m\} - \mathrm{card}\{i \in 0..n-1 | i = k \wedge tab(i) = m\}$$

en regroupant

$$\begin{array}{rcl}
\mathrm{Enum}_n\,(tab')\,(m) & = & \mathrm{card}\{i \in 0..n-1 | tab'(i) = m\} \\
& = & \mathrm{card}\{i \in 0..n-1 | i \neq k \wedge tab(i) = m\} \\
& + & \mathrm{card}\{i \in 0..n-1 | i = k \wedge v = m\} \\
& = & \mathrm{card}\{i \in 0..n-1 | tab(i) = m\} \\
& - & \mathrm{card}\{i \in 0..n-1 | i = k \wedge tab(i) = m\} \\
& + & \mathrm{card}\{i \in 0..n-1 | i = k \wedge v = m\} \\
& = & \mathrm{Enum}_n\,(tab)\,(m) - \chi_{\{tab(k)\}}(m) + \chi_{\{v\}}(m)
\end{array}$$

□

Pour les théorèmes suivants, nous considérons $n \in \mathbb{N}$ et $tab, tab_{out} \in \mathbb{N} \nrightarrow \mathbb{Z}$ tels que $0..n-1 \subseteq \mathrm{Dom}\,(tab)$ et $0..n-1 \subseteq \mathrm{Dom}\,(tab_{out})$.

Théorème 2.2 (Destruction)**.**
Soit $j, k \in 0..n-1$ et $tmp \in \mathbb{Z}$.
Les annotations suivantes sont justes :

$l_0 : \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) \wedge tmp = tab_{out}(k) \wedge j \neq k$
`tab_out(k) := tab_out(j)`
$l_1 : \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(j)\}} - \chi_{\{tmp\}}$

Théorème 2.3 (Échange)**.**
Soit $j, l \in 0..n-1$ et $tmp \in \mathbb{Z}$.
Les annotations suivantes sont justes :

$l_0 : \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(j)\}} - \chi_{\{tmp\}} \wedge l \neq j$
`tab_out(j) := tab_out(l)`
$l_1 : \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(l)\}} - \chi_{\{tmp\}}$

Théorème 2.4 (Reconstruction)**.**
Soit $l \in 0..n-1$ et $tmp \in \mathbb{Z}$.
Les annotations suivantes sont justes :

$l_0 : \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(l)\}} - \chi_{\{tmp\}}$
`tab_out(l) := tmp`
$l_1 : \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$

Les trois théorèmes se prouvent avec la technique de preuve classique sur les annotations en utilisant le lemme sur tab_{out}.

3 Bubble sort (Alexandre)

3.1 Code original

```
1  void bubble_sort(int *tab, unsigned int n) {
2      int c, d, swap;
3      c = 0;
4      while (c < (n - 1)) {
5          d = 0;
6          while (d < (n - c - 1)) {
7              if (tab[d] > tab[d + 1]) {
8                  swap = tab[d];
9                  tab[d] = tab[d + 1];
10                 tab[d + 1] = swap;
11             }
12             d++;
13         }
14         c++;
15     }
16 }
```

3.2 Réécriture et annotations

Lors de la réécriture du code C en algorithme prouvable, il est nécessaire de transformer/décomposer les `for` en brique de base que l'on sait prouver : la boucle `while`.

De plus, il faut introduire une boucle pour copier le tableau initial, afin de toujours conserver l'original pour pouvoir établir la post-condition.

N.B. les assertions logiques de la pré-condition sont considérées comme vraies et pouvant être utilisées pour prouver le système d'annotation. Pour être plus rigoureux, il aurait fallu propager les parties utiles de la pré-condition tout le long du système d'annotation, ce qui aurait encore alourdi la notation. De plus, les types des variables sont considérés comme des axiomes tout au long de la preuve et dans le système d'annotation (pour ne pas surcharger encore l'ensemble). Sous Rodin et TLA+, un invariant de vérification des types sera ajouté.

```
constants tab, n
variables tab_out, c, d, swap
```

l_0 : VRAI
`c := 0`
$l_1 : c = 0$
`while (c < n) {`
  $l_2 : 0 \leqslant c < n \land 0..c-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_c\,(tab_{out}) = \mathrm{Enum}_c\,(tab)$
  `tab_out(c) := tab(c)`
  $l_3 : 0 \leqslant c < n \land 0..c \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_{c+1}\,(tab_{out}) = \mathrm{Enum}_{c+1}\,(tab)$
  `c := c + 1`
  $l_4 : 0 \leqslant c \leqslant n \land 0..c-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_c\,(tab_{out}) = \mathrm{Enum}_c\,(tab)$
`}`
$l_5 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
`c := 0`
$l_6 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land c = 0 \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
`while (c < (n - 1)) {`

$l_7 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$

```
d := 0
```

$l_8 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge d = 0$

```
while (d < (n - c - 1)) {
```

$l_9 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d, tab_{out}(k) \leqslant tab_{out}(d)$

```
if (tab_out(d) > tab_out(d + 1)) {
```

$l_{10} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d, tab_{out}(k) \leqslant tab_{out}(d)$
$\quad \wedge tab_{out}(d) > tab_{out}(d+1)$

```
swap := tab_out(d)
```

$l_{11} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, swap \leqslant tab_{out}(k) \wedge (\forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k))$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d, tab_{out}(k) \leqslant tab_{out}(d)$
$\quad \wedge tab_{out}(d) > tab_{out}(d+1) \wedge swap = tab_{out}(d) \wedge swap > tab_{out}(d+1)$

```
tab_out(d) := tab_out(d + 1)
```

$l_{12} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1$
$\quad \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(d+1)\}} - \chi_{\{swap\}}$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, swap \leqslant tab_{out}(k) \wedge (\forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k))$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d+1, tab_{out}(k) \leqslant swap$

```
tab_out(d + 1) := swap
```

$l_{13} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d+1, tab_{out}(k) \leqslant tab_{out}(d+1)$

```
} else {
```

$l_{14} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d+1, tab_{out}(k) \leqslant tab_{out}(d+1)$

```
}
```

$l_{15} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge 0 \leqslant d < n-c-1 \wedge \forall 0 \leqslant k < d+1, tab_{out}(k) \leqslant tab_{out}(d+1)$

```
d := d + 1
```

$l_{16} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$
$\quad \wedge 0 \leqslant d \leqslant n-c-1 \wedge \forall 0 \leqslant k < d, tab_{out}(k) \leqslant tab_{out}(d)$

```
}
```

$l_{17} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c < n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c-1 \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$

```
c := c + 1
```

$l_{18} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge 0 \leqslant c \leqslant n-1 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall n-c \leqslant k \leqslant n-1, \forall 0 \leqslant l < k, tab_{out}(l) \leqslant tab_{out}(k)$

```
}
```

$l_{19} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \wedge \forall 0 \leqslant k \leqslant n-2, tab_{out}(k) \leqslant tab_{out}(k+1)$

3.3 Idée de la preuve

Si l'indice de l'élément en cours de déplacement est d, et l'indice de début la zone triée (en fin de tableau) est t, alors l'invariant est $\forall k \in 0..d-1, tab_{out}(k) \leqslant tab_{out}(d)$ (« l'élément courant est plus grand que tous ceux à sa gauche ») et $\forall k \in t..n-1, \forall l \in 0..k-1, tab_{out}(l) \leqslant tab_{out}(k)$ (« chaque élément de la zone triée est plus grand que tous les éléments à sa gauche »). Quand d arrive en bout de parcours (et vaut $t-1$), l'élément courant $tab_{out}(d)$ est plus grand que tous les éléments à sa gauche et plus petit que tous les éléments à sa droite. La zone des éléments triés s'en retrouve donc agrandie.

3.4 Absence d'erreurs à l'exécution

L'utilisation de Frama-C sur l'algorithme de bubble sort est délicate. En effet, pour vérifier l'absence d'erreurs à l'exécution, Frama-C va tenter de prouver l'absence d'overflow/underflow, ainsi que la validité des cases mémoires lors des déréférences de pointeurs.

Cependant, pour prouver l'absence d'erreurs, compte tenu de la complexité de l'algorithme de bubble sort, Frama-C a besoin d'un système d'annotations semblable à celui décrit dans ce document, à ceci près que le code C travaille en place, et que Frama-C le remodifie ensuite dans un format que l'outil digère plus facilement. Il faudrait donc adapter le système d'annotation qui a été utilisé sur papier, Rodin et TLA+ – un travail long et fastidieux – pour gagner seulement la preuve d'absence de RTE...

Mais en analysant le problème de manière plus détaillée :

— la validité des cases mémoires (les accès aux cellules de tab et tab_{out}) a été prouvée avec Rodin, avec toutes les annotations (validées) de la forme $0..n-1 \subseteq \mathrm{Dom}\,(tab)$;

— les annotations prouvées avec Rodin montrent que c et d sont toujours dans l'intervalle $0..n-1$. *swap* quant à lui n'est utilisé que pour sauver une valeur temporaire du tableau (qui est du type `int` aussi, donc aucune chance de RTE).

Ainsi, il semble que l'absence de RTE soit déjà validée grâce à Rodin, à condition que n ne dépasse pas la valeur maximale pour un entier signé, à savoir 2147483647 en 32 bit ou 9223372036854775807 en 64 bit.

Le code C du programme, avec le commentaire spécial au format ACSL (Frama-C) spécifiant la pré-condition et post-condition de l'algorithme est tout de même fourni dans les livrables du projet.

4 Shell sort (Thomas)

4.1 Code original

```
void shell_sort(int *tab, unsigned int n) {
    int i, j, inc, tmp;
    // --------------------
    inc = 3;
    while (inc > 0) {
        for (i = 0; i < n; i++) {
            j = i;
            tmp = tab[i];
            while ((j >= inc) && (tab[j - inc] > tmp)) {
                tab[j] = tab[j - inc];
                j = j - inc;
            }
            tab[j] = tmp;
        }
        if (inc / 2 != 0) {
            inc = inc / 2;
        } else if (inc == 1) {
            inc = 0;
        } else {
            inc = 1;
        }
    }
}
```

4.2 Réécriture et annotations

Lors de la réécriture du code C en algorithme prouvable, il est nécessaire de transformer/décomposer les `for` et les `else if` en briques de bases que l'on sait prouver : la boucle `while` et l'alternative `if`/`else`.

De plus, il faut introduire une boucle pour copier le tableau initial, afin de toujours conserver l'original pour pouvoir établir la post-condition.

N.B. les assertions logiques de la pré-condition sont considérées comme vraies et pouvant être utilisées pour prouver le système d'annotation. Pour être plus rigoureux, il aurait fallu propager les parties utiles de la pré-condition tout le long du système d'annotations, ce qui aurait encore alourdi la notation. De plus, les types des variables sont considérés comme des axiomes tout au long de la preuve et dans le système d'annotation (pour ne pas surcharger encore l'ensemble). Sous Rodin et TLA+, un invariant de vérification des types sera ajouté.

```
constants tab, n
variables tab_out, i, j, inc, tmp
```

l_0 : VRAI
`i := 0`
$l_1 : i = 0$
`while (i < n) {`
    $l_2 : 0 \leqslant i \wedge i < n \wedge 0..i-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge \mathrm{Enum}_i\,(tab_{out}) = \mathrm{Enum}_i\,(tab)$
    `tab_out(i) := tab(i)`
    $l_3 : 0 \leqslant i \wedge i < n \wedge 0..i \subseteq \mathrm{Dom}\,(tab_{out}) \wedge \mathrm{Enum}_{i+1}\,(tab_{out}) = \mathrm{Enum}_{i+1}\,(tab)$
    `i := i + 1`

$l_4 : 0 \leqslant i \wedge i \leqslant n \wedge 0..i-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge \mathrm{Enum}_i\,(tab_{out}) = \mathrm{Enum}_i\,(tab)$

```
}
```

$l_5 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$

```
inc := 3
```

$l_6 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc = 3 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$

```
while (inc > 0) {
```

$l_7 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$

```
i := 0
```

$l_8 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge i = 0 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$

```
while (i < n) {
```

$l_9 : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$

```
j := i
```

$l_{10} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))) \wedge j = i$

```
tmp := tab_out(i)
```

$l_{11} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))) \wedge j = i$
$\wedge\, tmp = tab_{out}(i)$

```
while ((j >= inc) and (tab_out(j - inc) > tmp)) {
```

$l_{12} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n$
$\wedge\, inc \leqslant j \wedge j \leqslant i \wedge tab_{out}(j - inc) \geqslant tmp$
$\wedge\, \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(j)\}} - \chi_{\{tmp\}}$
$\wedge\, (inc = 1 \implies ((\forall k \cdot k \in 0..i-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$
$\wedge\, (j = i \vee tab_{out}(i-1) \leqslant tab_{out}(i))))$

```
tab_out(j) := tab_out(j - inc)
```

$l_{13} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n$
$\wedge\, inc \leqslant j \wedge j \leqslant i \wedge tab_{out}(j - inc) \geqslant tmp$
$\wedge\, \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(j-inc)\}} - \chi_{\{tmp\}}$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-1 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$

```
j := j - inc
```

$l_{14} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n$
$\wedge\, 0 \leqslant j \wedge j \leqslant i-1 \wedge tab_{out}(j) \geqslant tmp$
$\wedge\, \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(j)\}} - \chi_{\{tmp\}}$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-1 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$

```
}
```

$l_{15} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n$
$\wedge\, 0 \leqslant j \wedge j < n \wedge tab_{out}(j) \geqslant tmp$
$\wedge\, \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab) + \chi_{\{tab_{out}(j)\}} - \chi_{\{tmp\}}$
$\wedge\, (inc = 1 \implies ((\forall k \cdot k \in 0..i-1 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$
$\wedge\, (\forall k \cdot k \in 0..j-1 \implies tab_{out}(k) \leqslant tmp)))$

```
tab_out(j) := tmp
```

$l_{16} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i < n$
$\wedge\, \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-1 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$

```
i := i + 1
```

$l_{17} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge 0 \leqslant i \wedge i \leqslant n$
$\wedge\, \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..i-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$

```
}
```

$l_{18} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \wedge inc > 0 \wedge \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\wedge\, (inc = 1 \implies (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$

```
    if (inc / 2 != 0) {
```
$l_{19} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land inc \geqslant 2 \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
```
        inc := inc / 2
```
$l_{20} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land inc \geqslant 1 \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
```
    } else {
```
$l_{21} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \land inc = 1 \land (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$
```
        if (inc = 1) {
```
$l_{22} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \land (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$
```
            inc := 0
```
$l_{23} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \land inc = 0 \land (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$
```
        } else {
```
$l_{24} :$ FAUX
```
            inc := 1
```
$l_{25} :$ FAUX
```
        }
```
$l_{26} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \land inc = 0 \land (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$
```
    }
```
$l_{27} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \land inc \geqslant 0 \land (inc = 0 \implies (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1)))$
```
}
```
$l_{28} : 0..n-1 \subseteq \mathrm{Dom}\,(tab_{out}) \land \mathrm{Enum}_n\,(tab_{out}) = \mathrm{Enum}_n\,(tab)$
$\quad \land (\forall k \cdot k \in 0..n-2 \implies tab_{out}(k) \leqslant tab_{out}(k+1))$

4.3 Idée de la preuve

Le shell sort est un dérivé du tri par insertion où l'on réalise des passes préliminaires pour pré-trier certaines valeurs du tableau. Comme ici on ne s'intéresse pas à montrer que le *shell sort* est plus efficace que le tri par insertion, mais seulement qu'il est partiellement correct, il suffit de montrer que pour $inc = 1$ la passe trie effectivement le tableau (puisque c'est équivalent à ce moment au tri par insertion), et que la suite des valeurs prises par inc se termine forcément par les valeurs $1, 0$. On vérifie également que toutes les passes conservent les éléments du tableau original.

4.4 Absence d'erreurs à l'exécution

L'utilisation de Frama-C sur l'algorithme de shell sort est délicate. En effet, pour vérifier l'absence d'erreurs à l'exécution, Frama-C va tenter de prouver l'absence d'overflow/underflow, ainsi que la validité des cases mémoires lors des déréférences de pointeurs.

Cependant, pour prouver l'absence d'erreurs, compte-tenu de la complexité de l'algorithme de shell sort, Frama-C a besoin d'un système d'annotations semblable à celui décrit dans ce document, à ceci près que le code C travaille en place, et que Frama-C le remodifie ensuite dans un format que l'outil digère plus facilement. Il faudrait donc adapter le système d'annotation qui a été utilisé sur papier, Rodin et TLA+ – un travail long et fastidieux – pour gagner seulement la preuve d'absence de RTE...

Mais en analysant le problème de manière plus détaillée :

— la validité des cases mémoires (les accès aux cellules de tab et tab_{out}) a été prouvée avec Rodin, avec toutes les annotations (validées) de la forme $0..n-1 \subseteq \mathrm{Dom}\,(tab)$;

— les annotations prouvées avec Rodin montrent que i, j sont toujours dans l'intervalle $0..n-1$, et inc dans l'intervalle $0..3$. tmp quant à lui n'est utilisé que pour sauver une valeur temporaire du tableau (qui est du type `int` aussi, donc aucune chance de RTE).

Ainsi, il semble que l'absence de RTE soit déjà validée grâce à Rodin, à condition que n ne dépasse pas la valeur maximale pour un entier signé, à savoir 2147483647 en 32 bit ou 9223372036854775807 en 64 bit.

Le code C du programme, avec le commentaire spécial au format ACSL (Frama-C) spécifiant la pré-condition et post-condition de l'algorithme est tout de même fourni dans les livrables du projet.

5 Remarques, résultats et pistes d'amélioration

5.1 Genrodin

Genrodin est un script Python qui a été utilisé pour générer facilement la structure des preuves avec Rodin. Il est disponible librement à l'adresse `https://github.com/tbagrel1/genrodin`.

5.2 Prouveurs utilisés sur Rodin

Voici la liste des prouveurs qui ont été utilisés sur Rodin :

— le prouveur par défaut ;

— les prouveurs SMT (utilisés en tactique automatique, ce qui a permis de décharger beaucoup de preuves) ;

— les prouveurs d'Atelier B (pour décharger certaines PO en sélectionnant manuellement les hypothèses) ;

— le prouveur ProB (pour décharger certaines hypothèses arithmétiques complexes).

5.3 Pas de gestion de Enum avec Rodin

Nous avons introduit dans ce document l'opérateur Enum, utilisé pour prouver la conservation des éléments dans les preuves de tri. Cependant, compte tenu de son expression complexe, et de la difficulté déjà importante pour décharger toutes les PO sur Rodin, nous avons choisi de ne pas l'ajouter aux invariants. La preuve réalisée sur Rodin prouve donc que le tableau est trié en sortie, mais pas qu'il contient les mêmes éléments que le tableau d'entrée (même si les théorèmes énoncés dans ce document semblent justes).

Une solution si la définition directe sur Rodin de l'opérateur Enum est trop complexe pourrait être de prouver effectivement à la main (sur papier) les trois théorèmes sur l'opérateur, et de les ajouter au contexte comme axiomes. Une meilleure méthode encore serait d'ajouter le lemme en tant qu'axiome, pour laisser Rodin vérifier les théorèmes.

5.4 Résultats obtenus sur Rodin

À l'issue de ce projet, la preuve obtenue sur Rodin pour les trois programmes (triangle de Floyd, bubble sort et shell sort) ne contient plus aucune PO non déchargée. On peut donc dire que les trois programmes ont été prouvés mécaniquement (sauf la conservation des éléments pour les algorithmes de tri, comme évoqué au paragraphe précédent).

5.5 Labels "ul" sur TLA+

Lors de la définition des algorithmes en PlusCal, nous avons choisi de placer tous nos labels sous la forme `lx: skip`, pour se rapprocher le plus possible des programmes annotés présentés dans ce document. Cependant, la syntaxe de PlusCal requiert qu'un label soit présent devant chaque instruction `while`. Nous avons donc introduit les labels `ul`x, pour UselessLabel$x$, qui ne sont d'aucune utilité dans le reste du model checking...

5.6 Définition de l'opérateur Enum sur TLA+

TLA+ permet la définition d'opérateurs personnalisés, et nous avons donc pu définir l'opérateur Enum. Cependant, tester l'égalité entre $\text{Enum}_n\,(tab)$ et $\text{Enum}_n\,(tab_{out})$ requiert de vérifier que deux fonctions dont le domaine de départ est infini ($\mathbb{Z}$) sont égales, ce dont TLA+ ne semble pas capable directement. Nous avons donc restreint l'ensemble de définition de la fonction $\text{Enum}_n\,(tab_{out})$ aux valeurs effectivement présentes dans le tableau de départ (tab), ce qui est possible car tab est une constante dans le code TLA+. Ce n'est pas gênant car il est facile de montrer que $\text{Enum}_n\,(tab) \neq \text{Enum}_n\,(tab_{out})$ si et seulement si il en est de même pour la version TLA+.

5.7 Valeur par défaut des fonctions sur TLA+

À notre grand étonnement, il semble qu'en PlusCal l'opération `f(i) := v` ne peut altérer la fonction f que si $i \in \mathrm{Dom}(f)$ (contrairement au comportement de Rodin, qui étend le domaine de définition de la fonction partielle si nécessaire pour réaliser l'affectation).

Pour pallier ce problème, nous avons défini à l'initialisation les fonctions avec un domaine égal à leur domaine futur, ce qui est possible car n, qui dans les trois programmes borne le tableau-fonction, est une constante pour TLA+.

5.8 Résultats obtenus sur TLA+

Nos trois programmes TLA+ contiennent l'intégralité des annotations présentées dans ce document, et le model checking passe sur des exemples. Les annotations, même celles avec Enum, ne sont donc pas réfutées par TLA+ et semblent correctes.

5.9 Résultats obtenus sur Frama-C

Les résultats que nous avons obtenus sur Frama-C sont plutôt maigres : en effet, il semblerait que pour prouver l'absence d'overflow/underflow et la validité des accès mémoire, il faille ajouter un système d'annotation semblable à celui développé dans ce document, ce qui requiert un temps conséquent. De plus, par la preuve Rodin de chacun de nos codes, nous avons pu prouver l'absence de RTE avec de simples arguments mathématiques. Nous avons donc pu d'une certaine manière nous passer de Frama-C (puisqu'en mode WP, il faisait doublon avec Rodin).

Nous aurions toutefois aimé pouvoir nous faire la main avec le logiciel qui semble très intéressant sur des applications pratiques de la preuve de programme. Peut-être pour un prochain projet !

6 Ressentis personnels et conclusion

6.1 Ressentis de Timothée

Le projet nous a demandé du temps et de l'investissement mais les résultats sont à la hauteur de nos attentes ce qui est très satisfaisant. De plus le projet m'a permis de découvrir plus en détail Rodin et le système d'annotations ainsi que leur potentiel. Thomas a su nous expliquer et nous a permis d'apprendre de nos erreurs ce qui a été très enrichissant et nous a très certainement permis d'apprendre plus de chose que sans son soutien. L'ambiance au sein du groupe a également permis d'établir un cadre de travail agréable.

6.2 Ressentis d'Alexandre

Projet très intéressant qui nous a demandé beaucoup de réflexion sur la gestion des tableaux (une piste d'amélioration sur la preuve avec Rodin reste d'ailleurs ouverte). Groupe de travail toujours aussi agréable et un grand merci a Thomas pour l'aide qu'il m'a apporté sur ce projet.

6.3 Ressentis de Thomas

Ce projet a été très instructif et intéressant à mener. Les programmes proposés présentaient des difficultés différentes, et chacun leur spécificité. Si les outils de preuves n'étaient pas forcément très simples à prendre en main, je suis content à la fin de ce projet d'avoir bien compris les possibilités, cas d'usages et limitations de ces derniers. Je suis également fier d'avoir réussi avec mon groupe à prouver les trois programmes, le tout dans une ambiance extrêmement agréable et conviviale.

6.4 Conclusion

Ce projet très complet nous à permis de mettre en pratique une grande partie des connaissances apprises durant les cours et TPs de MVSI. Nous avons également pu exploiter les outils Rodin et TLA à leur plein potentiel et découvrir l'outil Frama-C. Enfin, ce projet a également été l'occasion de renforcer l'efficacité de notre groupe de travail, en développant encore plus l'organisation et l'entraide.